

Oracle® Machine Learning for Python User's Guide



Release 2.0

F85504-04

May 2024

The Oracle logo, consisting of a solid red square with the word "ORACLE" in white, uppercase, sans-serif font centered within it.

ORACLE®

Oracle Machine Learning for Python User's Guide, Release 2.0

F85504-04

Copyright © 2024, 2024, Oracle and/or its affiliates.

Primary Author: Dhanish Kumar

Contributors: Andi Wang , Boriana Milenova , David McDermid , Feng Li , Mandeep Kaur , Mark Hornick, Qin Wang , Sherry Lamonica , Venkatanathan Varadarajan , Yu Xiang

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Audience	ix
Related Resources	ix
Conventions	ix
Documentation Accessibility	x

1 Changes in This Release for Oracle Machine Learning for Python

1.1 New Features in 23ai	1-2
--------------------------	-----

2 About Oracle Machine Learning for Python

2.1 What Is Oracle Machine Learning for Python?	2-1
2.2 Advantages of Oracle Machine Learning for Python	2-2
2.3 Manipulate database tables and views using familiar Python functions and syntax	2-4
2.4 About the Python Components and Libraries in OML4Py	2-6

3 Install OML4Py Client for Linux for Use With Autonomous Database Serverless

4 Install OML4Py for On-Premises Databases

4.1 OML4Py On Premises System Requirements	4-1
4.2 Build and Install Python for Linux for On-Premises Databases	4-2
4.2.1 Commands Summary for Building and Installing Python for Linux for On-Premises Databases	4-4
4.3 Install the Required Supporting Packages for Linux for On-Premises Databases	4-5
4.3.1 Commands Summary for Installing Required Supporting Packages for Linux for On-Premises Databases	4-6
4.4 Install OML4Py Server for On-Premises Oracle Database	4-7
4.4.1 Install OML4Py Server for Linux for On-Premises Oracle Database 23ai	4-7
4.4.1.1 Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases	4-11
4.4.2 Verify OML4Py Installation for On-Premises Database	4-11

4.4.3	Grant Users the Required Privileges for On-Premises Database	4-12
4.4.4	Create New Users for On-Premises Oracle Database	4-13
4.4.5	Uninstall the OML4Py Server from an On-Premises Database 23ai	4-15
4.5	Install OML4Py Client for On-Premises Oracle Database	4-16
4.5.1	Install Oracle Instant Client and the OML4Py Client for Linux	4-16
4.5.1.1	Install Oracle Instant Client for Linux for On-Premises Databases	4-17
4.5.1.2	Install OML4Py Client for Linux for On-Premises Databases	4-18
4.5.1.3	Commands Summary for Installing Oracle Instant Client and OML4Py Client for On-Premises Oracle Database	4-21
4.5.2	Verify OML4Py Client Installation for On-Premises Databases	4-21
4.5.3	Uninstall the OML4Py Client for On-Premises Databases	4-22

5 Install OML4Py on Exadata

5.1	About Oracle Machine Learning for Python on Exadata	5-1
5.2	Configure DCLI to install Python across Exadata compute nodes.	5-2
5.2.1	Install Python across Exadata compute nodes using DCLI	5-4
5.2.2	Install OML4Py across Exadata compute nodes using DCLI	5-5

6 Install Third-Party Packages

6.1	Conda Commands	6-1
6.2	Administrative Tasks for Creating and Saving a Conda Environment	6-9
6.3	OML User Tasks for Downloading an Available Conda Environment	6-13
6.4	Using Conda Environments with Embedded Python Execution	6-19

7 Get Started with Oracle Machine Learning for Python

7.1	Use OML4Py with Oracle Autonomous Database	7-1
7.2	Use OML4Py with an On-Premises Oracle Database	7-1
7.2.1	About Connecting to an On-Premises Oracle Database	7-2
7.2.2	About Oracle Wallets	7-3
7.2.3	Connect to an Oracle Database	7-4
7.3	Move Data Between the Database and a Python Session	7-8
7.3.1	About Moving Data Between the Database and a Python Session	7-9
7.3.2	Push Local Python Data to the Database	7-9
7.3.3	Pull Data from the Database to a Local Python Session	7-11
7.3.4	Create a Python Proxy Object for a Database Object	7-13
7.3.5	Create a Persistent Database Table from a Python Data Set	7-15
7.4	Save Python Objects in the Database	7-19
7.4.1	About OML4Py Datastores	7-20
7.4.2	Save Objects to a Datastore	7-21
7.4.3	Load Saved Objects From a Datastore	7-24

7.4.4	Get Information About Datastores	7-25
7.4.5	Get Information About Datastore Objects	7-27
7.4.6	Delete Datastore Objects	7-28
7.4.7	Manage Access to Stored Objects	7-30

8 Prepare and Explore Data

8.1	Prepare Data	8-1
8.1.1	About Preparing Data in the Database	8-1
8.1.2	Select Data	8-4
8.1.3	Combine Data	8-8
8.1.4	Clean Data	8-13
8.1.5	Split Data	8-15
8.2	Explore Data	8-17
8.2.1	About the Exploratory Data Analysis Methods	8-18
8.2.2	Correlate Data	8-20
8.2.3	Cross-Tabulate Data	8-22
8.2.4	Mutate Data	8-24
8.2.5	Sort Data	8-27
8.2.6	Summarize Data	8-30
8.2.7	Date, Time, and Integer Data	8-32
8.3	Render Graphics	8-40

9 OML4Py Classes That Provide Access to In-Database Machine Learning Algorithms

9.1	About Machine Learning Classes and Algorithms	9-2
9.2	About Model Settings	9-4
9.3	Shared Settings	9-5
9.4	Export Oracle Machine Learning for Python Models	9-7
9.5	Automatic Data Preparation	9-11
9.6	Model Explainability	9-12
9.7	Attribute Importance	9-18
9.8	Association Rules	9-21
9.9	Decision Tree	9-27
9.10	Expectation Maximization	9-34
9.11	Explicit Semantic Analysis	9-48
9.12	Generalized Linear Model	9-53
9.13	k-Means	9-63
9.14	Naive Bayes	9-69
9.15	Neural Network	9-77
9.16	Random Forest	9-86

9.17	Singular Value Decomposition	9-94
9.18	Support Vector Machine	9-100
9.19	Non-Negative Matrix Factorization	9-106
9.20	Exponential Smoothing Method	9-112
9.21	XGBoost	9-119

10 Automated Machine Learning

10.1	About Automated Machine Learning	10-1
10.2	Algorithm Selection	10-6
10.3	Feature Selection	10-8
10.4	Model Tuning	10-11
10.5	Model Selection	10-15

11 Convert Pretrained Models to ONNX Format

11.1	Convert Pretrained Models to ONNX Model: End-to-End Instructions	11-6
11.2	Python Classes to Convert Pretrained Models to ONNX Models	11-9

12 Embedded Python Execution

12.1	About Embedded Python Execution	12-2
12.1.1	Comparison of the Embedded Python Execution APIs	12-2
12.2	Parallelism with OML4Py Embedded Python Execution	12-5
12.3	Datastores Supporting Embedded Python Execution	12-6
12.3.1	ALL_PYQ_DATASTORE_CONTENTS View	12-7
12.3.2	ALL_PYQ_DATASTORES View	12-8
12.3.3	USER_PYQ_DATASTORES View	12-9
12.4	Script repository for user-defined Python functions supporting EPE	12-10
12.4.1	ALL_PYQ_SCRIPTS View	12-10
12.4.2	USER_PYQ_SCRIPTS View	12-11
12.5	Python API for Embedded Python Execution	12-11
12.5.1	About Python API for Embedded Python Execution	12-12
12.5.2	Run a User-Defined Python Function	12-14
12.5.3	Run a User-Defined Python Function on the Specified Data	12-15
12.5.4	Run a Python Function on Data Grouped By Column Values	12-18
12.5.5	Run a User-Defined Python Function on Sets of Rows	12-22
12.5.6	Run a User-Defined Python Function Multiple Times	12-25
12.5.7	Save and Manage User-Defined Python Functions in the Script Repository	12-27
12.5.7.1	About the Script Repository	12-28
12.5.7.2	Create and Store a User-Defined Python Function	12-28
12.5.7.3	List Available User-Defined Python Functions	12-31

12.5.7.4	Load a User-Defined Python Function	12-33
12.5.7.5	Drop a User-Defined Python Function from the Repository	12-34
12.6	SQL API for Embedded Python Execution with On-premises Database	12-35
12.6.1	About the SQL API for Embedded Python Execution with On-Premises Database	12-36
12.6.2	pyqEval Function (On-Premises Database)	12-37
12.6.3	pyqTableEval Function (On-Premises Database)	12-40
12.6.4	pyqRowEval Function (On-Premises Database)	12-43
12.6.5	pyqGroupEval Function (On-Premises Database)	12-47
12.6.6	pyqGrant Function (On-Premises Database)	12-50
12.6.7	pyqRevoke Function (On-Premises Database)	12-51
12.6.8	pyqScriptCreate Procedure (On-Premises Database)	12-52
12.6.9	pyqScriptDrop Procedure (On-Premises Database)	12-55
12.7	SQL API for Embedded Python Execution with Autonomous Database	12-56
12.7.1	Access and Authorization Procedures and Functions	12-56
12.7.1.1	pyqAppendHostACE Procedure	12-59
12.7.1.2	pyqGetHostACE Function	12-59
12.7.1.3	pyqRemoveHostACE Procedure	12-60
12.7.1.4	pyqSetAuthToken Procedure	12-60
12.7.1.5	pyqIsTokenSet Function	12-60
12.7.2	Embedded Python Execution Functions (Autonomous Database)	12-61
12.7.2.1	pyqListEnvs Function (Autonomous Database)	12-62
12.7.2.2	pyqEval Function (Autonomous Database)	12-62
12.7.2.3	pyqTableEval Function (Autonomous Database)	12-68
12.7.2.4	pyqRowEval Function (Autonomous Database)	12-72
12.7.2.5	pyqGroupEval Function (Autonomous Database)	12-80
12.7.2.6	pyqIndexEval Function (Autonomous Database)	12-88
12.7.2.7	pyqGrant Function (Autonomous Database)	12-112
12.7.2.8	pyqRevoke Function (Autonomous Database)	12-113
12.7.2.9	pyqScriptCreate Procedure (Autonomous Database)	12-114
12.7.2.10	pyqScriptDrop Procedure (Autonomous Database)	12-116
12.7.3	Asynchronous Jobs (Autonomous Database)	12-117
12.7.3.1	oml_async_flag Argument	12-117
12.7.3.2	pyqJobStatus Function	12-119
12.7.3.3	pyqJobResult Function	12-119
12.7.3.4	Asynchronous Job Example	12-121
12.7.4	Special Control Arguments (Autonomous Database)	12-126
12.7.5	Output Formats (Autonomous Database)	12-127

13 Administrative Tasks for Oracle Machine Learning for Python

Index

Preface

This publication describes Oracle Machine Learning for Python (OML4Py) and how to use it.

- [Audience](#)
- [Related Resources](#)
- [Conventions](#)
- [Documentation Accessibility](#)

Audience

Related Resources

Related documentation is in the following publications:

- [Oracle Machine Learning for Python API Reference](#)
- [Oracle Machine Learning for Python Known Issues](#)
- [Oracle Machine Learning for Python Licensing Information User Manual](#)
- [REST API for Embedded Python Execution](#)
- [Get Started with Notebooks for Data Analysis and Data Visualization in Using Oracle Machine Learning Notebooks](#)
- [Oracle Machine Learning AutoML User Interface](#)
- [REST API for Oracle Machine Learning Services](#)

For more information, see these Oracle resources:

- [Oracle Machine Learning Technologies](#)
- [Oracle Autonomous Database](#)

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.

Convention	Meaning
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

1

Changes in This Release for Oracle Machine Learning for Python

Describes changes in Oracle Machine Learning for Python User's Guide for Oracle Database 23ai.

New Features in 23ai

Table 1-1 New Features

Features	Description
Support for in-database machine learning Non-Negative Matrix Factorization algorithm, Exponential Smoothing Method algorithm and XGBoost algorithm.	The following functions are new in the package that use in-database algorithms: <ul style="list-style-type: none">• <code>oml.nmf</code>, Non-Negative Matrix Factorization model• <code>oml.esm</code>, Exponential Smoothing Method model• <code>oml.xgb</code>, XGboost model
Support for data types that enable you to manipulate date, time and integer.	The following data types are supported by OML4Py: <ul style="list-style-type: none">• <code>oml.Datetime</code>, to create date.• <code>oml.Timezone</code>, to create time and timezone that includes hour, minute, second, microsecond, and tzone.• <code>oml.Timedelta</code>, to perform simple arithmetic operations.• <code>oml.Integer</code> to represent the integer data type.

Convert Pretrained Models to ONNX Format

Oracle Machine Learning for Python supports ONNX format models. To learn more, see [Convert Pretrained Models to ONNX Format](#).

The following topic tells about new features added in 23ai.

Topic:

- [New Features in 23ai](#)
Oracle Machine Learning for Python: new features in Oracle Database 23ai.

1.1 New Features in 23ai

Oracle Machine Learning for Python: new features in Oracle Database 23ai.

Algorithm Enhancements

 **Note:**

New Algorithm Settings: You can find model settings and algorithm specific settings in *Oracle Database PL/SQL Packages and Types Reference* guide. See Oracle Database PL/SQL Packages and Types Reference guide.

- **GLM link functions**

`GLMS_LINK_FUNCTION`: this setting enables the user to specify the link function for building a generalized linear model. The additional link functions are: Logit, Probit, Cloglog, and Cauchit. See [Generalized Linear Model](#).

- **XGBoost**

The following new settings are added for XGBoost support for constraints and survival analysis.

 **Note:**

The XGBoost settings are case sensitive.

- **Interaction and Monotonic Constraints**

- * `xgboost_interaction_constraints`
- * `xgboost_decrease_constraints`
- * `xgboost_increase_constraints`

- **Support for Survival Analysis**

- * `objective: survival: aft`
- * `xgboost_aft_loss_distribution`
- * `xgboost_aft_loss_distribution_scale`
- * `xgboost_aft_right_bound_column_name`

Oracle Machine Learning supports XGBoost features such as monotonic and interaction constraints, as well as the AFT model for survival analysis. See [XGBoost](#).

- **Explicit Semantic Analysis (ESA)**

The following settings are added to support generate embeddings through Explicit Semantic Analysis embeddings:

- `ESAS_EMBEDDINGS`: when enabled, generates embeddings during scoring for feature extraction models.
- `ESAS_EMBEDDING_SIZE`: specifies the size of the vectors representing embeddings.

Supports embeddings for the Explicit Semantic Analysis (ESA) algorithm. ESA embeddings enables you to utilize ESA models to generate embeddings for any text or other ESA input. This functionality is equivalent to doc2vec (document to vector representation). See [Explicit Semantic Analysis](#).

- **Expectation Maximization**

`EMCS_OUTLIER_RATE`: identifies the frequency of outliers in the training data. See [Expectation Maximization](#).

- **Exponential Smoothing Model**

New settings for Exponential Smoothing to support Time Series regression models and initial value optimization for model build:

- **Multiple time series**

`EXSM_SERIES_LIST`: setting enables you to forecast up to twenty predictor series in addition to the target series.

- **Automated model type search**

`EXSM_INITVL_OPTIMIZE`: determines whether initial values are optimized during model build.

Exponential Smoothing is enhanced to support building of multiple time series models and time series regression is possible with the multi-series build. The behavior of Exponential Smoothing is modified such that it searches for an acceptable time series model automatically. Enables the algorithm to select the best model type automatically when you do not specify `EXSM_MODEL` setting. This leads to more accurate forecasting. For details, see [Exponential Smoothing Method](#).

- **K-Means**

`KMNS_WINSORIZE`: this setting restricts the data in a window size of six standard deviations around the mean. See [k-Means](#).

General Enhancements

- **New shared settings**

- `ODMS_BOXCOX`: this setting enables the Box-Cox variance-stabilization transformation.
- `ODMS_EXPLOSION_MIN_SUPP`: introduced more efficient data driven encoding for high cardinality categorical columns. You can define minimum support required for the categorical values in explosion mapping.

See [Shared Settings](#).

- **Convert Pretrained Models to ONNX Format**

OML4Py enables the use of text transformers from Hugging Face by converting them into ONNX format models. OML4Py also adds the necessary tokenization and post-processing. The resulting ONNX pipeline is then imported into the database and can be used to generate embeddings for AI Vector Search. See [Convert Pretrained Models to ONNX Format](#).

- **Model Includes Data Lineage**

In-database ML models now record the query string that was run to specify the build data within the model's metadata. The `build_source` parameter in the `all/user/dba_mining_models` view enables users to know the data query used to produce the model. See `ALL_MINING_MODELS`.

- **Improved Performance of Partitioned Models**

Performance of partitioned models with high number of partitions and dropping individual models within partition model is improved. To know more about partitioned models, see DDL in Partitioned model.

2

About Oracle Machine Learning for Python

The following topics describe Oracle Machine Learning for Python (OML4Py) and its advantages for the Python user.

- [What Is Oracle Machine Learning for Python?](#)
Oracle Machine Learning for Python (OML4Py) enables you to run Python commands for data transformations and for statistical, machine learning, and graphical analysis on data stored in or accessible through an Oracle database using a Python API. The OML4Py supports running user-defined Python functions through the database spawned and controlled Python engines, with optional built-in data-parallelism and task-parallelism. This embedded execution functionality enables invoking user-defined functions from SQL, and on ADB, REST. The OML4Py supports Automated Machine Learning (AutoML) for algorithm and feature selection, and model tuning and selection. You can augment the Python included functionality with third-party packages from the Python ecosystem.
- [Advantages of Oracle Machine Learning for Python](#)
Using OML4Py to prepare and analyze data in or accessible to an Oracle database has many advantages for a Python user.
- [Manipulate database tables and views using familiar Python functions and syntax](#)
With the transparency layer classes, you can manipulate database tables and views using familiar Python functions and syntax. For example, using DataFrame proxy objects that map to database data, users can invoke overloaded Pandas functions that transparently generate SQL that runs in the database, using the database as a high-performance compute engine.
- [About the Python Components and Libraries in OML4Py](#)
OML4Py requires an installation of Python, the specified Python libraries, as well as the OML4Py components.

2.1 What Is Oracle Machine Learning for Python?

Oracle Machine Learning for Python (OML4Py) enables you to run Python commands for data transformations and for statistical, machine learning, and graphical analysis on data stored in or accessible through an Oracle database using a Python API. The OML4Py supports running user-defined Python functions through the database spawned and controlled Python engines, with optional built-in data-parallelism and task-parallelism. This embedded execution functionality enables invoking user-defined functions from SQL, and on ADB, REST. The OML4Py supports Automated Machine Learning (AutoML) for algorithm and feature selection, and model tuning and selection. You can augment the Python included functionality with third-party packages from the Python ecosystem.

OML4Py is a Python module that enables Python users to manipulate data in database tables and views using Python syntax. OML4Py functions and methods transparently translate a select set of Python functions into SQL for in-database execution.

OML4Py is available in the following Oracle database environments:

- OML4Py is available in the Python interpreter in Oracle Machine Learning Notebooks in your Oracle Autonomous Database. For more information, see *Get Started with Notebooks for Data Analysis and Data Visualization* in *Using Oracle Machine Learning Notebooks*.

- An OML4Py client connection to OML4Py in an on-premises Oracle Database instance.
For this environment, you must install Python, the required Python libraries, and the OML4Py server components in the database, and you must install the OML4Py client. See [Install OML4Py for On-Premises Databases](#).

Designed for problems involving both large and small volumes of data, OML4Py integrates Python with the database. With OML4Py, you can do the following:

- Run overloaded Python functions and use native Python syntax to manipulate in-database data, without having to learn SQL.
- Use Automated Machine Learning (AutoML) to enhance user productivity and machine learning results through automated algorithm and feature selection, as well as model tuning and selection.
- Use Embedded Python Execution to run user-defined Python functions in Python engines spawned and managed by the database environment. The user-defined functions and data are automatically loaded to the engines as required, and when data-parallel and task-parallel execution is enabled. Develop, refine, and deploy user-defined Python functions and machine learning models that leverage the parallelism and scalability of the database to automate data preparation and machine learning.
- Use a natural Python interface to build in-database machine learning models.

2.2 Advantages of Oracle Machine Learning for Python

Using OML4Py to prepare and analyze data in or accessible to an Oracle database has many advantages for a Python user.

With OML4Py, you can do the following:

- **Operate on database data without using SQL**

OML4Py transparently translates many standard Python functions into SQL. With OML4Py, you can create Python proxy objects that access, analyze, and manipulate data that resides in the database. OML4Py can automatically optimize the SQL by taking advantage of column indexes, query optimization, table partitioning, and database parallelism.

OML4Py overloaded functions are available for many commonly used Python functions, including those on Pandas data frames for in-database execution.

See Also: [Manipulate database tables and views using familiar Python functions and syntax](#)

- **Automate common machine learning tasks**

By using Oracle's advanced Automated Machine Learning (AutoML) technology, both data scientists and beginner machine learning users can automate common machine learning modeling tasks such as algorithm selection and feature selection, and model tuning and selection, all of which leverage the parallel processing and scalability of the database.

See Also: [About Automated Machine Learning](#)

- **Minimize data movement**

By keeping data in the database whenever possible, you eliminate the time involved in transferring the data to your client Python engine and the need to store the data locally. You also eliminate the need to manage the locally stored data, which includes tasks such as distributing the data files to the appropriate locations, synchronizing the data with changes that are made in the production database, and so on.

See Also: [About Moving Data Between the Database and a Python Session](#)

- **Keep data secure**

By keeping the data in the database, you have the security, scalability, reliability, and backup features of the database for managing the data.

- **Use the power of the database**

By operating directly on data in the database, you can use the memory and processing power of the database and avoid the memory constraints of your client Python engine.

- **Use current data**

As data is refreshed in the database, you have immediate access to current data.

- **Save Python objects to a datastore in the database**

You can save Python objects to an OML4Py datastore for future use and for use by others.

See Also: [About OML4Py Datastores](#)

- **Build and store native Python models in the database**

Using Embedded Python Execution, you can build native Python models and store and manage them in an OML4Py datastore.

You can also build in-database models, with, for example, an `oml` class such as the Decision Tree class `oml.dt`. These in-database models have proxy objects that reference the actual models. Keeping with normal Python behavior, when the Python engine terminates, all in-memory objects, including models, are lost. To prevent an in-database model created using OML4Py from being deleted when the database connection is terminated, you must store its proxy object in a datastore.

See Also: [About Machine Learning Classes and Algorithms](#)

- **Score data**

For most of the OML4Py machine learning classes, you can use the `predict` and `predict_proba` methods of the model object to score new data.

For these OML4Py in-database models, you can also use the SQL `PREDICTION` function on the model proxy objects, which scores directly in the database. You can use in-database models directly from SQL if you prepare the data properly. For open source models, you can use Embedded Python Execution and enable data-parallel execution for performance and scalability.

- **Run user-defined Python functions in embedded Python engines**

Using OML4Py Embedded Python Execution, you can store user-defined Python functions in the OML4Py script repository, and run those functions in Python engines spawned by the database environment. When a user-defined Python function runs, the database starts, controls, and manages one or more Python engines that can run in parallel. With the Embedded Python Execution functionality, you can do the following:

- Use a select set of Python packages in user-defined functions that run in embedded Python engines
- Use other Python packages and third-party package in user-defined Python functions that run in embedded Python engines
- Operationalize user-defined Python functions for use in production applications and eliminate porting Python code and models into SQL, and on ADB, REST; avoid reinventing code to integrate Python results into existing applications

- Seamlessly leverage your Oracle database as a high-performance computing environment for user-defined Python functions, providing data parallelism and resource management
- Perform parallel simulations, for example, Monte Carlo analysis, using the `oml.index_apply` function
- Generate JSON images, PNG images and XML representations of both structured and image data, which can be used by Python clients and SQL-based applications. PNG images and structured data can be used for Python clients and applications that use REST APIs.

See Also: [About Embedded Python Execution](#)

2.3 Manipulate database tables and views using familiar Python functions and syntax

With the transparency layer classes, you can manipulate database tables and views using familiar Python functions and syntax. For example, using DataFrame proxy objects that map to database data, users can invoke overloaded Pandas functions that transparently generate SQL that runs in the database, using the database as a high-performance compute engine.

The OML4Py transparency layer does the following:

- Enables creating tables and views from `pandas.DataFrame` and getting proxy objects to tables and views.
- Overloads specific Python functions that transparently translate functionality to SQL
- Leverages proxy objects for database data
- Uses familiar Python syntax to manipulate database data

The following table lists the transparency layer functions for getting and creating proxy objects and tables/views.

Table 2-1 Transparency Layer Functions for getting and creating proxy objects and tables/views

Function	Description
<code>oml.create</code>	Creates a table in a the database schema from a Python data set.
<code>oml_object.pull</code>	Creates a local Python object that contains a copy of data fetched from database object referenced by the <code>oml</code> object.
<code>oml.push</code>	Pushes data from a Python session into an object in a database schema.
<code>oml.sync</code>	Creates a <code>DataFrame</code> proxy object in Python that represents a database table or view.
<code>oml.dir</code>	Return the names of <code>oml</code> objects in the Python session workspace.
<code>oml.drop</code>	Drops a persistent database table or view.

Transparency layer proxy classes map SQL data types or objects to corresponding Python types. The classes provide Python functions and operators that are the same as those on the mapped Python types. The following table lists the transparency layer data type classes.

Table 2-2 Transparency Layer Data Type Classes

Class	Description
<code>oml.Boolean</code>	A boolean series data class that represents a single column of 0, 1, and NULL values in database data.
<code>oml.Bytes</code>	A binary series data class that represents a single column of RAW or BLOB database data types.
<code>oml.Float</code>	A numeric series data class that represents a single column of NUMBER, BINARY_DOUBLE, or BINARY_FLOAT database data types.
<code>oml.String</code>	A character series data class that represents a single column of VARCHAR2, CHAR, or CLOB database data types.
<code>oml.DataFrame</code>	A tabular DataFrame class that represents multiple columns of <code>oml.Boolean</code> , <code>oml.Bytes</code> , <code>oml.Float</code> , and <code>oml.String</code> data.
<code>oml.Integer</code>	A data class that represents a single column of NUMBER(*,0) data in the database.
<code>oml.Datetime</code>	A series date class that represents a single column of TIMESTAMP or TIMESTAMP WITH TIME ZONE in Oracle Database. <code>oml.Timezone</code> A time class that is used with <code>oml.Datetime</code> to support TIME STAMP WITH TIME ZONE. <code>oml.Timedelta</code> A time class that represents a single column series of differences between two dates or times, or INTERVAL DAY TO SECOND in Oracle Database.
<code>oml.Timezone</code>	A time class that is used with <code>oml.Datetime</code> to support TIME STAMP WITH TIME ZONE.
<code>oml.Timedelta</code>	A time class that represents a single column series of differences between two dates or times, or INTERVAL DAY TO SECOND in Oracle Database.

The following table lists the mappings of Python data types for both the reading and writing of data between Python and the database.

Table 2-3 Python and SQL Data Type Equivalencies

Database Read	Python Data Types	Database Write
N/A	Bool	If <code>oracolumn == True</code> , then NUMBER (the default), else BINARY_DOUBLE.
BLOB	bytes	BLOB
RAW		RAW
BINARY_DOUBLE	float	If <code>oracolumn == True</code> , then NUMBER (the default), else BINARY_DOUBLE.
BINARY_FLOAT		
NUMBER		
CHAR	str	CHAR
CLOB		CLOB
VARCHAR2		VARCHAR2
NUMBER(*,0)	int	NUMBER(*,0)
TIMESTAMP or TIMESTAMP WITH TIME ZONE	<code>datetime.datetime</code>	TIMESTAMP or TIMESTAMP WITH TIME ZONE
TIMESTAMP WITH TIME ZONE	<code>datetime.timezone</code>	TIMESTAMP WITH TIME ZONE

Table 2-3 (Cont.) Python and SQL Data Type Equivalencies

Database Read	Python Data Types	Database Write
INTERVAL DAY TO SECOND	datetime.timedelta	INTERVAL DAY TO SECOND

2.4 About the Python Components and Libraries in OML4Py

OML4Py requires an installation of Python, the specified Python libraries, as well as the OML4Py components.

- In Oracle Autonomous Database, OML4Py is already installed. The OML4Py installation includes Python, additional required Python libraries, and the OML4Py server components. A Python interpreter is included with Oracle Machine Learning Notebooks in Autonomous Database.
- You can install third-party Python libraries in a conda environment through a conda interpreter for use within OML Notebooks sessions and OML4Py embedded execution invocations.
- You can install OML4Py in an on-premises Oracle Database. In this case, you must install Python, the additional required Python libraries, the OML4Py server components, and an OML4Py client. See [Install OML4Py for On-Premises Databases](#).

Python Version in Current Release of OML4Py

The current release of OML4Py is based on Python 3.12.0.

This version is in the current release of Oracle Autonomous Database.

Required Python Libraries

The following Python libraries must be included.

- oracledb 2.0.1
- cycler 0.10.0
- joblib 1.1.0
- kiwisolver 1.1.0
- matplotlib 3.8.4
- numpy 1.26.4
- pandas 2.1.1
- Pillow-8.2.0
- pyparsing 2.4.0
- python-dateutil 2.8.1
- pytz 2022.1
- scikit-learn 1.4.1post1
- scipy 1.12.0
- six 1.13.0

- `threadpoolctl 3.1.0`

All the above libraries are included with Python in the current release of Oracle Autonomous Database.

For an installation of OML4Py in an on-premises Oracle Database, you must install Python and additionally the libraries listed here. See [Install OML4Py for On-Premises Databases](#).

3

Install OML4Py Client for Linux for Use With Autonomous Database Serverless

You can install and use the OML4Py client for Linux to work with OML4Py in an Oracle Autonomous Database on Serverless Exadata infrastructure.

OML4Py on premises runs on 64-bit platforms only. For supported platforms see [OML4Py On Premises System Requirements](#).

The following instructions tell you how to download install Python, configure your environment, install manage your client credentials, install Oracle Instant Client, and install the OML4Py client:

1. Download the Python 3.12.0 source and untar it:

```
wget https://www.python.org/ftp/python/3.12.0/Python-3.12.0.tar.xz
tar xvf Python-3.12.0.tar.xz
```

2. OML4Py requires the presence of the `perl-Env`, `libffi-devel`, `openssl`, `openssl-devel`, `tk-devel`, `xz-devel`, `zlib-devel`, `bzip2-devel`, `readline-devel`, `libuuid-devel` and `ncurses-devel` libraries. Install these packages as `sudo` or root user:

```
sudo yum install perl-Env libffi-devel openssl openssl-devel tk-devel xz-devel
zlib-devel bzip2-devel readline-devel libuuid-devel ncurses-devel
```

3. To build Python, enter the following commands, where `PREFIX` is the directory in which you installed Python-3.12.0. Use `make altinstall` to avoid overriding the system default's Python installation.

```
export PREFIX=`pwd`/Python-3.12.0
cd $PREFIX
./configure --prefix=$PREFIX --enable-shared

make clean; make
make altinstall
```

4. Set environment variable `PYTHONHOME` and add it to your `PATH`, and set environment variable `LD_LIBRARY_PATH`:

```
export PYTHONHOME=$PREFIX
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
```

Create a symbolic link in your `$PYTHONHOME/bin` directory. You need to link it to your Python 3.12.0 executable, which you can do with the following commands:

```
cd $PYTHONHOME/bin
ln -s python3.12 python3
```

You can now start Python with the `python3` script:

```
python3
```

`pip` will return warnings during package installation if the latest version is not installed. You can upgrade the version of `pip` to avoid these warnings:

```
python3 -m pip install --upgrade pip
```

5. Install the Oracle Instant Client for Autonomous Database, as follows:

Download the Oracle Instant Client for your system. Go to the [Oracle Instant Client Downloads](#) page and select **Instant Client for Linux x86-64**. For more instruction see [Install Oracle Instant Client for Linux for On-Premises Databases](#).

For instruction on installing the Oracle instant client for on-premises see [Install OML4Py Client for On-Premises Oracle Database](#).

If you have root access to install an RPM on the client system. Alternatively, you can also download the zip file installer, unzip the file, and add the location of the unzipped file to `LD_LIBRARY_PATH` as done in next section.

```
wget https://download.oracle.com/otn_software/linux/instantclient/1914000/
oracle-instantclient19.14-basic-19.14.0.0-1.x86_64.rpm
rpm -ivh oracle-instantclient19.14-basic-19.14.0.0-1.x86_64.rpm
export LD_LIBRARY_PATH=/usr/lib/oracle/19.14/client64/lib:$LD_LIBRARY_PATH
```

If you do not have root access to install an RPM on the client system.

```
https://download.oracle.com/otn_software/linux/instantclient/2340000/
instantclient-basic-linux.x64-23.4.0.24.05dbru.zip
instantclient-basic-linux.x64-23.4.0.24.05dbru.zip
export LD_LIBRARY_PATH=/path/to/instantclient_23_4:$LD_LIBRARY_PATH
```

6. Download the client credentials (wallet) from your Autonomous database. Create a directory for the Wallet contents. Unzip the wallet zip file to the newly created directory:

 **Note:**

An mTLS connection using the client Wallet is required. TLS connections are not currently supported.

```
mkdir -p mywalletdir
unzip Wallet.name.zip -d mywalletdir
cd mywalletdir/
ls
```

```
README          ewallet.p12    ojdbc.properties  tnsnames.ora
cwallet.sso     keystore.jks  sqlnet.ora        truststore.jks
```

- Update `sqlnet.ora` with the wallet location. If you're working behind a proxy firewall, set the `SQLNET.USE_HTTPS_PROXY` environment variable to `on`:

```
WALLET_LOCATION = (SOURCE = (METHOD = file) (METHOD_DATA =
(DIRECTORY="mywalletdir")))
SSL_SERVER_DN_MATCH=yes
SQLNET.USE_HTTPS_PROXY=on
```

- Add proxy address information to all service levels in `tnsnames.ora`, and add the connection pools for all service levels. If you are behind a firewall, enter the proxy address and port number to all service levels in `tnsnames.ora`. You will also need to add three new entries for the AutoML connection pools as shown below.

 **Note:**

If the proxy server contains a firewall to terminate connections within a set time period, the database connection will also be terminated.

For example, `myadb_medium_pool` is another alias for the connection string with `SERVER=POOLED` added to the corresponding one for `myadb_medium`.

```
myadb_low = (description= (retry_count=20) (retry_delay=3)
(address=(https_proxy=your proxy address here) (https_proxy_port=80)
(protocol=tcps) (port=1522) (host=adb.us-sanjose-1.oraclecloud.com)
(connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com)
(security=(ssl_server_cert_dn="CN=adb.us-sanjose-1.oraclecloud.com,OU=Oracle
ADB SANJOSE,O=Oracle Corporation,L=Redwood City,ST=California,C=US"))))

myadb_medium = (description= (retry_count=20) (retry_delay=3)
(address=(https_proxy=your proxy address here) (https_proxy_port=80)
(protocol=tcps) (port=1522) (host=adb.us-sanjose-1.oraclecloud.com)
(connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com)
(security=(ssl_server_cert_dn="CN=adb.us-sanjose-1.oraclecloud.com,OU=Oracle
ADB SANJOSE,O=Oracle Corporation,L=Redwood City,ST=California,C=US"))))

myadb_high = (description= (retry_count=20) (retry_delay=3)
(address=(https_proxy=your proxy address here) (https_proxy_port=80)
(protocol=tcps) (port=1522) (host=adb.us-sanjose-1.oraclecloud.com)
(connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com)
(security=(ssl_server_cert_dn="CN=adb.us-sanjose-1.oraclecloud.com,OU=Oracle
ADB SANJOSE,O=Oracle Corporation,L=Redwood City,ST=California,C=US"))))

myadb_low_pool = (description= (retry_count=20) (retry_delay=3)
(address=(https_proxy=your proxy address here) (https_proxy_port=80)
(protocol=tcps) (port=1522) (host=adb.us-sanjose-1.oraclecloud.com)
(connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com)
(SERVER=POOLED)) (security=(ssl_server_cert_dn="CN=adb.us-
sanjose-1.oraclecloud.com,OU=Oracle ADB SANJOSE,O=Oracle Corporation,L=Redwood
City,ST=California,C=US"))))

myadb_medium_pool = (description= (retry_count=20) (retry_delay=3)
(address=(https_proxy=your proxy address here) (https_proxy_port=80)
(protocol=tcps) (port=1522) (host=adb.us-sanjose-1.oraclecloud.com)
(connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com)
(SERVER=POOLED)) (security=(ssl_server_cert_dn="CN=adb.us-
```



```
sanjose-1.oraclecloud.com,OU=Oracle ADB SANJOSE,O=Oracle Corporation,L=Redwood
City,ST=California,C=US"))
```

```
myadb_high_pool = (description= (retry_count=20)(retry_delay=3)
(address=(https_proxy=your proxy address here)(https_proxy_port=80)
(protocol=tcps)(port=1522)(host=adb.us-sanjose-1.oraclecloud.com))
(connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com)
(SERVER=POOLED))(security=(ssl_server_cert_dn="CN=adb.us-
sanjose-1.oraclecloud.com,OU=Oracle ADB SANJOSE,O=Oracle Corporation,L=Redwood
City,ST=California,C=US")))
```

9. Set TNS_ADMIN environment variable to the wallet directory:

```
export TNS_ADMIN=mywalletdir
```

10. Install OML4Py library dependencies. The versions listed here are the versions Oracle has tested and supports:

- pip3.12 install pandas==2.1.1
- pip3.12 install scipy==1.12.0
- pip3.12 install matplotlib==3.8.4
- pip3.12 install oracledb==2.0.1
- pip3.12 install threadpoolctl==3.1.0
- pip3.12 install joblib==1.2.0
- pip3.12 install scikit-learn==1.4.1post1
- pip3.12 install setuptools==68.0.0
- pip3.12 install numpy==1.26.4
- Install OML4Py client:

Download OML4Py client installation zip file, go to the [Oracle Machine Learning for Python Downloads](#) page on the Oracle Technology Network. For more instruction see [Install OML4Py Client for Linux for On-Premises Databases](#)

```
unzip oml4py-client-linux-x86_64-2.0.zip
perl -Iclient client/client.pl
```

Oracle Machine Learning for Python 2.0 Client.

Copyright (c) 2018, 2022 Oracle and/or its affiliates. All rights reserved.

```
Checking platform ..... Pass
Checking Python ..... Pass
Checking dependencies ..... Pass
```

```

Checking OML4P version ..... Pass
Current configuration
  Python Version ..... 3.12.0
  PYTHONHOME ..... /opt/Python-3.12.0
  Existing OML4P module version .... None

  Operation ..... Install/Upgrade

Proceed? [yes]
Processing ./client/oml-2.0-cp312-cp312-linux_x86_64.whl
Installing collected packages: oml
Successfully installed oml-2.0

Done

```

- **Start Python and load the `oml` library:**

```
python3
import oml
```

- **Create a database connection. The OML client connects using the wallet. Set the `dsn` and `automl` arguments to the `tnsnames` alias in the wallet:**

```
oml.connect(user="oml_user", password="oml_user_password",
            dsn="myadb_medium", automl="myadb_medium_pool")
```

To provide empty strings for the user and password parameters to connect without exposing your Oracle Machine Learning user credentials in clear text:

```
oml.connect(user="", password="", dsn="myadb_medium",
            automl="myadb_medium_pool")
```

4

Install OML4Py for On-Premises Databases

The following topics tell how to install and uninstall the server and client components required for using OML4Py with an on-premises Oracle Database.

Topics:

- [OML4Py On Premises System Requirements](#)
OML4Py on premises runs on 64-bit platforms only.
- [Build and Install Python for Linux for On-Premises Databases](#)
Instructions for installing Python for Linux for an on-premises Oracle database.
- [Install the Required Supporting Packages for Linux for On-Premises Databases](#)
Both the OML4Py server and client installations for an on-premises Oracle database require that you also install a set of supporting Python packages, as described below.
- [Install OML4Py Server for On-Premises Oracle Database](#)
The following instructions tell how to install and uninstall the OML4Py server components for an on-premises Oracle Database 23ai.
- [Install OML4Py Client for On-Premises Oracle Database](#)
Instructions for installing and uninstalling the on-premises OML4Py client.

4.1 OML4Py On Premises System Requirements

OML4Py on premises runs on 64-bit platforms only.

Both client and server on-premises components are supported on the Linux platforms listed in the table below.

Table 4-1 On-Premises OML4Py Platform Requirements

Operating System	Hardware Platform	Description
Oracle Linux x86-64 8.x	Intel	64-bit Oracle Linux Release 8

Table 4-2 On-Premises OML4Py Configuration Requirements and Server Support Matrix

Oracle Machine Learning for Python Version	Python Version	On-Premises Oracle Database Release
2.0	3.12.0	19c, 21c, 23ai

Note:

Plug in violations are seen after DB upgrade from 21c to 23c with OML4PY configurations.

4.2 Build and Install Python for Linux for On-Premises Databases

Instructions for installing Python for Linux for an on-premises Oracle database.

For 23ai, building Python is only necessary for the client. The database has Python 3.12.1 in `$ORACLE_HOME/python`.

 **Note:**

The feature supporting text transformers from Hugging Face by converting them into ONNX format models will only work on the OML4Py client. It is not supported on the OML4Py server.

Python 3.12.0 or Python 3.12.1 is required to install and use OML4Py.

These steps describe building and installing Python 3.12.0 for Linux.

1. Go to the [Python website](https://www.python.org/) and download the **Gzipped source tarball**. The downloaded file name is `Python-3.12.0.tgz`

```
wget https://www.python.org/ftp/python/3.12.0/Python-3.12.0.tgz
```

2. Create a directory, such as `/home/user/python`, and extract the contents to this directory:

```
mkdir -p /home/user/python  
tar -xvzf Python-3.12.0.tgz --strip-components=1 -C /home/user/python
```

The contents of the Gzipped source tarball will be copied directly to `/home/user/python`

3. Go to the new directory:

```
cd /home/user/python
```

4. OML4Py requires the presence of the `perl-Env`, `libffi-devel`, `openssl`, `openssl-devel`, `tk-devel`, `xz-devel`, `zlib-devel`, `bzip2-devel`, `readline-devel`, `libuuid-devel` and `ncurses-devel` libraries.

Install these packages as `sudo` or root user:

```
sudo yum install perl-Env libffi-devel openssl openssl-devel tk-devel xz-devel  
zlib-devel bzip2-devel readline-devel libuuid-devel ncurses-devel
```

5. To build Python 3.12.0, enter the following commands, where `/home/user/python` is the directory in which you installed `Python-3.12.0`. The command on the Oracle Machine Learning for Python server will be:

```
cd /home/user/python  
./configure --enable-shared --prefix=/home/user/python
```

```
make clean; make  
make altinstall
```

 **Note:**

Be sure to use the `--enable-shared` flag if you are going to use Embedded Python Execution; otherwise, using an Embedded Python Execution function results in an `extproc` error.

 **Note:**

Be sure to invoke `make altinstall` instead of `make install` to avoid overwriting the system Python.

6. Set environment variable `PYTHONHOME` and add it to your `PATH`, and set environment variable `LD_LIBRARY_PATH`:

```
export PYTHONHOME=/home/user/python
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
```

 **Note:**

In order to use Python for OML4Py, the variables must be set, and these variables must appear before system Python in `PATH` and `LD_LIBRARY_PATH`.

7. Create a symbolic link in your `/home/user/python/bin` directory to link to your `python3.12` executable, which you can do with the following commands:

```
cd /home/user/python/bin
ln -s python3.12 python3
```

`pip` will return warnings during package installation if the latest version is not installed. You can upgrade the version of `pip` to avoid these warnings:

```
python3 -m pip install --upgrade pip
```

You can now start Python by running the command `python3`. To verify the directory where Python is installed, use the `sys.executable` command from the `sys` package. For example:

```
python3

Python 3.12.0 (default, Feb 22 2022, 15:13:36)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44.0.3)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> print(sys.executable)
/home/user/python/bin/python3
```

This returns the absolute path of the Python executable binary.

If you run the command `python3` and you get the error `command not found`, then that means the system cannot find an executable named `python3` in `$PYTHONHOME/bin`. A symlink is required for the OML4Py server installation components. So, in that case, you need to create a symbolic link in your `/home/user/python/bin` directory to link to your `python3.12` executable as described in Step 6.

- [Commands Summary for Building and Installing Python for Linux for On-Premises Databases](#)

The commands used to build and install Python for Linux for On-Premises Databases are listed in the following example.

4.2.1 Commands Summary for Building and Installing Python for Linux for On-Premises Databases

The commands used to build and install Python for Linux for On-Premises Databases are listed in the following example.

Example 4-1 Build and Install Python for Linux for On-Premises Databases

```
wget https://www.python.org/ftp/python/3.12.0/Python-3.12.0.tgz

mkdir -p /home/user/python
tar -xvzf Python-3.12.0.tgz --strip-components=1 -C /home/user/python

cd /home/user/python

sudo yum install perl-Env libffi-devel openssl openssl-devel tk-devel xz-
devel zlib-devel bzip2-devel readline-devel libuuid-devel ncurses-devel

$ cd /home/user/python
./configure --enable-shared --prefix=/home/user/python

make clean; make
make altinstall

export PYTHONHOME=/home/user/python
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH

cd /home/user/python/bin
ln -s python3.12 python3
```

4.3 Install the Required Supporting Packages for Linux for On-Premises Databases

Both the OML4Py server and client installations for an on-premises Oracle database require that you also install a set of supporting Python packages, as described below.

Installing required packages on OML4Py client machine



Note:

`onnxruntime`, `onnxruntime-extensions`, `onnx`, `transformers`, and `sentencepiece` are to be installed on the client only. `onnxruntime`, `onnxruntime-extensions`, `onnx`, `transformers`, and `sentencepiece` support the ONNX conversion feature on the OML4Py client and should be installed on the client only. All other packages are installed on both the client and server.



Note:

`oracledb 2.1.0` or later version is required to invoke vector database SQL APIs in 23ai.

Use `pip3.12` to install the supporting packages. For OML4Py client installation of all the packages, run the following command, specifying the package:

```
pip3.12 install packagename
```

This command installs the required Python packages for on-premises OML4Py client:

```
pip3.12 install pandas==2.1.1
pip3.12 install setuptools==68.0.0
pip3.12 install scipy==1.12.0
pip3.12 install matplotlib==3.8.4
pip3.12 install oracledb==2.0.1
pip3.12 install scikit-learn==1.4.1post1
pip3.12 install numpy==1.26.4
pip3.12 install onnxruntime==1.17.0
pip3.12 install onnxruntime-extensions==0.10.1
pip3.12 install onnx==1.16.0
pip3.12 install --extra-index-url https://download.pytorch.org/whl/cpu
torch==2.2.0+cpu
pip3.12 install transformers==4.38.1
pip3.12 install sentencepiece==0.2.0
```

Installing required packages on OML4Py server machine

On the OML4Py server machine, all these packages must be installed into `$ORACLE_HOME/oml4py/modules` so they can be detected by the Embedded Python Execution process. Run

the following command, specifying the package and target directory, `$ORACLE_HOME/oml4py/modules`:

```
pip3.12 install packagename --target=$ORACLE_HOME/oml4py/modules
```

These command installs the required packages:

```
pip3.12 install pandas==2.1.1--target=$ORACLE_HOME/oml4py/modules
pip3.12 install setuptools==68.0.0 --target=$ORACLE_HOME/oml4py/modules
pip3.12 install scipy==1.12.0 --target=$ORACLE_HOME/oml4py/modules
pip3.12 install matplotlib==3.7.2 --target=$ORACLE_HOME/oml4py/modules
pip3.12 install oracledb==2.0.1 --target=$ORACLE_HOME/oml4py/modules
pip3.12 install joblib==1.2.0 --target=$ORACLE_HOME/oml4py/modules
pip3.12 install scikit-learn==1.2.1 --no-deps --target=$ORACLE_HOME/oml4py/modules
pip3.12 install numpy==1.26.4 --target=$ORACLE_HOME/oml4py/modules
```

Verify the Package Installation

Load the packages below to ensure they have been installed successfully. Start Python and run the following commands:

```
python3
```

```
Python 3.12.0 (default, Feb 22 2022, 15:13:36)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44.0.3)] on linux
Type "help", "copyright", "credits" or "license" for more information.
import numpy
import pandas
import scipy
import matplotlib
import oracledb
import sklearn
```

If all the packages are installed successfully, then no errors are returned.

- [Commands Summary for Installing Required Supporting Packages for Linux for On-Premises Databases](#)
The commands used for installing required supporting packages for linux for On-Premises databases are listed in the following example.

4.3.1 Commands Summary for Installing Required Supporting Packages for Linux for On-Premises Databases

The commands used for installing required supporting packages for linux for On-Premises databases are listed in the following example.

Example 4-2 Install required packages on OML4Py client machine

```
pip3.12 install pandas==2.1.1
pip3.12 install setuptools==68.0.0
pip3.12 install scipy==1.12.0
pip3.12 install matplotlib==3.8.4
```



```
pip3.12 install oracledb==2.0.1
pip3.12 install scikit-learn==1.4.1post1
pip3.12 install numpy==1.26.4
pip3.12 install onnxruntime==1.17.0
pip3.12 install onnxruntime-extensions==0.10.1
pip3.12 install onnx==1.16.0
pip3.12 install --extra-index-url https://download.pytorch.org/whl/cpu
torch==2.2.0+cpu
pip3.12 install transformers==4.38.1
pip3.12 install sentencepiece==0.2.0
```

4.4 Install OML4Py Server for On-Premises Oracle Database

The following instructions tell how to install and uninstall the OML4Py server components for an on-premises Oracle Database 23ai.

- [Install OML4Py Server for Linux for On-Premises Oracle Database 23ai](#)
Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 23ai.
- [Verify OML4Py Installation for On-Premises Database](#)
Verify the installation of the OML4Py server and client components for an on-premises database.
- [Grant Users the Required Privileges for On-Premises Database](#)
Instructions for granting the privileges required for using OML4Py with an on-premises database.
- [Create New Users for On-Premises Oracle Database](#)
The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.
- [Uninstall the OML4Py Server from an On-Premises Database 23ai](#)
Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle Database 23ai.

4.4.1 Install OML4Py Server for Linux for On-Premises Oracle Database 23ai

Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 23ai.

You can install OML4Py by using a Python script included in your 23ai database or by using the Database Configuration Assistant (DBCA).

Install OML4Py Using a SQL Script

To install the on-premises OML4Py server, the following are required:

- A connection to the internet.
- OML4Py supporting packages. For instructions on installing the required supporting packages see [Install the Required Supporting Packages for Linux for On-Premises Databases](#).
- Perl 5.8 or higher installed on your system.

 **Note:**

Perl requires the `perl-Env` package. You can install the package as root with the command `yum install perl-Env`.

To check for the existence of `perl-Env`, run the following command. The version will vary depending on your Operating System and version:

```
rpm -qa perl-Env
perl-Env-1.04-395.el8.noarch
```

- Write permission on the directories to which you download and install the server components.

 **Note:**

The following environment variables must be set up.

- Set environment variables: Set `PYTHONHOME` and add it to your `PATH`
- Set `ORACLE_HOME` and add it to your `PATH`
- Set `LD_LIBRARY_PATH`

```
export PYTHONHOME=$ORACLE_HOME/python
export PATH=$PYTHONHOME/bin:$ORACLE_HOME/bin:$PATH
export ORACLE_HOME=<ORACLE_HOME PATH>
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$ORACLE_HOME/lib:$LD_LIBRARY_PATH
```

To install the OML4Py server for Linux for an on-premises Oracle Database 23ai, run the server installation SQL script `pyqcfg.sql`.

 **Note:**

The OML4Py server needs to be installed on `CDB$ROOT` first, followed by installation on a PDB for Oracle Database 23ai.

1. The `pyqcfg.sql` script is under `$ORACLE_HOME/oml4py/server`. Change directory to `$ORACLE_HOME/oml4py/server`.

```
cd $ORACLE_HOME/oml4py/server
```

2. At your operating system prompt, start SQL*Plus, connect to the CDB\$ROOT and run the pyqcfg.sql script.

```
sqlplus / as sysdba
```

```
SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 12:40:18 2024  
Version 23.4.0.24.05
```

```
Copyright (c) 1982, 2024, Oracle. All rights reserved.
```

```
Connected to:  
Oracle Database 23ai Enterprise Edition Release 23.0.0.0.0 - Production  
Version 23.4.0.24.05
```

To capture the log, spool the installation steps to an external file `install_root.txt`.

```
SQL> spool install_root.txt
```

Verify that you are currently connected to the CDB\$ROOT container.

```
SQL> show con_name
```

```
CON_NAME  
-----  
CDB$ROOT
```

Run the `pyqcfg.sql` script to install OML4Py server in the CDB\$ROOT first.

```
SQL> @pyqcfg.sql SYSAUX TEMP
```

```
SQL> spool off;
```

where

- SYSAUX is the permanent tablespace for the PYQSYS schema.
- TEMP is the temporary tablespace for the PYQSYS schema.

Open the `install_root.txt` file to see if any errors occurred.

3. At your operating system prompt, start SQL*Plus, connect to your PDB and run the `pyqcfg.sql` script. To capture the log, spool the installation steps to an external file `install_pdb.txt`. The following example uses the PDB `ORCLPDB` and gives example values for the script arguments.

```
sqlplus / as sysdba
```

```
SQL> spool install_pdb.txt
```

Log into a PDB where you want to install OML4Py server.

```
SQL> alter session set container=ORCLPDB;
```

Run the pyqcfg.sql script to install OML4Py server in the PDB.

```
SQL> @pyqcfg.sql SYSAUX TEMP
```

```
SQL> spool off;
```

where

- SYSAUX is the permanent tablespace for the PYQSYS schema.
- TEMP is the temporary tablespace for the PYQSYS schema.

Open the install_pdb.txt file to see if any errors occurred

Verify the Server Installation

You can verify the database configuration of OML4Py as oracle user by doing the following:

1. On the OML4Py server database instance, start SQL*Plus as the OML user logging into the PDB, in this example, PDB1.

```
sqlplus oml_user/oml_user_password$ORCLPDB
```

2. Run the following command:

```
SELECT * FROM sys.pyq_config;
```

The expected output is as follows:

```
$ sqlplus / as sysdba;
SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 16:23:35 2024
```

```
Copyright (c) 1982, 2024, Oracle. All rights reserved.
```

```
Connected to:
Oracle Database 23ai Enterprise Edition Release 23.0.0.0.0 - Production
Version 23.4.0.24.05
```

```
SQL> alter session set container=ORCLPDB;
```

```
Session altered.
```

```
SQL> select * from sys.pyq_config;
```

```
NAME
-----
VALUE
-----
PYTHONHOME
/u01/app/oracle/product/23.4.0.0/dbhome_1/python
PYTHONPATH
/u01/app/oracle/product/23.4.0.0/dbhome_1/oml4y/modules
```

```
VERSION
2.0
```

```
NAME
```

```
-----
VALUE
-----
```

```
PLATFORM
```

```
ODB
```

```
DSWLIST
```

```
oml.*;pandas.*;numpy.*;matplotlib.*;sklearn.*
```

3. To verify the installation of the OML4Py server for an on-premises database see [Verify OML4Py Installation for On-Premises Database](#).
- [Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases](#)
The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.

4.4.1.1 Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases

The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.

Example 4-3 Install OML4Py Server for On-Premises Oracle Databases

```
export PYTHONHOME=$ORACLE_HOME/python
export PATH=$PYTHONHOME/bin:$ORACLE_HOME/bin:$PATH
export ORACLE_HOME=<ORACLE_HOME PATH>
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$ORACLE_HOME/lib:$LD_LIBRARY_PATH

cd $ORACLE_HOME/oml4py/server

$ sqlplus / as sysdba
SQL> spool install_root.txt
SQL> @pyqcfg.sql SYSAUX TEMP /u01/app/oracle/product/23.4.0.0/dbhome_1/python
SQL> spool off;

SQL> alter session set container=ORCLPDB;
SQL> @pyqcfg.sql SYSAUX TEMP /u01/app/oracle/product/23.4.0.0/dbhome_1/python
SQL> spool off;
```

4.4.2 Verify OML4Py Installation for On-Premises Database

Verify the installation of the OML4Py server and client components for an on-premises database.

1. In your local Python session, connect to the OML4Py server and invoke the same function by name. In the following example, replace the values for the parameters with those for your database.

```
import oml
oml.connect(user='oml_user', password='oml_user_password', host='myhost',
            port=1521, service_name='myservice')
```

2. Create a user-defined Python function and store it in the OML4Py script repository.

```
oml.script.create("TEST", func='def func():return 1 + 1', overwrite=True)
```

3. Call the user-defined function, using the `oml.do_eval` function.

```
res = oml.do_eval(func='TEST')
res
```

```
2
```

4. When you are finished testing, you can drop the test.

```
oml.script.drop("TEST")
```

4.4.3 Grant Users the Required Privileges for On-Premises Database

Instructions for granting the privileges required for using OML4Py with an on-premises database.

To use OML4Py (OML4Py), a user must have certain database privileges. To store and manage user-defined Python functions in the OML4Py script repository, a user must also have the PYQADMIN database role.

User Privileges

After installing the OML4Py server on an on-premises Oracle database server, grant the following privileges to any OML4Py user.

- CREATE SESSION
- CREATE TABLE
- CREATE VIEW
- CREATE PROCEDURE
- CREATE MINING MODEL
- EXECUTE ON CTXSYS.CTX_DDL (required for using Oracle Text Processing capability in the algorithm classes in the `oml.algo` package)

To grant all of these privileges, on the on-premises Oracle database server start SQL as a database administrator and run the following SQL statement, where `oml_user` is the OML4Py user:

```
GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW, CREATE PROCEDURE,
CREATE MINING MODEL, EXECUTE ON CTXSYS.CTX_DDL to oml_user;
```

Script Repository and Datastore Management

The OML4Py script repository stores user-defined Python functions that a user can invoke in an Embedded Python Execution function. An OML4Py datastore stores Python objects that can be used in subsequent Python sessions. A user-defined Python function in the script repository or a datastore can be available to any user or can be restricted for use by the owner only or by those granted access to it.

The OML4Py server installation script creates the PYQADMIN role in the database. A user must have that role to do the following:

- Store user-defined Python functions in the script repository.
- Drop user-defined Python function from the repository
- Grant or revoke permission to use a user-defined Python function in the script repository.
- Grant or revoke permission to use the objects in a datastore.

To grant this role to a user, on the on-premises Oracle database server start SQL as a database administrator and run the following SQL statement, where `oml_user` is your OML4Py user:

```
GRANT PYQADMIN to oml_user;
```

4.4.4 Create New Users for On-Premises Oracle Database

The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.

About the `pyquser.sql` Script

The `pyquser.sql` script is a component of the on-premises OML4Py server installation. The script is in the `server` directory of the installation. The `sysdba` privilege is required to run the script.

The `pyquser.sql` script grants the new user the required on-premises Oracle database privileges and, optionally, grants the PYQADMIN database role. The PYQADMIN role is required for creating and managing scripts in the OML4Py script repository for use in Embedded Python Execution.

The `pyquser.sql` script takes the following five positional arguments:

- Username
- User's permanent tablespace
- User's temporary tablespace
- Permanent tablespace quota
- PYQADMIN role

When you run the script, it prompts you for a password for the user.

Create a New User

To use the `pyquser.sql` script, go the `server` subdirectory of the directory that contains the extracted OML4Py server installation files. Run the script as a database administrator.

The following examples use SQL*Plus and the `sysdba` user to run the `pyquser.sql` script.

Example 4-4 Creating New Users

This example creates the user `oml_user` with the permanent tablespace `USERS` with an unlimited quota, the temporary tablespace `TEMP`, and grants the `PYQADMIN` role to the `oml_user`.

```
sqlplus / as sysdba
@pyquser.sql oml_user USERS TEMP unlimited pyqadmin
```

Enter value for password: <type your password>

For a pluggable database:

```
sqlplus / as sysdba
alter session set container=<PDBNAME>
@pyquser.sql oml_user USERS TEMP unlimited pyqadmin
```

The output is similar to the following:

```
SQL> @pyquser.sql oml_user USERS TEMP unlimited pyqadmin
Enter value for password: welcome1
old  1: create user &&1 identified by &password
new  1: create user oml_user identified by welcome1
old  2: default tablespace &&2
new  2: default tablespace USERS
old  3: temporary tablespace &&3
new  3: temporary tablespace TEMP
old  4: quota &&4 on &&2
new  4: quota unlimited on USERS
```

User created.

```
old  4:      'create procedure, create mining model to &&1';
new  4:      'create procedure, create mining model to pyquser';
old  6:  IF lower('&&5') = 'pyqadmin' THEN
new  6:  IF lower('pyqadmin') = 'pyqadmin' THEN
old  7:      execute immediate 'grant PYQADMIN to &&1';
new  7:      execute immediate 'grant PYQADMIN to pyquser';
```

PL/SQL procedure successfully completed.

This example creates the user `oml_user2` with 20 megabyte quota on the `USERS` tablespace, the temporary tablespace `TEMP`, and without the `PYQADMIN` role.

```
sqlplus / as sysdba
@pyquser.sql oml_user2 USERS TEMP 20M FALSE
```

Enter value for password: <type your password>

4.4.5 Uninstall the OML4Py Server from an On-Premises Database 23ai

Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle Database 23ai.

Uninstall the On-Premises OML4Py Server for Linux

To uninstall the on-premises OML4Py server for Linux, do the following:

1. Verify that the `PYTHONHOME` environment variable is set to the Python3.12 directory.

```
echo $PYTHONHOME
```

2. Verify that `PYTHONPATH` environment variable is set to the directory in which the `oml` modules are installed.

```
echo $PYTHONPATH
```

If it is not set to the proper directory, set it.

```
export PYTHONPATH=$ORACLE_HOME/oml4py/modules
```

3. Change directories to the directory containing the server installation zip file.

```
cd $ORACLE_HOME/oml4py
```

4. Run the server uninstall script `pyquncfg.sql`, the script is under `$ORACLE_HOME/oml4py/server`.

```
cd $ORACLE_HOME/oml4py/server
```

Start SQL*Plus, connect to the PDB.

```
$ sqlplus / as sysdba;
```

```
SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 12:40:18 2024  
Version 23.4.0.24.05
```

```
Copyright (c) 1982, 2024, Oracle. All rights reserved.
```

```
Connected to:  
Oracle Database 23ai Enterprise Edition Release 23.0.0.0.0 - Production  
Version 23.4.0.24.05
```

Run the `pyquncfg.sql` script to record the log, spool, and the installation steps to an external file.

```
SQL> spool install.txt
```

```
SQL> alter session set container=ORCLPDB;  
SQL> @pyquncfg.sql
```

4.5 Install OML4Py Client for On-Premises Oracle Database

Instructions for installing and uninstalling the on-premises OML4Py client.

For instructions on installing the OML4Py client on Autonomous Database, see [Install OML4Py Client for Linux for Use With Autonomous Database Serverless](#).

To use the OML4Py client, users will need to install the following:



Note:

The supporting packages need to be installed prior to the OML4Py client installation.

1. Install Python from source. See [Build and Install Python for Linux for On-Premises Databases](#).
2. Install the Oracle Instant Client. See [Install Oracle Instant Client for Linux for On-Premises Databases](#).
3. Install the OML4Py supporting packages. See [Install the Required Supporting Packages for Linux for On-Premises Databases](#).
4. Install the OML4Py client. See [Install OML4Py Client for Linux for On-Premises Databases](#).

Topics:

- [Install Oracle Instant Client and the OML4Py Client for Linux](#)
Instructions for installing Oracle Instant Client and the OML4Py client for Linux for an on-premises Oracle database.
- [Verify OML4Py Client Installation for On-Premises Databases](#)
Verify the installation of the OML4Py client components for an on-premises Oracle database.
- [Uninstall the OML4Py Client for On-Premises Databases](#)
Instructions for uninstalling the OML4Py client.

4.5.1 Install Oracle Instant Client and the OML4Py Client for Linux

Instructions for installing Oracle Instant Client and the OML4Py client for Linux for an on-premises Oracle database.

To connect the OML4Py client for Linux to an on-premises Oracle database, you must have Oracle Instant Client installed on your local system.

- [Install Oracle Instant Client for Linux for On-Premises Databases](#)
Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle database.
- [Install OML4Py Client for Linux for On-Premises Databases](#)
Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle database.

- [Commands Summary for Installing Oracle Instant Client and OML4Py Client for On-Premises Oracle Database](#)
The commands used for installing Oracle Instant Client and OML4Py client for On-Premises Oracle Database are listed in the following example.

4.5.1.1 Install Oracle Instant Client for Linux for On-Premises Databases

Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle database.

The OML4Py client requires Oracle Instant Client to connect to an Oracle database.

To install Oracle Instant Client, the following are required:

- A connection to the internet.
- Write permission on the directory in which you are installing the client.

To install Oracle Instant Client, do the following:

1. Download the Oracle Instant Client for your system. Go to the [Oracle Instant Client Downloads page](#) and select Instant Client for Linux x86-64.
2. Locate the section for your version of Oracle Database. These instructions use Version 23.4.0.0.0.
3. In the Base section, in the Download column, click the zip file for the Basic Package and save the file in an accessible directory on your system or use `wget` to download the file:

```
wget https://download.oracle.com/otn_software/linux/instantclient/2340000/instantclient-basic-linux.x64-23.4.0.24.05.zip
```

4. Unzip the package:

```
unzip instantclient-basic-linux.x64-23.4.0.24.05.zip
```

Extracting the package creates the subdirectory `instantclient_23_4`, which contains the Oracle Instant Client files.

5. To install the `libaio` package with `sudo` or as the root user, run the following command:

```
sudo yum install libaio
```

 **Note:**

In some Linux distributions, this package is called `libaio1`.

6. Add the directory that contains the Oracle Instant Client files to the beginning of your `LD_LIBRARY_PATH` environment variable:

```
export LD_LIBRARY_PATH=/opt/oracle/instantclient_23_04:$LD_LIBRARY_PATH
```

4.5.1.2 Install OML4Py Client for Linux for On-Premises Databases

Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle database.

Prerequisites

To download and install the on-premises OML4Py client, the following are required:

- A connection to the internet.
- Write permission on the directory in which you are installing the client.
- Perl 5.8 or higher installed on your system.
- Python 3.12.0. To know more about downloading and installing Python 3.12.0, see [Build and Install Python for Linux for On-Premises Databases](#)
- The OML4Py supporting packages. To know more about OML4Py the supporting packages, see [Install the Required Supporting Packages for Linux for On-Premises Databases](#).

To use the OML4Py client to connect to an on-premises Oracle database, the following are required:

- Oracle Instant Client must be installed on the client machine.
- The OML4Py server must be installed on the on-premises database server.

Download and Extract the OML4Py Client Installation File

To download and extract the OML4Py client installation file, do the following:

1. Download the client installation zip file.
 - a. Go to the [Oracle Machine Learning for Python Downloads](#) page on the Oracle Technology Network.
 - b. Accept the license agreement and select **Oracle Machine Learning for Python Downloads (v2.0)**.
 - c. Select **Oracle Machine Learning for Python Client Install for Oracle Database on Linux 64 bit**.
 - d. Save the zip file to an accessible directory.
2. Go to the directory to which you downloaded the zip file and unzip the file.

```
unzip oml4py-client-linux-x86_64-2.0.zip
```

The contents are extracted to a subdirectory named `client`, which contains these four files:

- OML4PInstallShared.pm
- oml-2.0-cp312-cp312-linux_x86_64.whl
- client.pl
- oml4py.ver

View the Optional Arguments to the Client Installation Perl Script

In the directory above the unzipped client folder, run the client installation Perl script with the `--help` option to display the arguments to the client installation Perl script.

The following command displays the available installation options:

```
perl -Iclient client/client.pl --help
```

```
Oracle Machine Learning for Python 2.0 Client.
```

```
Copyright (c) 2018, 2024 Oracle and/or its affiliates. All rights reserved.
```

```
Usage: client.pl [OPTION]...
```

```
Install, upgrade, or uninstall OML4P Client.
```

```
-i, --install install or upgrade (default)
-u, --uninstall uninstall
-y never prompt
--ask interactive mode (default)
--no-embed do not install embedded python functionality
--no-deps turn off dependencies checking
--target <dir> install client into <dir>
```

By default, the installation script installs the Embedded Python Execution modules. If you don't want to install this module, then you can use the `--no-embed` flag.

Also by default, the installation script checks for the existence and version of each of the supporting packages that the OML4Py client requires. If a required package is missing or does not meet the version requirement, the installation script displays an error message and exits. You can skip the dependency checking in the client installation by using the `--no-deps` flag. However, to use the `oml` module, you need to have installed acceptable versions of all of the supporting packages.

For a list of the required dependencies, see [Install the Required Supporting Packages for Linux for On-Premises Databases](#).

Run the OML4Py Client Installation Script

To install the OML4Py client, do the following:

1. In the directory above the unzipped client folder, run the script. The following command runs the Perl script in the current directory:

```
perl -Iclient client/client.pl
```

Alternatively, the following command runs the Perl script with the target directory specified:

```
perl -Iclient client/client.pl --target path_to_target_dir
```

The `--target` flag is optional, if you don't want to install it to the current directory.

When the script displays `Proceed?`, enter `y` or `yes`.

If you use the `--target <dir>` argument to install the `oml` module to the specified directory, then add that location to environment variable `PYTHONPATH` so that Python can find the module:

```
export PYTHONPATH=path_to_target_dir
```

The command displays the following:

```
perl -Iclient client/client.pl
```

```
Oracle Machine Learning for Python 2.0 Client.
```

```
Copyright (c) 2018, 2022 Oracle and/or its affiliates. All rights reserved.
Checking platform ..... Pass
Checking Python ..... Pass
Checking dependencies ..... Pass
Checking OML4P version ..... Pass
Current configuration
  Python Version ..... 3.12.0
  PYTHONHOME ..... /opt/Python-3.12.0
  Existing OML4P module version .... None
  Operation ..... Install/Upgrade
```

```
Proceed? [yes]
```

```
Processing ./client/oml-2.0-cp311-cp311-linux_x86_64.whl
Installing collected packages: oml
Successfully installed oml-2.0
```

2. To verify that `oml` modules are successfully installed and are ready to use, start Python and import `oml`. At the Linux prompt, enter `python3`.

```
python3
```

At the Python prompt, enter `import oml`

```
import oml
```

The output is similar to the following:

```
$ python3
Python 3.12.0 (default, Feb 23 2022, 17:12:33)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44.0.3)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import oml
>>>
```

3. Display the location of the installation directory. If you didn't use the `--target <dir>` argument, then the installed `oml` modules are stored under `$PYTHONHOME/lib/python3.12/site-packages/`. Again, you must have write permission for the target directory.

In Python, after importing the `oml` module, you can display the directory in which the client is installed. At the Python prompt, enter:

```
oml.__path__
```

Connect to the OML4Py Server

Start Python, import `oml`, and create a connection to your OML4Py server using an appropriate password, hostname, and system identifier. The following example uses `oml_user` as the user and has example argument values. Replace the username and other argument values with the values for your user and database.

```
import oml
oml.connect(user='oml_user', password='oml_user_password', host=myhost,
            port=1521, sid=mysid)
```

After connecting, you can run any of the examples in this publication. For example, you could run [Example 7-8](#).

Note:

To use the Embedded Python Execution examples, you must have installed the OML4Py client with the Embedded Python Execution option enabled. To use the Automatic Machine Learning (AutoML) examples, you must specify a running connection pool on the server in the `automl` argument in an `oml.connect` invocation.

4.5.1.3 Commands Summary for Installing Oracle Instant Client and OML4Py Client for On-Premises Oracle Database

The commands used for installing Oracle Instant Client and OML4Py client for On-Premises Oracle Database are listed in the following example.

Example 4-5 Command Summary for Installing Oracle Instant Client

```
wget https://download.oracle.com/otn_software/linux/instantclient/2340000/
instantclient-basic-linux.x64-23.4.0.24.05.zip
unzip instantclient-basic-linux.x64-23.4.0.0dbru.zip
export LD_LIBRARY_PATH=/opt/oracle/instantclient_23_4:$LD_LIBRARY_PATH
```

Example 4-6 Command Summary for Installing OML4Py Client

```
unzip oml4py-client-linux-x86_64-2.0.zip
perl -Iclient client/client.pl
```

4.5.2 Verify OML4Py Client Installation for On-Premises Databases

Verify the installation of the OML4Py client components for an on-premises Oracle database.

1. In your local Python session, connect to the OML4Py server and invoke the same function by name. In the following example, replace the values for the parameters with those for your database.

```
import oml
oml.connect(user='oml_user', password='oml_user_password', host='myhost',
            port=1521, sid='mysid')
```

2. Create a user-defined Python function and store it in the OML4Py script repository.

```
oml.script.create("TEST", func='def func():return 1 + 1', overwrite=True)
```

3. Call the user-defined function, using the `oml.do_eval` function.

```
res = oml.do_eval(func='TEST')
res
```

```
2
```

4. When you are finished testing, you can drop the test.

```
oml.script.drop("TEST")
```

4.5.3 Uninstall the OML4Py Client for On-Premises Databases

Instructions for uninstalling the OML4Py client.

Uninstall the On-Premises OML4Py Client for Linux

To uninstall the on-premises OML4Py client for Linux, from the directory containing the client installation zip file, run the client installation Perl script with the `-u` argument:

```
perl -Iclient client/client.pl -u
```

When the script displays `Proceed?`, enter `y` or `yes`.

If the client is successfully uninstalled, you'll see the following message:

```
Uninstalling oml-2.0:
Successfully uninstalled oml-2.0
```


5

Install OML4Py on Exadata

The following topics tell about OML4Py on Exadata and how to configure DCLI and install python, OML4Py across Exadata.

Topics:

- [About Oracle Machine Learning for Python on Exadata](#)
- [Configure DCLI to install Python across Exadata compute nodes.](#)
- [About Oracle Machine Learning for Python on Exadata](#)
Exadata is an ideal platform for OML4Py. The parallel resources of Python computations in OML4Py take advantage of the massively parallel grid infrastructure of Exadata.
- [Configure DCLI to install Python across Exadata compute nodes.](#)
Using Distributed Command Line Interface (DCLI) can simplify the installation of OML4Py on Exadata.

5.1 About Oracle Machine Learning for Python on Exadata

Exadata is an ideal platform for OML4Py. The parallel resources of Python computations in OML4Py take advantage of the massively parallel grid infrastructure of Exadata.



Note:

The version of OML4Py must be the same on the server and on each client computer. Also, the version of Python must be the same on the server and on each client computer. See table number 3-2 [OML4Py On Premises System Requirements](#) for supported configurations.

To install OML4Py on Exadata:

1. On all compute nodes:
 - Install Python
 - Verify and configure the environment
 - Install the OML4Py supporting packages
 - Install the OML4Py server components
2. On the first node only:
 - Install the OML4Py Server components including the database configuration.
 - Create an OML4Py user, if desired. Alternatively, configure an existing database user to use OML4Py. See [Create New Users for On-Premises Oracle Database](#).

You can simplify the Python installation on Exadata by using the Distributed Command Line Interface (DCLI).

5.2 Configure DCLI to install Python across Exadata compute nodes.

Using Distributed Command Line Interface (DCLI) can simplify the installation of OML4Py on Exadata.

With DCLI, you can use a single command to install Python across multiple Exadata compute nodes. The following example shows the output of the DCLI help option, which explains the basic syntax of the utility.

Example 5-1 DCLI Help Option Output

```
$ dcli -h

Distributed Shell for Oracle Storage

This script executes commands on multiple cells in parallel threads.
The cells are referenced by their domain name or ip address.
Local files can be copied to cells and executed on cells.
This tool does not support interactive sessions with host applications.
Use of this tool assumes ssh is running on local host and cells.
The -k option should be used initially to perform key exchange with
cells. User may be prompted to acknowledge cell authenticity, and
may be prompted for the remote user password. This -k step is serialized
to prevent overlaid prompts. After -k option is used once, then
subsequent commands to the same cells do not require -k and will not require
passwords for that user from the host.
Command output (stdout and stderr) is collected and displayed after the
copy and command execution has finished on all cells.
Options allow this command output to be abbreviated.

Return values:
 0 -- file or command was copied and executed successfully on all cells
 1 -- one or more cells could not be reached or remote execution
    returned non-zero status.
 2 -- An error prevented any command execution

Examples:
dcli -g mycells -k
dcli -c stsd2s2, stsd2s3 vmstat
dcli -g mycells cellcli -e alter iormplan active
dcli -g mycells -x reConfig.scl

Usage: dcli [options] [command]

Options:
--version          show program's version number and exit
--batchsize=MAXTHDS limit the number of target cells on which to run the
                  command or file copy in parallel
-c CELLS          comma-separated list of cells
--ctimeout=CTIMEOUT Maximum time in seconds for initial cell connect
-d DESTFILE       destination directory or file
-f FILE           files to be copied
-g GROUPFILE      file containing list of cells
```

```

-h, --help                show help message and exit
--hidestderr              hide stderr for remotely executed commands in ssh
-k                        push ssh key to cell's authorized_keys file
--key-with-one-password  apply one credential for pushing ssh key to
                        authorized_keys files
-l USERID                user to login as on remote cells (default: celladmin)
--root-exadatatmp        root user login using directory /var/log/exadatatmp/
--maxlines=MAXLINES      limit output lines from a cell when in parallel
                        execution over multiple cells (default: 100000)
-n                        abbreviate non-error output
-r REGEXP                abbreviate output lines matching a regular expression
-s SSHOPTIONS            string of options passed through to ssh
--scp=SCOPTIONS          string of options passed through to scp if different
                        from sshoptions
--serial                 serialize execution over the cells
--showbanner             show banner of the remote node in ssh
-t                        list target cells
--unkey                  drop keys from target cells' authorized_keys file
-v                        print extra messages to stdout
--vmstat=VMSTATOPS      vmstat command options
-x EXECFILE              file to be copied and executed

```

Configure the Exadata environment to enable automatic authentication for DCLI on each compute node.

1. Generate an SSH public-private key for the root user. Execute the following command as root on any node:

```
$ ssh-keygen -N '' -f /.ssh/id_dsa -t dsa
```

This command generates public and private key files in the `.ssh` subdirectory of the home directory of the root user.

2. In a text editor, create a file that contains the names of all the compute nodes in the rack. Specify each node name on a separate line. For example, the `nodes` file for a 2-node cluster could contain entries like the following:

```
$ cat nodes
exadb01
exadb02
```

3. Run the DCLI command with the `-k` option to establish SSH trust across all the nodes. The `-k` option causes DCLI to contact each node sequentially (not in parallel) and prompts you to enter the password for each node.

```
$ dcli -t -g nodes -l root -k -s "\-o StrictHostkeyChecking=no"
```

DCLI with `-k` establishes SSH Trust and User Equivalence. Subsequent DCLI commands will not prompt for passwords.

- [Install Python across Exadata compute nodes using DCLI](#)
Instructions for installing Python across Exadata compute nodes using DCLI.

- [Install OML4Py across Exadata compute nodes using DCLI](#)
Instructions for installing OML4Py across Exadata compute nodes using DCLI.

5.2.1 Install Python across Exadata compute nodes using DCLI

Instructions for installing Python across Exadata compute nodes using DCLI.

These steps describe building and installing Python for Exdata.

1. Go to the [Python website](#) and download the Python 3.12.0 **XZ compressed source tarball** and untar it. The downloaded file name is `Python-3.12.0.tgz`

```
$ wget https://www.python.org/ftp/python/3.12.0/Python-3.12.0.tgz
$ tar xvf Python-3.12.0.tgz
```

2. OML4Py requires the presence of the `perl-Env libffi-devel, openssl, openssl-devel, tk-devel, xz-devel, zlib-devel, bzip2-devel, readline-devel` and `libuuid-devel` libraries. Install these libraries using the command:

```
# dcli -t -g nodes -l root "yum -y install perl-Env libffi-devel openssl
openssl-devel tk-devel xz-devel zlib-devel bzip2-devel readline-devel
libuuid-devel"
```

3. Set the `PYTHONHOME` environment on each node:

```
# dcli -t -g nodes -l oracle "export PYTHONHOME=$ORACLE_HOME/python;
export PATH=$ORACLE_HOME/python/bin:$PATH; export
LD_LIBRARY_PATH=$ORACLE_HOME/python/lib:$LD_LIBRARY_PATH; export
PIP_REQUIRE_VIRTUALENV=false"
```

```
# dcli -t -g nodes -l oracle "tar xvfz $ORACLE_HOME/Python-3.12.0.tgz -
C $ORACLE_HOME/python"
```

```
# dcli -t -g nodes -l oracle "cd $ORACLE_HOME/python; ./configure --enable-
shared --prefix=$ORACLE_HOME/python"
```

```
# dcli -t -g nodes -l oracle "cd $ORACLE_HOME/python; make clean; make"
```

```
# dcli -t -g nodes -l oracle "cd $ORACLE_HOME/python; make altinstall"
```

4. Create a symbolic link in your `$PYTHONHOME/bin` directory. You need to link it to your Python-3.12 executable, which you can do with the following commands:

```
# dcli -t -g nodes -l oracle "cd $PYTHONHOME/bin"
# dcli -t -g nodes -l oracle "ln -s python3.12 python3"
```

5. Set environment variable `PYTHONHOME` and add it to your `PATH`, and set environment variable `LD_LIBRARY_PATH`:

```
# dcli -t -g nodes -l oracle "export PYTHONHOME=$ORACLE_HOME/python"
# dcli -t -g nodes -l oracle "export PATH=$PYTHONHOME/bin:$PATH"
# dcli -t -g nodes -l oracle "export LD_LIBRARY_PATH=$PYTHONHOME/
lib:$LD_LIBRARY_PATH"
# dcli -t -g nodes -l oracle "export PIP_REQUIRE_VIRTUALENV=false"
```

6. You can now start Python by running the command `python3`. For example:

```
# dcli -t -g nodes -l oracle "python3"
exadb01: Python 3.12.0 (default, Feb 10 2022, 14:38:12)
      [GCC 4.8.5 20150623 (Red Hat 4.8.5-44.0.3)] on linux
      Type "help", "copyright", "credits" or "license" for more information.

exadb02: Python 3.12.0 (default, Feb 10 2022, 14:38:12)
      [GCC 4.8.5 20150623 (Red Hat 4.8.5-44.0.3)] on linux
      Type "help", "copyright", "credits" or "license" for more information.
```

5.2.2 Install OML4Py across Exadata compute nodes using DCLI

Instructions for installing OML4Py across Exadata compute nodes using DCLI.

To install OML4Py on Exadata using DCLI, follow the steps:

1. First install the OML4Py supporting packages to `$ORACLE_HOME/oml4py/modules` on each node. The OML4Py supporting packages must be installed individually on each compute node. DCLI cannot be used because it uses the system default Python and cause conflicts with the Python installed for use with OML4Py.

```
$ pip3.12 install pandas==2.1.1 --target=$ORACLE_HOME/oml4py/modules
$ pip3.12 install scipy==1.11.3 --target=$ORACLE_HOME/oml4py/modules
$ pip3.12 install matplotlib==3.7.2 --target=$ORACLE_HOME/oml4py/modules
$ pip3.12 install python-oracledb==1.4.0 --target=$ORACLE_HOME/oml4py/
modules
$ pip3.12 install scikit-learn==1.2.1 --no-deps --target=$ORACLE_HOME/
oml4py/modules
$ pip3.12 install numpy==1.26.0 --target=$ORACLE_HOME/oml4py/modules
```

2. Set the `PYTHONPATH` environment variable to the location of the OML4Py modules:

```
$ export PYTHONPATH=$ORACLE_HOME/oml4py/modules
```

3. Download the installation file for your system.
 - a. Go to the [Oracle Machine Learning for Python Downloads](#) page on the Oracle Technology Network.
 - b. Accept the license agreement and **select Oracle Machine Learning for Python Downloads (v2.0)**.
 - c. Select **Oracle Machine Learning for Python Server Install for Oracle Database on Linux 64 bit**.
 - d. Save the file to the `$ORACLE_HOME/oml4py` directory.

To extract the installation file to `$ORACLE_HOME/oml4py` directory, use the command:

```
$ unzip oml4py-server-linux-x86_64-2.0.zip -d $ORACLE_HOME/oml4py
```

The files are extracted to the `$ORACLE_HOME/oml4py/server` subdirectory.

4. On the first node, from the `$ORACLE_HOME/oml4py` directory, run the server installation script. The following command runs the script in interactive mode:

```
$ perl -Iserver server/server.pl
```

To run the server script in non-interactive mode, pass the parameters for the pluggable database, and permanent and temporary tablespaces to the script

```
$ perl -Iserver server/server.pl -y --pdb PDB11 --perm SYSTEM --temp TEMP
```

Run the server script with the `--no-db` flag on all remaining compute nodes. This sets up the OML4Py server configuration and skips the database configuration steps already performed on the first node:

```
$ perl -Iserver server/server.pl --no-db
```

6

Install Third-Party Packages

Oracle Machine Learning Notebooks in the Autonomous Database provides a conda interpreter to install third-party Python libraries in a conda environment for use within OML Notebooks sessions and OML4Py embedded execution invocations. Conda is an open-source package and environment management system that enables the use of environments containing third-party Python libraries.

Administrators create conda environments and install packages that can then be accessed by non-administrator users and loaded into their OML Notebooks session. The conda environments can be used by OML4Py Python, SQL, and REST APIs.

Note:

- None of the OML features that come with ADB require the customer to install any additional third-party software via the conda feature.
- When installing third-party software using the conda feature, vulnerability management and license compliance of that software is the sole responsibility of the customer who installed it, not Oracle.

Topics:

- [Conda Commands](#)
This topic contains common commands used by ADMIN while creating and testing conda environments in Autonomous Databases. Conda is an open-source package and environment management system that enables the use of environments containing third-party Python libraries.
- [Administrative Tasks for Creating and Saving a Conda Environment](#)
In OML Notebooks, user ADMIN can manage the lifecycle of the OML user's conda environments, including creating and deleting environments and installing and deleting packages.
- [OML User Tasks for Downloading an Available Conda Environment](#)
Once user ADMIN installs the environment in Object Storage in the Autonomous Database, as an OML user, you can download, activate, and use it in Python paragraphs in notebooks and with embedded execution.
- [Using Conda Environments with Embedded Python Execution](#)
This topic explains the usage of conda environments by running user-defined functions (UDFs) in SQL and REST APIs for embedded Python execution.

6.1 Conda Commands

This topic contains common commands used by ADMIN while creating and testing conda environments in Autonomous Databases. Conda is an open-source package and environment

management system that enables the use of environments containing third-party Python libraries.

Refer to Conda Interpreter Commands for a table of supported conda commands.

Conda Help

To get help for conda commands, run the command name followed by the `--help` flag.



Note:

The conda command is not run explicitly because the `%conda` interpreter provides the conda context.

- Get help for all conda commands

```
%conda
```

```
--help
```

- Get help for a specific conda command. Run the following command to get help with the `install` command:

```
%conda
```

```
install --help
```

Conda Info

The `info` command displays information about the conda installation, including the conda version and available channels.

```
%conda
```

```
info
```

Conda Search

The `search` command allows the user to search for packages and display associated information, including the package version and the channel where it resides.

- Search for a specific package. Run the following command to search for the package `scikit-learn`.

```
%conda
```

```
search scikit-learn
```

- Search for packages containing 'scikit' in the package name.

```
%conda
```

```
search '*scikit*'
```


- Search for a specific version of a package.

```
%conda  
  
search 'numpy==1.12'
```

```
%conda  
  
search 'numpy>=1.12'
```

- Search for a specific version on a specific channel.

```
%conda  
  
search conda-forge::numpy
```

Enhanced Conda Commands

A set of enhanced conda commands in the conda environment lifecycle management package `env-lcm` supports the management of environments saved to Object Storage, including uploading, downloading, listing, and deleting available environments.

Help for conda lifecycle environment commands.

```
%conda  
  
env-lcm --help
```

```
Usage: conda-env-lcm [OPTIONS] COMMAND [ARGS]...
```

```
ADB-S Command Line Interface (CLI) to manage persistence of conda  
environments
```

Options:

```
-v, --version Show the version and exit.  
--help Show this message and exit.
```

Commands:

```
delete Delete a saved conda environment  
download Download a saved conda environment  
import Create or update a conda environment from saved metadata  
list-local-envs List locally available environments for use  
list-saved-envs List saved conda environments  
upload Save conda environment for later use
```

Creating Conda Environments

This section demonstrates creating and installing packages to a conda environment, then removing the environment. Here commonly used options available for environment creation and testing are illustrated. The environment exists for the duration of the notebook session and does not persist between sessions unless it is saved to Object Storage. For instructions that include both creating and persisting an environment for OML users, refer to Administrative task to create and save the conda environments. As an ADMIN user:

1. Use the create command to create an environment `myenv` and install the Python keras package.
2. Verify that the new environment is created, and activate the environment.
3. Install, then uninstall an additional Python package, `pytorch`, in the environment.
4. Deactivate and remove the environment.

 **Note:**

The ADMIN user can access the conda environment from Python and R, but does not have the capability to run embedded Python and R execution commands.

For help with the conda `create` command, enter `create --help` in a `%conda` paragraph.

List Environments

Start by listing the environments available by default. Conda contains default environments with some core system libraries and conda dependencies. The active environment is marked with an asterisk (*).

```
%conda
env list

# conda environments:
#
base                * /usr
conda-pack-env      /usr/envs/conda-pack-env
```

Create Conda Environment

Create conda environment called `myenv` with Python 3.10 for OML4Py compatibility and install the keras package.

```
%conda
create -n myenv python=3.10 keras
```

Verify Environment Creation

Verify the `myenv` environment is in the list of environments. The asterisk (*) indicates active environments. The new environment is created but not activated.

```
%conda
env list

# conda environments:
#
myenv                /u01/.conda/envs/myenv
```

```
base                * /usr
conda-pack-env      /usr/envs/conda-pack-env
```

Activate the Environment

Activate the myenv environment and list the environments to verify the activation. The asterisk (*) next to the environment name confirms the activation.

```
%conda

activate myenv

Conda environment 'myenv' activated
```

List the environments available by default.

```
%conda

env list

# conda environments:
#
myenv                * /u01/.conda/envs/myenv
base                 /usr
conda-pack-env      /usr/envs/conda-pack-env
```

Installing and Uninstalling Libraries

The ADMIN user can install and uninstall libraries into an environment using the `install` and `uninstall` commands. For help with the `conda install` and `conda uninstall` commands, type `install --help` and `uninstall --help` in a `%conda` paragraph.

 **Note:**

When conda installs a package into an environment, it also installs any required dependencies. As shown here, it's possible to install packages to an existing environment. As a best practice, to avoid dependency conflicts, simultaneously install all the packages you need in a specific environment.

Install Additional Packages

Install the pytorch package into the activated myenv environment.

```
%conda

install pytorch
```

List Packages in the Current Environment

List the packages installed in the current environment, and confirm that keras and pytorch are installed.

```
%conda  
  
list
```

The output is similar to the following:

```
# packages in environment at /u01/.conda/envs/myenv:  
#  
# Name                               Version                               Build  Channel  
_libgcc_mutex                         0.1                                   main  
_openmp_mutex                         5.1                                   1_gnu  
blas                                   1.0                                   mkl  
.  
.  
.  
fftw                                   3.3.9                                h27cfd23_1  
future                                0.18.2                               py310h06a4308_1  
intel-openmp                          2021.4.0                             h06a4308_3561  
keras                                  2.10.0                               py310h06a4308_0  
keras-preprocessing                   1.1.2                               pyhd3eb1b0_0  
ld_impl_linux-64                     2.38                                  h1181459_1  
libffi                                 3.3                                   he6710b0_2  
libgcc-ng                              11.2.0                               h1234567_1  
.  
.  
.  
numpy-base                            1.23.3                               py310h8e6c178_1  
openssl                                1.1.1s                               h7f8727e_0  
pip                                    22.2.2                              py310h06a4308_0  
pyparser                              2.21                                 pypi_0      pypi  
python                                 3.10.6                               haa1d7c7_1  
pytorch                               1.10.2                              cpu_py310h6894f24_0  
readline                              8.2                                  h5eee18b_0  
.  
.  
.  
xz                                     5.2.6                               h5eee18b_0  
zlib                                   1.2.13                              h5eee18b_0
```

The output above has been truncated and does not show the complete list of packages.

Uninstall Package

Libraries can be uninstalled from an environment using the `uninstall` command. Let's uninstall the `pytorch` package from the current environment.

```
%conda  
  
uninstall pytorch
```

Verify Package was Uninstalled

List packages in current environment and verify that the pytorch package was uninstalled.

```
%conda
list
```

The output shown below does not contain the pytorch package.

```
# packages in environment at /u01/.conda/envs/myenv:
#
# Name                               Version                               Build Channel
_libgcc_mutex                        0.1                                   main
_openmp_mutex                        5.1                                   1_gnu
blas                                  1.0                                   mkl
bzip2                                 1.0.8                                h7b6447c_0
ca-certificates                      2022.10.11                           h06a4308_0
certifi                              2022.9.24                             py310h06a4308_0
cffi                                  1.15.1                                py310h74dc2b5_0
fftw                                  3.3.9                                 h27cfd23_1
future                               0.18.2                                py310h06a4308_1
intel-openmp                         2021.4.0                              h06a4308_3561
keras                                 2.10.0                                py310h06a4308_0
keras-preprocessing                 1.1.2                                 pyhd3eb1b0_0
ld_impl_linux-64                    2.38                                  h1181459_1
libffi                                3.3                                   he6710b0_2
libgcc-ng                            11.2.0                               h1234567_1
libgfortran-ng                      11.2.0                               h00389a5_1
libgfortran5                        11.2.0                               h1234567_1
libgomp                              11.2.0                               h1234567_1
libstdcxx-ng                        11.2.0                               h1234567_1
libuuid                              1.0.3                                 h7f8727e_2
mkl                                  2021.4.0                              h06a4308_640
mkl-service                         2.4.0                                py310h7f8727e_0
mkl_fft                             1.3.1                                py310hd6ae3a3_0
mkl_random                          1.2.2                                py310h00e6091_0
ncurses                              6.3                                   h5eee18b_3
ninja                                1.10.2                               h06a4308_5
ninja-base                          1.10.2                               hd09550d_5
numpy                                 1.23.3                               py310hd5efca6_1
numpy-base                          1.23.3                               py310h8e6c178_1
openssl                              1.1.1s                                h7f8727e_0
pip                                  22.2.2                               py310h06a4308_0
pyparser                             2.21                                  pypi_0 pypi
python                               3.10.6                               haa1d7c7_1
readline                             8.2                                   h5eee18b_0
scipy                                 1.9.3                                py310hd5efca6_0
setuptools                           65.5.0                               py310h06a4308_0
six                                   1.16.0                               pyhd3eb1b0_1
sqlite                                3.39.3                               h5082296_0
tk                                    8.6.12                               h1ccaba5_0
typing-extensions                    4.3.0                                py310h06a4308_0
typing_extensions                    4.3.0                                py310h06a4308_0
tzdata                               2022f                                 h04d1e81_0
wheel                                 0.37.1                               pyhd3eb1b0_0
```

xz	5.2.6	h5eee18b_0
zlib	1.2.13	h5eee18b_0

Removing Environments

If you don't intend to upload the environment to Object Storage for the OML users in the database, you can simply exit the notebook session and it will go out of scope. Alternatively, it can be explicitly removed using the `env remove` command. Remove the `myenv` environment and verify it was removed. A best practice is to deactivate the environment prior to removal. For help on the `env remove` command, type `env remove --help` in the `%conda` interpreter.

- Deactivate the environment.

```
%conda
deactivate

Conda environment deactivated
```

- Remove the environment.

```
%conda
env remove -n myenv
```

List the environment to see if the environment is removed or not.

```
env list

# conda environments:
#
myenv          /u01/.conda/envs/myenv
base           * /usr
conda-pack-env /usr/envs/conda-pack-env
```

Remove all packages in environment `/u01/.conda/envs/myenv`.

Specify Packages for Installation

Install Packages from the conda-forge Channel

Conda channels are the locations where packages are stored. They serve as the base for hosting and managing packages. Conda packages are downloaded from remote channels, which are URLs to directories containing conda packages. The `conda` command searches a set of channels. By default, packages are automatically downloaded and updated from the default channel. The `conda-forge` channel is free for all to use. You can modify what remote channels are automatically searched. You might want to do this to maintain a private or internal channel. We use the `conda-forge` channel, a community channel made up of thousands of contributors, in the following examples.

- Install a specific version of a Package.
To install a specific version of a package, use `<package_name>=<version>`.

- Create an environment using conda-forge.

```
%conda  
  
create -n mychannelenv -c conda-forge python=3.10  
  
activate mychannelenv
```

- Install a package from conda-forge by specifying the channel.

```
%conda  
  
install scipy --channel conda-forge
```

- Install a specific version of a package.

```
%conda  
  
install scipy=0.15.0
```

6.2 Administrative Tasks for Creating and Saving a Conda Environment

In OML Notebooks, user ADMIN can manage the lifecycle of the OML user's conda environments, including creating and deleting environments and installing and deleting packages.

The conda environments created by user ADMIN are stored in an Object Storage bucket folder associated with the Autonomous Database instance. OML users can download these conda environments using enhanced conda commands. Conda environments are available after they are downloaded and activated using the download and activate functions in a `%conda` paragraph. An activated environment is available until it is deactivated.

Create a Conda environment

As an ADMIN user in an OML notebook, specify a conda interpreter in a paragraph using `%conda`, then use the `create` command to create a conda environment named `sbenv` to install the seaborn package. Specify the Python version using the `python` parameter. Here, Python 3.10 is used for compatibility with OML4Py.

 **Note:**

When conda installs a package into an environment, it also installs any required dependencies. As a best practice, to avoid dependency conflicts, simultaneously install all the packages you need in a specific environment.

**Note:**

Specify `python=3.10.8` when creating a conda environment for a 3rd-party package to avoid inconsistencies.

```
%conda  
  
create -n sbenv python==3.10.8 seaborn
```

Upload the environment to Object Storage

Upload the environment to the Object Storage associated with the Autonomous Database instance. Here you provide an environment description and a tag corresponding to an application name, OML4Py.

```
%conda  
  
upload sbenv --description 'Conda environment with seaborn' -t application  
"OML4PY"
```

```
Uploading conda environment sbenv  
Upload successful for conda environment sbenv
```

The environment is now available for an OML user to download. The uploaded environment will persist in Object Storage until it is deleted. The application tag is required for use with embedded execution. For example, OML4Py embedded Python execution works with conda environments containing the OML4Py tag, and OML4R embedded R execution works with conda environments containing the OML4R tag.

There is one Object Storage bucket for each data center region. The conda environments are saved to a folder in Object Storage corresponding to the tenancy and database. The folder is managed by Autonomous Database and only available to users through OML Notebooks. There is an 8G maximum size for a single conda environment, and no size limit on Object Storage.

Logged in as a non-administrator user, specify the conda interpreter in a notebook paragraph using `%conda`. Get the list of environments saved in Object Storage using the `list-saved-envs` command.

```
%conda  
  
list-saved-envs
```

Provide the environment name as an argument to the `-e` parameter and request a list of packages installed in the environment.

```
%conda  
  
list-saved-envs -e sbenv --installed-packages
```


The output is similar to the following:

```
{
  "name": "sbenv",
  "size": "1.7 GiB",
  "description": "Conda environment with seaborn",
  "tags": {
    "application": "OML4PY"
  },
  "number_of_installed_packages": 78,
  "installed_packages": [
    "blas-1.0-mkl",
    "bottleneck-1.3.5-py39h7deecbd_0",
    "brotli-1.0.9-h5eee18b_7",
    "brotli-bin-1.0.9-h5eee18b_7",
    "ca-certificates-2022.07.19-h06a4308_0",
    "certifi-2022.9.14-py39h06a4308_0",
    "cycler-0.11.0-pyhd3eb1b0_0",
    "dbus-1.13.18-hb2f20db_0",
    "expat-2.4.4-h295c915_0",
    "fftw-3.3.9-h27cfd23_1",
    "fontconfig-2.13.1-h6c09931_0",
    "fonttools-4.25.0-pyhd3eb1b0_0",
    "freetype-2.11.0-h70c0345_0",
    "giflib-5.2.1-h7b6447c_0",
    "glib-2.69.1-h4ff587b_1",
    "gst-plugins-base-1.14.0-h8213a91_2",
    "gststreamer-1.14.0-h28cd5cc_2",
    "icu-58.2-he6710b0_3",
    "intel-openmp-2021.4.0-h06a4308_3561",
    "jpeg-9e-h7f8727e_0",
    "kiwisolver-1.4.2-py39h295c915_0",
    "lcms2-2.12-h3be6417_0",
    "ld_impl_linux-64-2.38-h1181459_1",
    "lerc-3.0-h295c915_0",
    "libbrotlicommon-1.0.9-h5eee18b_7",
    "libbrotlidec-1.0.9-h5eee18b_7",
    "libbrotlienc-1.0.9-h5eee18b_7",
    "libdeflate-1.8-h7f8727e_5",
    "libffi-3.3-he6710b0_2",
    "libgcc-ng-11.2.0-h1234567_1",
    "libgfortran-ng-11.2.0-h00389a5_1",
    "libgfortran5-11.2.0-h1234567_1",
    "libpng-1.6.37-hbc83047_0",
    "libstdcxx-ng-11.2.0-h1234567_1",
    "libtiff-4.4.0-hecacb30_0",
    "libuuid-1.0.3-h7f8727e_2",
    "libwebp-1.2.2-h55f646e_0",
    "libwebp-base-1.2.2-h7f8727e_0",
    "libxcb-1.15-h7f8727e_0",
    "libxml2-2.9.14-h74e7548_0",
    "lz4-c-1.9.3-h295c915_1",
    "matplotlib-3.5.2-py39h06a4308_0",
    "matplotlib-base-3.5.2-py39hf590b9c_0",
    "mkl-2021.4.0-h06a4308_640",
    "mkl-service-2.4.0-py39h7f8727e_0",
```

```

"mkl_fft-1.3.1-py39hd3c417c_0",
"mkl_random-1.2.2-py39h51133e4_0",
"munkres-1.1.4-py_0",
"ncurses-6.3-h5eee18b_3",
"numexpr-2.8.3-py39h807cd23_0",
"numpy-1.22.3-py39he7a7128_0",
"numpy-base-1.22.3-py39hf524024_0",
"openssl-1.1.1q-h7f8727e_0",
"packaging-21.3-pyhd3eb1b0_0",
"pandas-1.4.4-py39h6a678d5_0",
"pcre-8.45-h295c915_0",
"pillow-9.2.0-py39hace64e9_1",
"pip-22.1.2-py39h06a4308_0",
"pyparsing-3.0.9-py39h06a4308_0",
"pyqt-5.9.2-py39h2531618_6",
"python-3.9.0-hdb3f193_2",
"python-dateutil-2.8.2-pyhd3eb1b0_0",
"pytz-2022.1-py39h06a4308_0",
"qt-5.9.7-h5867ecd_1",
"readline-8.1.2-h7f8727e_1",
"scipy-1.7.3-py39h6c91a56_2",
"seaborn-0.11.2-pyhd3eb1b0_0",
"setuptools-63.4.1-py39h06a4308_0",
"sip-4.19.13-py39h295c915_0",
"six-1.16.0-pyhd3eb1b0_1",
"sqlite-3.39.2-h5082296_0",
"tk-8.6.12-h1ccaba5_0",
"tornado-6.2-py39h5eee18b_0",
"tzdata-2022c-h04d1e81_0",
"wheel-0.37.1-pyhd3eb1b0_0",
"xz-5.2.5-h7f8727e_1",
"zlib-1.2.12-h5eee18b_3",
"zstd-1.5.2-ha4553b6_0"
]
}

```

Delete an environment saved in an Object Storage

Use the `delete` command to delete an environment saved in an Object Storage.



Note:

Only user ADMIN can delete an environment saved in an Object Storage.

```
%conda
```

```
delete sbenv
```

```
Deleting conda environment sbenv
```

```
Deletion successful for conda environment sbenv
```

6.3 OML User Tasks for Downloading an Available Conda Environment

Once user ADMIN installs the environment in Object Storage in the Autonomous Database, as an OML user, you can download, activate, and use it in Python paragraphs in notebooks and with embedded execution.

List all environments persisted in Object Storage

Get the list of environments saved in Object Storage using the `list-saved-envs` command.

```
%conda  
  
list-saved-envs
```

Get information on a named environment persisted in Object Storage

Provide the environment name as an argument to the `-e` parameter and request information on the environment.

```
%conda  
  
list-saved-envs -e sbenv
```

The output is similar to the following:

```
{  
  "name": "sbenv",  
  "size": "1.2 GiB",  
  "description": "Conda environment with seaborn",  
  "tags": {  
    "application": "OML4PY"  
  },  
  "number_of_installed_packages": 60  
}
```

Download and activate the environment

Use the `download` command to download an environment from Object Storage. To activate the downloaded environment, use the `activate` command.



Note:

The paragraph that contains the download command must be the first paragraph in the notebook.

```
%conda
```

```
download sbenv
activate sbenv
```

```
Downloading conda environment sbenv
Download successful for conda environment sbenv
```

List the packages available in the environment

Get the list of all the packages in an active environment using the `list` command.

```
%conda

list
```

The output is similar to the following:

```
# packages in environment at /u01/.conda/envs/sbenv:
#
# Name                Version                Build Channel
blas                  1.0                    mkl
bottleneck            1.3.5                  py39h7deecbd_0
brotli                1.0.9                  h5eee18b_7
brotli-bin            1.0.9                  h5eee18b_7
ca-certificates      2022.07.19             h06a4308_0
certifi               2022.9.14              py39h06a4308_0
cyclер                0.11.0                 pyhd3eb1b0_0
dbus                  1.13.18                hb2f20db_0
expat                 2.4.4                  h295c915_0
fftw                  3.3.9                  h27cfd23_1
fontconfig            2.13.1                 h6c09931_0
fonttools             4.25.0                 pyhd3eb1b0_0
freetype              2.11.0                 h70c0345_0
giflib                5.2.1                  h7b6447c_0
glib                  2.69.1                 h4ff587b_1
gst-plugins-base     1.14.0                 h8213a91_2
gstreamer             1.14.0                 h28cd5cc_2
icu                   58.2                   he6710b0_3
intel-openmp          2021.4.0               h06a4308_3561
jpeg                  9e                     h7f8727e_0
kiwisolver            1.4.2                  py39h295c915_0
lcms2                 2.12                   h3be6417_0
ld_impl_linux-64     2.38                   h1181459_1
lerc                  3.0                    h295c915_0
libbrotlicommon      1.0.9                  h5eee18b_7
libbrotlidec         1.0.9                  h5eee18b_7
libbrotlienc         1.0.9                  h5eee18b_7
libdeflate           1.8                    h7f8727e_5
libffi                3.3                    he6710b0_2
libgcc-ng            11.2.0                 h1234567_1
libgfortran-ng       11.2.0                 h00389a5_1
libgfortran5         11.2.0                 h1234567_1
libpng                1.6.37                 hbc83047_0
libstdcxx-ng         11.2.0                 h1234567_1
libtiff               4.4.0                  hecacb30_0
```

libuuid	1.0.3	h7f8727e_2
libwebp	1.2.2	h55f646e_0
libwebp-base	1.2.2	h7f8727e_0
libxcb	1.15	h7f8727e_0
libxml2	2.9.14	h74e7548_0
lz4-c	1.9.3	h295c915_1
matplotlib	3.5.2	py39h06a4308_0
matplotlib-base	3.5.2	py39hf590b9c_0
mkl	2021.4.0	h06a4308_640
mkl-service	2.4.0	py39h7f8727e_0
mkl_fft	1.3.1	py39hd3c417c_0
mkl_random	1.2.2	py39h51133e4_0
munkres	1.1.4	py_0
ncurses	6.3	h5eee18b_3
numexpr	2.8.3	py39h807cd23_0
numpy	1.22.3	py39he7a7128_0
numpy-base	1.22.3	py39hf524024_0
openssl	1.1.1q	h7f8727e_0
packaging	21.3	pyhd3eb1b0_0
pandas	1.4.4	py39h6a678d5_0
pcre	8.45	h295c915_0
pillow	9.2.0	py39hace64e9_1
pip	22.1.2	py39h06a4308_0
pyparsing	3.0.9	py39h06a4308_0
pyqt	5.9.2	py39h2531618_6
python	3.9.0	hdb3f193_2
python-dateutil	2.8.2	pyhd3eb1b0_0
pytz	2022.1	py39h06a4308_0
qt	5.9.7	h5867ecd_1
readline	8.1.2	h7f8727e_1
scipy	1.7.3	py39h6c91a56_2
seaborn	0.11.2	pyhd3eb1b0_0
setuptools	63.4.1	py39h06a4308_0
sip	4.19.13	py39h295c915_0
six	1.16.0	pyhd3eb1b0_1
sqlite	3.39.2	h5082296_0
tk	8.6.12	h1ccaba5_0
tornado	6.2	py39h5eee18b_0
tzdata	2022c	h04d1e81_0
wheel	0.37.1	pyhd3eb1b0_0
xz	5.2.5	h7f8727e_1
zlib	1.2.12	h5eee18b_3
zstd	1.5.2	ha4553b6_0

Example 6-1 Create a visualization using seaborn

The following example shows the use of the available packages in the installed and activated environment. It imports `pandas`, `seaborn`, and `matplotlib` packages and loads the `iris` dataset from the `seaborn` library as a `pandas` dataframe. The `pairplot` `seaborn` function plots the pair-wise relationship between all the variables of the dataset.

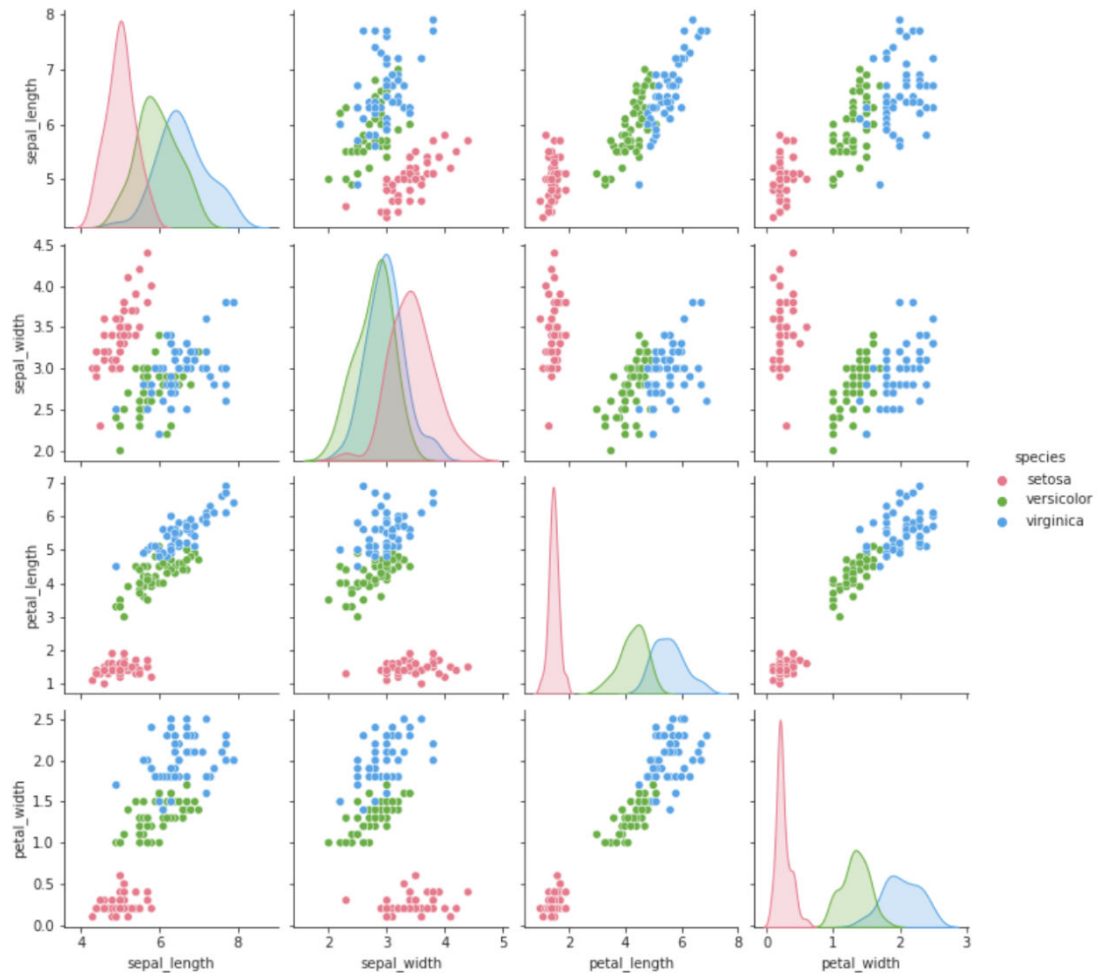
```
%python

import pandas as pd
import seaborn as sb
from matplotlib import pyplot as plt
```

```
df = sb.load_dataset('iris')
sb.set_style("ticks")
sb.pairplot(df,hue = 'species',diag_kind = "kde",kind = "scatter",palette =
"husl")
plt.show()
```

The output of the example is the following.

Figure 6-1 Iris pair plot



Example 6-2 Create a string representation of the function and save it to the OML4Py script repository

With OML4Py, functions are saved to the script repository using their string definition representation so they can be run in embedded Python execution. Create a function `sb_plot`, after verifying the function behaves as expected, provide it as a string (within triple quotes for formatting), and save it to the OML4Py script repository. Use the `oml.script.create` function to store a single user-defined Python function in the OML4Py script repository. The parameter

"sb_plot" is a string that specifies the name of the user-defined function. The parameter `func=sb_plot` is the Python function to run.

```
%python

sb_plot = """def sb_plot():
    import pandas as pd
    import seaborn as sb
    from matplotlib import pyplot as plt
    df = sb.load_dataset('iris')
    sb.set_style("ticks")
    sb.pairplot(df,hue = 'species',diag_kind = "kde",kind = "scatter",palette
= "husl")
    plt.show()"""

oml.script.create("sb_plot", func=sb_plot)
```

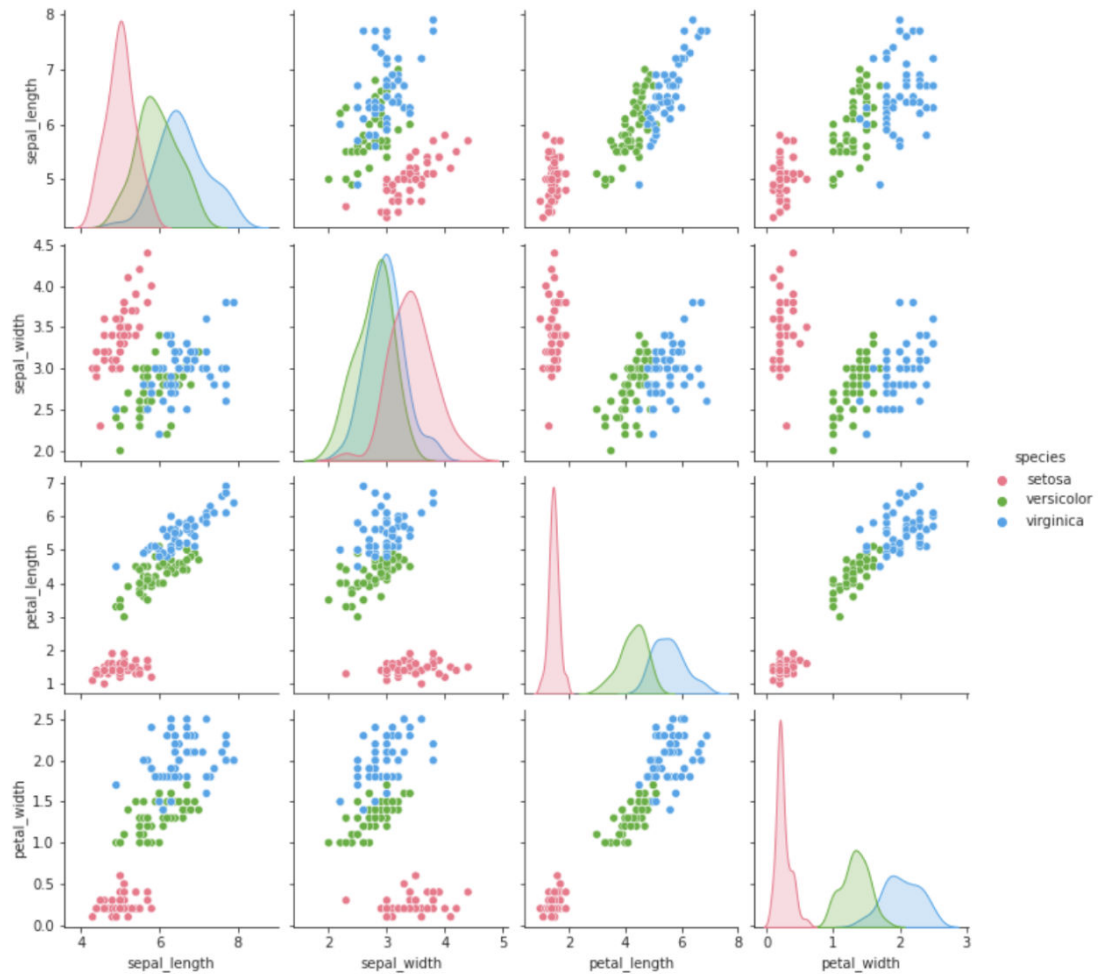
Use the Python API for embedded Python execution to run the user-defined Python function you saved in the script repository.

```
%python

oml.do_eval(func="sb_plot", graphics=True)
```

The output of the example is the following:

Figure 6-2 Iris pair plot



Deactivate the current environment

Use the `deactivate` command to deactivate an environment.



Note:

At a given time, only one active environment is supported. So, a newly activated environment would replace an old environment. As a best practice, deactivate an environment before logging off.

```
%conda
```

```
deactivate
```

```
Conda environment deactivated
```


6.4 Using Conda Environments with Embedded Python Execution

This topic explains the usage of conda environments by running user-defined functions (UDFs) in SQL and REST APIs for embedded Python execution.

Running UDFs in the SQL and REST APIs for embedded Python execution

The conda environments can be used by OML4Py Python, SQL, and REST APIs. To use the SQL and REST API for embedded Python execution, the following information is needed.

1. The token URL from the OML service console in Autonomous Database. For more information on how to obtain the token URL and set the access token see Access and Authorization Procedures and Functions (Autonomous Database).
2. A script containing a user-defined Python function in the Oracle Machine Learning for Python (OML4Py) script repository. For information on creating a script and saving it to the script repository, see About Embedded Python Execution and the Script Repository.

Note:

To use a conda environment when calling OML4Py script execution endpoints, specify the conda environment in the `env_name` field when using SQL, and the `envName` field when using REST.

Run the Python UDF using the SQL API for embedded Python execution - Asynchronous mode

Run a `SELECT` statement that calls the `pyqEval` function. The `PAR_LST` argument specifies the special control argument `oml_graphics_flag` to `true` so that the web server can capture images rendered in the invoked script, the `oml_async_flag` is set to `true` to submit the job asynchronously. In the `OUT_FMT` argument, the string `'PNG'`, specifies that the table returns the response in a table with fixed columns (including an image bytes column). The `SCR_NAME` parameter specifies the function `sb_plot` stored in the script repository. The `ENV_NAME` specifies the environment name `mysbenv` in which the script is called.

```
%script

set long 2000

SELECT * FROM table(pyqEval(
  par_lst => '{"oml_graphics_flag":true, "oml_async_flag":true}',
  out_fmt => 'PNG',
  scr_name => 'sb_plot',
  scr_owner=> NULL,
  env_name => 'mysbenv'));
```

The output is similar to the following:

```
NAME VALUE
-----
```

```
https://gcc59e2cf7a6f5f-oml4.adb-compdev1.us-
phoenix-1.oraclecloudapps.com/oml/api/py-scripts/v1/jobs/b82947a7-ec3a-4ca6-
bf86-54b3f2b3a4b0
```

Get the job status

Poll the job status using the `pyqJobStatus` function. If the job is still running, the return value will note that the job is still running. When the job completes, a job ID and result location are returned.

```
%script

set long 1000
SELECT VALUE from pyqJobStatus(job_id => 'b82947a7-ec3a-4ca6-
bf86-54b3f2b3a4b0');
```

The output returns a job ID:

```
NAME VALUE
-----
https://gcc59e2cf7a6f5f-oml4.adb-compdev1.us-
phoenix-1.oraclecloudapps.com/oml/api/py-scripts/v1/jobs/b82947a7-ec3a-4ca6-
bf86-54b3f2b3a4b0/result
```

Retrieve the result

```
%script

set long 500
SELECT NAME, ID, VALUE, dbms_lob.substr(image,100,1) image FROM
pyqJobResult(job_id => 'b82947a7-ec3a-4ca6-bf86-54b3f2b3a4b0',
out_fmt=>'PNG');
```

The output is similar to the following:

```
NAME ID VALUE IMAGE
-----
1
[{"0":0.0,"1":0.0,"2":0.2333333333,"accuracy":0.2333333333,"macro
avg":0.0777777778,"weighted avg":0.0544444444},
{"0":0.0,"1":0.0,"2":1.0,"accuracy":0.2333333333,"macro
avg":0.3333333333,"weighted avg":0.2333333333},
{"0":0.0,"1":0.0,"2":0.3783783784,"accuracy":0.2333333333,"macro
avg":0.1261261261,"weighted avg":0.0882882883},
{"0":11.0,"1":12.0,"2":7.0,"accuracy":0.2333333333,"macro avg":30.0,"weighted
avg":30.0}]
89504E470D0A1A0A0000000D494844520000046A000003E808060000008668185B000000397445
5874536F667477617265004D6174706C6F746C69622076657273696F6E332E362E322C20687474
70733A2F2F6D6174706C6F746C69622E6F72672F28E8
```

Run the Python UDF using the REST API for embedded Python execution

The following example runs the script named `sb_plot` in the OML4Py REST API for embedded Python execution. The environment name parameter `envName` is set to `mysbenv`. The `graphicsFlag` parameter is set to `true` to return the PNG image and the data from the function in JSON format.

```
$ curl -i -X POST --header "Authorization: Bearer ${token}" \
--header 'Content-Type: application/json' --header 'Accept: application/json' \
\
-d '{"envName":"mysbenv", "graphicsFlag":true, "service":"LOW"}' \
"${omlserver}/oml/api/py-scripts/v1/do-eval/sb_plot"
```

The output is similar to the following:

```
NAME ID VALUE IMAGE
-----
1 [{"0":0.0,"1":0.0,"2":0.2333333333,"accuracy":0.2333333333,"macro
avg":0.0777777778,"weighted avg":0.0544444444},
{"0":0.0,"1":0.0,"2":1.0,"accuracy":0.2333333333,"macro
avg":0.3333333333,"weighted avg":0.2333333333},
{"0":0.0,"1":0.0,"2":0.3783783784,"accuracy":0.2333333333,"macro
avg":0.1261261261,"weighted avg":0.0882882883},
{"0":11.0,"1":12.0,"2":7.0,"accuracy":0.2333333333,"macro avg":30.0,"weighted
avg":30.0}]
89504E470D0A1A0A0000000D494844520000046A000003E808060000008668185B000000397445
5874536F667477617265004D6174706C6F746C69622076657273696F6E332E362E322C20687474
70733A2F2F6D6174706C6F746C69622E6F72672F28E8
```

7

Get Started with Oracle Machine Learning for Python

Learn how to use OML4Py in Oracle Machine Learning Notebooks and how to move data between the local Python session and the database.

These actions are described in the following topics.

- [Use OML4Py with Oracle Autonomous Database](#)
- [Move Data Between the Database and a Python Session](#)
- [Save Python Objects in the Database](#)
- [Use OML4Py with Oracle Autonomous Database](#)
OML4Py is available through the Python interpreter in Oracle Machine Learning Notebooks in Oracle Autonomous Database.
- [Use OML4Py with an On-Premises Oracle Database](#)
After the OML4Py server and client components have been installed on your on-premises Oracle database server and you have installed the OML4Py client on your local system, you can connect your client Python session to the OML4Py server.
- [Move Data Between the Database and a Python Session](#)
With OML4Py functions, you can interact with data structures in a database schema.
- [Save Python Objects in the Database](#)
You can save Python objects in OML4Py datastores, which persist in the database.

7.1 Use OML4Py with Oracle Autonomous Database

OML4Py is available through the Python interpreter in Oracle Machine Learning Notebooks in Oracle Autonomous Database.

For more information, see *Get Started with Notebooks for Data Analysis and Data Visualization in Using Oracle Machine Learning Notebooks*.

7.2 Use OML4Py with an On-Premises Oracle Database

After the OML4Py server and client components have been installed on your on-premises Oracle database server and you have installed the OML4Py client on your local system, you can connect your client Python session to the OML4Py server.

To connect an OML4Py client to an on-premises Oracle database, you first import the `oml` module and then connect as described in the following topics.

- [About Connecting to an On-Premises Oracle Database](#)
OML4Py client components connect a Python session to the OML4Py server components on an on-premises Oracle database server.
- [About Oracle Wallets](#)
An Oracle wallet is a secure software container that stores authentication and signing credentials for an Oracle Database.

- [Connect to an Oracle Database](#)
Establish an OML4Py connection to an on-premises Oracle database with `oml.connect`.

7.2.1 About Connecting to an On-Premises Oracle Database

OML4Py client components connect a Python session to the OML4Py server components on an on-premises Oracle database server.

The connection makes the data in an on-premises Oracle database schema available to the Python user. It also makes the processing power, memory, and storage capacities of the database server available to the Python session through the OML4Py client interface. To use that data and those capabilities, you must create a connection to the Oracle database server.

To use the Automatic Machine Learning (AutoML) capabilities of OML4Py, the following must be true:

- A connection pool must be running on the server.
- You must explicitly use the `automl` argument in an `oml.connect` invocation to specify the running connection pool on the server.

Note:

Before you can create an AutoML connection, a database administrator must first activate the database-resident connection pool in your on-premises Oracle database by issuing the following SQL statement:

```
EXECUTE DBMS_CONNECTION_POOL.START_POOL();
```

Once started, the connection pool remains in this state until a database administrator explicitly stops it by issuing the following command:

```
EXECUTE DBMS_CONNECTION_POOL.STOP_POOL();
```

Note:

Because an AutoML connection requires more database resources than an `oml.connect` connection without AutoML does, you should create an AutoML connection only if you are going to use the AutoML classes.

 **Note:**

- Only one type of connection can be active during a Python session: either a connection with AutoML enabled or one without it enabled. You can, however, terminate one type of connection and initiate the other type during the same Python session. Terminating either type of connection results in the automatic clean up of any temporary objects created in the session during that connection.

If you want to save any objects that you created in one type of connection before changing to the other type, then save the objects in an OML4Py datastore before invoking `oml.connect` again. You can then reload the objects after reconnecting.
- The `oml.connect` function uses the `cx_Oracle` Python package for database connectivity. In some cases, you might want to use the `cx_Oracle.connect` function of that package to connect to a database. That function has advantages such as the following:

- Allows multiple connections to a multiple databases, which might be useful in an running Embedded Python Execution functions
- Permits some SQL data manipulation language (DML) operations that are not available in an `oml.connect` connection

For information on the `cx_Oracle.connect` function, see [Connecting to Oracle Database](#) in the `cx_Oracle` documentation.

OML4Py Connection Functions

The OML4Py functions related to database connections are the following.

Table 7-1 Connection Functions for OML4Py

Function	Description
<code>oml.connect</code>	Establishes an OML4Py connection to an Oracle database.
<code>oml.disconnect</code>	Terminates the Oracle database connection.
<code>oml.isconnected</code>	Indicates whether an active Oracle database connection exists.
<code>oml.check_embed</code>	Indicates whether Embedded Python Execution is enabled in the connected Oracle database.

7.2.2 About Oracle Wallets

An Oracle wallet is a secure software container that stores authentication and signing credentials for an Oracle Database.

You can create an OML4Py connection to an Oracle Database instance by specifying an Oracle wallet. For instructions on creating an Oracle wallet, see *Managing the Secure External Password Store for Password Credentials* in *Oracle Database Security Guide*.

The Oracle wallet must contain a credential that specifies a `tnsnames.ora` entry such as the following:

```
waltcon = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp) (HOST=myhost) (PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=myserv.example.com)))
```

To be able to use an Oracle wallet to create an OML4Py connection in which you can use Automatic Machine Learning (AutoML), the wallet must also have a credential that has a `tnsnames.ora` entry for a server connection pool such as the following:

```
waltcon_pool = (DESCRIPTION= (ADDRESS=(PROTOCOL=tcp)(HOST=myhost)
(PORT=1521)) (CONNECT_DATA=(SID=mysid) (SERVER=pooled)))
```



Note:

Before you can create an AutoML connection, a database administrator must first activate the database-resident connection pool in your on-premises Oracle database by issuing the following SQL statement:

```
EXECUTE DBMS_CONNECTION_POOL.START_POOL();
```

Once started, the connection pool remains in this state until a database administrator explicitly stops it by issuing the following command:

```
EXECUTE DBMS_CONNECTION_POOL.STOP_POOL();
```

For examples of creating a connection using an Oracle wallet, see [Example 7-6](#) and [Example 7-7](#).

7.2.3 Connect to an Oracle Database

Establish an OML4Py connection to an on-premises Oracle database with `oml.connect`.

The `oml.connect` function establishes a connection to the user's schema in an on-premises Oracle database.

The syntax of the `oml.connect` function is the following.

```
oml.connect(user=None, password=None, host=None, port=None, sid=None,
service_name=None, dsn=None, encoding='UTF-8', nencoding='UTF-8', automl=None)
```

To create a basic connection to the database, you can specify arguments to the `oml.connect` function in the following mutually exclusive combinations:

- `user, password, dsn`
- `user, password, host, port, sid`
- `user, password, host, port, service_name`

The arguments specify the following values

Table 7-2 Parameters to `oml.connect`

Parameter	Description
<code>user</code>	A string specifying a username.
<code>password</code>	A string specifying the password for the user.
<code>host</code>	A string specifying the name of the host machine on which the OML4Py server is installed.

Table 7-2 (Cont.) Parameters to oml.connect

Parameter	Description
port	An int or a string specifying the Oracle database port number on the host machine.
sid	A string specifying the system identifier (SID) of the Oracle database.
service_name	A string specifying the service name of the Oracle database.
dsn	A string specifying a data source name, which can be a TNS entry for the database or a TNS alias in an Oracle Wallet.
encoding	A string specifying the encoding to use for regular database strings.
nencoding	A string specifying the encoding to use for national character set database strings.
automl	A string or a boolean specifying whether to enable an Automatic Machine Learning (AutoML) connection, which uses the database-resident connection pool. If there is a connection pool running for a host, port, SID (or service name), then you can specify that host, port, SID (or service name) and <code>automl=True</code> . If the <code>dsn</code> argument is a data source name, then the <code>automl</code> argument must be a data source name for a running connection pool. If the <code>dsn</code> argument is a TNS alias, then the <code>automl</code> argument must be a TNS alias for a connection pool specified in an Oracle Wallet.

To use the AutoML capabilities of OML4Py, the following must be true:

- A connection pool must be running on the server.
- You must explicitly use the `automl` argument in an `oml.connect` invocation to specify the running connection pool on the server.

 **Note:**

Before you can create an AutoML connection, a database administrator must first activate the database-resident connection pool in your on-premises Oracle database by issuing the following SQL statement:

```
EXECUTE DBMS_CONNECTION_POOL.START_POOL();
```

Once started, the connection pool remains in this state until a database administrator explicitly stops it by issuing the following command:

```
EXECUTE DBMS_CONNECTION_POOL.STOP_POOL();
```

Only one active OML4Py connection can exist at a time during a Python session. If you call `oml.connect` when an active connection already exists, then the `oml.disconnect` function is implicitly invoked, any temporary objects that you created in the previous session are discarded, and the new connection is established. Before attempting to connect, you can discover whether an active connection exists by using the `oml.isconnected` function.

You explicitly end a connection with the `oml.disconnect` function. If you do not invoke `oml.disconnect`, then the connection is automatically terminated when the Python session ends.

Examples

In the following examples, the values of some of the arguments to the `oml.connect` function are string variables that are not declared in the example. To use any of the following examples, replace the username, password, port, and variable argument values with the values for your user and database.

Example 7-1 Connecting with a Host, Port, and SID

This example uses the `host`, `port`, and `sid` arguments. It also shows the use of the `oml.isconnected`, `oml.check_embed`, and `oml.disconnect` functions.

```
import oml

oml.connect(user='oml_user', password='oml_user_password', host='myhost',
            port=1521, sid='mysid')

# Verify that the connection exists.
oml.isconnected()

# Find out whether Embedded Python Execution is enabled in the
# database instance.
oml.check_embed()

# Disconnect from the database.
oml.disconnect()

# Verify that the connection has been terminated.
oml.isconnected()
```

Listing for This Example

```
>>> import oml
>>>
>>> oml.connect(user='oml_user', password='oml_user_password', host='myhost',
...           port=1521, sid='mysid')
>>>
>>> # Verify that the connection exists.
... oml.isconnected()
True
>>>
>>> # Find out whether Embedded Python Execution is enabled in the
... # database instance.
... oml.check_embed()
True
>>>
>>> # Disconnect from the database.
... oml.disconnect()
>>>
>>> # Verify that the connection has been terminated.
... oml.isconnected()
False
```

Example 7-2 Connecting with Host, Port, and Service Name

This example uses the `host`, `port`, and `service_name` arguments.

```
import oml

oml.connect(user='oml_user', password='oml_user_password', host='myhost',
            port=1521, service_name='myservice')
```

Example 7-3 Connecting with a DSN Containing a SID

This example uses the `dsn` argument to specify a SID.

```
import oml

mydsn = "(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myhost)(PORT=1521))\
        (CONNECT_DATA=(SID=mysid)))"
oml.connect(user='oml_user', password='oml_user_password', dsn=mydsn)
```

Example 7-4 Connecting with a DSN Containing a Service Name

This example uses the `dsn` argument to specify a service name.

```
import oml

myinst = "(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myhost)\
        (PORT=1521))\
        (CONNECT_DATA=(SERVICE_NAME=myservice.example.com)))"
oml.connect(user='oml_user', password='oml_user_password', dsn=myinst)
```

Example 7-5 Creating a Connection with a DSN and with AutoML Enabled

This example creates an OML4Py connection with AutoML enabled. The example connects to a local database.

```
import oml

mydsn = "(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myhost)\
        (PORT=1521))(CONNECT_DATA=(SID=mysid)))"

dsn_pool = "(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myhost)\
        (PORT=1521))\
        (CONNECT_DATA=(SERVICE_NAME=myservice.example.com)\
        (SERVER=POOLED)))"

oml.connect(user='oml_user', password='oml_user_password',
            dsn=mydsn, automl=dsn_pool)

# Verify that the connection exists and that AutoML is enabled.
oml.isconnected(check_automl=True)
```

Example 7-6 Connecting with an Oracle Wallet

This example creates a connection using the `dsn` argument to specify an Oracle wallet. The `dsn` value, `waltcon` in the example, must refer to the alias in the database `tnsnames.ora` file that was used to create the appropriate credential in the wallet.

```
import oml

oml.connect(user='', password='', dsn='waltcon')
```



See Also:

[About Oracle Wallets](#)

Example 7-7 Connecting with an Oracle Wallet with AutoML Enabled

This example connects using an Oracle wallet to establish a connection with AutoML enabled by using the `dsn` and `automl` arguments. The example then verifies that the connection has AutoML enabled. The `dsn` and `automl` values, `waltcon` and `waltcon_pool` in the example, must refer to aliases in the database `tnsnames.ora` file that were used to create the appropriate credentials in the wallet.

```
import oml

oml.connect(user='', password='', dsn='waltcon', automl='waltcon_pool')
oml.isconnected(check_automl=True)
```

7.3 Move Data Between the Database and a Python Session

With OML4Py functions, you can interact with data structures in a database schema.

In your Python session, you can move data to and from the database and create temporary or persistent database tables. The OML4Py functions that perform these actions are described in the following topics.

- [About Moving Data Between the Database and a Python Session](#)
- [Push Local Python Data to the Database](#)
- [Pull Data from the Database to a Local Python Session](#)
- [Create a Python Proxy Object for a Database Object](#)
- [Create a Persistent Database Table from a Python Data Set](#)
- [About Moving Data Between the Database and a Python Session](#)
Using the functions described in this topic, you can move data between the your local Python session and an Oracle database schema.
- [Push Local Python Data to the Database](#)
Use the `oml.push` function to push data from your local Python session to a temporary table in your Oracle database schema.
- [Pull Data from the Database to a Local Python Session](#)
Use the `pull` method of an `oml` proxy object to create a Python object in your local Python session.

- [Create a Python Proxy Object for a Database Object](#)
Use the `oml.sync` function to create a Python object as a proxy for a database table, view, or SQL statement.
- [Create a Persistent Database Table from a Python Data Set](#)
Use the `oml.create` function to create a persistent table in your database schema from data in your Python session.

7.3.1 About Moving Data Between the Database and a Python Session

Using the functions described in this topic, you can move data between the your local Python session and an Oracle database schema.

The following functions create proxy `oml` Python objects from database objects, create database tables from Python objects, list the objects in the workspace, and drop tables and views.

Function	Definition
<code>oml.create</code>	Creates a persistent database table from a Python data set.
<code>oml.cursor</code>	Returns a <code>cx_Oracle cursor</code> object for the current OML4Py database connection.
<code>oml.dir</code>	Returns the names of the <code>oml</code> objects in the workspace.
<code>oml.drop</code>	Drops a persistent database table or view.
<code>oml_object.pull</code>	Creates a local Python object that contains a copy of the database data referenced by the <code>oml</code> object.
<code>oml.push</code>	Pushes data from the OML Notebooks Python session memory into a temporary table in the database.
<code>oml.sync</code>	Creates an <code>oml.DataFrame</code> proxy object in Python that represents a database table, view, or query.

With the `pull` method of an `oml` object, you can create a local Python object that contains a copy of the database data represented by an `oml` proxy object.

The `oml.push` function implicitly coerces Python data types to `oml` data types and the `pull` method on `oml` objects coerces `oml` data types to Python data types.

With the `oml.create` function, you can create a persistent database table and a corresponding `oml.DataFrame` proxy object from a Python data set.

With the `oml.sync` function, you can synchronize the metadata of a database table or view with the `oml` object representing the database object.

With the `oml.cursor` function, you can create a `cx_Oracle cursor` object for the current database connection. You can use the `cursor` to run queries against the database, as shown in [Example 7-13](#).

7.3.2 Push Local Python Data to the Database

Use the `oml.push` function to push data from your local Python session to a temporary table in your Oracle database schema.

The `oml.push` function creates a temporary table in the user's database schema and inserts data into the table. It also creates and returns a corresponding proxy `oml.DataFrame` object

that references the table in the Python session. The table exists as long as an `oml` object exists that references it, either in the Python session memory or in an OML4Py datastore.

The syntax of the `oml.push` function is the following:

```
oml.push(x, oranumber=True, dbtypes=None)
```

The `x` argument may be a `pandas.DataFrame` or a list of tuples of equal size that contain the data for the table. For a list of tuples, each tuple represents a row in the table and the column names are set to `COL1`, `COL2`, and so on.

The SQL data types of the columns are determined by the following:

- OML4Py determines default column types by looking at 20 random rows sampled from the table. For tables with less than 20 rows, it uses all rows in determining the column type. If the values in a column are all `None`, or if a column has inconsistent data types that are not `None` in the sampled rows, then a default column type cannot be determined and a `ValueError` is raised unless a SQL type for the column is specified by the `dbtypes` argument.
- For numeric columns, the `oranumber` argument, which is a `bool`, determines the SQL data type. If `True` (the default), then the SQL data type is `NUMBER`. If `False`, then the data type is `BINARY_DOUBLE`.
If the data in `x` contains `NaN` values, then you should set `oranumber` to `False`.
- For string columns, the default type is `VARCHAR2(4000)`.
- For bytes columns, the default type is `BLOB`.

With the `dbtypes` argument, you can specify the SQL data types for the table columns. The values of `dbtypes` may be either a `dict` that maps `str` to `str` values or a list of `str` values. For a `dict`, the keys are the names of the columns.

Example 7-8 Pushing Data to a Database Table

This example creates `pd_df`, a `pandas.core.frame.DataFrame` object with columns of various data types. It pushes `pd_df` to a temporary database table, which creates the `oml_df` object, which references the table. It then pulls the data from the `oml_df` object to the `df` object in local memory.

```
import oml
import pandas as pd

pd_df = pd.DataFrame({'numeric': [1, 1.4, -4, 3.145, 5, None],
                      'string' : [None, None, 'a', 'a', 'a', 'b'],
                      'bytes'  : [b'a', b'b', b'c', b'c', b'd', b'e]})

# Push the data set to a database table with the specified dbtypes
# for each column.
oml_df = oml.push(pd_df, dbtypes = {'numeric': 'BINARY_DOUBLE',
                                   'string': 'CHAR(1)',
                                   'bytes': 'RAW(1)'})

# Display the data type of oml_df.
type(oml_df)

# Pull the data from oml_df into local memory.
```

```

df = oml_df.pull()

# Display the data type of df.
type(df)

# Create a list of tuples.
lst = [(1, None, b'a'), (1.4, None, b'b'), (-4, 'a', b'c'),
       (3.145, 'a', b'c'), (5, 'a', b'd'), (None, 'b', b'e')]

# Create an oml.DataFrame using the list.
oml_df2 = oml.push(lst, dbtypes = ['BINARY_DOUBLE', 'CHAR(1)', 'RAW(1)'])

type(oml_df2)

```

Listing for This Example

```

>>> import oml
>>> import pandas as pd
>>>
>>> pd_df = pd.DataFrame({'numeric': [1, 1.4, -4, 3.145, 5, None],
...                       'string' : [None, None, 'a', 'a', 'a', 'b'],
...                       'bytes'  : [b'a', b'b', b'c', b'c', b'd', b'e]})
>>>
>>> # Push the data set to a database table with the specified dbtypes
... # for each column.
... oml_df = oml.push(pd_df, dbtypes = {'numeric': 'BINARY_DOUBLE',
...                                     'string': 'CHAR(1)',
...                                     'bytes': 'RAW(1)'})
>>>
>>> # Display the data type of oml_df.
... type(oml_df)
<class 'oml.core.frame.DataFrame'>
>>>
>>> # Pull the data from oml_df into local memory.
... df = oml_df.pull()
>>>
>>> # Display the data type of df.
... type(df)
<class 'pandas.core.frame.DataFrame'>
>>>
>>> # Create a list of tuples.
... lst = [(1, None, b'a'), (1.4, None, b'b'), (-4, 'a', b'c'),
...        (3.145, 'a', b'c'), (5, 'a', b'd'), (None, 'b', b'e')]
>>>
>>> # Create an oml.DataFrame using the list.
... oml_df2 = oml.push(lst, dbtypes = ['BINARY_DOUBLE', 'CHAR(1)', 'RAW(1)'])
>>>
>>> type(oml_df2)
<class 'oml.core.frame.DataFrame'>

```

7.3.3 Pull Data from the Database to a Local Python Session

Use the `pull` method of an `oml` proxy object to create a Python object in your local Python session.

 **Note:**

You can pull data to a local `pandas.DataFrame` only if the data can fit into the local Python session memory. Also, even if the data fits in memory but is still very large, you may not be able to perform many, or any, Python functions in the local Python session.

Example 7-9 Pulling Data into Local Memory

This example loads the iris data set and creates the IRIS database table and the `oml_iris` proxy object that references that table. It displays the type of the `oml_iris` object, then pulls the data from it to the `iris` object in local memory and displays its type.

```
import oml
from sklearn.datasets import load_iris
import pandas as pd

iris = load_iris()
x = pd.DataFrame(iris.data, columns = ['SEPAL_LENGTH', 'SEPAL_WIDTH',
    'PETAL_LENGTH', 'PETAL_WIDTH'])
y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:
    'virginica'}[x], iris.target)), columns = ['SPECIES'])
iris_df = pd.concat([x, y], axis=1)

oml_iris = oml.create(iris_df, table = 'IRIS')

# Display the data type of oml_iris.
type(oml_iris)

# Pull the data from oml_iris into local memory.
iris = oml_iris.pull()

# Display the data type of iris.
type(iris)

# Drop the IRIS database table.
oml.drop('IRIS')
```

Listing for This Example

```
>>> import oml
>>> from sklearn.datasets import load_iris
>>> import pandas as pd
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
>>> iris = datasets.load_iris()

>>> iris = load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['SEPAL_LENGTH', 'SEPAL_WIDTH',
    'PETAL_LENGTH', 'PETAL_WIDTH'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:
    'virginica'}[x], iris.target)), columns = ['SPECIES'])
```

```
>>> iris_df = pd.concat([x, y], axis=1)

>>> oml_iris = oml.create(iris_df, table = 'IRIS')

>>>
>>> # Display the data type of oml_iris.
... type(oml_iris)
<class 'oml.core.frame.DataFrame'>
>>>
>>> # Pull the data from oml_iris into local memory.
... iris = oml_iris.pull()
>>>
>>> # Display the data type of iris.
... type(iris)
<class 'pandas.core.frame.DataFrame'>
>>>
>>> # Drop the IRIS database table.
... oml.drop('IRIS')
```

7.3.4 Create a Python Proxy Object for a Database Object

Use the `oml.sync` function to create a Python object as a proxy for a database table, view, or SQL statement.

The `oml.sync` function returns an `oml.DataFrame` object or a dictionary of `oml.DataFrame` objects. The `oml.DataFrame` object returned by `oml.sync` is a proxy for the database object.

You can use the proxy `oml.DataFrame` object to select data from the table. When you run a Python function that selects data from the table, the function returns the current data from the database object. However, if some application has added a column to the table, or has otherwise changed the metadata of the database object, the `oml.DataFrame` proxy object does not reflect such a change until you again invoke `oml.sync` for the database object.

Tip:

To conserve memory resources and save time, you should only create proxies for the tables that you want to use in your Python session.

You can use the `oml.dir` function to list the `oml.DataFrame` proxy objects in the environment for a schema.

The syntax of the `oml.sync` function is the following:

```
oml.sync(schema=None, regex_match=False, table=None, view=None, query=None)
```

The `schema` argument in `oml.sync` specifies the name of the schema where the database object exists. If `schema=None`, which is the default, then the current schema is used.

To create an `oml.DataFrame` object for a table, use the `table` parameter. To create one for a view, use the `view` parameter. To create one for a SQL `SELECT` statement, use the `query` parameter. You can only specify one of these parameters in an `oml.sync` invocation: the argument for one of the parameters must be a string and the argument for each of the other two parameters must be `None`.

Creating a proxy object for a query enables you to create an `oml.DataFrame` object without creating a view in the database. This can be useful when you do not have the `CREATE VIEW` system privilege for the current schema. You cannot use the `schema` parameter and the `query` parameter in the same `ore.sync` invocation.

With the `regex_match` argument, you can specify whether the value of the `table` or `view` argument is a regular expression. If `regex_match=True`, then `oml.sync` creates `oml.DataFrame` objects for each database object that matches the pattern. The matched tables or views are returned in a `dict` with the table or view names as keys.

Example 7-10 Creating a Python Object for a Database Table

This example creates an `oml.DataFrame` Python object as a proxy for a database table. For this example, the table `COFFEE` exists in the user's schema.

```
import oml

# Create the Python object oml_coffee as a proxy for the
# database table COFFEE.
oml_coffee = oml.sync(table = 'COFFEE')
type(oml_coffee)

# List the proxy objects in the schema.
oml.dir()

oml_coffee.head()
```

Listing for This Example

```
>>> import oml
>>>
>>> # Create the Python object oml_coffee as a proxy for the
... # database table COFFEE.
... oml_coffee = oml.sync(table = 'COFFEE')
>>> type(oml_coffee)
<class 'oml.core.frame.DataFrame'>
>>>
>>> # List the proxy objects in the schema.
... oml.dir()
['oml_coffee']
>>>
>>> oml_coffee.head()
   ID COFFEE WINDOW
0    1    esp      w
1    2    cap      d
2    3    cap      w
3    4    kon      w
4    5    ice      w
```

Example 7-11 Using the `regex_match` Argument

This example uses the `regex_match` argument in creating a `dict` object that contains `oml.DataFrame` proxy objects for tables whose names start with `C`. For this example, the

COFFEE and COLOR tables exist in the user's schema and are the only tables whose names start with C.

```
# Create a dict of oml.DataFrame proxy objects for tables
# whose names start with 'C'.
oml_cdat = oml.sync(table="^C", regex_match=True)

oml_cdat.keys()
oml_cdat['COFFEE'].columns
oml_cdat['COLOR'].columns
```

Listing for This Example

```
>>> # Create a dict of oml.DataFrame proxy objects for tables
... # whose names start with 'C'.
... oml_cdat = oml.sync(table="^C", regex_match=True)
>>>
>>> oml_cdat.keys()
dict_keys(['COFFEE', 'COLOR'])
>>> oml_cdat['COFFEE'].columns
['ID', 'COFFEE', 'WINDOW']
>>> oml_cdat['COLOR'].columns
['REGION', 'EYES', 'HAIR', 'COUNT']
```

Example 7-12 Synchronizing an Updated Table

This example uses `oml.sync` to create an `oml.DataFrame` for the database table COFFEE. For the example, the new column BREW has been added to the database table by some other database process after the first invocation of `oml.sync`. Invoking `oml.sync` again synchronizes the metadata of the `oml.DataFrame` with those of the table.

```
oml_coffee = oml.sync(table = "COFFEE")
oml_coffee.columns

# After a new column has been inserted into the table.
oml_coffee = oml.sync(table = "COFFEE")
oml_coffee.columns
```

Listing for This Example

```
>>> oml_coffee = oml.sync(table = "COFFEE")
>>> oml_coffee.columns
['ID', 'COFFEE', 'WINDOW']
>>>
>>> # After a new column has been inserted into the table.
... oml_coffee = oml.sync(table = "COFFEE")
>>> oml_coffee.columns
['ID', 'COFFEE', 'WINDOW', 'BREW']
```

7.3.5 Create a Persistent Database Table from a Python Data Set

Use the `oml.create` function to create a persistent table in your database schema from data in your Python session.

The `oml.create` function creates a table in the database schema and returns an `oml.DataFrame` object that is a proxy for the table. The proxy `oml.DataFrame` object has the same name as the table.

 **Note:**

When creating a table in Oracle Machine Learning for Python, if you use lowercase or mixed case for the name of the table, then you must use the same lowercase or mixed case name in double quotation marks when using the table in a SQL query or function. If, instead, you use an all uppercase name when creating the table, then the table name is case-insensitive: you can use uppercase, lowercase, or mixed case when using the table without using double quotation marks. The same is true for naming columns in a table.

You can delete the persistent table in a database schema with the `oml.drop` function.

 **Caution:**

Use the `oml.drop` function to delete a persistent database table. Use the `del` statement to remove an `oml.DataFrame` proxy object and its associated temporary table; `del` does not delete a persistent table.

The syntax of the `oml.create` function is the following:

```
oml.create(x, table, oranumber=True, dbtypes=None, append=False)
```

The `x` argument is a `pandas.DataFrame` or a list of tuples of equal size that contain the data for the table. For a list of tuples, each tuple represents a row in the table and the column names are set to `COL1`, `COL2`, and so on. The `table` argument is a string that specifies a name for the table.

The SQL data types of the columns are determined by the following:

- OML4Py determines default column types by looking at 20 random rows sampled from the table. For tables with less than 20 rows, it uses all rows in determining the column type. If the values in a column are all `None`, or if a column has inconsistent data types that are not `None` in the sampled rows, then a default column type cannot be determined and a `ValueError` is raised unless a SQL type for the column is specified by the `dbtypes` argument.
- For numeric columns, the `oranumber` argument, which is a `bool`, determines the SQL data type. If `True` (the default), then the SQL data type is `NUMBER`. If `False`, then the data type is `BINARY DOUBLE`.
If the data in `x` contains `NaN` values, then you should set `oranumber` to `False`.
- For string columns, the default type is `VARCHAR2(4000)`.
- For bytes columns, the default type is `BLOB`.

With the `dbtypes` parameter, you can specify the SQL data types for the table columns. The values of `dbtypes` may be either a `dict` that maps `str` to `str` values or a list of `str` values. For

a dict, the keys are the names of the columns. The `dbtypes` parameter is ignored if the `append` argument is `True`.

The `append` argument is a `bool` that specifies whether to append the `x` data to an existing table.

Example 7-13 Creating Database Tables from a Python Data Set

This example creates a `cursor` object for the database connection, creates a `pandas.core.frame.DataFrame` with columns of various data types, then creates a series of tables using different `oml.create` parameters and shows the SQL data types of the table columns.

```
import oml

# Create a cursor object for the current OML4Py database
# connection to run queries and get information from the database.
cr = oml.cursor()

import pandas as pd

df = pd.DataFrame({'numeric': [1, 1.4, -4, 3.145, 5, 2],
                  'string' : [None, None, 'a', 'a', 'a', 'b'],
                  'bytes' : [b'a', b'b', b'c', b'c', b'd', b'e]})

# Get the order of the columns
df.columns

# Create a table with the default parameters.
oml_df1 = oml.create(df, table = 'tbl1')

# Show the default SQL data types of the columns.
_ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl1'")
cr.fetchall()

# Create a table with oranumber set to False.
oml_df2 = oml.create(df, table = 'tbl2', oranumber = False)

# Show the SQL data typea of the columns.
_ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl2'")
cr.fetchall()

# Create a table with dbtypes specified as a dict mapping column names
# to SQL data types.
oml_df3 = oml.create(df, table = 'tbl3',
                    dbtypes = {'numeric': 'BINARY_DOUBLE',
                               'bytes': 'RAW(1)'})

# Show the SQL data types of the columns.
_ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl3'")
cr.fetchall()

# Create a table with dbtypes specified as a list of SQL data types
# matching the order of the columns.
oml_df4 = oml.create(df, table = 'tbl4',
```

```

dbtypes = ['BINARY_DOUBLE', 'VARCHAR2', 'RAW(1)']

# Show the SQL data type of the columns.
_ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl4'")
cr.fetchall()

# Create a table from a list of tuples.
lst = [(1, None, b'a'), (1.4, None, b'b'), (-4, 'a', b'c'),
       (3.145, 'a', b'c'), (5, 'a', b'd'), (None, 'b', b'e')]
oml_df5 = oml.create(lst, table = 'tbl5',
                    dbtypes = ['BINARY_DOUBLE', 'CHAR(1)', 'RAW(1)'])

# Close the cursor
cr.close()

# Drop the tables.
oml.drop('tbl1')
oml.drop('tbl2')
oml.drop('tbl3')
oml.drop('tbl4')
oml.drop('tbl5')

```

Listing for This Example

```

>>> import oml
>>>
>>> # Create a cursor object for the current OML4Py database
... # connection to run queries and get information from the database.
... cr = oml.cursor()
>>>
>>> import pandas as pd
>>>
>>> df = pd.DataFrame({'numeric': [1, 1.4, -4, 3.145, 5, 2],
...                    'string' : [None, None, 'a', 'a', 'a', 'b'],
...                    'bytes' : [b'a', b'b', b'c', b'c', b'd', b'e]})
>>>
>>> # Get the order of the columns.
... df.columns
Index(['numeric', 'string', 'bytes'], dtype='object')
>>>
>>> # Create a table with the default parameters.
... oml_df1 = oml.create(df, table = 'tbl1')
>>>
>>> # Show the default SQL data types of the columns.
... _ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl1'")
>>> cr.fetchall()
[('NUMBER',), ('VARCHAR2',), ('BLOB',)]
>>>
>>> # Create a table with oranumber set to False.
... oml_df2 = oml.create(df, table = 'tbl2', oranumber = False)
>>>
>>> # Show the SQL data types of the columns.
... _ = cr.execute("select data_type from all_tab_columns where table_name =

```

```
'tbl2'")
>>> cr.fetchall()
[('BINARY_DOUBLE',), ('VARCHAR2',), ('BLOB',)]
>>>
>>> # Create a table with dbtypes specified as a dict mapping column names
... # to SQL data types.
... oml_df3 = oml.create(df, table = 'tbl3',
...                       dbtypes = {'numeric': 'BINARY_DOUBLE',
...                                   'bytes': 'RAW(1)'})
>>>
>>> # Show the SQL data type of the columns.
... _ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl3'")
>>> cr.fetchall()
[('BINARY_DOUBLE',), ('VARCHAR2',), ('RAW',)]
>>>
>>> # Create a table with dbtypes specified as a list of SQL data types
... # matching the order of the columns.
... oml_df4 = oml.create(df, table = 'tbl4',
...                       dbtypes = ['BINARY_DOUBLE', 'CHAR(1)', 'RAW(1)'])
>>>
>>> # Show the SQL data type of the columns
... _ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl4'")
>>> cr.fetchall()
[('BINARY_DOUBLE',), ('CHAR',), ('RAW',)]
>>>
>>> # Create a table from a list of tuples.
... lst = [(1, None, b'a'), (1.4, None, b'b'), (-4, 'a', b'c'),
...         (3.145, 'a', b'c'), (5, 'a', b'd'), (None, 'b', b'e')]
>>> oml_df5 = oml.create(lst, table = 'tbl5',
...                       dbtypes = ['BINARY_DOUBLE', 'CHAR(1)', 'RAW(1)'])
>>>
>>> # Show the SQL data type of the columns.
... _ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl5'")
>>> cr.fetchall()
[('BINARY_DOUBLE',), ('CHAR',), ('RAW',)]
>>>
>>> # Close the cursor.
... cr.close()
>>>
>>> # Drop the tables
... oml.drop('tbl1')
>>> oml.drop('tbl2')
>>> oml.drop('tbl3')
>>> oml.drop('tbl4')
>>> oml.drop('tbl5')
```

7.4 Save Python Objects in the Database

You can save Python objects in OML4Py datastores, which persist in the database.

You can grant or revoke read privilege access to a datastore or its objects to one or more users. You can restore the saved objects in another Python session.

The following topics describe the OML4Py functions for creating and managing datastores:

- [About OML4Py Datastores](#)
- [Save Objects to a Datastore](#)
- [Load Saved Objects From a Datastore](#)
- [Get Information About Datastores](#)
- [Get Information About Datastore Objects](#)
- [Delete Datastore Objects](#)
- [Manage Access to Stored Objects](#)
- [About OML4Py Datastores](#)

In an OML4Py datastore, you can store Python objects, which you can then use in subsequent Python sessions; you can also make them available to other users or programs.
- [Save Objects to a Datastore](#)

The `oml.ds.save` function saves one or more Python objects to a datastore.
- [Load Saved Objects From a Datastore](#)

The `oml.ds.load` function loads one or more Python objects from a datastore into a Python session.
- [Get Information About Datastores](#)

The `oml.ds.dir` function provides information about datastores.
- [Get Information About Datastore Objects](#)

The `oml.ds.describe` function provides information about the objects in a datastore.
- [Delete Datastore Objects](#)

The `oml.ds.delete` function deletes datastores or objects in a datastore.
- [Manage Access to Stored Objects](#)

The `oml.grant` and `oml.revoke` functions grant or revoke the read privilege to datastores or to user-defined Python functions in the script repository.

7.4.1 About OML4Py Datastores

In an OML4Py datastore, you can store Python objects, which you can then use in subsequent Python sessions; you can also make them available to other users or programs.

Python objects, including OML4Py proxy objects, exist only for the duration of the current Python session unless you explicitly save them. You can save a Python object, including `oml` proxy objects, to a named datastore and then load that object in a later Python session, including an Embedded Python Execution session. OML4Py creates the datastore in the user's database schema. A datastore, and the objects it contains, persist in the database until you delete them.

You can grant or revoke read privilege permission to another user to a datastore that you created or to objects in a datastore.

OML4Py has Python functions for managing objects in a datastore. It also has PL/SQL procedures for granting or revoking the read privilege and database views for listing available datastores and their contents.

Using a datastore, you can do the following:

- Save OML4Py and other Python objects that you create in one Python session and load them in another Python session.

- Pass arguments to Python functions for use in Embedded Python Execution.
- Pass objects for use in Embedded Python Execution. You could, for example, use the `oml.glm` class to build an Oracle Machine Learning model and save it in a datastore. You could then use that model to score data in the database through Embedded Python Execution.

Python Interface for Datastores

The following table lists the Python functions for saving and managing objects in a datastore.

Function	Description
<code>oml.ds.delete</code>	Deletes one or more datastores or Python objects from a datastore.
<code>oml.ds.dir</code>	Lists the datastores available to the current user.
<code>oml.ds.load</code>	Loads Python objects from a datastore into the user's session.
<code>oml.ds.save</code>	Saves Python objects to a named datastore in the user's database schema.

The following table lists the Python functions for managing access to datastores and datastore objects.

Function	Description
<code>oml.grant</code>	Grants read privilege permission to another user to a datastore or a user-defined Python function in the script repository owned by the current user.
<code>oml.revoke</code>	Revokes the read privilege permission that was granted to another user to a datastore or a user-defined Python function in the script repository owned by the current user.

7.4.2 Save Objects to a Datastore

The `oml.ds.save` function saves one or more Python objects to a datastore.

OML4Py creates the datastore in the current user's schema.

The syntax of `oml.ds.save` is the following:

```
oml.ds.save(objs, name, description=' ', grantable=None,
            overwrite=False, append=False, compression=False)
```

The `objs` argument is a dict that contains the name and object pairs to save to the datastore specified by the `name` argument.

With the `description` argument, you can provide some descriptive text that appears when you get information about the datastore. The `description` parameter has no effect when used with the `append` parameter.

With the `grantable` argument, you can specify whether the read privilege to the datastore may be granted to other users.

If you set the `overwrite` argument to `TRUE`, then you can replace an existing datastore with another datastore of the same name.

If you set the `append` argument to `TRUE`, then you can add objects to an existing datastore. The `overwrite` and `append` arguments are mutually exclusive.

If you set `compression` to `True`, then the serialized Python objects are compressed in the datastore.

Example 7-14 Saving Python Objects to a Datastore

This example demonstrates creating datastores.

```
import oml
from sklearn import datasets
from sklearn import linear_model
import pandas as pd

# Load three data sets and create oml.DataFrame objects for them.
wine = datasets.load_wine()
x = pd.DataFrame(wine.data, columns = wine.feature_names)
y = pd.DataFrame(wine.target, columns = ['Class'])

# Create the database table WINE.
oml_wine = oml.create(pd.concat([x, y], axis=1), table = 'WINE')
oml_wine.columns

diabetes = datasets.load_diabetes()
x = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
y = pd.DataFrame(diabetes.target, columns=['disease_progression'])
oml_diabetes = oml.create(pd.concat([x, y], axis=1),
                          table = "DIABETES")
oml_diabetes.columns

boston = datasets.load_boston()
x = pd.DataFrame(boston.data, columns = boston.feature_names.tolist())
y = pd.DataFrame(boston.target, columns = ['Value'])
oml_boston = oml.create(pd.concat([x, y], axis=1), table = "BOSTON")
oml_boston.columns

# Save the wine Bunch object to the datastore directly,
# along with the oml.DataFrame proxy object for the BOSTON table.
oml.ds.save(objs={'wine':wine, 'oml_boston':oml_boston},
            name="ds_pydata", description = "python datasets")

# Save the oml_diabetes proxy object to an existing datastore.
oml.ds.save(objs={'oml_diabetes':oml_diabetes},
            name="ds_pydata", append=True)

# Save the oml_wine proxy object to another datastore.
oml.ds.save(objs={'oml_wine':oml_wine},
            name="ds_wine_data", description = "wine dataset")

# Create regression models using sklearn and oml.
# The regr1 linear model is a native Python object.
regr1 = linear_model.LinearRegression()
regr1.fit(boston.data, boston.target)
# The regr2 GLM model is an oml object.
regr2 = oml.glm("regression")
X = oml_boston.drop('Value')
y = oml_boston['Value']
regr2 = regr2.fit(X, y)
```

```
# Save the native Python object and the oml proxy object to a datastore
# and allow the read privilege to be granted to them.
oml.ds.save(objs={'regr1':regr1, 'regr2':regr2},
            name="ds_pymodel", grantable=True)

# Grant the read privilege to the datastore to every user.
oml.grant(name="ds_pymodel", typ="datastore", user=None)

# List the datastores to which the read privilege has been granted.
oml.ds.dir(dstype="grant")
```

Listing for This Example

```
>>> import oml
>>> from sklearn import datasets
>>> from sklearn import linear_model
>>> import pandas as pd
>>>
>>> # Load three data sets and create oml.DataFrame objects for them.
>>> wine = datasets.load_wine()
>>> x = pd.DataFrame(wine.data, columns = wine.feature_names)
>>> y = pd.DataFrame(wine.target, columns = ['Class'])
>>>
>>> # Create the database table WINE.
... oml_wine = oml.create(pd.concat([x, y], axis=1), table = 'WINE')
>>> oml_wine.columns
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium',
'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins',
'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline', 'Class']
>>>
>>> diabetes = datasets.load_diabetes()
>>> x = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
>>> y = pd.DataFrame(diabetes.target, columns=['disease_progression'])
>>> oml_diabetes = oml.create(pd.concat([x, y], axis=1),
...                          table = "DIABETES")
>>> oml_diabetes.columns
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6',
'disease_progression']
>>>
>>> boston = datasets.load_boston()
>>> x = pd.DataFrame(boston.data, columns = boston.feature_names.tolist())
>>> y = pd.DataFrame(boston.target, columns = ['Value'])
>>> oml_boston = oml.create(pd.concat([x, y], axis=1), table = "BOSTON")
>>> oml_boston.columns
['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
'PTRATIO', 'B', 'LSTAT', 'Value']
>>>
>>> # Save the wine Bunch object to the datastore directly,
... # along with the oml.DataFrame proxy object for the BOSTON table.
... oml.ds.save(objs={'wine':wine, 'oml_boston':oml_boston},
...             name="ds_pydata", description = "python datasets")
>>>
>>> # Save the oml_diabetes proxy object to an existing
datastore.
```

```
... oml.ds.save(objs={'oml_diabetes':oml_diabetes},
...               name="ds_pydata", append=True)
>>>
>>> # Save the oml_wine proxy object to another datastore.
... oml.ds.save(objs={'oml_wine':oml_wine},
...               name="ds_wine_data", description = "wine dataset")
>>>
>>> # Create regression models using sklearn and oml.
... # The regr1 linear model is a native Python object.
... regr1 = linear_model.LinearRegression()
>>> regr1.fit(boston.data, boston.target)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
>>> # The regr2 GLM model is an oml proxy object.
... regr2 = oml.glm("regression")
>>> X = oml_boston.drop('Value')
>>> y = oml_boston['Value']
>>> regr2 = regr2.fit(X, y)
>>>
>>> # Save the native Python object and the oml proxy object to a datastore
... # and allow the read privilege to be granted to them.
... oml.ds.save(objs={'regr1':regr1, 'regr2':regr2},
...               name="ds_pymodel", grantable=True)
>>>
>>> # Grant the read privilege to the ds_pymodel datastore to every user.
... oml.grant(name="ds_pymodel", typ="datastore", user=None)
>>>
>>> # List the datastores to which the read privilege has been granted.
... oml.ds.dir(dstype="grant")
    datastore_name grantee
0      ds_pymodel PUBLIC
```

7.4.3 Load Saved Objects From a Datastore

The `oml.ds.load` function loads one or more Python objects from a datastore into a Python session.

The syntax of `oml.ds.load` is the following:

```
oml.ds.load(name, objs=None, owner=None, to_globals=True)
```

The `name` argument specifies the datastore that contains the objects to load.

With the `objs` argument, you identify a specific object or a list of objects to load.

With the boolean `to_globals` parameter, you can specify whether the objects are loaded to a global workspace or to a dictionary object. If the argument to `to_globals` is `True`, then `oml.ds.load` function loads the objects into the global workspace. If the argument is `False`, then the function returns a `dict` object that contains pairs of object names and values.

The `oml.ds.load` function raises a `ValueError` if the `name` argument is an empty string or if the owner of the datastore is not the current user and the read privilege for the datastore has not been granted to the current user.

Example 7-15 Loading Objects from Datastores

This example loads objects from datastores. For the creation of the datastores used in this example, see [Example 7-14](#).

```
import oml

# Load all Python objects from a datastore to the global workspace.
sorted(oml.ds.load(name="ds_pydata"))

# Load the named Python object from the datastore to the global workspace.
oml.ds.load(name="ds_pymodel", objs=["regr2"])

# Load the named Python object from the datastore to the user's workspace.
oml.ds.load(name="ds_pymodel", objs=["regr1"], to_globals=False)
```

Listing for This Example

```
>>> import oml
>>>
>>> # Load all Python objects from a datastore to the current workspace.
... sorted(oml.ds.load(name="ds_pydata"))
['oml_boston', 'oml_diabetes', 'wine']
>>>
>>> # Load the named Python object from the datastore to the global workspace.
... oml.ds.load(name="ds_pymodel", objs=["regr2"])
['regr2']
>>>
>>> # Load the named Python object from the datastore to the user's workspace.
... oml.ds.load(name="ds_pymodel", objs=["regr1"], to_globals=False)
{'regr1': LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1,
normalize=False)}
```

7.4.4 Get Information About Datastores

The `oml.ds.dir` function provides information about datastores.

The syntax of `oml.ds.dir` is the following:

```
oml.ds.dir(name=None, regex_match=False, dstype='user')
```

Use the `name` parameter to get information about a specific datastore.

Optionally, you can use the `regex_match` and `dstype` parameters to get information about datastores with certain characteristics. The valid arguments for `dstype` are the following:

Argument	Description
<code>all</code>	Lists all of the datastores to which the current user has the read privilege.
<code>grant</code>	Lists the datastores for which the current user has granted read privilege to other users.
<code>granted</code>	Lists the datastores for which other users have granted read privilege to the current user.

Argument	Description
grantable	Lists the datastores that the current user can grant the read privilege to.
user	Lists the datastores created by current user.
private	Lists the datastores that the current user cannot grant the read privileges to.

The `oml.ds.dir` function returns a `pandas.DataFrame` object that contains different columns depending on which `dstype` argument you use. The following table lists the arguments and the columns returned for the values supplied.

dstype Argument	Columns in the DataFrame Returned
user	DSNAME, which contains the datastore name
private	NOBJ, which contains the number of objects in the datastore
grantable	DSIZE, which contains the size in bytes of each object in the datastore CDATE, which contains the creation date of the datastore DESCRIPTION, which contains the optional description of the datastore
all	All of the columns returned by the <code>user</code> , <code>private</code> , and <code>grantable</code> values, plus this additional column:
granted	DSOWNER, which contains the owner of the datastore
grant	DSNAME, which contains the datastore name GRANTEE, which contains the name of the user to which the read privilege to the datastore has been granted by the current session user

Example 7-16 Getting Information About Datastores

This example demonstrates using different combinations of arguments to the `oml.ds.dir` function. It demonstrates using `oml.dir` to list some or all of the datastores. For the creation of the datastores used in this example, see [Example 7-14](#).

```
import oml

# Show all saved datastores.
oml.ds.dir(dstype="all")[['owner', 'datastore_name', 'object_count']]

# Show datastores to which other users have been granted the read
# privilege.
oml.ds.dir(dstype="grant")

# Show datastores whose names match a pattern.
oml.ds.dir(name='pydata', regex_match=True)\
    [['datastore_name', 'object_count']]
```

Listing for This Example

```
>>> import oml
>>>
>>> # Show all saved datastores.
```

```

... oml.ds.dir(dstype="all")[['owner', 'datastore_name', 'object_count']]
   owner  datastore_name  object_count
0  OML_USER      ds_pydata           3
1  OML_USER      ds_pymodel          2
2  OML_USER      ds_wine_data         1
>>>
>>> # Show datastores to which other users have been granted the read
>>> # privilege.
... oml.ds.dir(dstype="grant")
   datastore_name  grantee
0      ds_pymodel  PUBLIC
>>>
>>> oml.ds.dir(name='pydata', regex_match=True)\
...      [['datastore_name', 'object_count']]
   datastore_name  object_count
0      ds_pydata           3

```

7.4.5 Get Information About Datastore Objects

The `oml.ds.describe` function provides information about the objects in a datastore.

The syntax of `oml.ds.describe` is the following:

```
oml.ds.describe(name, owner=None))
```

The `name` argument is a string that specifies the name of a datastore.

The `owner` argument is a string that specifies the owner of the datastore or `None` (the default). If you do not specify the owner, then the function returns information about the datastore if it is owned by the current user.

The `oml.ds.describe` function returns a `pandas.DataFrame` object, each row of which represents an object in the datastore. The columns of the `DataFrame` are the following:

- `object_name`, which specifies the name of the object
- `class`, which specifies the class of the object
- `size`, which specifies the size of the object in bytes
- `length`, which specifies the length of the object
- `row_count`, which specifies the rows of the object
- `col_count`, which specifies the columns of the object

This function raises a `ValueError` if the following occur:

- The current user is not the owner of the datastore and has not been granted read privilege for the datastore.
- The datastore does not exist.

Example 7-17 Getting Information About Datastore Objects

This example demonstrates the using the `oml.ds.describe` function. For the creation of the datastore used in this example, see [Example 7-14](#).

```
import oml
```

```
# Describe the contents of the ds_pydata datastore.
oml.ds.describe(name='ds_pydata')
oml.ds.describe(name="ds_pydata") [['object_name', 'class']]
```

Listing for This Example

```
>>> import oml
>>>
>>> # Describe the contents of the ds_pydata datastore.
... oml.ds.describe(name='ds_pydata')
  object_name      class  size  length  row_count  col_count
0  oml_boston  oml.DataFrame  1073    506      506      14
1  oml_diabetes oml.DataFrame   964    442      442      11
2      wine      Bunch  24177     5        1        5
>>> oml.ds.describe(name="ds_pydata") [['object_name', 'class']]
  object_name      class
0  oml_boston  oml.DataFrame
1  oml_diabetes oml.DataFrame
2      wine      Bunch
```

7.4.6 Delete Datastore Objects

The `oml.ds.delete` function deletes datastores or objects in a datastore.

Use the `oml.ds.delete` function to delete one or more datastores in your database schema or to delete objects in a datastore.

The syntax of `oml.ds.delete` is the following:

```
oml.ds.delete(name, objs=None, regex_match=False)
```

The argument to the `name` parameter may be one of the following:

- A string that specifies the name of the datastore to modify or delete, or a regular expression that matches the datastores to delete.
- A list of `str` objects that name the datastores from which to delete objects.

The `objs` parameter specifies the objects to delete from a datastore. The argument to the `objs` parameter may be one of the following:

- A string that specifies the object to delete from one or more datastores, or a regular expression that matches the objects to delete.
- `None` (the default), which deletes the entire datastore or datastores.

The `regex_match` parameter is a `bool` that indicates whether the `name` or `objs` arguments are regular expressions. The default value is `False`. The `regex_match` parameter operates as follows:

- If `regex_match=False` and if `name` is not `None`, and:
 - If `objs=None`, then `oml.ds.delete` deletes the datastore or datastores specified in the `name` argument.
 - If you specify one or more datastores with the `name` argument and one or more datastore objects with the `objs` argument, then `oml.ds.delete` deletes the specified Python objects from the datastores.

- If `regex_match=True` and:
 - If `objs=None`, then `oml.ds.delete` deletes the datastores you specified in the `name` argument.
 - If the `name` argument is a string and you specify one or more datastore objects with the `objs` argument, then `oml.ds.delete` deletes from the datastore the objects whose names match the regular expression specified in the `objs` argument.
 - If the `name` argument is a list of `str` objects, then the `objs` argument must be a list of `str` objects of the same length as `name`, and `oml.ds.delete` deletes from the datastores the objects whose names match the regular expressions specified in `objs`.

This function raises an error if the following occur:

- A specified datastore does not exist.
- Argument `regex_match` is `False` and argument `name` is a list of `str` objects larger than 1 and argument `objs` is not `None`.
- Argument `regex_match` is `True` and arguments `name` and `objs` are lists that are not the same length.

Example 7-18 Deleting Datastore Objects

This example demonstrates the using the `oml.ds.delete` function. For the creation of the datastores used in this example, see [Example 7-14](#).

```
import oml

# Show the existing datastores.
oml.ds.dir()

# Show the Python objects in the ds_pydata datastore.
oml.ds.describe(name='ds_pydata')

# Delete some objects from the datastore.
oml.ds.delete(name="ds_pydata", objs=["wine", "oml_boston"])

# Delete a datastore.
oml.ds.delete(name="ds_pydata")

# Delete all datastores whose names match a pattern.
oml.ds.delete(name="_pymodel", regex_match=True)

# Show the existing datastores again.
oml.ds.dir()
```

Listing for This Example

```
>>> import oml
>>>
>>> # Show the existing datastores.
... oml.ds.dir()
  datastore_name  object_count  size          date          description
0      ds_pydata           3  26214 2019-05-18 21:04:06  python datasets
1      ds_pymodel          2   6370 2019-05-18 21:08:18           None
2      ds_wine_data         1   1410 2019-05-18 21:06:53    wine dataset
>>>
```



```

>>> # Show the Python objects in the ds_pydata datastore.
... oml.ds.describe(name='ds_pydata')
   object_name      class  size  length  row_count  col_count
0  oml_boston  oml.DataFrame  1073    506         506         14
1  oml_diabetes oml.DataFrame   964    442         442         11
2      wine      Bunch  24177     5           1           5
>>>
>>> # Delete some objects from a datastore.
... oml.ds.delete(name="ds_pydata", objs=["wine", "oml_boston"])
{'wine', 'oml_boston'}
>>>
>>> # Delete a datastore.
... oml.ds.delete(name="ds_pydata")
'ds_pydata'
>>>
>>> # Delete all datastores whose names match a pattern.
... oml.ds.delete(name="_pymodel", regex_match=True)
{'ds_pymodel'}
>>>
>>> # Show the existing datastores again.
... oml.ds.dir()
   datastore_name  object_count  size      date  description
0  ds_wine_data           1  1410  2019-05-18  21:06:53  wine dataset

```

7.4.7 Manage Access to Stored Objects

The `oml.grant` and `oml.revoke` functions grant or revoke the read privilege to datastores or to user-defined Python functions in the script repository.

The `oml.grant` function grants the read privilege to another user to a datastore or to a user-defined Python function in the OML4Py script repository. The `oml.revoke` function revokes that privilege.

The syntax of these functions is the following:

```

oml.grant(name, typ='datastore', user=None)
oml.revoke(name, typ='datastore', user=None)

```

The `name` argument is a string that specifies the name of the user-defined Python function in the script repository or the name of a datastore.

The `typ` parameter must be specified. The argument is a string that is either `'datastore'` or `'pyqscript'`.

The `user` argument is a string that specifies the user to whom read privilege to the named datastore or user-defined Python function is granted or from whom it is revoked, or `None` (the default). If you specify `None`, then the read privilege is granted to or revoked from all users.

Example 7-19 Granting and Revoking Access to Datastores

This example displays the datastores to which the read privilege has been granted to all users. It revokes read privilege from the `ds_pymodel` datastore and displays the datastores with public read privilege again. It next grants the read privilege to the user SH and finally displays once

more the datastores to which read privilege has been granted. For the creation of the datastores used in this example, see [Example 7-14](#).

```
import oml

# Show datastores to which other users have been granted read privilege.
oml.ds.dir(dstype="grant")

# Revoke the read privilege from every user.
oml.revoke(name="ds_pymodel", typ="datastore", user=None)

# Again show datastores to which read privilege has been granted.
oml.ds.dir(dstype="grant")

# Grant the read privilege to the user SH.
oml.grant(name="ds_pymodel", typ="datastore", user="SH")

oml.ds.dir(dstype="grant")
```

Listing for This Example

```
>>> import oml
>>>
>>> # Show datastores to which other users have been granted read privilege.
... oml.ds.dir(dstype="grant")
   datastore_name grantee
0      ds_pymodel  PUBLIC
>>>
>>> # Revoke the read privilege from every user.
... oml.revoke(name="ds_pymodel", typ="datastore", user=None)
>>>
>>> # Again show datastores to which read privilege has been granted to other
users.
... oml.ds.dir(dstype="grant")
Empty DataFrame
Columns: [datastore_name, grantee]
Index: []
>>>
>>> # Grant the read privilege to the user SH.
... oml.grant(name="ds_pymodel", typ="datastore", user="SH")
>>>
>>> oml.ds.dir(dstype="grant")
   datastore_name grantee
0      ds_pymodel      SH
```

Example 7-20 Granting and Revoking Access to User-Defined Python Functions

This example grants the read privilege to the MYLM user-defined Python function to the user SH and then revokes that privilege. For the creation of the user-defined Python functions used in this example, see [Example 12-11](#).

```
# List the user-defined Python functions available only to the current user.
oml.script.dir(sctype='user')

# Grant the read privilege to the MYLM user-defined Python function to the
```

```
user SH.
oml.grant(name="MYLM", typ="pyqscript", user="SH")

# List the user-defined Python functions to which read privilege has been
granted.
oml.script.dir(sctype="grant")

# Revoke the read privilege to the MYLM user-defined Python function from the
user SH.
oml.revoke(name="MYLM", typ="pyqscript", user="SH")

# List the granted user-defined Python functions again to see if the
revocation was successful.
oml.script.dir(sctype="grant")
```

Listing for This Example

```
>>> # List the user-defined Python functions available only to the current
user.
oml.script.dir(sctype='user')
      name                script
0 MYLM def build_lm1(dat):\n from sklearn import lin...
>>>
>>># Grant the read privilege to the MYLM user-defined Python function to the
user SH.
...oml.grant(name="MYLM", typ="pyqscript", user="SH")
>>>
>>> # List the user-defined Python functions to which read privilege has been
granted.
... oml.script.dir(sctype="grant")
      name grantee
0 MYLM      SH
>>>
>>> # Revoke the read privilege to the MYLM user-defined Python function from
the user SH.
... oml.revoke(name="MYLM", typ="pyqscript", user="SH")
>>>
>>> # List the granted user-defined Python functions again to see if the
revocation was successful.
... oml.script.dir(sctype="grant")
Empty DataFrame
Columns: [name, grantee]
Index: []
```

8

Prepare and Explore Data

Use OML4Py methods to prepare data for analysis and to perform exploratory analysis of the data.

Methods of the OML4Py data type classes make it easier for you to prepare very large enterprise database-resident data for modeling. These methods are described in the following topics.

- [Prepare Data](#)
Using methods of OML4Py data type classes, you can prepare data for analysis in the database, as described in the following topics.
- [Explore Data](#)
OML4Py provides methods that enable you to perform exploratory data analysis and common statistical operations.
- [Render Graphics](#)
OML4Py provides functions for rendering graphical displays of data.

8.1 Prepare Data

Using methods of OML4Py data type classes, you can prepare data for analysis in the database, as described in the following topics.

- [About Preparing Data in the Database](#)
OML4Py data type classes have methods that enable you to use Python to prepare database data for analysis.
- [Select Data](#)
A typical step in preparing data for analysis is selecting or filtering values of interest from a larger data set.
- [Combine Data](#)
You can join data from `oml.DataFrame` objects that represent database tables by using the `append`, `concat`, and `merge` methods.
- [Clean Data](#)
In preparing data for analysis, a typical step is to transform data by dropping some values.
- [Split Data](#)
Sample and randomly partition data with the `split` and `KFold` methods.

8.1.1 About Preparing Data in the Database

OML4Py data type classes have methods that enable you to use Python to prepare database data for analysis.

You can perform data preparation operations on large quantities of data in the database and then continue operating on that data in-database or pull a subset of the results to your local Python session where, for example, you can use third-party Python packages to perform other operations.

The following table lists methods with which you can perform common data preparation tasks and indicates whether the OML4Py data type class supports the method.

Table 8-1 Methods Supported by Data Types

Method	Description	oml.Boolean	oml.Bytes	oml.Float	oml.String	oml.DataFrame	oml.Date	oml.Time	oml.Timestamp	oml.Integer
append	Appends another oml data object of the same class to an oml object.	✓	✓	✓	✓	✓	✓	✓	✓	✓
ceil	Computes the ceiling of each element in an oml.Float series data object.	✗	✗	✓	✗	✗	✗	✗	✗	✗
concat	Combines an oml data object column-wise with one or more other data objects.	✓	✓	✓	✓	✓	✓	✓	✓	✓
count_pattern	Counts the number of occurrences of a pattern in each string.	✗	✗	✗	✓	✗	✗	✗	✗	✗
create_view	Creates an Oracle Database view for the data represented by the OML4Py data object.	✗	✗	✗	✗	✓	✓	✓	✓	✓
dot	Calculates the inner product of the current oml.Float object with another oml.Float, or does matrix multiplication with an oml.DataFrame.	✗	✗	✓	✗	✗	✗	✗	✗	✓
drop	Drops specified columns in an oml.DataFrame.	✗	✗	✗	✗	✓	✗	✗	✗	✗
drop_duplicates	Removes duplicated elements from an oml series data object or duplicated rows from an oml.DataFrame.	✓	✓	✓	✓	✓	✓	✓	✓	✓
dropna	Removes missing elements from an oml series data object, or rows containing missing values from an oml.DataFrame.	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 8-1 (Cont.) Methods Supported by Data Types

Method	Description	oml.Boolean	oml.Bytes	oml.Float	oml.String	oml.DataFrame	oml.Date	oml.Timezone	oml.Timestamp	oml.Integer
exp	Computes element-wise e to the power of values in an <code>oml.Float</code> series data object.	✗	✗	✓	✗	✗	✗	✗	✗	✓
find	Finds the lowest index in each string in which a substring is found that is greater than or equal to a start index.	✗	✗	✗	✓	✗	✗	✗	✗	✗
floor	Computes the floor of each element in an <code>oml.Float</code> series data object.	✗	✗	✓	✗	✗	✗	✗	✗	✗
head	Returns the first n elements of an <code>oml</code> series data object or the first n rows of an <code>oml.DataFrame</code> .	✓	✓	✓	✓	✓	✓	✓	✓	✓
KFold	Splits the <code>oml</code> data object randomly into k consecutive folds.	✓	✓	✓	✓	✓	✓	✓	✓	✓
len	Computes the length of each string in an <code>oml.Bytes</code> or <code>oml.String</code> series data object.	✗	✓	✗	✓	✗	✗	✗	✗	✗
log	Calculates an element-wise logarithm, to the given base, of values in the <code>oml.Float</code> series data object.	✗	✗	✓	✗	✗	✗	✗	✗	✓
materialize	Pushes the contents represented by an OML4Py proxy object (a view, a table, and so on) into a table in Oracle Database.	✗	✗	✗	✗	✓	✗	✗	✗	✗
merge	Joins another <code>oml.DataFrame</code> to an <code>oml.DataFrame</code> .	✗	✗	✗	✗	✓	✗	✗	✗	✗
replace	Replaces an existing value with another value.	✗	✗	✓	✓	✓	✓	✗	✗	✓

Table 8-1 (Cont.) Methods Supported by Data Types

Method	Description	oml.Boolean	oml.Bytes	oml.Float	oml.String	oml.DataFrame	oml.Date	oml.Timezone	oml.Timestamp	oml.Integer
rename	Renames columns of an <code>oml.DataFrame</code> .	✗	✗	✗	✗	✓	✗	✗	✗	✗
round	Rounds <code>oml.Float</code> values to the specified decimal place.	✗	✗	✓	✗	✗	✗	✗	✗	✗
select_types	Returns the subset of columns that are included or excluded based on their <code>oml</code> data type.	✗	✗	✗	✗	✓	✗	✗	✗	✗
split	Splits an <code>oml</code> data object randomly into multiple sets.	✓	✓	✓	✓	✓	✓	✓	✓	✓
sqrt	Computes the square root of each element in an <code>oml.Float series</code> data object.	✗	✗	✓	✗	✗	✗	✗	✗	✓
tail	Returns the last n elements of an <code>oml series</code> data object or the last n rows of an <code>oml.DataFrame</code> .	✓	✓	✓	✓	✓	✓	✓	✓	✓

8.1.2 Select Data

A typical step in preparing data for analysis is selecting or filtering values of interest from a larger data set.

The examples in this section demonstrate selecting data from an `oml.DataFrame` object by rows, by columns, and by value.

The examples use the `oml_iris` object created by the following code, which imports the `sklearn.datasets` package and loads the `iris` data set. It creates the `x` and `y` variables, and then creates the persistent database table `IRIS` and the `oml.DataFrame` object `oml.iris` as a proxy for the table.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width',
                                     'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor',
                                   2: 'virginica'}[x], iris.target)),
```

```
columns = ['Species'])

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

The examples are in the following topics:

- [Select the First or Last Number of Rows](#)
- [Select Data by Column](#)
- [Select Data by Value](#)

Select the First or Last Number of Rows

The `head` and `tail` methods return the first or last number of elements.

The default number of rows selected is 5.

Example 8-1 Selecting the First and Last Number of Rows

This example selects rows from the `oml.DataFrame` object `oml_iris`. It displays the first five rows and ten rows of `oml_iris` and then the last five and ten rows.

```
# Display the first 5 rows.
oml_iris.head()

# Display the first 10 rows.
oml_iris.head(10)

# Display the last 5 rows.
oml_iris.tail()

# Display the last 10 rows.
oml_iris.tail(10)
```

Listing for This Example

```
>>> # Display the first 5 rows.
... oml_iris.head()
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width  Species
0             5.1           3.5           1.4           0.2  setosa
1             4.9           3.0           1.4           0.2  setosa
2             4.7           3.2           1.3           0.2  setosa
3             4.6           3.1           1.5           0.2  setosa
4             5.0           3.6           1.4           0.2  setosa
>>>
>>> # Display the first 10 rows.
... oml_iris.head(10)
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width  Species
0             5.1           3.5           1.4           0.2  setosa
1             4.9           3.0           1.4           0.2  setosa
2             4.7           3.2           1.3           0.2  setosa
3             4.6           3.1           1.5           0.2  setosa
4             5.0           3.6           1.4           0.2  setosa
5             5.4           3.9           1.7           0.4  setosa
6             4.6           3.4           1.4           0.3  setosa
```



```

7          5.0          3.4          1.5          0.2 setosa
8          4.4          2.9          1.4          0.2 setosa
9          4.9          3.1          1.5          0.1 setosa
>>>
>>> # Display the last 5 rows.
... oml_iris.tail()
   Sepal_Length Sepal_Width Petal_Length Petal_Width Species
0            6.7          3.0          5.2          2.3 virginica
1            6.3          2.5          5.0          1.9 virginica
2            6.5          3.0          5.2          2.0 virginica
3            6.2          3.4          5.4          2.3 virginica
4            5.9          3.0          5.1          1.8 virginica

>>>
>>> # Display the last 10 rows.
... oml_iris.tail(10)
   Sepal_Length Sepal_Width Petal_Length Petal_Width Species
0            6.7          3.1          5.6          2.4 virginica
1            6.9          3.1          5.1          2.3 virginica
2            5.8          2.7          5.1          1.9 virginica
3            6.8          3.2          5.9          2.3 virginica
4            6.7          3.3          5.7          2.5 virginica
5            6.7          3.0          5.2          2.3 virginica
6            6.3          2.5          5.0          1.9 virginica
7            6.5          3.0          5.2          2.0 virginica
8            6.2          3.4          5.4          2.3 virginica
9            5.9          3.0          5.1          1.8 virginica

```

Select Data by Column

Example 8-2 Selecting Data by Columns

The example selects two columns from `oml_iris` and creates the `oml.DataFrame` object `iris_projected1` with them. It then displays the first three rows of `iris_projected1`. The example also selects a range of columns from `oml_iris`, creates `iris_projected2`, and displays its first three rows. Finally, the example selects columns from `oml_iris` by data types, creates `iris_projected3`, and displays its first three rows.

```

# Select all rows with the specified column names.
iris_projected1 = oml_iris[:, ["Sepal_Length", "Petal_Length"]]
iris_projected1.head(3)

# Select all rows with columns whose indices are in the range [1, 4).
iris_projected2 = oml_iris[:, 1:4]
iris_projected2.head(3)

# Select all rows with columns of oml.String data type.
iris_projected3 = oml_iris.select_types(include=[oml.String])
iris_projected3.head(3)

```

Listing for This Example

```

>>> # Select all rows with specified column names.
... iris_projected1 = oml_iris[:, ["Sepal_Length", "Petal_Length"]]
>>> iris_projected1.head(3)

```

```

      Sepal_Length  Petal_Length
0           5.1           1.4
1           4.9           1.4
2           4.7           1.3
>>>
>>> # Select all rows with columns whose indices are in range [1, 4).
... iris_projected2 = oml_iris[:, 1:4]
>>> iris_projected2.head(3)
      Sepal_Width  Petal_Length  Petal_Width
0           3.5           1.4           0.2
1           3.0           1.4           0.2
2           3.2           1.3           0.2
>>>
>>> # Select all rows with columns of oml.String data type.
... iris_projected3 = oml_iris.select_types(include=[oml.String])
>>> iris_projected3.head(3)
      Species
0  setosa
1  setosa
2  setosa

```

Select Data by Value

Example 8-3 Selecting Data by Value

This example filters `oml_iris` to produce `iris_of_filtered1`, which contains the values from the rows of `oml_iris` that have a petal length of less than 1.5 and that are in the `Sepal_Length` and `Petal_Length` columns. The example also filters the data using conditions, so that `oml_iris_filtered2` contains the values from `oml_iris` that have a petal length of less than 1.5 or a sepal length equal to 5.0 and `oml_iris_filtered3` contains the values from `oml_iris` that have a petal length of less than 1.5 and a sepal length larger than 5.0.

```

# Select sepal length and petal length where petal length
# is less than 1.5.
oml_iris_filtered1 = oml_iris[oml_iris["Petal_Length"] < 1.5,
                             ["Sepal_Length", "Petal_Length"]]

len(oml_iris_filtered1)
oml_iris_filtered1.head(3)

### Using the AND and OR conditions in filtering.
# Select all rows in which petal length is less than 1.5 or sepal length
# sepal length is 5.0.
oml_iris_filtered2 = oml_iris[(oml_iris["Petal_Length"] < 1.5) |
                              (oml_iris["Sepal_Length"] == 5.0), :]

len(oml_iris_filtered2)
oml_iris_filtered2.head(3)

# Select all rows in which petal length is less than 1.5 and
# sepal length is larger than 5.0.
oml_iris_filtered3 = oml_iris[(oml_iris["Petal_Length"] < 1.5) &
                              (oml_iris["Sepal_Length"] > 5.0), :]

len(oml_iris_filtered3)
oml_iris_filtered3.head()

```

Listing for This Example

```

>>> # Select sepal length and petal length where petal length
... # is less than 1.5.
... oml_iris_filtered1 = oml_iris[oml_iris["Petal_Length"] < 1.5,
...                               ["Sepal_Length", "Petal_Length"]]
>>> len(oml_iris_filtered1)
24
>>> oml_iris_filtered1.head(3)
   Sepal_Length  Petal_Length
0             5.1           1.4
1             4.9           1.4
2             4.7           1.3
>>>
>>> ### Using the AND and OR conditions in filtering.
... # Select all rows in which petal length is less than 1.5 or
... # sepal length is 5.0.
... oml_iris_filtered2 = oml_iris[(oml_iris["Petal_Length"] < 1.5) |
...                               (oml_iris["Sepal_Length"] == 5.0), :]
>>> len(oml_iris_filtered2)
30
>>> oml_iris_filtered2.head(3)
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width  Species
0             5.1           3.5           1.4           0.2  setosa
1             4.9           3.0           1.4           0.2  setosa
2             4.7           3.2           1.3           0.2  setosa
>>>
>>> # Select all rows in which petal length is less than 1.5
... # and sepal length is larger than 5.0.
... oml_iris_filtered3 = oml_iris[(oml_iris["Petal_Length"] < 1.5) &
...                               (oml_iris["Sepal_Length"] > 5.0), :]
>>> len(oml_iris_filtered3)
7
>>> oml_iris_filtered3.head()
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width  Species
0             5.1           3.5           1.4           0.2  setosa
1             5.8           4.0           1.2           0.2  setosa
2             5.4           3.9           1.3           0.4  setosa
3             5.1           3.5           1.4           0.3  setosa
4             5.2           3.4           1.4           0.2  setosa

```

8.1.3 Combine Data

You can join data from `oml.DataFrame` objects that represent database tables by using the `append`, `concat`, and `merge` methods.

Examples of using these methods are in the following topics.

- [Append Data from One Object to Another Object](#)
- [Combine Two Objects](#)
- [Join Data From Two Objects](#)

Append Data from One Object to Another Object

Use the `append` method to join two objects of the same data type.

Example 8-4 Appending Data from Two Tables

This example first appends the `oml.Float` series object `num1` to another `oml.Float` series object, `num2`. It then appends an `oml.DataFrame` object to another `oml.DataFrame` object, which has the same column types.

```
import oml
import pandas as pd

df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
                  "val" : ["a", "b", "c", "d", "e"],
                  "ch" : ["p", "q", "r", "a", "b"],
                  "num" : [4, 3, 6.7, 7.2, 5]})
oml_df = oml.push(df)

# Append an oml.Float series object to another.
num1 = oml_df['id']
num2 = oml_df['num']
num1.append(num2)

# Append an oml.DataFrame object to another.
x = oml_df[['id', 'val']] # 1st column oml.Float, 2nd column oml.String
y = oml_df[['num', 'ch']] # 1st column oml.Float, 2nd column oml.String
x.append(y)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>>
>>> df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
...                   "val" : ["a", "b", "c", "d", "e"],
...                   "ch" : ["p", "q", "r", "a", "b"],
...                   "num" : [4, 3, 6.7, 7.2, 5]})
>>> oml_df = oml.push(df)
>>>
>>> # Append an oml.Float series object to another.
... num1 = oml_df['id']
>>> num2 = oml_df['num']
>>> num1.append(num2)
[1, 2, 3, 4, 5, 4, 3, 6.7, 7.2, 5]
>>>
>>> # Explicitly convert oml.Integer to oml.Float
>>> oml.Float(num1).append(num2)
>>> # Append an oml.DataFrame object to another.
... x = oml_df[['id', 'val']] # 1st column oml.Float, 2nd column oml.String
>>> y = oml_df[['num', 'ch']] # 1st column oml.Float, 2nd column oml.String
>>> x.append(y)
   id val
0  1.0  a
1  2.0  b
2  3.0  c
3  4.0  d
4  5.0  e
5  4.0  p
```

```
6 3.0 q
7 6.7 r
8 7.2 a
9 5.0 b
```

Combine Two Objects

Use the `concat` method to combine columns from one object with those of another object. The `auto_name` argument of the `concat` method controls whether to invoke automatic name conflict resolution. You can also perform customized renaming by passing in a dictionary mapping strings to objects.

To combine two objects with the `concat` method, both objects must represent data from the same underlying database table, view, or query.

Example 8-5 Combining Data Column-Wise

This example first combines the two `oml.DataFrame` objects `x` and `y` column-wise. It then concatenates object `y` with the `oml.Float` series object `w`.

```
import oml
import pandas as pd
from collections import OrderedDict

df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
                  "val" : ["a", "b", "c", "d", "e"],
                  "ch" : ["p", "q", "r", "a", "b"],
                  "num" : [4, 3, 6.7, 7.2, 5]})
oml_df = oml.push(df)

# Create two oml.DataFrame objects and combine the objects column-wise.
x = oml_df[['id', 'val']]
y = oml_df[['num', 'ch']]
x.concat(y)

# Create an oml.Float object with the rounded exponential of two times
# the values in the num column of the oml_df object, then
# concatenate it with the oml.DataFrame object y using a new column name.
w = (oml_df['num']*2).exp().round(decimals=2)
y.concat({'round(exp(2*num))':w})

# Concatenate object x with multiple objects and turn on automatic
# name conflict resolution.
z = oml_df[:, 'id']
x.concat([z, w, y], auto_name=True)

# Concatenate multiple oml data objects and perform customized renaming.
x.concat(OrderedDict([('ID', z), ('round(exp(2*num))', w), ('New_', y)]))
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from collections import OrderedDict
>>>
>>> df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
```

```

...         "val" : ["a", "b", "c", "d", "e"],
...         "ch"  : ["p", "q", "r", "a", "b"],
...         "num" : [4, 3, 6.7, 7.2, 5])
>>> oml_df = oml.push(df)

>>> # Create two oml.DataFrame objects and combine the objects column-wise.
... x = oml_df[['id', 'val']]
>>> y = oml_df[['num', 'ch']]
>>> x.concat(y)
   id val  num  ch
0   1  a  4.0  p
1   2  b  3.0  q
2   3  c  6.7  r
3   4  d  7.2  a
4   5  e  5.0  b
>>>
>>> # Create an oml.Float object with the rounded exponential of two times
... # the values in the num column of the oml_df object, then
... # concatenate it with the oml.DataFrame object y using a new column name.
... w = (oml_df['num']*2).exp().round(decimals=2)
>>> y.concat({'round(exp(2*num))':w})
   num ch  round(exp(2*num))
0  4.0  p           2980.96
1  3.0  q           403.43
2  6.7  r        660003.22
3  7.2  a       1794074.77
4  5.0  b        22026.47
>>>
>>> # Concatenate object x with multiple objects and turn on automatic
... # name conflict resolution.
... z = oml_df[:, 'id']
>>> x.concat([z, w, y], auto_name=True)
   id val  id3      num  num5  ch
0  1  a    1    2980.96  4.0  p
1  2  b    2     403.43  3.0  q
2  3  c    3   660003.22  6.7  r
3  4  d    4  1794074.77  7.2  a
4  5  e    5    22026.47  5.0  b
>>>
>>> # Concatenate multiple oml data objects and perform customized renaming.
... x.concat(OrderedDict([('ID', z), ('round(exp(2*num))', w), ('New_', y)]))
   id val  ID  round(exp(2*num))  New_num  New_ch
0  1  a    1           2980.96         4.0      p
1  2  b    2            403.43         3.0      q
2  3  c    3       660003.22         6.7      r
3  4  d    4     1794074.77         7.2      a
4  5  e    5        22026.47         5.0      b

```

Join Data From Two Objects

Use the `merge` method to join data from two objects.

Example 8-6 Joining Data from Two Tables

This example first performs a cross join on the `oml.DataFrame` objects `x` and `y`, which creates the `oml.DataFrame` object `xy`. The example performs a left outer join on the first four rows of `x`

with the `oml.DataFrame` object `other` on the shared column `id` and applies the suffixes `.l` and `.r` to column names on the left and right side, respectively. The example then performs a right outer join on the `id` column on the left side object `x` and the `num` column on the right side object `y`.

```
import oml
import pandas as pd

df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
                  "val" : ["a", "b", "c", "d", "e"],
                  "ch" : ["p", "q", "r", "a", "b"],
                  "num" : [4, 3, 6.7, 7.2, 5]})
oml_df = oml.push(df)

x = oml_df[['id', 'val']]
y = oml_df[['num', 'ch']]

# Perform a cross join.
xy = x.merge(y)
xy

# Perform a left outer join.
x.head(4).merge(other=oml_df[['id', 'num']], on="id",
               suffixes=['.l', '.r'])

# Perform a right outer join.
x.merge(other=y, left_on="id", right_on="num", how="right")
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>>
>>> df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
...                   "val" : ["a", "b", "c", "d", "e"],
...                   "ch" : ["p", "q", "r", "a", "b"],
...                   "num" : [4, 3, 6.7, 7.2, 5]})
>>> oml_df = oml.push(df)
>>>
>>> x = oml_df[['id', 'val']]
>>> y = oml_df[['num', 'ch']]
>>>
>>> # Perform a cross join.
... xy = x.merge(y)
>>> xy
   id_l val_l  num_r ch_r
0     1    a    4.0    p
1     1    a    3.0    q
2     1    a    6.7    r
3     1    a    7.2    a
4     1    a    5.0    b
5     2    b    4.0    p
6     2    b    3.0    q
7     2    b    6.7    r
8     2    b    7.2    a
```

```

9      2      b      5.0      b
10     3      c      4.0      p
11     3      c      3.0      q
12     3      c      6.7      r
13     3      c      7.2      a
14     3      c      5.0      b
15     4      d      4.0      p
16     4      d      3.0      q
17     4      d      6.7      r
18     4      d      7.2      a
19     4      d      5.0      b
20     5      e      4.0      p
21     5      e      3.0      q
22     5      e      6.7      r
23     5      e      7.2      a
24     5      e      5.0      b
>>>
>>> # Perform a left outer join.
... x.head(4).merge(other=o1_df[['id', 'num']], on="id",
...                 suffixes=['.l', '.r'])
   id val.l num.r
0   1     a   4.0
1   2     b   3.0
2   3     c   6.7
3   4     d   7.2
>>>
>>> # Perform a right outer join.
... x.merge(other=y, left_on="id", right_on="num", how="right")
   id_l val_l num_r ch_r
0   3.0     c   3.0    q
1   4.0     d   4.0    p
2   5.0     e   5.0    b
3   NaN  None   6.7    r
4   NaN  None   7.2    a

```

8.1.4 Clean Data

In preparing data for analysis, a typical step is to transform data by dropping some values.

You can filter out unneeded data by using the `drop`, `drop_duplicates`, and `dropna` methods.

Example 8-7 Filtering Data

This example demonstrates ways of dropping columns with the `drop` method, dropping missing values with the `dropna` method, and dropping duplicate values with the `drop_duplicates` method.

```

import pandas as pd
import oml

df = pd.DataFrame({'numeric': [1, 1.4, -4, -4, 5.432, None, None],
                  'string1': [None, None, 'a', 'a', 'a', 'b', None],
                  'string2': ['x', None, 'z', 'z', 'z', 'x', None]})
o1_df = oml.push(df, dbtypes = {'numeric': 'BINARY_DOUBLE',
                              'string1': 'CHAR(1)',
                              'string2': 'CHAR(1)'})

```



```
# Drop rows with any missing values.
oml_df.dropna(how='any')

# Drop rows in which all column values are missing.
oml_df.dropna(how='all')

# Drop rows in which any numeric column values are missing.
oml_df.dropna(how='any', subset=['numeric'])

# Drop duplicate rows.
oml_df.drop_duplicates()

# Drop rows that have the same value in column 'string1' and 'string2'.
oml_df.drop_duplicates(subset=['string1', 'string2'])

# Drop column 'string2'
oml_df.drop('string2')
```

Listing for This Example

```
>>> import pandas as pd
>>> import oml
>>>
>>> df = pd.DataFrame({'numeric': [1, 1.4, -4, -4, 5.432, None, None],
...                   'string1': [None, None, 'a', 'a', 'a', 'b', None],
...                   'string2': ['x', None, 'z', 'z', 'z', 'x', None]})
>>> oml_df = oml.push(df, dbtypes = {'numeric': 'BINARY_DOUBLE',
...                                 'string1': 'CHAR(1)',
...                                 'string2': 'CHAR(1)'})
>>>
>>> # Drop rows with any missing values.
... oml_df.dropna(how='any')
   numeric string1 string2
0  -4.000      a      z
1  -4.000      a      z
2   5.432      a      z
>>>
>>> # Drop rows in which all column values are missing.
... oml_df.dropna(how='all')
   numeric string1 string2
0   1.000     None      x
1   1.400     None     None
2  -4.000      a      z
3  -4.000      a      z
4   5.432      a      z
5     NaN      b      x
>>>
>>> # Drop rows in which any numeric column values are missing.
... oml_df.dropna(how='any', subset=['numeric'])
   numeric string1 string2
0   1.000     None      x
1   1.400     None     None
2  -4.000      a      z
3  -4.000      a      z
```

```

4    5.432    a    z
>>>
>>> # Drop duplicate rows.
... oml_df.drop_duplicates()
   numeric string1 string2
0    5.432    a    z
1    1.000  None    x
2   -4.000    a    z
3     NaN    b    x
4    1.400  None  None
5     NaN  None  None
>>>
>>> # Drop rows that have the same value in columns 'string1' and 'string2'.
... oml_df.drop_duplicates(subset=['string1', 'string2'])
   numeric string1 string2
0    -4.0    a    z
1     1.4  None  None
2     1.0  None    x
3     NaN    b    x
>>>
>>> # Drop the column 'string2'.
... oml_df.drop('string2')
   numeric string1
0     1.000  None
1     1.400  None
2    -4.000    a
3    -4.000    a
4     5.432    a
5     NaN    b
6     NaN  None

```

8.1.5 Split Data

Sample and randomly partition data with the `split` and `KFold` methods.

In analyzing large data sets, a typical operation is to randomly partition the data set into subsets for training and testing purposes, which you can do with these methods. You can also sample data with the `split` method.

Example 8-8 Splitting Data into Multiple Sets

This example demonstrates splitting data into multiple sets and into k consecutive folds, which can be used for k -fold cross-validation.

```

import oml
import pandas as pd
from sklearn import datasets

digits = datasets.load_digits()
pd_digits = pd.DataFrame(digits.data,
                          columns=['IMG'+str(i) for i in
                                   range(digits['data'].shape[1])])
pd_digits = pd.concat([pd_digits,
                       pd.Series(digits.target,
                                  name = 'target')],
                       axis = 1)

```

```
oml_digits = oml.push(pd_digits)

# Sample 20% and 80% of the data.
splits = oml_digits.split(ratio=(.2, .8), use_hash = False)
[len(split) for split in splits]

# Split the data into four sets.
splits = oml_digits.split(ratio = (.25, .25, .25, .25),
                          use_hash = False)
[len(split) for split in splits]

# Perform stratification on the target column.
splits = oml_digits.split(strata_cols=['target'])
[split.shape for split in splits]

# Verify that the stratified sampling generates splits in which
# all of the different categories of digits (digits 0~9)
# are present in each split.
[split['target'].drop_duplicates().sort_values().pull()
 for split in splits]

# Hash on the target column.
splits = oml_digits.split(hash_cols=['target'])
[split.shape for split in splits]

# Verify that the different categories of digits (digits 0~9) are present
# in only one of the splits generated by hashing on the category column.
[split['target'].drop_duplicates().sort_values().pull()
 for split in splits]

# Split the data randomly into 4 consecutive folds.
folds = oml_digits.KFold(n_splits=4)
[(len(fold[0]), len(fold[1])) for fold in folds]
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> digits = datasets.load_digits()
>>> pd_digits = pd.DataFrame(digits.data,
...                          columns=['IMG'+str(i) for i in
...                                  range(digits['data'].shape[1])])
>>> pd_digits = pd.concat([pd_digits,
...                        pd.Series(digits.target,
...                                  name = 'target')],
...                        axis = 1)
>>> oml_digits = oml.push(pd_digits)
>>>
>>> # Sample 20% and 80% of the data.
... splits = oml_digits.split(ratio=(.2, .8), use_hash = False)
>>> [len(split) for split in splits]
[351, 1446]
>>>
```

```
>>> # Split the data into four sets.
... splits = oml_digits.split(ratio = (.25, .25, .25, .25),
...                             use_hash = False)
>>> [len(split) for split in splits]
[432, 460, 451, 454]
>>>
>>> # Perform stratification on the target column.
... splits = oml_digits.split(strata_cols=['target'])
>>> [split.shape for split in splits]
[(1285, 65), (512, 65)]
>>>
>>> # Verify that the stratified sampling generates splits in which
... # all of the different categories of digits (digits 0~9)
... # are present in each split.
... [split['target'].drop_duplicates().sort_values().pull()
... for split in splits]
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]]
>>>
>>> # Hash on the target column
... splits = oml_digits.split(hash_cols=['target'])
>>> [split.shape for split in splits]
[(899, 65), (898, 65)]
>>>
>>> # Verify that the different categories of digits (digits 0~9) are present
... # in only one of the splits generated by hashing on the category column.
... [split['target'].drop_duplicates().sort_values().pull()
... for split in splits]
[[0, 1, 3, 5, 8], [2, 4, 6, 7, 9]]
>>>
>>> # Split the data randomly into 4 consecutive folds.
... folds = oml_digits.KFold(n_splits=4)
>>> [(len(fold[0]), len(fold[1])) for fold in folds]
[(1352, 445), (1336, 461), (1379, 418), (1325, 472)]
```

8.2 Explore Data

OML4Py provides methods that enable you to perform exploratory data analysis and common statistical operations.

These methods are described in the following topics.

- [About the Exploratory Data Analysis Methods](#)
OML4Py provides methods that enable you to perform exploratory data analysis.
- [Correlate Data](#)
Use the `corr` method to perform Pearson, Spearman, or Kendall correlation analysis across columns where possible in an `oml.DataFrame` object.
- [Cross-Tabulate Data](#)
Use the `crosstab` method to perform cross-column analysis of an `oml.DataFrame` object and the `pivot_table` method to convert an `oml.DataFrame` to a spreadsheet-style pivot table.
- [Mutate Data](#)
In preparing data for analysis, a typical operation is to mutate data by reformatting it or deriving new columns and adding them to the data set.

- [Sort Data](#)
The `sort_values` function enables flexible sorting of an `oml.DataFrame` along one or more columns specified by the `by` argument, and returns an `oml.DataFrame`.
- [Summarize Data](#)
The `describe` method calculates descriptive statistics that summarize the central tendency, dispersion, and shape of the data in each column.
- [Date, Time, and Integer Data](#)
OML4Py provides the data types that enable you to manipulate date, time and integer.

8.2.1 About the Exploratory Data Analysis Methods

OML4Py provides methods that enable you to perform exploratory data analysis.

The following table lists methods of OML4Py data type classes with which you can perform common statistical operations and indicates whether the class supports the method.

Table 8-2 Data Exploration Methods Supported by Data Type Classes

Method	Description	oml.Boolean	oml.Bytes	oml.Float	oml.String	oml.DataFrame	oml.Date	oml.Time	oml.Timezone	oml.Integer
<code>corr</code>	Computes pairwise correlation between all columns in an <code>oml.DataFrame</code> where possible, given the type of coefficient.	✗	✗	✗	✗	✓	✗	✗	✗	✗
<code>count</code>	Computes the number of elements that are not NULL in the series data object or in each column of an <code>oml.DataFrame</code> .	✓	✓	✓	✓	✓	✓	✓	✓	✓
<code>crosstab</code>	Computes a cross-tabulation of two or more columns in an <code>oml.DataFrame</code> .	✗	✗	✗	✗	✓	✗	✗	✗	✗
<code>cumsum</code>	Computes the cumulative sum after an <code>oml.Float</code> series data object is sorted, or of each float or Boolean column after an <code>oml.DataFrame</code> object is sorted.	✗	✗	✓	✗	✓	✗	✗	✗	✓

Table 8-2 (Cont.) Data Exploration Methods Supported by Data Type Classes

Method	Description	oml.Boolean	oml.Bytes	oml.Float	oml.String	oml.DataFrame	oml.Date	oml.Time	oml.Interval	oml.Integer
describe	Computes descriptive statistics that summarize the central tendency, dispersion, and shape of an oml series data distribution, or of each column in an oml.DataFrame.	✓	✓	✓	✓	✓	✓	✓	✓	✓
kurtosis	Computes the kurtosis of the values in an oml.Float series data object, or for each float column in an oml.DataFrame.	✗	✗	✓	✗	✓	✗	✗	✗	✓
max	Returns the maximum value in a series data object or in each column in an oml.DataFrame.	✓	✓	✓	✓	✓	✓	✓	✓	✓
mean	Computes the mean of the values in an oml.Float series object, or for each float or Boolean column in an oml.DataFrame.	✗	✗	✓	✗	✓	✗	✗	✗	✓
median	Computes the median of the values in an oml.Float series object, or for each float column in an oml.DataFrame.	✗	✗	✓	✗	✓	✗	✗	✗	✓
min	Returns the minimum value in a series data object or of each column in an oml.DataFrame.	✓	✓	✓	✓	✓	✓	✓	✓	✓
nunique	Computes the number of unique values in a series data object or in each column of an oml.DataFrame.	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 8-2 (Cont.) Data Exploration Methods Supported by Data Type Classes

Method	Description	oml.Boolean	oml.Bytes	oml.Float	oml.String	oml.DataFrame	oml.Date	oml.Time	oml.Interval	oml.Integer
<code>pivot_table</code>	Converts an <code>oml.DataFrame</code> to a spreadsheet-style pivot table.	✗	✗	✗	✗	✓	✗	✗	✗	✗
<code>sort_values</code>	Sorts the values in a series data object or sorts the rows in an <code>oml.DataFrame</code> .	✓	✓	✓	✓	✓	✓	✓	✓	✓
<code>skew</code>	Computes the skewness of the values in an <code>oml.Float</code> data series object or of each float column in an <code>oml.DataFrame</code> .	✗	✗	✓	✗	✓	✗	✗	✗	✓
<code>std</code>	Computes the standard deviation of the values in an <code>oml.Float</code> data series object or in each float or Boolean column in an <code>oml.DataFrame</code> .	✗	✗	✓	✗	✓	✗	✗	✗	✓
<code>sum</code>	Computes the sum of the values in an <code>oml.Float</code> data series object or of each float or Boolean column in an <code>oml.DataFrame</code> .	✗	✗	✓	✗	✓	✗	✗	✗	✓

8.2.2 Correlate Data

Use the `corr` method to perform Pearson, Spearman, or Kendall correlation analysis across columns where possible in an `oml.DataFrame` object.

For details about the function arguments, invoke `help(oml.DataFrame.corr)` or see [Oracle Machine Learning for Python API Reference](#).

Example 8-9 Performing Basic Correlation Calculations

This example first creates a temporary database table, with its corresponding proxy `oml.DataFrame` object `oml_df1`, from the `pandas.DataFrame` object `df`. It then verifies the correlation computed between columns A and B, which gives 1, as expected. The values in B are twice the values in A element-wise. The example also changes a value field in `df` and creates a `NaN` entry. It then creates a temporary database table, with the corresponding proxy

`oml.DataFrame` object `oml_df2`. Finally, it invokes the `corr` method on `oml_df2` with `skipna` set to `True` (the default) and then `False` to compare the results.

```
import oml
import pandas as pd

df = pd.DataFrame({'A': range(4), 'B': [2*i for i in range(4)]})
oml_df1 = oml.push(df)

# Verify that the correlation between column A and B is 1.
oml_df1.corr()

# Change a value to test the change in the computed correlation result.
df.loc[2, 'A'] = 1.5

# Change an entry to NaN (not a number) to test the 'skipna'
# parameter in the corr method.
df.loc[1, 'B'] = None

# Push df to the database using the floating point column type
# because NaNs cannot be used in Oracle numbers.
oml_df2 = oml.push(df, oranumber=False)

# By default, 'skipna' is True.
oml_df2.corr()
oml_df2.corr(skipna=False)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>>
>>> df = pd.DataFrame({'A': range(4), 'B': [2*i for i in range(4)]})
>>> oml_df1 = oml.push(df)
>>>
>>> # Verify that the correlation between column A and B is 1.
... oml_df1.corr()
   A  B
A  1  1
B  1  1
>>>
>>> # Change a value to test the change in the computed correlation result.
... df.loc[2, 'A'] = 1.5
>>>
>>> # Change an entry to NaN (not a number) so to test the 'skipna'
... # parameter in the corr method.
... df.loc[1, 'B'] = None
>>>
>>> # Push df to the database using the floating point column type
... # because NaNs cannot be used in Oracle numbers.
... oml_df2 = oml.push(df, oranumber=False)
>>>
>>> # By default, 'skipna' is True.
... oml_df2.corr()
   A  B
```



```

A 1.000000 0.981981
B 0.981981 1.000000
>>> oml_df2.corr(skipna=False)
      A      B
A 1.0  NaN
B NaN  1.0

```

8.2.3 Cross-Tabulate Data

Use the `crosstab` method to perform cross-column analysis of an `oml.DataFrame` object and the `pivot_table` method to convert an `oml.DataFrame` to a spreadsheet-style pivot table.

Cross-tabulation is a statistical technique that finds an interdependent relationship between two columns of values. The `crosstab` method computes a cross-tabulation of two or more columns. By default, it computes a frequency table for the columns unless a column and an aggregation function have been passed to it.

The `pivot_table` method converts a data set into a pivot table. Due to the database 1000 column limit, pivot tables with more than 1000 columns are automatically truncated to display the categories with the most entries for each column value.

For details about the method arguments, invoke `help(oml.DataFrame.crosstab)` or `help(oml.DataFrame.pivot_table)`, or see [Oracle Machine Learning for Python API Reference](#).

Example 8-10 Producing Cross-Tabulation and Pivot Tables

This example demonstrates the use of the `crosstab` and `pivot_table` methods.

```

import pandas as pd
import oml

x = pd.DataFrame({
    'GENDER': ['M', 'M', 'F', 'M', 'F', 'M', 'F', 'F',
              None, 'F', 'M', 'F'],
    'HAND': ['L', 'R', 'R', 'L', 'R', None, 'L', 'R',
            'R', 'R', 'R', 'R'],
    'SPEED': [40.5, 30.4, 60.8, 51.2, 54, 29.3, 34.1,
              39.6, 46.4, 12, 25.3, 37.5],
    'ACCURACY': [.92, .94, .87, .9, .85, .97, .96, .93,
                 .89, .84, .91, .95]
})
x = oml.push(x)

# Find the categories that the most entries belonged to.
x.crosstab('GENDER', 'HAND').sort_values('count', ascending=False)

# For each gender value and across all entries, find the ratio of entries
# with different hand values.
x.crosstab('GENDER', 'HAND', pivot = True, margins = True, normalize = 0)

# Find the mean speed across all gender and hand combinations.
x.pivot_table('GENDER', 'HAND', 'SPEED')

# Find the median accuracy and speed for every gender and hand combination.
x.pivot_table('GENDER', 'HAND', aggfunc = oml.DataFrame.median)

```

```
# Find the max and min speeds for every gender and hand combination and
# across all combinations.
x.pivot_table('GENDER', 'HAND', 'SPEED',
              aggfunc = [oml.DataFrame.max, oml.DataFrame.min],
              margins = True)
```

Listing for This Example

```
>>> import pandas as pd
>>> import oml
>>>
>>> x = pd.DataFrame({
...     'GENDER': ['M', 'M', 'F', 'M', 'F', 'M', 'F', 'F',
...               None, 'F', 'M', 'F'],
...     'HAND': ['L', 'R', 'R', 'L', 'R', None, 'L', 'R',
...              'R', 'R', 'R', 'R'],
...     'SPEED': [40.5, 30.4, 60.8, 51.2, 54, 29.3, 34.1,
...               39.6, 46.4, 12, 25.3, 37.5],
...     'ACCURACY': [.92, .94, .87, .9, .85, .97, .96, .93,
...                  .89, .84, .91, .95]
... })
>>> x = oml.push(x)
>>>
>>> # Find the categories that the most entries belonged to.
... x.crosstab('GENDER', 'HAND').sort_values('count', ascending=False)
GENDER HAND count
0      F      R      5
1      M      L      2
2      M      R      2
3      M  None      1
4      F      L      1
5  None      R      1
>>>
>>> # For each gender value and across all entries, find the ratio of entries
... # with different hand values.
... x.crosstab('GENDER', 'HAND', pivot = True, margins = True, normalize = 0)
GENDER count_(L) count_(R) count_(None)
0  None  0.000000  1.000000  0.000000
1      F  0.166667  0.833333  0.000000
2      M  0.400000  0.400000  0.200000
3  All   0.250000  0.666667  0.083333
>>>
>>> # Find the mean speed across all gender and hand combinations.
... x.pivot_table('GENDER', 'HAND', 'SPEED')
GENDER mean(SPEED)_(L) mean(SPEED)_(R) mean(SPEED)_(None)
0  None                NaN                46.40                NaN
1      F                34.10                40.78                NaN
2      M                45.85                27.85                29.3
>>>
>>> # Find the median accuracy and speed for every gender and hand
combination.
... x.pivot_table('GENDER', 'HAND', aggfunc = oml.DataFrame.median)
GENDER median(ACCURACY)_(L) median(ACCURACY)_(R)
median(ACCURACY)_(None) \
```

```

0   None           NaN           0.890
NaN
1     F           0.96           0.870
NaN
2     M           0.91           0.925
0.97

      median(SPEED)_(L)  median(SPEED)_(R)  median(SPEED)_(None)
0                NaN           46.40           NaN
1            34.10           39.60           NaN
2            45.85           27.85           29.3
>>>
>>> # Find the max and min speeds for every gender and hand combination and
... # across all combinations.
... x.pivot_table('GENDER', 'HAND', 'SPEED',
...               aggfunc = [oml.DataFrame.max, oml.DataFrame.min],
...               margins = True)
      GENDER  max(SPEED)_(L)  max(SPEED)_(R)  max(SPEED)_(None)
max(SPEED)_(All) \
0   None           NaN           46.4           NaN
46.4
1     F           34.1           60.8           NaN
60.8
2     M           51.2           30.4           29.3
51.2
3   All           51.2           60.8           29.3
60.8

      min(SPEED)_(L)  min(SPEED)_(R)  min(SPEED)_(None)  min(SPEED)_(All)
0                NaN           46.4           NaN           46.4
1            34.1           12.0           NaN           12.0
2            40.5           25.3           29.3           25.3
3            34.1           12.0           29.3           12.0

```

8.2.4 Mutate Data

In preparing data for analysis, a typical operation is to mutate data by reformatting it or deriving new columns and adding them to the data set.

These examples demonstrate methods of formatting data and deriving columns.

```

import pandas as pd
import oml

# Create a shopping cart data set.
shopping_cart = pd.DataFrame({
    'Item_name': ['paper_towel', 'ground_pork', 'tofu', 'eggs',
                 'pork_loin', 'whole_milk', 'egg_custard'],
    'Item_type': ['grocery', 'meat', 'grocery', 'dairy', 'meat',
                 'dairy', 'bakery'],
    'Quantity': [1, 2.6, 4, 1, 1.9, 1, 1],
    'Unit_price': [1.19, 2.79, 0.99, 2.49, 3.19, 2.5, 3.99]
})
oml_cart = oml.push(shopping_cart)
oml_cart

```

```
# Add a column 'Price' multiplying 'Quantity' with 'Unit_price',
# rounded to 2 decimal places.
price = oml_cart['Quantity']*(oml_cart['Unit_price'])
type(price)
price
oml_cart = oml_cart.concat({'Price': price.round(2)})

# Count the pattern 'egg' in the 'Item_name' column.
egg_pattern = oml_cart['Item_name'].count_pattern('egg')
type(egg_pattern)
oml_cart.concat({'Egg_pattern': egg_pattern})

# Find the start index of substring 'pork' in the 'Item_name' column.
pork_startInd = oml_cart['Item_name'].find('pork')
type(pork_startInd)
oml_cart.concat({'Pork_startInd': pork_startInd})

# Check whether items are of grocery category.
is_grocery=oml_cart['Item_type']=='grocery'
type(is_grocery)
oml_cart.concat({'Is_grocery': is_grocery})

# Calculate the length of item names.
name_length=oml_cart['Item_name'].len()
type(name_length)
oml_cart.concat({'Name_length': name_length})

# Get the ceiling, floor, exponential, logarithm and square root
# of the 'Price' column.
oml_cart['Price'].ceil()
oml_cart['Price'].floor()
oml_cart['Price'].exp()
oml_cart['Price'].log()
oml_cart['Price'].sqrt()
```

Listing for This Example

```
>>> import pandas as pd
>>> import oml
>>>
>>> # Create a shopping cart data set.
... shopping_cart = pd.DataFrame({
...     'Item_name': ['paper_towel', 'ground_pork', 'tofu', 'eggs',
...                  'pork_loin', 'whole_milk', 'egg_custard'],
...     'Item_type': ['grocery', 'meat', 'grocery', 'dairy', 'meat',
...                  'dairy', 'bakery'],
...     'Quantity': [1, 2.6, 4, 1, 1.9, 1, 1],
...     'Unit_price': [1.19, 2.79, 0.99, 2.49, 3.19, 2.5, 3.99]
... })
>>> oml_cart = oml.push(shopping_cart)
>>> oml_cart
```

	Item_name	Item_type	Quantity	Unit_price
0	paper_towel	grocery	1.0	1.19
1	ground_pork	meat	2.6	2.79

```

2      tofu  grocery      4.0      0.99
3      eggs   dairy       1.0      2.49
4  pork_loin  meat        1.9      3.19
5  whole_milk dairy       1.0      2.50
6  egg_custard bakery     1.0      3.99
>>>
>>> # Add a column 'Price' multiplying 'Quantity' with 'Unit_price',
... # rounded to 2 decimal places.
... price = oml_cart['Quantity']*(oml_cart['Unit_price'])
>>> type(price)
<class 'oml.core.float.Float'>
>>> price
[1.19, 7.254, 3.96, 2.49, 6.061, 2.5, 3.99]
>>> oml_cart = oml_cart.concat({'Price': price.round(2)})
>>>
>>> # Count the pattern 'egg' in the 'Item_name' column.
... egg_pattern = oml_cart['Item_name'].count_pattern('egg')
>>> type(egg_pattern)
<class 'oml.core.float.Float'>
>>> oml_cart.concat({'Egg_pattern': egg_pattern})

```

	Item_name	Item_type	Quantity	Unit_price	Price	Egg_pattern
0	paper_towel	grocery	1.0	1.19	1.19	0
1	ground_pork	meat	2.6	2.79	7.25	0
2	tofu	grocery	4.0	0.99	3.96	0
3	eggs	dairy	1.0	2.49	2.49	1
4	pork_loin	meat	1.9	3.19	6.06	0
5	whole_milk	dairy	1.0	2.50	2.50	0
6	egg_custard	bakery	1.0	3.99	3.99	1

```

>>>
>>> # Find the start index of substring 'pork' in the 'Item_name' column.
... pork_startInd = oml_cart['Item_name'].find('pork')
>>> type(pork_startInd)
<class 'oml.core.float.Float'>
>>> oml_cart.concat({'Pork_startInd': pork_startInd})

```

	Item_name	Item_type	Quantity	Unit_price	Price	Pork_startInd
0	paper_towel	grocery	1.0	1.19	1.19	-1
1	ground_pork	meat	2.6	2.79	7.25	7
2	tofu	grocery	4.0	0.99	3.96	-1
3	eggs	dairy	1.0	2.49	2.49	-1
4	pork_loin	meat	1.9	3.19	6.06	0
5	whole_milk	dairy	1.0	2.50	2.50	-1
6	egg_custard	bakery	1.0	3.99	3.99	-1

```

>>>
>>> # Check whether items are of grocery category.
... is_grocery=oml_cart['Item_type']=='grocery'
>>> type(is_grocery)
<class 'oml.core.boolean.Boolean'>
>>> oml_cart.concat({'Is_grocery': is_grocery})

```

	Item_name	Item_type	Quantity	Unit_price	Price	Is_grocery
0	paper_towel	grocery	1.0	1.19	1.19	True
1	ground_pork	meat	2.6	2.79	7.25	False
2	tofu	grocery	4.0	0.99	3.96	True
3	eggs	dairy	1.0	2.49	2.49	False
4	pork_loin	meat	1.9	3.19	6.06	False
5	whole_milk	dairy	1.0	2.50	2.50	False
6	egg_custard	bakery	1.0	3.99	3.99	False

```

>>>
>>> # Calculate the length of item names.
... name_length=oml_cart['Item_name'].len()
>>> type(name_length)
<class 'oml.core.float.Float'>
>>> oml_cart.concat({'Name_length': name_length})
   Item_name Item_type  Quantity  Unit_price  Price  Name_length
0  paper_towel  grocery     1.0         1.19   1.19         11
1  ground_pork   meat     2.6         2.79   7.25         11
2         tofu  grocery     4.0         0.99   3.96          4
3         eggs   dairy     1.0         2.49   2.49          4
4   pork_loin   meat     1.9         3.19   6.06          9
5  whole_milk   dairy     1.0         2.50   2.50         10
6  egg_custard  bakery     1.0         3.99   3.99         11
>>>
>>> # Get the ceiling, floor, exponential, logarithm and square root
... # of the 'Price' column.
... oml_cart['Price'].ceil()
[2, 8, 4, 3, 7, 3, 4]
>>> oml_cart['Price'].floor()
[1, 7, 3, 2, 6, 2, 3]
>>> oml_cart['Price'].exp()
[3.2870812073831184, 1408.1048482046956, 52.45732594909905,
12.061276120444719, 428.37543685928694, 12.182493960703473, 54.05488936332659]
>>> oml_cart['Price'].log()
[0.173953307123438, 1.9810014688665833, 1.3762440252663892,
0.9122827104766162, 1.801709800081223, 0.9162907318741551, 1.3837912309017721]
>>> oml_cart['Price'].sqrt()
[1.0908712114635715, 2.692582403567252, 1.98997487421324, 1.57797338380595,
2.4617067250182343, 1.5811388300841898, 1.997498435543818]

```

8.2.5 Sort Data

The `sort_values` function enables flexible sorting of an `oml.DataFrame` along one or more columns specified by the `by` argument, and returns an `oml.DataFrame`.

Example 8-11 Sorting Data

The following example demonstrates these operations.

```

import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

# Create the IRIS database table and the proxy object for the table.

```

```
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Modify the data set by replacing a few entries with NaNs to test
# how the na_position parameter works in the sort_values method.
Iris = oml_iris.pull()
Iris['Sepal_Width'].replace({3.5: None}, inplace=True)
Iris['Petal_Length'].replace({1.5: None}, inplace=True)
Iris['Petal_Width'].replace({2.3: None}, inplace=True)

# Create another table using the changed data.
oml_iris2 = oml.create(Iris, table = 'IRIS2')

# Sort the data set first by Sepal_Length then by Sepal_Width
# in descending order and display the first 5 rows of the
# sorted result.
oml_iris2.sort_values(by = ['Sepal_Length', 'Sepal_Width'],
                     ascending=False).head()

# Display the last 5 rows of the data set.
oml_iris2.tail()

# Sort the last 5 rows of the iris data set first by Petal_Length
# then by Petal_Width. By default, rows with NaNs are placed
# after the other rows when the sort keys are the same.
oml_iris2.tail().sort_values(by = ['Petal_Length', 'Petal_Width'])

# Sort the last 5 rows of the iris data set first by Petal_Length
# and then by Petal_Width. When the values in these two columns
# are the same, place the row with a NaN before the other row.
oml_iris2.tail().sort_values(by = ['Petal_Length', 'Petal_Width'],
                             na_position = 'first')

oml.drop('IRIS')
oml.drop('IRIS2')
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                             'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Modify the data set by replacing a few entries with NaNs to test
```

```
... # how the na_position parameter works in the sort_values method.
... Iris = oml_iris.pull()
>>> Iris['Sepal_Width'].replace({3.5: None}, inplace=True)
>>> Iris['Petal_Length'].replace({1.5: None}, inplace=True)
>>> Iris['Petal_Width'].replace({2.3: None}, inplace=True)
>>>
>>> # Create another table using the changed data.
... oml_iris2 = oml.create(Iris, table = 'IRIS2')
>>>
>>> # Sort the data set first by 'Sepal_Length' then by 'Sepal_Width'
... # in descending order and displays the first 5 rows of the
... # sorted result.
... oml_iris2.sort_values(by = ['Sepal_Length', 'Sepal_Width'],
...
...                          ascending=False).head()
      Sepal_Length  Sepal_Width  Petal_Length  Petal_Width  Species
0                7.9          3.8           6.4          2.0  virginica
1                7.7          3.8           6.7          2.2  virginica
2                7.7          3.0           6.1          NaN  virginica
3                7.7          2.8           6.7          2.0  virginica
4                7.7          2.6           6.9          NaN  virginica
>>>
>>> # Display the last 5 rows of the data set.
... oml_iris2.tail()
      Sepal_Length  Sepal_Width  Petal_Length  Petal_Width  Species
0                6.7          3.0           5.2          NaN  virginica
1                6.3          2.5           5.0          1.9  virginica
2                6.5          3.0           5.2          2.0  virginica
3                6.2          3.4           5.4          NaN  virginica
4                5.9          3.0           5.1          1.8  virginica
>>>
>>> # Sort the last 5 rows of the iris data set first by 'Petal_Length'
... # then by 'Petal_Width'. By default, rows with NaNs are placed
... # after the other rows when the sort keys are the same.
... oml_iris2.tail().sort_values(by = ['Petal_Length', 'Petal_Width'])
      Sepal_Length  Sepal_Width  Petal_Length  Petal_Width  Species
0                6.3          2.5           5.0          1.9  virginica
1                5.9          3.0           5.1          1.8  virginica
2                6.5          3.0           5.2          2.0  virginica
3                6.7          3.0           5.2          NaN  virginica
4                6.2          3.4           5.4          NaN  virginica
>>>
>>> # Sort the last 5 rows of the iris data set first by 'Petal_Length'
... # and then by 'Petal_Width'. When the values in these two columns
... # are the same, place the row with a NaN before the other row.
... oml_iris2.tail().sort_values(by = ['Petal_Length', 'Petal_Width'],
...
...                          na_position = 'first')
      Sepal_Length  Sepal_Width  Petal_Length  Petal_Width  Species
0                6.3          2.5           5.0          1.9  virginica
1                5.9          3.0           5.1          1.8  virginica
2                6.7          3.0           5.2          NaN  virginica
3                6.5          3.0           5.2          2.0  virginica
4                6.2          3.4           5.4          NaN  virginica
>>>
>>> oml.drop('IRIS')
>>> oml.drop('IRIS2')
```


8.2.6 Summarize Data

The `describe` method calculates descriptive statistics that summarize the central tendency, dispersion, and shape of the data in each column.

You can also specify the types of columns to include or exclude from the results.

With the `sum` and `cumsum` methods, you can compute the sum and cumulative sum of each Float or Boolean column of an `oml.DataFrame`.

The `describe` method supports finding the following statistics:

- Mean, minimum, maximum, median, top character, standard deviation
- Number of not-Null values, unique values, top characters
- Percentiles between 0 and 1

Example 8-12 Calculating Descriptive Statistics

The following example demonstrates these operations.

```
import pandas as pd
import oml

df = pd.DataFrame({'numeric': [1, 1.4, -4, 3.145, 5, None],
                  'string' : [None, None, 'a', 'a', 'a', 'b'],
                  'bytes' : [b'a', b'b', b'c', b'c', b'd', b'e]})

oml_df = oml.push(df, dbtypes = {'numeric': 'BINARY_DOUBLE',
                                'string': 'CHAR(1)',
                                'bytes': 'RAW(1)'})

# Combine a Boolean column with oml_df.
oml_bool = oml_df['numeric'] > 3
oml_df = oml_df.concat(oml_bool)
oml_df.rename({'COL4': 'boolean'})

# Describe all of the columns.
oml_df.describe(include='all')

# Exclude Float columns.
oml_df.describe(exclude=[oml.Float])

# Get the sum of values in each Float or Boolean column.
oml_df.sum()

# Find the cumulative sum of values in each Float or Boolean column
# after oml_df is sorted by the bytes column in descending order.
oml_df.cumsum(by = 'bytes', ascending = False)

# Compute the skewness of values in the Float columns.
oml_df.skew()

# Find the median value of Float columns.
oml_df.median()
```

```
# Calculate the kurtosis of Float columns.
oml_df.kurtosis()
```

Listing for This Example

```
>>> import pandas as pd
>>> import oml
>>>
>>> df = pd.DataFrame({'numeric': [1, 1.4, -4, 3.145, 5, None],
...                   'string' : [None, None, 'a', 'a', 'a', 'b'],
...                   'bytes'  : [b'a', b'b', b'c', b'c', b'd', b'e]})
>>>
>>> oml_df = oml.push(df, dbtypes = {'numeric': 'BINARY_DOUBLE',
...                                 'string': 'CHAR(1)',
...                                 'bytes': 'RAW(1)'})
>>>
>>> # Combine a Boolean column with oml_df.
... oml_bool = oml_df['numeric'] > 3
>>> oml_df = oml_df.concat(oml_bool)
>>> oml_df.rename({'COL4': 'boolean'})
  bytes  numeric string  boolean
0 b'a'    1.000  None   False
1 b'b'    1.400  None   False
2 b'c'   -4.000    a   False
3 b'c'    3.145    a    True
4 b'd'    5.000    a    True
5 b'e'     NaN    b    True
>>>
>>> # Describe all of the columns.
... oml_df.describe(include='all')
      bytes  numeric string  boolean
count      6  5.000000     4         6
unique      5      NaN     2         2
top      b'c'      NaN     a   False
freq        2      NaN     3         3
mean      NaN  1.309000   NaN      NaN
std       NaN  3.364655   NaN      NaN
min       NaN -4.000000   NaN      NaN
25%      NaN  1.000000   NaN      NaN
50%      NaN  1.400000   NaN      NaN
75%      NaN  3.145000   NaN      NaN
max       NaN  5.000000   NaN      NaN
>>>
>>> # Exclude Float columns.
... oml_df.describe(exclude=[oml.Float])
      bytes string  boolean
count      6     4         6
unique      5     2         2
top      b'c'    a   False
freq        2     3         3
>>>
>>> # Get the sum of values in each Float or Boolean column.
... oml_df.sum()
numeric    6.545
boolean    3.000
```

```
dtype: float64
>>>
>>> # Find the cumulative sum of values in each Float or Boolean column
... # after oml_df is sorted by the bytes column in descending order.
... oml_df.cumsum(by = 'bytes', ascending = False)
   numeric  boolean
0      NaN         1
1    5.000         2
2    1.000         2
3    4.145         3
4    5.545         3
5    6.545         3
>>>
>>> # Compute the skewness of values in the Float columns.
... oml_df.skew()
numeric  -0.683838
dtype: float64
>>>
>>> # Find the median value of Float columns.
... oml_df.median()
numeric    1.4
dtype: float64
>>>
>>> # Calculate the kurtosis of Float columns.
... oml_df.kurtosis()
numeric  -0.582684
dtype: float64
```

8.2.7 Date, Time, and Integer Data

OML4Py provides the data types that enable you to manipulate date, time and integer.

The following newly added data types are now supported in OML4Py:

- `oml.Datetime`
- `oml.Timezone`
- `oml.Timedelta`
- `oml.Integer`

For information on the attributes and methods of `oml.Datetime`, `oml.Timezone`, `oml.Timedelta` and `oml.Integer`, see [Oracle Machine Learning for Python API Reference](#).

oml.Datetime

To create a date, you can use the `datetime` class of the `datetime` module, which is included in OML4Py.

The `datetime` class requires three parameters: `year`, `month`, and `day`. It also contains optional parameters for time and timezone that includes `hour`, `minute`, `second`, `microsecond`, and `tzone`.

Example 8-13 Using the oml.Datetime Function

This example creates a proxy object from a table with DATE column.

```
import oml
import pandas as pd
import numpy as np
import datetime
from datetime import datetime, timezone, timedelta
SALES = oml.sync(schema="SH", table="SALES")
z.show(SALES.head())

# Use the following command to compute the statistics on table columns:
pd.set_option('display.max_columns', 50)
pd.set_option('display.width', 1000)
SALES.describe(include='all')

# Use the following command to compute statistics on DATE column TIME_ID:
SALES['TIME_ID'].describe()

# Use the following command to extract date-related features:
date = SALES['TIME_ID']
SALES2 = SALES.concat({'YEAR': date.year, 'MONTH': date.month})
SALES2.head()
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> import numpy as np
>>> import datetime
>>> from datetime import datetime, timezone, timedelta
>>> SALES = oml.sync(schema="SH", table="SALES")
>>> z.show(SALES.head())
>>>  PROD_ID  CUST_ID  TIME_ID  CHANNEL_ID  PROMO_ID
QUANTITY_SOLD  AMOUNT_SOLD
      13      524   1998-01-20 00:00:00  2          999
1.0           1205.99
      13      2128   1998-04-05 00:00:00  2          999
1.0           1250.25
      13      3212   1998-04-05 00:00:00  2          999
1.0           1250.25
      13      3375   1998-04-05 00:00:00  2          999
1.0           1250.25
      13      5204   1998-04-05 00:00:00  2          999
1.0           1250.25
>>> pd.set_option('display.max_columns', 50)
>>> pd.set_option('display.width', 1000)
>>> SALES.describe(include='all')
>>>
          PROD_ID      CUST_ID      TIME_ID
CHANNEL_ID      PROMO_ID  QUANTITY_SOLD      AMOUNT_SOLD
count  918843.000000  918843.000000      918843
918843.000000  918843.000000      918843.0      918843.000000
unique  NaN          NaN          1460
NaN          NaN          NaN          NaN
top     NaN          NaN          2001-10-18 00:00:00
```

```

NaN          NaN          NaN          NaN
freq  NaN          NaN          NaN          2940
NaN          NaN          NaN          NaN
mean  78.183945    7289.807720          NaN
2.861603    976.396093    1.0          106.879882
std   49.008014    8948.653221    0.0          259.780490
0.686874    121.829887
min   13.000000    2.000000          NaN
2.000000    33.000000    1.0          6.400000
25%   31.000000    2383.000000          NaN
2.000000    999.000000    1.0          17.380000
50%   48.000000    4927.000000          NaN
3.000000    999.000000    1.0          34.240000
75%  127.000000    9163.000000          NaN
3.000000    999.000000    1.0          53.890000
max   148.000000   10100.000000          NaN
9.000000    999.000000    1.0          1782.720000
>>> SALES['TIME_ID'].describe()
>>> count          918843
      unique         1460
      top      2001-10-18 00:00:00
      freq          2940
      Name: TIME_ID, dtype: object
>>> date = SALES['TIME_ID']
>>> SALES2 = SALES.concat({'YEAR': date.year, 'MONTH': date.month})
>>> SALES2.head()
>>> PROD_ID  CUST_ID  TIME_ID  CHANNEL_ID  PROMO_ID  QUANTITY_SOLD
AMOUNT_SOLD  YEAR  MONTH
0           13      524 1998-01-20          2          999          1.0
1205.99  1998          1
1           13     2128 1998-04-05          2          999          1.0
1250.25  1998          4
2           13     3212 1998-04-05          2          999          1.0
1250.25  1998          4
3           13     3375 1998-04-05          2          999          1.0
1250.25  1998          4
4           13     5204 1998-04-05          2          999          1.0
1250.25  1998          4

```

Example 8-14 Using the oml.Datetime Function

This example creates a datetime object with the year, month, day, then creates a temporary proxy object using `oml.push`. The `oml.push` function requires a data frame or list as input, so the date object is converted to a list. The resulting object is an object of class `oml.Datetime`.

```

import oml
import pandas as pd
import numpy as np

import datetime
from datetime import datetime, timezone, timedelta

d1 = datetime(year=2004, month=7, day=24)

print ('d1:', d1)
print('d1 year:', d1.year)

```

```
print('d1 month:', d1.month)
d1_lst = [d1]
D1 = oml.push(d1_lst)

print ('type', type(D1))
print ('D1:', D1)
print ('year:', D1.year)
print ('month:', D1.month)
print ('day:', D1.day)
d2 = datetime.fromisoformat('2004-07-24 00:05:23+04:00')
d2_lst=[d2]

D2 = oml.push(d2_lst)

print('D2', D2)
print('type', type(D2))
D2.strftime()
D2.strftime()
d3 = "14-Jul-05 20:01:01"
d3_lst = [d3]
D3 = oml.push(d3_lst)
oml.Datetime.strptime(D3, "DD-Mon-RR HH24:MI:SS")
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> import numpy as np

>>> import datetime
>>> from datetime import datetime, timezone, timedelta

>>> d1 = datetime(year=2004, month=7, day=24)

>>> print ('d1:', d1)
>>> d1: 2004-07-24 00:00:00
>>> print('d1 year:', d1.year)
>>> d1 year: 2004
>>> print('d1 month:', d1.month)
>>> d1 month: 7

>>> d1_lst = [d1]
>>> D1 = oml.push(d1_lst)

>>> print ('type', type(D1))
>>> type <class 'oml.core.datetime.Datetime'>
>>> print ('D1:', D1)
>>> D1: [datetime.datetime(2004, 7, 24, 0, 0)]
>>> print ('year:', D1.year)
>>> year: [2004]
>>> print ('month:', D1.month)
>>> month: [7]
>>> print ('day:', D1.day)
>>> day: [24]
```

```

>>> d2 = datetime.fromisoformat('2004-07-24 00:05:23+04:00')
>>> d2_lst=[d2]
>>> D2 = oml.push(d2_lst)
>>> print('D2', D2)
>>> D2 [datetime.datetime(2004, 7, 24, 0, 5, 23,
tzinfo=datetime.timezone(datetime.timedelta(seconds=14400)))]
>>> print('type', type(D2))
>>> type <class 'oml.core.datetime.Datetime'>

>>> D2.strftime()
>>> ['2004-07-24 00:05:23+04:00']
>>> d3 = "14-Jul-05 20:01:01"
>>> d3_lst = [d3]

>>> D3 = oml.push(d3_lst)

>>> oml.Datetime.strptime(D3, "DD-Mon-RR HH24:MI:SS")
>>> [datetime.datetime(2005, 7, 14, 20, 1, 1)]

```

oml.Timedelta

`oml.Timedelta` objects represent a span of time, which can be used to perform simple arithmetic operations on `oml.Datetime` objects. The `oml.Timedelta` objects can be multiplied by an integer value or a floating point value. Subtracting dates creates an `oml.Timedelta` object that can be added, subtracted, or multiplied by a `oml.Timedate` object to produce another date.

Example 8-15 Using the `oml.Timedelta` Function

This example creates a time-based reference using the current date and time, and creates an `oml.Timedelta` object named `DELT1` to determine a past and future dates.

```

import oml
import pandas as pd
import numpy as np

import datetime
from datetime import datetime, timezone, timedelta
today = datetime.now()
print('today:', today)

delt1 = timedelta(days=1, hours=2, seconds=5)
print('delt1:', delt1)

dat = pd.DataFrame({'datetime': [today], 'timedelta': [delt1]})
DAT = oml.push(dat, dbtypes = ['TIMESTAMP', 'INTERVAL DAY TO SECOND'])

TODAY = DAT['datetime']
DELT1 = DAT['timedelta']

print('TODAY:', today)
print('DELT1:', DELT1)
past_date1 = TODAY - DELT1
print('past date 1:', past_date1)

past_date2 = TODAY - (DELT1 * 3)

```

```
print('past date 2:', past_date2)

future_date1 = TODAY + DELT1
print('future date 1:', future_date1)

future_date2 = TODAY + (DELT1 *3)
print('future date 2:', future_date2)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> import numpy as np

>>> import datetime
>>> from datetime import datetime, timezone, timedelta
>>> today = datetime.now()
>>> print('today:', today)
>>> today: 2022-12-27 06:00:14.555899

>>> delt1 = timedelta(days=1, hours=2, seconds=5)
>>> print('delt1:', delt1)
>>> delt1: 1 day, 2:00:05

>>> dat = pd.DataFrame({'datetime': [today], 'timedelta': [delt1]})
>>> DAT = oml.push(dat, dbtypes = ['TIMESTAMP', 'INTERVAL DAY TO SECOND'])

>>> TODAY = DAT['datetime']
>>> DELT1 = DAT['timedelta']

>>> print('TODAY:', today)
>>> TODAY: 2022-12-27 06:00:14.555899
>>> print('DELT1:', DELT1)
>>> DELT1: [datetime.timedelta(days=1, seconds=7205)]
>>> past_date1 = TODAY - DELT1
>>> print('past date 1:', past_date1)
>>> past date 1: [datetime.datetime(2022, 12, 26, 4, 0, 9, 555899)]
>>> past_date2 = TODAY - (DELT1 *3)
>>> print('past date 2:', past_date2)
>>> past date 2: [datetime.datetime(2022, 12, 23, 23, 59, 59, 555899)]

>>> future_date1 = TODAY + DELT1
>>> print('future date 1:', future_date1)
>>> future date 1: [datetime.datetime(2022, 12, 28, 8, 0, 19, 555899)]

>>> future_date2 = TODAY + (DELT1 *3)
>>> print('future date 2:', future_date2)
>>> future date 2: [datetime.datetime(2022, 12, 30, 12, 0, 29, 555899)]
```

oml.Integer

The `oml.Integer` class represents the integer data type.

Example 8-16 Using the oml.Integer Function

This example creates an `oml.Integer` object named `INTEGER1` to represent the integer data type.

```
import oml
import pandas as pd
integer1 = oml.push(pd.DataFrame({'INTEGER': [0, -12, 1234, 40, 95]}),
dbtypes = "NUMBER(*, 0)")
integer1
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> integer1 = oml.push(pd.DataFrame({'INTEGER': [0, -12, 1234, 40, 95]}),
dbtypes = "NUMBER(*, 0)")
>>> integer1
      INTEGER
0          0
1         -12
2        1234
3          40
4          95
```

Compare two `oml.Datetime` columns in a proxy object using standard arithmetic comparison operators.

Example 8-17 Using value comparison Function

This example compares two `oml.Datetime` columns in a proxy object using standard arithmetic comparison operators.

```
d1 = datetime(2005, 7, 14, 5, 10, 30)
d2 = datetime(2004, 6, 30, 1, 22, 46)
d3 = datetime(2003, 12, 10, 12, 50, 25)
d4 = datetime(2002, 3, 20, 20, 42, 59)

d3 = pd.DataFrame({'X': [d1, d2], 'Y': [d3,
d4]})
D3 = oml.push(d3, dbtypes = ['TIMESTAMP', 'TIMESTAMP'])

print(D3)
D4 = D3['X']
D5 = D3['Y']

print("D4:", D4)
print("D4 type:", type(D4))

print("D5:", D5)
print("D5 type:", type(D5))

print(D4 == D5)
```

```
print(D4 != D5)

print(D4 > D5)

print(D4 >= D5)

print(D4 < D5)

print(D4 <= D5)

print("max:", D3['X'].max())
print("min:", D3['Y'].min())
```

Listing for This Example

```
>>> d1 = datetime(2005, 7, 14, 5, 10, 30)
>>> d2 = datetime(2004, 6, 30, 1, 22, 46)
>>> d3 = datetime(2003, 12, 10, 12, 50, 25)
>>> d4 = datetime(2002, 3, 20, 20, 42, 59)

>>> d3 = pd.DataFrame({'X': [d1, d2], 'Y': [d3,
d4]})
>>> D3 = oml.push(d3, dbtypes = ['TIMESTAMP', 'TIMESTAMP'])

>>> print(D3)
>>>
           X                               Y
>>> 0 2005-07-14 05:10:30 2003-12-10 12:50:25
>>> 1 2004-06-30 01:22:46 2002-03-20 20:42:59
>>> D4 = D3['X']
>>> D5 = D3['Y']

>>> print("D4:", D4)
>>> print("D4 type:", type(D4))
>>> D4: [datetime.datetime(2005, 7, 14, 5, 10, 30), datetime.datetime(2004,
6, 30, 1, 22, 46)]
>>> D4 type: <class 'oml.core.datetime.Datetime'>

>>> print("D5:", D5)
>>> print("D5 type:", type(D5))
>>> D5: [datetime.datetime(2003, 12, 10, 12, 50, 25), datetime.datetime(2002,
3, 20, 20, 42, 59)]
>>> D5 type: <class 'oml.core.datetime.Datetime'>

>>> print(D4 == D5)
>>> [False, False]

>>> print(D4 != D5)
>>> [True, True]

>>> print(D4 > D5)
>>> [True, True]

>>> print(D4 >= D5)
```

```
>>> [True, True]

>>> print(D4 < D5)
>>> [False, False]

>>> print(D4 <= D5)
>>> [False,
False]

>>> [True, True] [True, True] [True, True] [False,
False] [False,

>>> print("max:", D3['X'].max())
>>> max: 2005-07-14 05:10:30
>>> print("min:", D3['Y'].min())
>>> min: 2002-03-20 20:42:59
```

Value Replacement

This function updates the elements of an `oml.Datetime` object, such as year, month, and day.

Example 8-18 Using value replacement function

```
D4 = D3['X'].replace(year=2000)
print("D4:", D4)

D5 = D3['X'].replace(month=11)
print("D5:", D5)

D6 = D3['X'].replace(day=6)
print("D6:", D6)
```

Listing for This Example

```
>>> D4 = D3['X'].replace(year=2000)
>>> print("D4:", D4)
>>> D4: [datetime.datetime(2000, 7, 14, 5, 10, 30), datetime.datetime(2000,
6, 30, 1, 22, 46)]

>>> D5 = D3['X'].replace(month=11)
>>> print("D5:", D5)
>>> D5: [datetime.datetime(2005, 11, 14, 5, 10, 30), datetime.datetime(2004,
11, 30, 1, 22, 46)]

>>> D6 = D3['X'].replace(day=6)
>>> print("D6:", D6)
>>> D6: [datetime.datetime(2005, 7, 6, 5, 10, 30), datetime.datetime(2004, 6,
6, 1, 22, 46)]
```

8.3 Render Graphics

OML4Py provides functions for rendering graphical displays of data.

The `oml.boxplot` and `oml.hist` functions compute the statistics necessary to generate box and whisker plots or histograms in-database for scalability and performance.

OML4Py uses the `matplotlib` library to render the output. You can use methods of `matplotlib.pyplot` to customize the created images and `matplotlib.pyplot.show` to show the images. By default, rendered graphics have the same properties as those stored in `matplotlib.rcParams`.

For the parameters of the `oml.boxplot` and `oml.hist` functions, invoke `help(oml.boxplot)` or `help(oml.hist)`, or see [Oracle Machine Learning for Python API Reference](#).

Generate a Box Plot

Use the `oml.boxplot` function to generate a box and whisker plot for every column of `x` or for every column object in `x`.

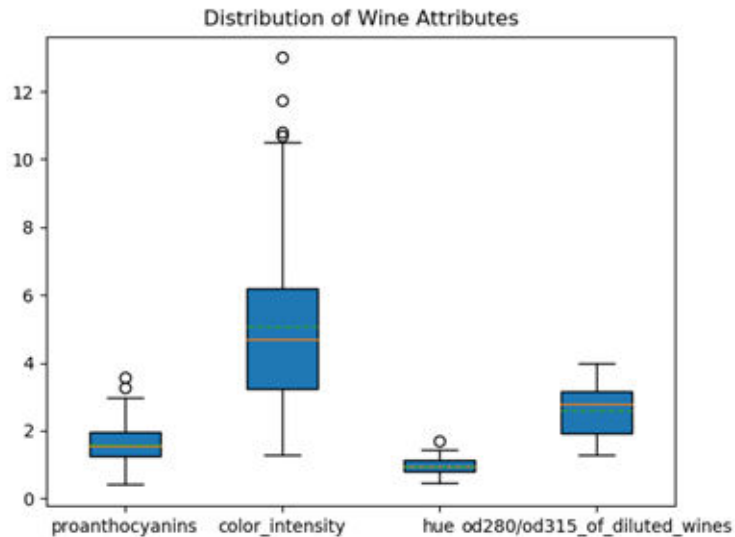
Example 8-19 Using the `oml.boxplot` Function

This example first loads the wine data set from `sklearn` and creates the `pandas.DataFrame` object `wine_data`. It then creates a temporary database table, with its corresponding proxy `oml.DataFrame` object `oml_wine`, from `wine_data`. It draws a box and whisker plot on every column with the index ranging from 8 to 12 (not including 12) in `oml_wine`. The arguments `showmeans` and `meanline` are set to `True` to show the arithmetic means and to render the mean as a line spanning the full width of the box. The argument `patch_artist` is set to `True` to have the boxes drawn with Patch artists.

```
import oml
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets

wine = datasets.load_wine()
wine_data = pd.DataFrame(wine.data, columns = wine.feature_names)
oml_wine = oml.push(wine_data)
oml.graphics.boxplot(oml_wine[:,8:12], showmeans=True,
                    meanline=True, patch_artist=True,
                    labels=oml_wine.columns[8:12])
plt.title('Distribution of Wine Attributes')
plt.show()
```

The output of the example is the following.



The image shows a box and whisker plot for each of the four columns of the wine data set: Proanthocyanins, Color intensity, Hue, and OD280/OD315 of diluted wines. The boxes extend from the lower to upper quartile values of the data, with a solid orange line at the median. The whiskers that extend from the box show the range of the data. The caps are the horizontal lines at the ends of the whiskers. Flier or outlier points are those past the ends of the whiskers. The mean is shown as a green dotted line spanning the width of the each box.

Generate a Histogram

Use the `oml.hist` function to compute and draw a histogram for every data set column contained in `x`.

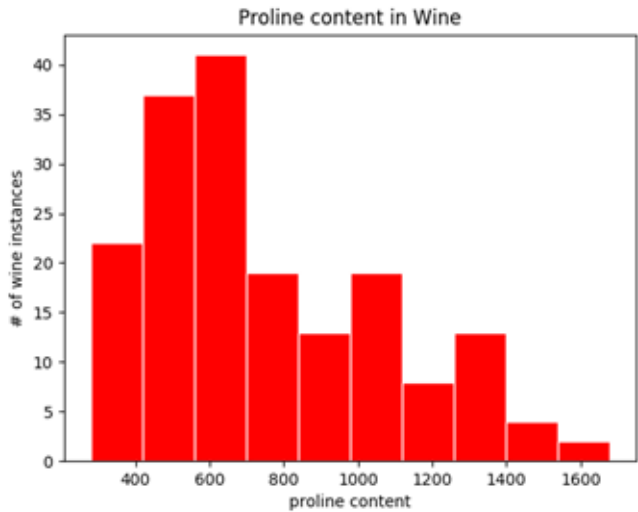
Example 8-20 Using the `oml.hist` Function

This example first loads the wine data set from `sklearn` and creates the `pandas.DataFrame` object `wine_data`. It then creates a temporary database table, with its corresponding proxy `oml.DataFrame` object `oml_wine`, from `wine_data`. Next it draws a histogram on the `proline` column of `oml_wine`. The argument `bins` specifies generating ten equal-width bins. Argument `color` specifies filling the bars with the color purple. Arguments `linestyle` and `edgecolor` are set to draw the bar edges as solid lines in pink.

```
import oml
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine

wine = load_wine()
wine_data = pd.DataFrame(wine.data, columns = wine.feature_names)
oml_wine = oml.push(wine_data)
oml.graphics.hist(oml_wine['proline'], bins=10, color='red',
                 linestyle='solid', edgecolor='white')
plt.title('Proline content in Wine')
plt.xlabel('proline content')
plt.ylabel('# of wine instances')
plt.show()
```

The output of the example is the following.



The image shows a traditional bar-type histogram for the Proline column of the wine data set. The range of proline values is divided into 10 bins of equal size. The height of the rectangular bar for each bin indicates the number of wine instances in each bin. The bars are red with solid white edges.

9

OML4Py Classes That Provide Access to In-Database Machine Learning Algorithms

OML4Py has classes that provide access to in-database Oracle Machine Learning algorithms.

These classes are described in the following topics.

- [About Machine Learning Classes and Algorithms](#)
These classes provide access to in-database machine learning algorithms.
- [About Model Settings](#)
You can specify settings that affect the characteristics of a model.
- [Shared Settings](#)
These settings are common to all of the OML4Py machine learning classes.
- [Export Oracle Machine Learning for Python Models](#)
You can export an `oml` model from Python and then score it in SQL.
- [Automatic Data Preparation](#)
Oracle Machine Learning for Python supports Automatic Data Preparation (ADP) and user-directed general data preparation.
- [Model Explainability](#)
Use the OML4Py Explainability module to identify the important features that impact a trained model's predictions.
- [Attribute Importance](#)
The `oml.ai` class computes the relative attribute importance, which ranks attributes according to their significance in predicting a classification or regression target.
- [Association Rules](#)
The `oml.ar` class implements the Apriori algorithm to find frequent itemsets and association rules, all as part of an association model object.
- [Decision Tree](#)
The `oml.dt` class uses the Decision Tree algorithm for classification.
- [Expectation Maximization](#)
The `oml.em` class uses the Expectation Maximization (EM) algorithm to create a clustering model.
- [Explicit Semantic Analysis](#)
The `oml.esa` class extracts text-based features from a corpus of documents and performs document similarity comparisons.
- [Generalized Linear Model](#)
The `oml.glm` class builds a Generalized Linear Model (GLM) model.
- [k-Means](#)
The `oml.km` class uses the *k*-Means (KM) algorithm, which is a hierarchical, distance-based clustering algorithm that partitions data into a specified number of clusters.
- [Naive Bayes](#)
The `oml.nb` class creates a Naive Bayes (NB) model for classification.

- **Neural Network**
The `oml.nn` class creates a Neural Network (NN) model for classification and regression.
- **Random Forest**
The `oml.rf` class creates a Random Forest (RF) model that provides an ensemble learning technique for classification.
- **Singular Value Decomposition**
Use the `oml.svd` class to build a model for feature extraction.
- **Support Vector Machine**
The `oml.svm` class creates a Support Vector Machine (SVM) model for classification, regression, or anomaly detection.
- **Non-Negative Matrix Factorization**
The `oml.nmf` class creates a Non-Negative Matrix Factorization (NMF) model for feature extraction.
- **Exponential Smoothing Method**
The `oml.esm` function uses the Exponential Smoothing Method (ESM) algorithm to create a time series model.
- **XGBoost**
The `oml.xgb` class supports the in-database scalable gradient tree boosting algorithm for both classification, regression specifications, ranking models, and survival models. It makes available the open source gradient boosting framework. It prepares the categorical encoding and missing value replacement from the OML infrastructure, calls the in-database XGBoost, builds and persists a model as a first-class database model object, and supports using the model for prediction.

9.1 About Machine Learning Classes and Algorithms

These classes provide access to in-database machine learning algorithms.

Algorithm Classes

Class	Algorithm	Function of Algorithm	Description
<code>oml.ai</code>	Minimum Description Length	Attribute importance for classification or regression	Ranks attributes according to their importance in predicting a target.
<code>oml.ar</code>	Apriori	Association rules	Performs market basket analysis by identifying co-occurring items (frequent itemsets) within a set.
<code>oml.dt</code>	Decision Tree	Classification	Extracts predictive information in the form of human-understandable rules. The rules are if-then-else expressions; they explain the decisions that lead to the prediction.
<code>oml.em</code>	Expectation Maximization	Clustering	Performs probabilistic clustering based on a density estimation algorithm.
<code>oml.esa</code>	Explicit Semantic Analysis	Feature extraction	Extracts text-based features from a corpus of documents. Performs document similarity comparisons.
<code>oml.glm</code>	Generalized Linear Model	Classification Regression	Implements logistic regression for classification of binary targets and linear regression for continuous targets.

Class	Algorithm	Function of Algorithm	Description
<code>oml.km</code>	<i>k</i> -Means	Clustering	Uses unsupervised learning to group data based on similarity into a predetermined number of clusters.
<code>oml.nb</code>	Naive Bayes	Classification	Makes predictions by deriving the probability of a prediction from the underlying evidence, as observed in the data.
<code>oml.nn</code>	Neural Network	Classification Regression	Learns from examples and tunes the weights of the connections among the neurons during the learning process.
<code>oml.rf</code>	Random Forest	Classification	Provides an ensemble learning technique for classification of data.
<code>oml.svd</code>	Singular Value Decomposition	Feature extraction	Performs orthogonal linear transformations that capture the underlying variance of the data by decomposing a rectangular matrix into three matrices.
<code>oml.svm</code>	Support Vector Machine	Anomaly detection Classification Regression	Builds a model that is a profile of a class, which, when the model is applied, identifies cases that are somehow different from that profile.
<code>oml.nmf</code>	Non-Negative Matrix Factorization	Clustering	A state of the art feature extraction algorithm used when there are many attributes and the attributes are ambiguous or have weak predictability.
<code>oml.xgb</code>	XGBoost	Classification Regression	Can be used as a stand-alone predictor or incorporate it into real-world production pipelines for a wide range of problems such as ad click-through rate prediction, hazard risk prediction, web text classification, and so on.

Repeatable Results

You can use the `case_id` parameter in the `fit` method of the OML4Py machine learning algorithm classes to achieve repeatable sampling, data splits (train and held aside), and random data shuffling.

Persisting Models

In-database models created through the OML4Py API exist as temporary objects that are dropped when the database connection ends unless you take one of the following actions:

- Save a default-named model object in a datastore, as in the following example:

```
regr2 = oml.glm("regression")
oml.ds.save(regr2, 'regression2')
```

- Use the `model_name` parameter in the `fit` function when building the model, as in the following example:

```
regr2 = regr2.fit(X, y, model_name = 'regression2')
```

- Change the name of an existing model using the `model_name` function of the model, as in the following example:

```
regr2(model_name = 'myRegression2')
```

To drop a persistent named model, use the `oml.drop` function.

Creating a Model from an Existing In-Database Model

You can create an OML4Py model as a proxy object for an existing in-database machine learning model. The in-database model could have been created through OML4Py, OML4SQL, or OML4R. To do so, when creating the OML4Py, specify the name of the existing model and, optionally, the name of the owner of the model, as in the following example.

```
ar_mod = oml.ar(model_name = 'existing_ar_model', model_owner = 'SH',  
**setting)
```

An OML4Py model created this way persists until you drop it with the `oml.drop` function.

Scoring New Data with a Model

For most of the OML4Py machine learning classes, you can use the `predict` and `predict_proba` methods of the model object to score new data.

For in-database models, you can use the SQL `PREDICTION` function on model proxy objects, which scores directly in the database. You can use in-database models directly from SQL if you prepare the data properly. For open source models, you can use Embedded Python Execution and enable data-parallel execution for performance and scalability.

Deploying Models Through a REST API

The [REST API for Oracle Machine Learning Services](#) provides REST endpoints hosted on an Oracle Autonomous Database instance. These endpoints allow you to store OML models along with their metadata, and to create scoring endpoints for the models.

9.2 About Model Settings

You can specify settings that affect the characteristics of a model.

Some settings are general, some are specific to an Oracle Machine Learning function, and some are specific to an algorithm.

All settings have default values. If you want to override one or more of the settings for a model, then you must specify the settings with the `**params` parameter when instantiating the model or later by using the `set_params` method of the model.

For the `_init_` method, the argument can be key-value pairs or a `dict`. Each list element's name and value refer to a machine learning algorithm parameter setting name and value, respectively. The setting value must be numeric or a string.

The argument for the `**params` parameter of the `set_params` method is a `dict` object mapping a `str` to a `str`. The key should be the name of the setting, and the value should be the new setting.

Example 9-1 Specifying Model Settings

This example shows the creation of an Expectation Maximization (EM) model and the changing of a setting. For the complete code of the EM model example, see [Example 9-10](#).

```
# Specify settings.
setting = {'emcs_num_iterations': 100}
# Create an EM model object
em_mod = em(n_clusters = 2, **setting)

# Intervening code not shown.

# Change the random seed and refit the model.
em_mod.set_params(EMCS_RANDOM_SEED = '5').fit(train_dat)
```

9.3 Shared Settings

These settings are common to all of the OML4Py machine learning classes.

The following table lists the settings that are shared by all OML4Py models.

Table 9-1 Shared Model Settings

Setting Name	Setting Value	Description
ODMS_DETAILS	ODMS_ENABLE ODMS_DISABLE	Helps to control model size in the database. Model details can consume significant disk space, especially for partitioned models. The default value is ODMS_ENABLE. If the setting value is ODMS_ENABLE, then model detail tables and views are created along with the model. You can query the model details using SQL. If the value is ODMS_DISABLE, then model detail tables are not created and tables relevant to model details are also not created. The reduction in the space depends on the algorithm. Model size reduction can be on the order of 10x .
ODMS_MAX_PARTITIONS	1 < value <= 1000000	Controls the maximum number of partitions allowed for a partitioned model. The default is 1000.
ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO ODMS_MISSING_VALUE_MEAN_MODE ODMS_MISSING_VALUE_DELETE_ROW	Indicates how to treat missing values in the training data. This setting does not affect the scoring data. The default value is ODMS_MISSING_VALUE_AUTO. ODMS_MISSING_VALUE_MEAN_MODE replaces missing values with the mean (numeric attributes) or the mode (categorical attributes) both at build time and apply time where appropriate. ODMS_MISSING_VALUE_AUTO performs different strategies for different algorithms. When ODMS_MISSING_VALUE_TREATMENT is set to ODMS_MISSING_VALUE_DELETE_ROW, the rows in the training data that contain missing values are deleted. However, if you want to replicate this missing value treatment in the scoring data, then you must perform the transformation explicitly. The value ODMS_MISSING_VALUE_DELETE_ROW is applicable to all algorithms.

Table 9-1 (Cont.) Shared Model Settings

Setting Name	Setting Value	Description
ODMS_PARTITION_BUILD_TYPE	ODMS_PARTITION_BUILD_INTRANTRA ODMS_PARTITION_BUILD_INTERNTER ODMS_PARTITION_BUILD_HYBRID	Controls the parallel building of partitioned models. ODMS_PARTITION_BUILD_INTRANTRA builds each partition in parallel using all slaves. ODMS_PARTITION_BUILD_INTERNTER builds each partition entirely in a single slave, but multiple partitions may be built at the same time because multiple slaves are active. ODMS_PARTITION_BUILD_HYBRID combines the other two types and is recommended for most situations to adapt to dynamic environments. This is the default value.
ODMS_PARTITION_COLUMNS	Comma separated list of machine learning attributes	Requests the building of a partitioned model. The setting value is a comma-separated list of the machine learning attributes to be used to determine the in-list partition key values. These attributes are taken from the input columns, unless an XFORM_LIST parameter is passed to the model. If XFORM_LIST parameter is passed to the model, then the attributes are taken from the attributes produced by these transformations.
ODMS_TABLESPACE_NAME	<i>tablespace_name</i>	Specifies the tablespace in which to store the model. If you explicitly set this to the name of a tablespace (for which you have sufficient quota), then the specified tablespace storage creates the resulting model content. If you do not provide this setting, then the your default tablespace creates the resulting model content.
ODMS_SAMPLE_SIZE	0 < value	Determines how many rows to sample (approximately). You can use this setting only if ODMS_SAMPLING is enabled. The default value is system determined.
ODMS_SAMPLING	ODMS_SAMPLING_ENABLE ODMS_SAMPLING_DISABLE	Allows the user to request sampling of the build data. The default is ODMS_SAMPLING_DISABLE.
ODMS_TEXT_MAX_FEATURES	1 <= value	The maximum number of distinct features, across all text attributes, to use from a document set passed to the model. The default is 3000. An oml.esa model has the default value of 300000.
ODMS_TEXT_MIN_DOCUMENTS	Non-negative value	This text processing setting controls how many documents a token needs to appear in to be used as a feature. The default is 1. An oml.esa model has the default value of 3.
ODMS_TEXT_POLICY_NAME	The name of an Oracle Text POLICY created using CTX_DDL.CREATE_POLICY.	Affects how individual tokens are extracted from unstructured text. For details about CTX_DDL.CREATE_POLICY, see <i>Oracle Text Reference</i> .
PREP_AUTO	PREP_AUTO_ON PREP_AUTO_OFF	This data preparation setting enables fully automated data preparation. The default is PREP_AUTO_ON.

Table 9-1 (Cont.) Shared Model Settings

Setting Name	Setting Value	Description
PREP_SCALE_2DNUM	pPREP_SCALE_STDDEV PREP_SCALE_RANGE	This data preparation setting enables scaling data preparation for two-dimensional numeric columns. PREP_AUTO must be OFF for this setting to take effect. The following are the possible values: PREP_SCALE_STDDEV: A request to divide the column values by the standard deviation of the column and is often provided together with PREP_SHIFT_MEAN to yield z-score normalization. PREP_SCALE_RANGE: A request to divide the column values by the range of values and is often provided together with PREP_SHIFT_MIN to yield a range of [0,1].
PREP_SCALE_NNUM	PREP_SCALE_MAXABS	This data preparation setting enables scaling data preparation for nested numeric columns. PREP_AUTO must be OFF for this setting to take effect. If specified, then the valid value for this setting is PREP_SCALE_MAXABS, which yields data in the range of [-1,1].
PREP_SHIFT_2DNUM	PREP_SHIFT_MEAN PREP_SHIFT_MIN	This data preparation setting enables centering data preparation for two-dimensional numeric columns. PREP_AUTO must be OFF for this setting to take effect. The following are the possible values: PREP_SHIFT_MEAN: Results in subtracting the average of the column from each value. PREP_SHIFT_MIN: Results in subtracting the minimum of the column from each value.

9.4 Export Oracle Machine Learning for Python Models

You can export an `oml` model from Python and then score it in SQL.

Export a Model

With the `export_sermodel` function of an OML4Py algorithm model, you can export the model in a serialized format. You can then score that model in SQL. To save a model to a permanent table, you must pass in a name for the new table. If the model is partitioned, then you can optionally select an individual partition to export; otherwise all partitions are exported.



Note:

Any data transformations you apply to the data for model building you must also apply to the data for scoring with the imported model.

Example 9-2 Export a Trained `oml.svm` Model to a Database Table

This example creates the `x` and `y` variables using the iris data set. It then creates the persistent database table `IRIS` and the `oml.DataFrame` object `oml_iris` as a proxy for the table.

This example preprocesses the `iris` data set and splits the data set into training data and test data. It then fits an `oml.svm` model according to the training data of the data set, and saves the fitted model in a serialized format to a new table named `svm_sermod` in the database.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
    oml.drop('IRIS_TEST_DATA')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

df = oml.sync(table = "IRIS").pull()

# Add a case identifier column.
df.insert(0, 'ID', range(0, len(df)))

# Create training data and test data.
IRIS_TMP = oml.push(df).split()
train_x = IRIS_TMP[0].drop('Species')
train_y = IRIS_TMP[0]['Species']
test_dat = IRIS_TMP[1]

# Create the iris_test_data database table.
oml_test_dat = oml.create(test_dat.pull(), table = "IRIS_TEST_DATA")

# Create an oml SVM model object.
svm_mod = oml.svm('classification',
                 svms_kernel_function =
                 'dbms_data_mining.svms_linear')

# Fit the SVM model with the training data.
svm_mod = svm_mod.fit(train_x, train_y, case_id = 'ID')

# Export the oml.svm model to a new table named 'svm_sermod'
# in the database.
svm_export = svm_mod.export_sermodel(table='svm_sermod')
type(svm_export)

# Show the first 10 characters of the BLOB content from the
```

```
# model export.  
svm_export.pull()[0][1:10]
```

Listing for This Example

```
>>> import oml  
>>> import pandas as pd  
>>> from sklearn import datasets  
>>>  
>>> # Load the iris data set and create a pandas.DataFrame for it.  
... iris = datasets.load_iris()  
>>> x = pd.DataFrame(iris.data,  
...                  columns = ['Sepal_Length', 'Sepal_Width',  
...                            'Petal_Length', 'Petal_Width'])  
>>> y = pd.DataFrame(list(map(lambda x:  
...                          {0: 'setosa', 1: 'versicolor',  
...                          2: 'virginica'}[x], iris.target)),  
...                  columns = ['Species'])  
>>>  
>>> try:  
...     oml.drop('IRIS')  
...     oml.drop('IRIS_TEST_DATA')  
...except:  
...     pass  
>>> # Create the IRIS database table.  
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')  
>>>  
>>> df = oml.sync(table = "IRIS").pull()  
>>>  
>>> # Add a case identifier column.  
... df.insert(0, 'ID', range(0, len(df)))  
>>>  
>>> # Create training data and test data.  
... IRIS_TMP = oml.push(df).split()  
>>> train_x = IRIS_TMP[0].drop('Species')  
>>> train_y = IRIS_TMP[0]['Species']  
>>> test_dat = IRIS_TMP[1]  
>>>  
>>> # Create the iris_test_data database table.  
... oml_test_dat = oml.create(test_dat.pull(), table = "IRIS_TEST_DATA")  
>>>  
>>> # Create an oml SVM model object.  
... svm_mod = oml.svm('classification',  
...                   svms_kernel_function =  
...                   'dbms_data_mining.svms_linear')  
>>>  
>>> # Fit the SVM model with the training data.  
... svm_mod = svm_mod.fit(train_x, train_y, case_id='ID')  
>>>  
>>> # Export the oml.svm model to a new table named 'svm_sermod'  
... # in the database.  
... svm_export = svm_mod.export_sermodel(table='svm_sermod')  
>>> type(svm_export)  
<class 'oml.core.bytes.Bytes'>  
>>>
```

```
>>> # Show the first 10 characters of the BLOB content from the
... # model export.
... svm_export.pull()[0][1:10]
b'\xff\xfc|\x00\x00\x02\x9c\x00\x00'
```

Import a Model

In SQL, you can import the serialized format of an OML4Py model into an Oracle Machine Learning for SQL model with the `DBMS_DATA_MINING.IMPORT_SERMODEL` procedure. To that procedure, you pass the BLOB content from the table to which the model was exported and the name of the model to be created. The import procedure provides the ability to score the model. It does not create model views or tables that are needed for querying model details. You can use the SQL function `PREDICTION` to apply the imported model to the test data and get the prediction results.

Example 9-3 Import a Serialized SVM Model as an OML4SQL Model in SQL

This example retrieves the serialized content of the SVM classification model from the `svm_sermod` table. It uses the `IMPORT_SERMODEL` procedure to create a model named `my_iris_svm_classifier` with the content from the table. It also predicts test data saved in the `iris_test_data` table with the newly imported model `my_iris_svm_classifier`, and compares the prediction results with the target classes.

```
-- After starting SQL*Plus as the OML4Py user.
-- Import the model from the serialized content.

DECLARE
    v_blob blob;

BEGIN
    SELECT SERVAL INTO v_blob FROM "svm_sermod";
    dbms_data_mining.import_sermodel(v_blob, 'my_iris_svm_classifier');
END;
/

-- Set the output column format.
column TARGET_SPECIES format a15
column PREDICT_SPECIES format a15

-- Make predictions and display cases where mod(ID,3) equals 0.
SELECT ID, "Species" AS TARGET_SPECIES,
       PREDICTION(my_iris_svm_classifier USING "Sepal_Length", "Sepal_Width",
                 "Petal_Length", "Petal_Width")
       AS PREDICT_SPECIES
FROM "IRIS_TEST_DATA" WHERE MOD(ID,3) = 0;

-- Drop the imported model
BEGIN
    DBMS_DATA_MINING.DROP_MODEL(model_name => 'my_iris_svm_classifier');
END;
/
```

The prediction produces the following results.

```
ID TARGET_SPECIES PREDICT_SPECIES
-- -----
```



```

0 setosa      setosa
24 setosa      setosa
27 setosa      setosa
33 setosa      setosa
36 setosa      setosa
39 setosa      setosa
48 setosa      setosa
54 versicolor versicolor
57 versicolor versicolor
93 versicolor versicolor
114 virginica  virginica
120 virginica  virginica
132 virginica  virginica
13 rows selected.

```

9.5 Automatic Data Preparation

Oracle Machine Learning for Python supports Automatic Data Preparation (ADP) and user-directed general data preparation.

The `PREP_*` settings enable you to request fully automated (ADP) or manual data preparation. By default, ADP is enabled (`PREP_AUTO_ON`). When performed manually, data preparation requirements of each algorithm must be addressed.

When you enable ADP, the model uses heuristics to transform the build data according to the requirements of the algorithm. Instead of ADP, you can request that the data be shifted and/or scaled with the `PREP_SCALE_*` and `PREP_SHIFT_*` settings. The transformation instructions are stored with the model and reused whenever the model is applied. The model settings can be viewed in `USER_MINING_MODEL_SETTINGS`.

PREP_* Settings

The values for the `PREP_*` settings are described in the following table.

Table 9-2 title

Setting Name	Setting Value	Description
PREP_AUTO	PREP_AUTO_ON	This setting enables fully automated data preparation. The default is <code>PREP_AUTO_ON</code> .
	PREP_AUTO_OFF	
PREP_SCALE_2DNUM	PREP_SCALE_STDDEV	This setting enables scaling data preparation for two-dimensional numeric columns. <code>PREP_AUTO</code> must be <code>OFF</code> for this setting to take effect. The following are the possible values. <code>PREP_SCALE_STDDEV</code> : A request to divide the column values by the standard deviation of the column and is often provided together with <code>PREP_SHIFT_MEAN</code> to yield z-score normalization. <code>PREP_SCALE_RANGE</code> : A request to divide the column values by the range of values and is often provided together with <code>PREP_SHIFT_MIN</code> to yield a range of [0,1].
	PREP_SCALE_RANGE	

Table 9-2 (Cont.) title

Setting Name	Setting Value	Description
PREP_SCALE_NNUM	PREP_SCALE_MAXABS	This setting enables scaling data preparation for nested numeric columns. <code>PREP_AUTO</code> must be <code>OFF</code> for this setting to take effect. If specified, then the valid value for this setting is <code>PREP_SCALE_MAXABS</code> , which yields data in the range of <code>[-1,1]</code> .
PREP_SHIFT_2DNUM	PREP_SHIFT_MEAN PREP_SHIFT_MIN	This setting enables centering data preparation for two-dimensional numeric columns. <code>PREP_AUTO</code> must be <code>OFF</code> for this setting to take effect. The following are the possible values: <code>PREP_SHIFT_MEAN</code> : Results in subtracting the average of the column from each value. <code>PREP_SHIFT_MIN</code> : Results in subtracting the minimum of the column from each value.

 **See Also:**

- [About Model Settings](#)
- [Shared Settings](#)

9.6 Model Explainability

Use the OML4Py Explainability module to identify the important features that impact a trained model's predictions.

Machine Learning Explainability (MLX) is the process of explaining and interpreting machine learning models. The OML MLX Python module supports the ability to help better understand a model's behavior and why it makes its predictions. MLX currently provides model-agnostic explanations for classification and regression tasks where explanations treat the ML model as a black-box, instead of using properties from the model to guide the explanation.

The global feature importance explainer object is the interface to the MLX permutation importance explainer. The global feature importance explainer identifies the most important features for a given model and data set. The explainer is model-agnostic and currently supports tabular classification and regression data sets with both numerical and categorical features.

The algorithm estimates feature importance by evaluating the model's sensitivity to changes in a specific feature. Higher sensitivity suggests that the model places higher importance on that feature when making its predictions than on another feature with lower sensitivity.

For information on the `oml.GlobalFeatureImportance` class attributes and methods, call `help(oml.mlz.GlobalFeatureImportance)` or see Oracle Machine Learning for Python API Reference.

Example 9-4 Binary Classification

This example uses the Breast Cancer binary classification data set. Load the data set into the database and a unique case id column.

```
import oml
from oml.mlx import GlobalFeatureImportance
import pandas as pd
import numpy as np
from sklearn import datasets

bc_ds = datasets.load_breast_cancer()
bc_data = bc_ds.data.astype(float)
X = pd.DataFrame(bc_data, columns=bc_ds.feature_names)
y = pd.DataFrame(bc_ds.target, columns=['TARGET'])
row_id = pd.DataFrame(np.arange(bc_data.shape[0]),
                      columns=['CASE_ID'])
df = oml.create(pd.concat([X, y, row_id], axis=1),
               table='BreastCancer')
```

Split the data set into train and test variables.

```
train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
                      seed=32)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']
```

Train a Random Forest model.

```
model = oml.algo.rf(ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
"RF accuracy score = {:.2f}".format(model.score(X_test, y_test))
```

Create the MLX Global Feature Importance explainer, using the binary f1 metric.

```
gfi = GlobalFeatureImportance(mining_function='classification',
                              score_metric='f1', random_state=32,
                              parallel=4)
```

Run the explainer to generate the global feature importance. Here we construct an explanation using the train data set and then display the explanation.

```
explanation = gfi.explain(model, X, y, case_id='CASE_ID', n_iter=10)
explanation
```

Drop the BreastCancer table.

```
oml.drop('BreastCancer')
```

Listing for This Example

```
>>> import oml
>>> from oml.mlx import GlobalFeatureImportance
```

```
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets
>>>
>>> bc_ds = datasets.load_breast_cancer()
>>> bc_data = bc_ds.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns=bc_ds.feature_names)
>>> y = pd.DataFrame(bc_ds.target, columns=['TARGET'])
>>> row_id = pd.DataFrame(np.arange(bc_data.shape[0]),
...                       columns=['CASE_ID'])
>>> df = oml.create(pd.concat([X, y, row_id], axis=1),
...               table='BreastCancer')
>>>
>>> train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
...                       seed=32)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
>>>
>>> model = oml.algo.rf(ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
...       "RF accuracy score = {:.2f}".format(model.score(X_test, y_test))
'RF accuracy score = 0.95'
>>>
>>> gfi = GlobalFeatureImportance(mining_function='classification',
...                               score_metric='f1', random_state=32,
...                               parallel=4)
>>>
>>> explanation = gfi.explain(model, X, y, case_id='CASE_ID', n_iter=10)
>>> explanation
Global Feature Importance:
[0] worst concave points: Value: 0.0263, Error: 0.0069
[1] worst perimeter: Value: 0.0077, Error: 0.0027
[2] worst radius: Value: 0.0076, Error: 0.0031
[3] worst area: Value: 0.0045, Error: 0.0037
[4] mean concave points: Value: 0.0034, Error: 0.0033
[5] worst texture: Value: 0.0017, Error: 0.0015
[6] area error: Value: 0.0012, Error: 0.0014
[7] worst concavity: Value: 0.0008, Error: 0.0008
[8] worst symmetry: Value: 0.0004, Error: 0.0007
[9] mean texture: Value: 0.0003, Error: 0.0007
[10] mean perimeter: Value: 0.0003, Error: 0.0015
[11] mean radius: Value: 0.0000, Error: 0.0000
[12] mean smoothness: Value: 0.0000, Error: 0.0000
[13] mean compactness: Value: 0.0000, Error: 0.0000
[14] mean concavity: Value: 0.0000, Error: 0.0000
[15] mean symmetry: Value: 0.0000, Error: 0.0000
[16] mean fractal dimension: Value: 0.0000, Error: 0.0000
[17] radius error: Value: 0.0000, Error: 0.0000
[18] texture error: Value: 0.0000, Error: 0.0000
[19] smoothness error: Value: 0.0000, Error: 0.0000
[20] compactness error: Value: 0.0000, Error: 0.0000
[21] concavity error: Value: 0.0000, Error: 0.0000
[22] concave points error: Value: 0.0000, Error: 0.0000
[23] symmetry error: Value: 0.0000, Error: 0.0000
[24] fractal dimension error: Value: 0.0000, Error: 0.0000
[25] worst compactness: Value: 0.0000, Error: 0.0000
[26] worst fractal dimension: Value: 0.0000, Error: 0.0000
```

```
[27] mean area: Value: -0.0001, Error: 0.0011
[28] worst smoothness: Value: -0.0003, Error: 0.0013

oml.drop('BreastCancer')
```

Example 9-5 Multi-Class Classification

This example uses the Iris multi-class classification data set. Load the data set into the database, adding a unique case id column.

```
import oml
from oml.mlx import GlobalFeatureImportance
import pandas as pd
import numpy as np
from sklearn import datasets

iris_ds = datasets.load_iris()
iris_data = iris_ds.data.astype(float)
X = pd.DataFrame(iris_data, columns=iris_ds.feature_names)
y = pd.DataFrame(iris_ds.target, columns=['TARGET'])
row_id = pd.DataFrame(np.arange(iris_data.shape[0]),
                      columns=['CASE_ID'])
df = oml.create(pd.concat([X, y, row_id], axis=1), table='Iris')
```

Split the data set into train and test variables.

```
train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
                      seed=32)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']
```

Train an SVM model.

```
model = oml.algo.svm(ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
"SVM accuracy score = {:.2f}".format(model.score(X_test, y_test))
```

Create the MLX Global Feature Importance explainer, using the `f1_weighted` metric.

```
gfi = GlobalFeatureImportance(mining_function='classification',
                              score_metric='f1_weighted',
                              random_state=32, parallel=4)
```

Run the explainer to generate the global feature importance. Here, we use the test data set. Display the explanation.

```
explanation = gfi.explain(model, X_test, y_test,
                        case_id='CASE_ID', n_iter=10)
explanation
```

Drop the Iris table.

```
oml.drop('Iris')
```

Listing for This Example

```
>>> import oml
>>> from oml.mlx import GlobalFeatureImportance
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets
>>>
>>> iris_ds = datasets.load_iris()
>>> iris_data = iris_ds.data.astype(float)
>>> X = pd.DataFrame(iris_data, columns=iris_ds.feature_names)
>>> y = pd.DataFrame(iris_ds.target, columns=['TARGET'])
>>> row_id = pd.DataFrame(np.arange(iris_data.shape[0]),
...                       columns=['CASE_ID'])
>>> df = oml.create(pd.concat([X, y, row_id], axis=1), table='Iris')
>>>
>>> train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
...                       seed=32)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
>>>
>>> model = oml.algo.svm(ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
>>> "SVM accuracy score = {:.2f}".format(model.score(X_test, y_test))
'SVM accuracy score = 0.94'
>>>
>>> gfi = GlobalFeatureImportance(mining_function='classification',
...                              score_metric='f1_weighted',
...                              random_state=32, parallel=4)
>>>
>>> explanation = gfi.explain(model, X_test, y_test,
...                           case_id='CASE_ID', n_iter=10)
>>> explanation
Global Feature Importance:
[0] petal length (cm): Value: 0.3462, Error: 0.0824
[1] petal width (cm): Value: 0.2417, Error: 0.0687
[2] sepal width (cm): Value: 0.0926, Error: 0.0452
[3] sepal length (cm): Value: 0.0253, Error: 0.0152

>>> oml.drop('Iris')
```

Example 9-6 Regression

This example uses the Boston regression data set. Load the data set into the database, adding a unique case id column.

```
import oml
from oml.mlx import GlobalFeatureImportance
import pandas as pd
import numpy as np
from sklearn import datasets

boston_ds = datasets.load_boston()
boston_data = boston_ds.data
X = pd.DataFrame(boston_data, columns=boston_ds.feature_names)
y = pd.DataFrame(boston_ds.target, columns=['TARGET'])
row_id = pd.DataFrame(np.arange(boston_data.shape[0]),
```

```
        columns=['CASE_ID'])
df = oml.create(pd.concat([X, y, row_id], axis=1), table='Boston')
```

Split the data set into train and test variables.

```
train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID', seed=32)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']
```

Train a Neural Network regression model.

```
model = oml.algo.nn(mining_function='regression',
                   ODS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
"NN R^2 score = {:.2f}".format(model.score(X_test, y_test))
```

Create the MLX Global Feature Importance explainer, using the r^2 metric.

```
gfi = GlobalFeatureImportance(mining_function='regression',
                              score_metric='r2', random_state=32,
                              parallel=4)
```

Run the explainer to generate the global feature importance. Here, we use the test data set. Display the explanation.

```
explanation = gfi.explain(model, df, 'TARGET',
                        case_id='CASE_ID', n_iter=10)
explanation
```

Drop the Boston table.

```
oml.drop('Boston')
```

Listing for This Example

```
>>> import oml
>>> from oml.mlx import GlobalFeatureImportance
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets
>>>
>>> boston_ds = datasets.load_boston()
>>> boston_data = boston_ds.data
>>> X = pd.DataFrame(boston_data, columns=boston_ds.feature_names)
>>> y = pd.DataFrame(boston_ds.target, columns=['TARGET'])
>>> row_id = pd.DataFrame(np.arange(boston_data.shape[0]),
...                       columns=['CASE_ID'])
>>> df = oml.create(pd.concat([X, y, row_id], axis=1), table='Boston')
>>>
>>> train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
...                       seed=32)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
```

```
>>>
>>> model = oml.algo.nn(mining_function='regression',
..                      ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
>>> "NN R^2 score = {:.2f}".format(model.score(X_test, y_test))
'NN R^2 score = 0.85'
>>>
>>> gfi = GlobalFeatureImportance(mining_function='regression',
...                               score_metric='r2', random_state=32,
...                               parallel=4)
>>>
>>> explanation = gfi.explain(model, df, 'TARGET',
...                            case_id='CASE_ID', n_iter=10)
>>> explanation
Global Feature Importance:
[0] LSTAT: Value: 0.7686, Error: 0.0513
[1] RM: Value: 0.5734, Error: 0.0475
[2] CRIM: Value: 0.5131, Error: 0.0345
[3] DIS: Value: 0.4170, Error: 0.0632
[4] NOX: Value: 0.2592, Error: 0.0206
[5] AGE: Value: 0.2083, Error: 0.0212
[6] RAD: Value: 0.1956, Error: 0.0188
[7] INDUS: Value: 0.1792, Error: 0.0199
[8] B: Value: 0.0982, Error: 0.0146
[9] PTRATIO: Value: 0.0822, Error: 0.0069
[10] TAX: Value: 0.0566, Error: 0.0139
[11] ZN: Value: 0.0397, Error: 0.0081
[12] CHAS: Value: 0.0125, Error: 0.0045

>>> oml.drop('Boston')
```

9.7 Attribute Importance

The `oml.ai` class computes the relative attribute importance, which ranks attributes according to their significance in predicting a classification or regression target.

The `oml.ai` class uses the Minimum Description Length (MDL) algorithm to calculate attribute importance. MDL assumes that the simplest, most compact representation of the data is the best and most probable explanation of the data.

You can use methods of the `oml.ai` class to compute the relative importance of predictor variables when predicting a response variable.



Note:

Oracle Machine Learning does not support the scoring operation for `oml.ai`.

The results of `oml.ai` are the attributes of the build data ranked according to their predictive influence on a specified target attribute. You can use the ranking and the measure of importance for selecting attributes.

For information on the `oml.ai` class attributes and methods, invoke `help(oml.ai)` or see Oracle Machine Learning for Python API Reference.

 **See Also:**

- [About Model Settings](#)
- [Shared Settings](#)

Example 9-7 Ranking Attribute Significance with oml.ai

This example creates the `x` and `y` variables using the iris data set. It then creates the persistent database table `IRIS` and the `oml.DataFrame` object `oml_iris` as a proxy for the table.

This example demonstrates the use of various methods of the `oml.ai` class.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Specify settings.
setting = {'ODMS_SAMPLING': 'ODMS_SAMPLING_DISABLE'}

# Create an AI model object.
ai_mod = oml.ai(**setting)

# Fit the AI model according to the training data and parameter
# settings.
ai_mod = ai_mod.fit(train_x, train_y)

# Show the model details.
ai_mod
```

Listing for This Example

```

>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                            'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
>>>
>>> # Specify settings.
... setting = {'ODMS_SAMPLING': 'ODMS_SAMPLING_DISABLE'}
>>>
>>> # Create an AI model object.
... ai_mod = oml.ai(**setting)
>>>
>>> # Fit the AI model according to the training data and parameter
... # settings.
>>> ai_mod = ai_mod.fit(train_x, train_y)
>>>
>>> # Show the model details.
... ai_mod

```

Algorithm Name: Attribute Importance

Mining Function: ATTRIBUTE_IMPORTANCE

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_AI_MDL
1	ODMS_DETAILS	ODMS_ENABLE
2	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
3	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
4	PREP_AUTO	ON

Global Statistics:

```

    attribute name      attribute value
0      NUM_ROWS      104

```

```

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

```

```

Partition: NO

```

```

Importance:

```

```

    variable  importance  rank
0  Petal_Width  0.615851  1
1  Petal_Length  0.362519  2
2  Sepal_Length  0.042751  3
3  Sepal_Width  -0.155867  4

```

9.8 Association Rules

The `oml.ar` class implements the Apriori algorithm to find frequent itemsets and association rules, all as part of an association model object.

The Apriori algorithm is efficient and scales well with respect to the number of transactions, number of items, and number of itemsets and rules produced.

Use the `oml.ar` class to identify frequent itemsets within large volumes of transactional data, such as in market basket analysis. The results of an association model are the rules that identify patterns of association within the data.

An association rule identifies a pattern in the data in which the appearance of a set of items in a transactional record implies another set of items. The groups of items used to form rules must pass a minimum threshold according to how often they occur (the *support* of the rule) and how often the consequent follows the antecedent (the *confidence* of the rule). Association models generate all rules that have support and confidence greater than user-specified thresholds.

Oracle Machine Learning does not support the scoring operation for association modeling.

For information on the `oml.ar` class attributes and methods, invoke `help(oml.ar)` or see Oracle Machine Learning for Python API Reference.

Settings for an Association Rules Model

The following table lists the settings applicable to association rules models.

Table 9-3 Association Rules Models Settings

Setting Name	Setting Value	Description
ASSO_ABS_ERROR	0 < ASSO_ABS_ERRORMAX (ASSO_MIN_SUPPORT, ASSO_MIN_CONFIDENCE)	<p>Specifies the absolute error for the association rules sampling.</p> <p>A smaller value of ASSO_ABS_ERROR obtains a larger sample size that gives accurate results but takes longer to compute. Set a reasonable value for ASSO_ABS_ERROR, such as the default value, to avoid too large a sample size. The default value is $0.5 * \text{MAX}(\text{ASSO_MIN_SUPPORT}, \text{ASSO_MIN_CONFIDENCE})$.</p>
ASSO_AGGREGATES	NULL	<p>Specifies the columns to aggregate. It is a comma separated list of strings containing the names of the columns for aggregation. The number of columns in the list must be ≤ 10.</p> <p>You can set ASSO_AGGREGATES if you have specified a column name with ODMS_ITEM_ID_COLUMN_NAME. The data table must have valid column names such as ITEM_ID and CASE_ID which are derived from ODMS_ITEM_ID_COLUMN_NAME.</p> <p>An item value is not mandatory.</p> <p>The default value is NULL.</p> <p>For each item, you may supply several columns to aggregate. However, doing so requires more memory to buffer the extra data and also affects performance because of the larger input data set and increased operations.</p>
ASSO_ANT_IN_RULES	NULL	<p>Sets Including Rules for the antecedent: it is a comma separated list of strings, at least one of which must appear in the antecedent part of each reported association rule.</p> <p>The default value is NULL.</p>
ASSO_ANT_EX_RULES	NULL	<p>Sets Excluding Rules for the antecedent: it is a comma separated list of strings, none of which can appear in the antecedent part of each reported association rule.</p> <p>The default value is NULL.</p>
ASSO_CONF_LEVEL	0 ASSO_CONF_LEVEL 1	<p>Specifies the confidence level for an association rules sample.</p> <p>A larger value of ASSO_CONF_LEVEL obtains a larger sample size. Any value between 0.9 and 1 is suitable. The default value is 0.95.</p>

Table 9-3 (Cont.) Association Rules Models Settings

Setting Name	Setting Value	Description
ASSO_CONS_IN_RULES	NULL	<p>Sets Including Rules for the consequent: it is a comma separated list of strings, at least one of which must appear in the consequent part of each reported association rule.</p> <p>The default value is NULL.</p>
ASSO_CONS_EX_RULES	NULL	<p>Sets Excluding Rules for the consequent: it is a comma separated list of strings, none of which can appear in the consequent part of a reported association rule.</p> <p>You can use the excluding rule to reduce the data that must be stored, but you may be required to build extra models for executing different Including or Excluding Rules.</p> <p>The default value is NULL.</p>
ASSO_EX_RULES	NULL	<p>Sets Excluding Rules applied for each association rule: it is a comma separated list of strings that cannot appear in an association rule. No rule can contain any item in the list.</p> <p>The default value is NULL.</p>
ASSO_IN_RULES	NULL	<p>Sets Including Rules applied for each association rule: it is a comma separated list of strings, at least one of which must appear in each reported association rule, either as antecedent or as consequent</p> <p>The default value NULL, which specifies that filtering is not applied.</p>
ASSO_MAX_RULE_LENGTH	TO_CHAR(2<= numeric_expr <=20)	<p>Maximum rule length for association rules.</p> <p>The default value is 4.</p>
ASSO_MIN_CONFIDENCE	TO_CHAR(0<= numeric_expr <=1)	<p>Minimum confidence for association rules.</p> <p>The default value is 0.1.</p>
ASSO_MIN_REV_CONFIDENCE	TO_CHAR(0<= numeric_expr <=1)	<p>Sets the Minimum Reverse Confidence that each rule should satisfy.</p> <p>The Reverse Confidence of a rule is defined as the number of transactions in which the rule occurs divided by the number of transactions in which the consequent occurs.</p> <p>The value is real number between 0 and 1.</p> <p>The default value is 0.</p>
ASSO_MIN_SUPPORT	TO_CHAR(0<= numeric_expr <=1)	<p>Minimum support for association rules.</p> <p>The default value is 0.1.</p>
ASSO_MIN_SUPPORT_INT	TO_CHAR(0<= numeric_expr <=1)	<p>Minimum absolute support that each rule must satisfy. The value must be an integer.</p> <p>The default value is 1.</p>

Table 9-3 (Cont.) Association Rules Models Settings

Setting Name	Setting Value	Description
ASSO_CONS_EX_RULES		
ODMS_ITEM_ID_COLUMN_NAME	<i>column_name</i>	<p>The name of a column that contains the items in a transaction. When you specify this setting, the algorithm expects the data to be presented in native transactional format, consisting of two columns:</p> <ul style="list-style-type: none"> • Case ID, either categorical or numeric • Item ID, either categorical or numeric
ODMS_ITEM_VALUE_COLUMN_NAME	<i>column_name</i>	<p>The name of a column that contains a value associated with each item in a transaction. Use this setting only when you have specified a value for <code>ODMS_ITEM_ID_COLUMN_NAME</code>, indicating that the data is presented in native transactional format.</p> <p>If you also use <code>ASSO_AGGREGATES</code>, then the build data must include the following three columns and the columns specified in the <code>AGGREGATES</code> setting.</p> <ul style="list-style-type: none"> • Case ID, either categorical or numeric • Item ID, either categorical or numeric, specified by <code>ODMS_ITEM_ID_COLUMN_NAME</code> • Item value, either categorical or numeric, specified by <code>ODMS_ITEM_VALUE_COLUMN_NAME</code> <p>If <code>ASSO_AGGREGATES</code>, Case ID, and Item ID columns are present, then the Item Value column may or may not appear.</p> <p>The Item Value column may specify information such as the number of items (for example, three apples) or the type of the item (for example, macintosh apples).</p>

**See Also:**

- [About Model Settings](#)
- [Shared Settings](#)

Example 9-8 Using the oml.ar Class

This example uses methods of the `oml.ar` class.

```
import pandas as pd
from sklearn import datasets
import oml

# Load the iris data set and create a pandas.DataFrame for it.
```

```
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species']))

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training data.
train_dat = oml.sync(table = 'IRIS')

# Specify settings.
setting = {'asso_min_support': '0.1', 'asso_min_confidence': '0.1'}

# Create an AR model object.
ar_mod = oml.ar(**setting)

# Fit the model according to the training data and parameter
# settings.
ar_mod = ar_mod.fit(train_dat)

# Show details of the model.
ar_mod
```

Listing for This Example

```
>>> import pandas as pd
>>> from sklearn import datasets
>>> import oml
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                 columns = ['Sepal_Length', 'Sepal_Width',
...                           'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                          {0: 'setosa', 1: 'versicolor',
...                           2: 'virginica'}[x], iris.target)),
...                 columns = ['Species'])
>>>
>>> try:
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create the IRIS database table.
```

```

... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training data.
... train_dat = oml.sync(table = 'IRIS')
>>>
>>> # Specify settings.
... setting = {'asso_min_support':'0.1', 'asso_min_confidence':'0.1'}
>>>
>>> # Create an AR model object.
... ar_mod = oml.ar(**setting)
>>>
>>> # Fit the model according to the training data and parameter
... # settings.
>>> ar_mod = ar_mod.fit(train_dat)
>>>
>>> # Show details of the model.
... ar_mod

```

Algorithm Name: Association Rules

Mining Function: ASSOCIATION

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_APRIORI_ASSOCIATION_RULES
1	ASSO_MAX_RULE_LENGTH	4
2	ASSO_MIN_CONFIDENCE	0.1
3	ASSO_MIN_REV_CONFIDENCE	0
4	ASSO_MIN_SUPPORT	0.1
5	ASSO_MIN_SUPPORT_INT	1
6	ODMS_DETAILS	ODMS_ENABLE
7	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
8	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
9	PREP_AUTO	ON

Global Statistics:

	attribute name	attribute value
0	ITEMSET_COUNT	6.000000
1	MAX_SUPPORT	0.333333
2	NUM_ROWS	150.000000
3	RULE_COUNT	2.000000
4	TRANSACTION_COUNT	150.000000

Attributes:

Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Itemsets:

	ITEMSET_ID	SUPPORT	NUMBER_OF_ITEMS	ITEM_NAME	ITEM_VALUE
0	1	0.193333	1	Petal_Width	.200000000000000001

1	2	0.173333	1	Sepal_Width	3
2	3	0.333333	1	Species	setosa
3	4	0.333333	1	Species	versicolor
4	5	0.333333	1	Species	virginica
5	6	0.193333	2	Petal_Width	.200000000000000001
6	6	0.193333	2	Species	setosa

Rules:

RULE_ID	NUMBER_OF_ITEMS	LHS_NAME	LHS_VALUE	RHS_NAME	\
0	1	Species	setosa	Petal_Width	
1	2	Petal_Width	.200000000000000001	Species	

RHS_VALUE	SUPPORT	CONFIDENCE	REVCONFIDENCE	LIFT	
0	None	0.186667	0.58	1.00	3
1	None	0.186667	1.00	0.58	3

9.9 Decision Tree

The `oml.dt` class uses the Decision Tree algorithm for classification.

Decision Tree models are classification models that contain axis-parallel rules. A rule is a conditional statement that can be understood by humans and may be used within a database to identify a set of records.

A decision tree predicts a target value by asking a sequence of questions. At a given stage in the sequence, the question that is asked depends upon the answers to the previous questions. The goal is to ask questions that, taken together, uniquely identify specific target values. Graphically, this process forms a tree structure.

During the training process, the Decision Tree algorithm must repeatedly find the most efficient way to split a set of cases (records) into two child nodes. The `oml.dt` class offers two homogeneity metrics, gini and entropy, for calculating the splits. The default metric is gini.

For information on the `oml.dt` class attributes and methods, invoke `help(oml.dt)` or see Oracle Machine Learning for Python API Reference.

Settings for a Decision Tree Model

The following table lists settings that apply to Decision Tree models.

Table 9-4 Decision Tree Model Settings

Setting Name	Setting Value	Description
CLAS_COST_TABLE_NAME	<i>table_name</i>	<p>The name of a table that stores a cost matrix for the algorithm to use in building and applying the model. The cost matrix specifies the costs associated with misclassifications.</p> <p>The cost matrix table is user-created. The following are the column requirements for the table.</p> <ul style="list-style-type: none"> • Column Name: ACTUAL_TARGET_VALUE Data Type: Valid target data type • Column Name: PREDICTED_TARGET_VALUE Data Type: Valid target data type • Column Name: COST Data Type: NUMBER
CLAS_MAX_SUP_BINS	<i>2 <= a number <= 2147483647</i>	<p>Specifies the maximum number of bins for each attribute.</p> <p>The default value is 32.</p>
CLAS_WEIGHTS_BALANCED	ON OFF	<p>Indicates whether the algorithm must create a model that balances the target distribution. This setting is most relevant in the presence of rare targets, as balancing the distribution may enable better average accuracy (average of per-class accuracy) instead of overall accuracy (which favors the dominant class). The default value is OFF.</p>
TREE_IMPURITY_METRIC	TREE_IMPURITY_ENTROPY TREE_IMPURITY_GINI	<p>Tree impurity metric for a Decision Tree model.</p> <p>Tree algorithms seek the best test question for splitting data at each node. The best splitter and split value are those that result in the largest increase in target value homogeneity (purity) for the entities in the node. Purity is measured in accordance with a metric. Decision trees can use either gini (TREE_IMPURITY_GINI) or entropy (TREE_IMPURITY_ENTROPY) as the purity metric. By default, the algorithm uses TREE_IMPURITY_GINI.</p>
TREE_TERM_MAX_DEPTH	<i>2 <= a number <= 100</i>	<p>Criteria for splits: maximum tree depth (the maximum number of nodes between the root and any leaf node, including the leaf node).</p> <p>The default is 7.</p>
TREE_TERM_MINPCT_NODE	<i>0 < = a number <= 10</i>	<p>The minimum number of training rows in a node expressed as a percentage of the rows in the training data.</p> <p>The default value is 0.05, indicating 0.05%.</p>

Table 9-4 (Cont.) Decision Tree Model Settings

Setting Name	Setting Value	Description
TREE_TERM_MINPCT_SPLIT	$0 < a \text{ number} \leq 20$	Minimum number of rows required to consider splitting a node expressed as a percentage of the training rows. The default value is 0.1, indicating 0.1%.
TREE_TERM_MINREC_NODE	$A \text{ number} \geq 0$	Minimum number of rows in a node. The default value is 10.
TREE_TERM_MINREC_SPLIT	$A \text{ number} > 1$	Criteria for splits: minimum number of records in a parent node expressed as a value. No split is attempted if the number of records is below this value. The default value is 20.

 **See Also:**

- [About Model Settings](#)
- [Shared Settings](#)

Example 9-9 Using the oml.dt Class

This example demonstrates the use of various methods of the `oml.dt` class. In the listing for this example, some of the output is not shown as indicated by ellipses.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('COST_MATRIX')
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
```

```
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Create a cost matrix table in the database.
cost_matrix = [['setosa', 'setosa', 0],
               ['setosa', 'virginica', 0.2],
               ['setosa', 'versicolor', 0.8],
               ['virginica', 'virginica', 0],
               ['virginica', 'setosa', 0.5],
               ['virginica', 'versicolor', 0.5],
               ['versicolor', 'versicolor', 0],
               ['versicolor', 'setosa', 0.4],
               ['versicolor', 'virginica', 0.6]]
cost_matrix = oml.create(
    pd.DataFrame(cost_matrix,
                 columns = ['ACTUAL_TARGET_VALUE',
                           'PREDICTED_TARGET_VALUE', 'COST']),
    table = 'COST_MATRIX')

# Specify settings.
setting = {'TREE_TERM_MAX_DEPTH': '2'}

# Create a DT model object.
dt_mod = oml.dt(**setting)

# Fit the DT model according to the training data and parameter
# settings.
dt_mod.fit(train_x, train_y, cost_matrix = cost_matrix)

# Use the model to make predictions on the test data.
dt_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
                                                'Sepal_Width',
                                                'Petal_Length',
                                                'Species']])

# Return the prediction probability.
dt_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
                                                'Sepal_Width',
                                                'Species']],
               proba = True)

# Make predictions and return the probability for each class
# on new data.
dt_mod.predict_proba(test_dat.drop('Species'),
                    supplemental_cols = test_dat[:,
                                                  ['Sepal_Length',
                                                   'Species']]).sort_values(by = ['Sepal_Length',
                                                                 'Species'])

dt_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                   columns = ['Sepal_Length', 'Sepal_Width',
...                               'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                             {0: 'setosa', 1: 'versicolor',
...                               2: 'virginica'}[x], iris.target)),
...                   columns = ['Species'])
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> try:
...     oml.drop('COST_MATRIX')
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
>>>
>>> # Create a cost matrix table in the database.
... cost_matrix = [['setosa', 'setosa', 0],
...                 ['setosa', 'virginica', 0.2],
...                 ['setosa', 'versicolor', 0.8],
...                 ['virginica', 'virginica', 0],
...                 ['virginica', 'setosa', 0.5],
...                 ['virginica', 'versicolor', 0.5],
...                 ['versicolor', 'versicolor', 0],
...                 ['versicolor', 'setosa', 0.4],
...                 ['versicolor', 'virginica', 0.6]]
>>> cost_matrix = oml.create(
...     pd.DataFrame(cost_matrix,
...                   columns = ['ACTUAL_TARGET_VALUE',
...                               'PREDICTED_TARGET_VALUE',
...                               'COST']),
...     table = 'COST_MATRIX')
>>>
>>> # Specify settings.
... setting = {'TREE_TERM_MAX_DEPTH': '2'}
>>>
>>> # Create a DT model object.
... dt_mod = oml.dt(**setting)
>>>
>>> # Fit the DT model according to the training data and parameter
... # settings.
```

```
>>> dt_mod.fit(train_x, train_y, cost_matrix = cost_matrix)
```

Algorithm Name: Decision Tree

Mining Function: CLASSIFICATION

Target: Species

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_DECISION_TREE
1	CLAS_COST_TABLE_NAME	"OML_USER"."COST_MATRIX"
2	CLAS_MAX_SUP_BINS	32
3	CLAS_WEIGHTS_BALANCED	OFF
4	ODMS_DETAILS	ODMS_ENABLE
5	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
6	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
7	PREP_AUTO	ON
8	TREE_IMPURITY_METRIC	TREE_IMPURITY_GINI
9	TREE_TERM_MAX_DEPTH	2
10	TREE_TERM_MINPCT_NODE	.05
11	TREE_TERM_MINPCT_SPLIT	.1
12	TREE_TERM_MINREC_NODE	10
13	TREE_TERM_MINREC_SPLIT	20

Global Statistics:

	attribute name	attribute value
0	NUM_ROWS	104

Attributes:

Petal_Length
Petal_Width

Partition: NO

Distributions:

	NODE_ID	TARGET_VALUE	TARGET_COUNT
0	0	setosa	36
1	0	versicolor	35
2	0	virginica	33
3	1	setosa	36
4	2	versicolor	35
5	2	virginica	33

Nodes:

	parent	node.id	row.count	prediction	\
0	0.0	1	36	setosa	
1	0.0	2	68	versicolor	
2	NaN	0	104	setosa	
				split	\
0	(Petal_Length <=(2.4500000000000002E+000))				
1	(Petal_Length >(2.4500000000000002E+000))				
2	None				

```

surrogate \
0 Petal_Width <=(8.0000000000000004E-001)
1 Petal_Width >(8.0000000000000004E-001)
2 None

full.splits
0 (Petal_Length <=(2.4500000000000002E+000))
1 (Petal_Length >(2.4500000000000002E+000))
2 (
>>>
>>> # Use the model to make predictions on the test data.
... dt_mod.predict(test_dat.drop('Species'),
...                 supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                  'Sepal_Width',
...                                                  'Petal_Length',
...                                                  'Species']])
...
Sepal_Length Sepal_Width Petal_Length Species PREDICTION
0 4.9 3.0 1.4 setosa setosa
1 4.9 3.1 1.5 setosa setosa
2 4.8 3.4 1.6 setosa setosa
3 5.8 4.0 1.2 setosa setosa
... ..
44 6.7 3.3 5.7 virginica versicolor
45 6.7 3.0 5.2 virginica versicolor
46 6.5 3.0 5.2 virginica versicolor
47 5.9 3.0 5.1 virginica versicolor
>>>
>>> # Return the prediction probability.
... dt_mod.predict(test_dat.drop('Species'),
...                 supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                  'Sepal_Width',
...                                                  'Species']],
...                 proba = True)
...
Sepal_Length Sepal_Width Species PREDICTION PROBABILITY
0 4.9 3.0 setosa setosa 1.000000
1 4.9 3.1 setosa setosa 1.000000
2 4.8 3.4 setosa setosa 1.000000
3 5.8 4.0 setosa setosa 1.000000
... ..
44 6.7 3.3 virginica versicolor 0.514706
45 6.7 3.0 virginica versicolor 0.514706
46 6.5 3.0 virginica versicolor 0.514706
47 5.9 3.0 virginica versicolor 0.514706

>>> # Make predictions and return the probability for each class
>>> # on new data.
>>> dt_mod.predict_proba(test_dat.drop('Species'),
...                       supplemental_cols = test_dat[:,
...                                                  ['Sepal_Length',
...                                                  'Species']]).sort_values(by = ['Sepal_Length',
...                                                  'Species'])
...
Sepal_Length Species PROBABILITY_OF_SETOSA \
0 4.4 setosa 1.0
1 4.4 setosa 1.0
2 4.5 setosa 1.0

```

```

3          4.8      setosa          1.0
...          ...          ...          ...
42         6.7    virginica          0.0
43         6.9  versicolor          0.0
44         6.9    virginica          0.0
45         7.0  versicolor          0.0

      PROBABILITY_OF_VERSICOLOR  PROBABILITY_OF_VIRGINICA
0          0.000000          0.000000
1          0.000000          0.000000
2          0.000000          0.000000
3          0.000000          0.000000
...          ...          ...
42         0.514706          0.485294
43         0.514706          0.485294
44         0.514706          0.485294
45         0.514706          0.485294
>>>
>>> dt_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.645833

```

9.10 Expectation Maximization

The `oml.em` class uses the Expectation Maximization (EM) algorithm to create a clustering model.

EM is a density estimation algorithm that performs probabilistic clustering. In density estimation, the goal is to construct a density function that captures how a given population is distributed. The density estimate is based on observed data that represents a sample of the population.

For information on the `oml.em` class methods, invoke `help(oml.em)` or see Oracle Machine Learning for Python API Reference..

Settings for an Expectation Maximization Model

The following table lists settings for data preparation and analysis for EM models.

Table 9-5 Expectation Maximization Settings for Data Preparation and Analysis



Setting Name	Setting Value	Description
EMCS_ATTRIBUTE_FILTER	EMCS_ATTR_FILTER_ENABLE EMCS_ATTR_FILTER_DISABLE	Whether or not to include uncorrelated attributes in the model. When EMCS_ATTRIBUTE_FILTER is enabled, uncorrelated attributes are not included.
		<div style="border: 1px solid #0070C0; padding: 5px; background-color: #E6F2FF;">  Note: This setting applies only to attributes that are not nested. </div>
		The default value is system-determined.
EMCS_MAX_NUM_ATTR_2D	TO_CHAR(numeric_expr >= 1)	Maximum number of correlated attributes to include in the model.
		<div style="border: 1px solid #0070C0; padding: 5px; background-color: #E6F2FF;">  Note: This setting applies only to attributes that are not nested (2D). </div>
		The default value is 50.
EMCS_NUM_DISTRIBUTION	EMCS_NUM_DISTR_BERNOULLI EMCS_NUM_DISTR_GAUSSIAN EMCS_NUM_DISTR_SYSTEM	<p>The distribution for modeling numeric attributes. Applies to the input table or view as a whole and does not allow per-attribute specifications.</p> <p>The options include Bernoulli, Gaussian, or system-determined distribution. When Bernoulli or Gaussian distribution is chosen, all numeric attributes are modeled using the same type of distribution. When the distribution is system-determined, individual attributes may use different distributions (either Bernoulli or Gaussian), depending on the data.</p> <p>The default value is EMCS_NUM_DISTR_SYSTEM.</p>

Table 9-5 (Cont.) Expectation Maximization Settings for Data Preparation and Analysis

Setting Name	Setting Value	Description
EMCS_NUM_EQUIWIDTH_BINS	TO_CHAR(1 < numeric_expr <= 255)	Number of equi-width bins that will be used for gathering cluster statistics for numeric columns. The default value is 11.
EMCS_NUM_PROJECTIONS	TO_CHAR(numeric_expr >= 1)	Specifies the number of projections to use for each nested column. If a column has fewer distinct attributes than the specified number of projections, then the data is not projected. The setting applies to all nested columns. The default value is 50.
EMCS_NUM_QUANTILE_BINS	TO_CHAR(1 < numeric_expr <= 255)	Specifies the number of quantile bins to use for modeling numeric columns with multivalued Bernoulli distributions. The default value is system-determined.
EMCS_NUM_TOPN_BINS	TO_CHAR(1 < numeric_expr <= 255)	Specifies the number of top-N bins to use for modeling categorical columns with multivalued Bernoulli distributions. The default value is system-determined.

The following table lists settings for learning for EM models.

Table 9-6 Expectation Maximization Settings for Learning

Setting Name	Setting Value	Description
EMCS_CONVERGENCE_CRITERION	EMCS_CONV_CRIT_HELDASIDE EMCS_CONV_CRIT_BIC	The convergence criterion for EM. The convergence criterion may be based on a held-aside data set or it may be Bayesian Information Criterion. The default value is system determined.
EMCS_LOGLIKE_IMPROVEMENT	TO_CHAR(0 < numeric_expr < 1)	When the convergence criterion is based on a held-aside data set (EMCS_CONVERGENCE_CRITERION = EMCS_CONV_CRIT_HELDASIDE), this setting specifies the percentage improvement in the value of the log likelihood function that is required for adding a new component to the model.
EMCS_MODEL_SEARCH	EMCS_MODEL_SEARCH_ENABLE EMCS_MODEL_SEARCH_DISABLE	Enables model search in EM where different model sizes are explored and the best size is selected. The default value is EMCS_MODEL_SEARCH_DISABLE.

Table 9-6 (Cont.) Expectation Maximization Settings for Learning

Setting Name	Setting Value	Description
EMCS_NUM_COMPONENTS	TO_CHAR(<i>numeric_expr</i> >= 1)	Maximum number of components in the model. If model search is enabled, the algorithm automatically determines the number of components based on improvements in the likelihood function or based on regularization, up to the specified maximum. The number of components must be greater than or equal to the number of clusters. The default value is 20.
EMCS_NUM_ITERATIONS	TO_CHAR(<i>numeric_expr</i> >= 1)	Specifies the maximum number of iterations in the EM algorithm. The default value is 100.
EMCS_RANDOM_SEED	Non-negative integer	Controls the seed of the random generator used in EM. The default value is 0.
EMCS_REMOVE_COMPONENTS	EMCS_REMOVE_COMPS_ENABLE EMCS_REMOVE_COMPS_DISABLE	Allows the EM algorithm to remove a small component from the solution. The default value is EMCS_REMOVE_COMPS_ENABLE.

The following table lists the settings for component clustering for EM models.

Table 9-7 Expectation Maximization Settings for Component Clustering

Setting Name	Setting Value	Description
CLUS_NUM_CLUSTERS	TO_CHAR(<i>numeric_expr</i> >= 1)	The maximum number of leaf clusters generated by the algorithm. The algorithm may return fewer clusters than the specified number, depending on the data, but it cannot return more clusters than the number of components, which is governed by algorithm-specific settings. (See Table 9-6 .) Depending on these settings, there may be fewer clusters than components. If component clustering is disabled, then the number of clusters equals the number of components. The default value is system-determined.

Table 9-7 (Cont.) Expectation Maximization Settings for Component Clustering

Setting Name	Setting Value	Description
EMCS_CLUSTER_COMPONENTS	EMCS_CLUSTER_COMP_ENABLE EMCS_CLUSTER_COMP_DISABLE	Enables or disables the grouping of EM components into high-level clusters. When disabled, the components themselves are treated as clusters. When component clustering is enabled, model scoring through the SQL CLUSTER function produces assignments to the higher level clusters. When clustering is disabled, the CLUSTER function produces assignments to the original components. The default value is EMCS_CLUSTER_COMP_ENABLE.
EMCS_CLUSTER_THRESH	TO_CHAR(numeric_expr >= 1)	Dissimilarity threshold that controls the clustering of EM components. When the dissimilarity measure is less than the threshold, the components are combined into a single cluster. A lower threshold may produce more clusters that are more compact. A higher threshold may produce fewer clusters that are more spread out. The default value is 2.
EMCS_LINKAGE_FUNCTION	EMCS_LINKAGE_SINGLE EMCS_LINKAGE_AVERAGE EMCS_LINKAGE_COMPLETE	Allows the specification of a linkage function for the agglomerative clustering step. EMCS_LINKAGE_SINGLE uses the nearest distance within the branch. The clusters tend to be larger and have arbitrary shapes. EMCS_LINKAGE_AVERAGE uses the average distance within the branch. There is less chaining effect and the clusters are more compact. EMCS_LINKAGE_COMPLETE uses the maximum distance within the branch. The clusters are smaller and require strong component overlap. The default value is EMCS_LINKAGE_SINGLE.

The following table lists the settings for cluster statistics for EM models.

Table 9-8 Expectation Maximization Settings for Cluster Statistics

Setting Name	Setting Value	Description
EMCS_CLUSTER_STATISTICS	EMCS_CLUS_STATS_ENABLE EMCS_CLUS_STATS_DISABLE	Enables or disables the gathering of descriptive statistics for clusters (centroids, histograms, and rules). When statistics are disabled, model size is reduced. The default value is EMCS_CLUS_STATS_ENABLE.
EMCS_MIN_PCT_ATTR_SUPPORT	TO_CHAR(0 < numeric_expr < 1)	Minimum support required for including an attribute in the cluster rule. The support is the percentage of the data rows assigned to a cluster that must have non-null values for the attribute. The default value is 0.1.

 **See Also:**

- [About Model Settings](#)
- [Shared Settings](#)

Example 9-10 Using the oml.em Class

This example creates an EM model and uses some of the methods of the `oml.em` class.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
```

```
train_dat = dat[0]
test_dat = dat[1]

# Specify settings.
setting = {'emcs_num_iterations': 100}

# Create an EM model object
em_mod = oml.em(n_clusters = 2, **setting)

# Fit the EM model according to the training data and parameter
# settings.
em_mod = em_mod.fit(train_dat)

# Show details of the model.
em_mod

# Use the model to make predictions on the test data.
em_mod.predict(test_dat)

# Make predictions and return the probability for each class
# on new data.
em_mod.predict_proba(test_dat,
    supplemental_cols = test_dat[:,
        ['Sepal_Length', 'Sepal_Width',
        'Petal_Length']]).sort_values(by = ['Sepal_Length',
        'Sepal_Width', 'Petal_Length',
        'PROBABILITY_OF_2', 'PROBABILITY_OF_3'])

# Change the random seed and refit the model.
em_mod.set_params(EMCS_RANDOM_SEED = '5').fit(train_dat)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                            'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                          {0: 'setosa', 1: 'versicolor',
...                           2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
```

```
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_dat = dat[0]
>>> test_dat = dat[1]
>>>
>>> # Specify settings.
... setting = {'emcs_num_iterations': 100}
>>>
>>> # Create an EM model object.
... em_mod = oml.em(n_clusters = 2, **setting)
>>>
>>> # Fit the EM model according to the training data and parameter
... # settings.
>>> em_mod = em_mod.fit(train_dat)
>>>
>>> # Show details of the model.
... em_mod
```

Algorithm Name: Expectation Maximization

Mining Function: CLUSTERING

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_EXPECTATION_MAXIMIZATION
1	CLUS_NUM_CLUSTERS	2
2	EMCS_CLUSTER_COMPONENTS	EMCS_CLUSTER_COMP_ENABLE
3	EMCS_CLUSTER_STATISTICS	EMCS_CLUS_STATS_ENABLE
4	EMCS_CLUSTER_THRESH	2
5	EMCS_LINKAGE_FUNCTION	EMCS_LINKAGE_SINGLE
6	EMCS_LOGLIKE_IMPROVEMENT	.001
7	EMCS_MAX_NUM_ATTR_2D	50
8	EMCS_MIN_PCT_ATTR_SUPPORT	.1
9	EMCS_MODEL_SEARCH	EMCS_MODEL_SEARCH_DISABLE
10	EMCS_NUM_COMPONENTS	20
11	EMCS_NUM_DISTRIBUTION	EMCS_NUM_DISTR_SYSTEM
12	EMCS_NUM_EQUIWIDTH_BINS	11
13	EMCS_NUM_ITERATIONS	100
14	EMCS_NUM_PROJECTIONS	50
15	EMCS_RANDOM_SEED	0
16	EMCS_REMOVE_COMPONENTS	EMCS_REMOVE_COMPS_ENABLE
17	ODMS_DETAILS	ODMS_ENABLE
18	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
19	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
20	PREP_AUTO	ON

Computed Settings:

	setting name	setting value
0	EMCS_ATTRIBUTE_FILTER	EMCS_ATTR_FILTER_DISABLE
1	EMCS_CONVERGENCE_CRITERION	EMCS_CONV_CRIT_BIC
2	EMCS_NUM_QUANTILE_BINS	3
3	EMCS_NUM_TOPN_BINS	3

Global Statistics:

	attribute name	attribute value
0	CONVERGED	YES

```

1      LOGLIKELIHOOD      -2.10044
2      NUM_CLUSTERS        2
3      NUM_COMPONENTS     8
4      NUM_ROWS           104
5      RANDOM_SEED        0
6      REMOVED_COMPONENTS 12

```

Attributes:

```

Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

```

Partition: NO

Clusters:

	CLUSTER_ID	CLUSTER_NAME	RECORD_COUNT	PARENT	TREE_LEVEL \
0	1	1	104	NaN	1
1	2	2	68	1.0	2
2	3	3	36	1.0	2
	LEFT_CHILD_ID	RIGHT_CHILD_ID			
0	2.0	3.0			
1	NaN	NaN			
2	NaN	NaN			

Taxonomy:

	PARENT_CLUSTER_ID	CHILD_CLUSTER_ID
0	1	2.0
1	1	3.0
2	2	NaN
3	3	NaN

Centroids:

	CLUSTER_ID	ATTRIBUTE_NAME	MEAN	MODE_VALUE	VARIANCE
0	1	Petal_Length	3.721154	None	3.234694
1	1	Petal_Width	1.155769	None	0.567539
2	1	Sepal_Length	5.831731	None	0.753255
3	1	Sepal_Width	3.074038	None	0.221358
4	1	Species	NaN	setosa	NaN
5	2	Petal_Length	4.902941	None	0.860588
6	2	Petal_Width	1.635294	None	0.191572
7	2	Sepal_Length	6.266176	None	0.545555
8	2	Sepal_Width	2.854412	None	0.128786
9	2	Species	NaN	versicolor	NaN
10	3	Petal_Length	1.488889	None	0.033016
11	3	Petal_Width	0.250000	None	0.012857
12	3	Sepal_Length	5.011111	None	0.113016
13	3	Sepal_Width	3.488889	None	0.134159
14	3	Species	NaN	setosa	NaN

Leaf Cluster Counts:

CLUSTER_ID	CNT
0	2 68
1	3 36

Attribute Importance:

ATTRIBUTE_NAME	ATTRIBUTE_IMPORTANCE_VALUE	ATTRIBUTE_RANK
0 Petal_Length	0.558311	2
1 Petal_Width	0.556300	3
2 Sepal_Length	0.469978	4
3 Sepal_Width	0.196211	5
4 Species	0.612463	1

Components:

COMPONENT_ID	CLUSTER_ID	PRIOR_PROBABILITY
0	1 2	0.115366
1	2 2	0.079158
2	3 3	0.113448
3	4 2	0.148059
4	5 3	0.126979
5	6 2	0.134402
6	7 3	0.105727
7	8 2	0.176860

Cluster Hists:

cluster.id	variable	bin.id	lower.bound	upper.bound	\
0	1 Petal_Length	1	1.00	1.59	
1	1 Petal_Length	2	1.59	2.18	
2	1 Petal_Length	3	2.18	2.77	
3	1 Petal_Length	4	2.77	3.36	
...
137	3 Sepal_Width	11	NaN	NaN	
138	3 Species:'Other'	1	NaN	NaN	
139	3 Species:setosa	2	NaN	NaN	
140	3 Species:versicolor	3	NaN	NaN	

label	count
0 1:1.59	25
1 1.59:2.18	11
2 2.18:2.77	0
3 2.77:3.36	3
...	...
137 :	0
138 :	0
139 :	36
140 :	0

[141 rows x 7 columns]

Rules:

cluster.id	rhs.support	rhs.conf	lhr.support	lhs.conf	lhs.var	\
0	1	104	1.000000	93	0.892157	Sepal_Width
1	1	104	1.000000	93	0.892157	Sepal_Width

```

2          1          104 1.000000          99 0.892157 Petal_Length
3          1          104 1.000000          99 0.892157 Petal_Length
...      ...      ...      ...      ...      ...
26         3          36 0.346154          36 0.972222 Petal_Length
27         3          36 0.346154          36 0.972222 Sepal_Length
28         3          36 0.346154          36 0.972222 Sepal_Length
29         3          36 0.346154          36 0.972222 Species

```

```

      lhs.var.support  lhs.var.conf      predicate
0          93      0.400000      Sepal_Width <= 3.92
1          93      0.400000      Sepal_Width > 2.48
2          93      0.222222      Petal_Length <= 6.31
3          93      0.222222      Petal_Length >= 1
...      ...      ...      ...
26         35      0.134398      Petal_Length >= 1
27         35      0.094194      Sepal_Length <= 5.74
28         35      0.094194      Sepal_Length >= 4.3
29         35      0.281684      Species = setosa

```

[30 rows x 9 columns]

```
>>> # Use the model to make predictions on the test data.
```

```
... em_mod.predict(test_dat)
```

```
CLUSTER_ID
```

```
0          3
```

```
1          3
```

```
2          3
```

```
3          3
```

```
...      ...
```

```
42         2
```

```
43         2
```

```
44         2
```

```
45         2
```

```
>>> # Make predictions and return the probability for each class
```

```
... # on new data.
```

```
>>> em_mod.predict_proba(test_dat,
```

```
...   supplemental_cols = test_dat[:,
```

```
...     ['Sepal_Length', 'Sepal_Width',
```

```
...     'Petal_Length']).sort_values(by = ['Sepal_Length',
```

```
...     'Sepal_Width', 'Petal_Length',
```

```
...     'PROBABILITY_OF_2', 'PROBABILITY_OF_3'])
```

```
   Sepal_Length  Sepal_Width  Petal_Length  PROBABILITY_OF_2 \
```

```
0          4.4          3.0          1.3      4.680788e-20
```

```
1          4.4          3.2          1.3      1.052071e-20
```

```
2          4.5          2.3          1.3      7.751240e-06
```

```
3          4.8          3.4          1.6      5.363418e-19
```

```
...      ...      ...      ...      ...
```

```
43         6.9          3.1          4.9      1.000000e+00
```

```
44         6.9          3.1          5.4      1.000000e+00
```

```
45         7.0          3.2          4.7      1.000000e+00
```

```
PROBABILITY_OF_3
```

```
0          1.000000e+00
```

```
1          1.000000e+00
```

```
2          9.999922e-01
```

```

3      1.000000e+00
...      ...
43     3.295578e-97
44     6.438740e-137
45     3.853925e-89

>>>
>>> # Change the random seed and refit the model.
... em_mod.set_params(EMCS_RANDOM_SEED = '5').fit(train_dat)

```

Algorithm Name: Expectation Maximization

Mining Function: CLUSTERING

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_EXPECTATION_MAXIMIZATION
1	CLUS_NUM_CLUSTERS	2
2	EMCS_CLUSTER_COMPONENTS	EMCS_CLUSTER_COMP_ENABLE
3	EMCS_CLUSTER_STATISTICS	EMCS_CLUS_STATS_ENABLE
4	EMCS_CLUSTER_THRESH	2
5	EMCS_LINKAGE_FUNCTION	EMCS_LINKAGE_SINGLE
6	EMCS_LOGLIKE_IMPROVEMENT	.001
7	EMCS_MAX_NUM_ATTR_2D	50
8	EMCS_MIN_PCT_ATTR_SUPPORT	.1
9	EMCS_MODEL_SEARCH	EMCS_MODEL_SEARCH_DISABLE
10	EMCS_NUM_COMPONENTS	20
11	EMCS_NUM_DISTRIBUTION	EMCS_NUM_DISTR_SYSTEM
12	EMCS_NUM_EQUIWIDTH_BINS	11
13	EMCS_NUM_ITERATIONS	100
14	EMCS_NUM_PROJECTIONS	50
15	EMCS_RANDOM_SEED	5
16	EMCS_REMOVE_COMPONENTS	EMCS_REMOVE_COMPS_ENABLE
17	ODMS_DETAILS	ODMS_ENABLE
18	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
19	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
20	PREP_AUTO	ON

Computed Settings:

	setting name	setting value
0	EMCS_ATTRIBUTE_FILTER	EMCS_ATTR_FILTER_DISABLE
1	EMCS_CONVERGENCE_CRITERION	EMCS_CONV_CRIT_BIC
2	EMCS_NUM_QUANTILE_BINS	3
3	EMCS_NUM_TOPN_BINS	3

Global Statistics:

	attribute name	attribute value
0	CONVERGED	YES
1	LOGLIKELIHOOD	-1.75777
2	NUM_CLUSTERS	2
3	NUM_COMPONENTS	9
4	NUM_ROWS	104
5	RANDOM_SEED	5
6	REMOVED_COMPONENTS	11

Attributes:

Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Clusters:

	CLUSTER_ID	CLUSTER_NAME	RECORD_COUNT	PARENT	TREE_LEVEL	LEFT_CHILD_ID
\						
0	1	1	104	NaN	1	
2.0						
1	2	2	36	1.0	2	
NaN						
2	3	3	68	1.0	2	
NaN						

	RIGHT_CHILD_ID
0	3.0
1	NaN
2	NaN

Taxonomy:

	PARENT_CLUSTER_ID	CHILD_CLUSTER_ID
0	1	2.0
1	1	3.0
2	2	NaN
3	3	NaN

Centroids:

	CLUSTER_ID	ATTRIBUTE_NAME	MEAN	MODE_VALUE	VARIANCE
0	1	Petal_Length	3.721154	None	3.234694
1	1	Petal_Width	1.155769	None	0.567539
2	1	Sepal_Length	5.831731	None	0.753255
3	1	Sepal_Width	3.074038	None	0.221358
4	1	Species	NaN	setosa	NaN
5	2	Petal_Length	1.488889	None	0.033016
6	2	Petal_Width	0.250000	None	0.012857
7	2	Sepal_Length	5.011111	None	0.113016
8	2	Sepal_Width	3.488889	None	0.134159
9	2	Species	NaN	setosa	NaN
10	3	Petal_Length	4.902941	None	0.860588
11	3	Petal_Width	1.635294	None	0.191572
12	3	Sepal_Length	6.266176	None	0.545555
13	3	Sepal_Width	2.854412	None	0.128786
14	3	Species	NaN	versicolor	NaN

Leaf Cluster Counts:

	CLUSTER_ID	CNT
0	2	36
1	3	68

Attribute Importance:

	ATTRIBUTE_NAME	ATTRIBUTE_IMPORTANCE_VALUE	ATTRIBUTE_RANK
0	Petal_Length	0.558311	2
1	Petal_Width	0.556300	3
2	Sepal_Length	0.469978	4
3	Sepal_Width	0.196211	5
4	Species	0.612463	1

Components:

	COMPONENT_ID	CLUSTER_ID	PRIOR_PROBABILITY
0	1	2	0.113452
1	2	2	0.105727
2	3	3	0.114202
3	4	3	0.086285
4	5	3	0.067294
5	6	2	0.124365
6	7	3	0.126975
7	8	3	0.105761
8	9	3	0.155939

Cluster Hists:

	cluster.id	variable	bin.id	lower.bound	upper.bound	\
0	1	Petal_Length	1	1.00	1.59	
1	1	Petal_Length	2	1.59	2.18	
2	1	Petal_Length	3	2.18	2.77	
3	1	Petal_Length	4	2.77	3.36	
...
137	3	Sepal_Width	11	NaN	NaN	
138	3	Species:'Other'	1	NaN	NaN	
139	3	Species:setosa	3	NaN	NaN	
140	3	Species:versicolor	2	NaN	NaN	

	label	count
0	1:1.59	25
1	1.59:2.18	11
2	2.18:2.77	0
3	2.77:3.36	3
...
137	:	0
138	:	33
139	:	0
140	:	35

[141 rows x 7 columns]

Rules:

	cluster.id	rhs.support	rhs.conf	lhr.support	lhs.conf	lhs.var	\
0	1	104	1.000000	93	0.894231	Sepal_Width	
1	1	104	1.000000	93	0.894231	Sepal_Width	
2	1	104	1.000000	99	0.894231	Petal_Length	
3	1	104	1.000000	99	0.894231	Petal_Length	

```

...      ...      ...      ...      ...      ...
26      3      68  0.653846      68  0.955882      Sepal_Length
27      3      68  0.653846      68  0.955882      Sepal_Length
28      3      68  0.653846      68  0.955882      Species
29      3      68  0.653846      68  0.955882      Species

      lhs.var.support  lhs.var.conf      predicate
0      93      0.400000      Sepal_Width <= 3.92
1      93      0.400000      Sepal_Width > 2.48
2      93      0.222222      Petal_Length <= 6.31
3      93      0.222222      Petal_Length >= 1
...      ...      ...      ...
26      65      0.026013      Sepal_Length <= 7.9
27      65      0.026013      Sepal_Length > 4.66
28      65      0.125809      Species IN 'Other'
29      65      0.125809      Species IN versicolor

```

9.11 Explicit Semantic Analysis

The `oml.esa` class extracts text-based features from a corpus of documents and performs document similarity comparisons.

Explicit Semantic Analysis (ESA) is an unsupervised algorithm for feature extraction. ESA does not discover latent features but instead uses explicit features based on an existing knowledge base.

Explicit knowledge often exists in text form. Multiple knowledge bases are available as collections of text documents. These knowledge bases can be generic, such as Wikipedia, or domain-specific. Data preparation transforms the text into vectors that capture attribute-concept associations.

ESA uses concepts of an existing knowledge base as features rather than latent features derived by latent semantic analysis methods such as Singular Value Decomposition and Latent Dirichlet Allocation. Each row, for example, in a document in the training data maps to a feature, that is, a concept. ESA has multiple applications in the area of text processing, most notably semantic relatedness (similarity) and explicit topic modeling. Text similarity use cases might involve, for example, resume matching, searching for similar blog postings, and so on.

For information on the `oml.esa` class attributes and methods, invoke `help(oml.esa)` or see Oracle Machine Learning for Python API Reference.

Settings for an Explicit Semantic Analysis Model

The following table lists settings for ESA models.

Table 9-9 Explicit Semantic Analysis Settings

Setting Name	Setting Value	Description
ESAS_MIN_ITEMS	A non-negative number	Determines the minimum number of non-zero entries required in an input row. The default value is 100 for text input and 0 for non-text input.
ESAS_TOPN_FEATURES	A positive integer	Controls the maximum number of features per attribute. The default value is 1000.

Table 9-9 (Cont.) Explicit Semantic Analysis Settings

Setting Name	Setting Value	Description
ESAS_VALUE_THRESHOLD	A non-negative number	Sets the threshold to a small value for attribute weights in the transformed build data. The default value is $1e-8$.
FEAT_NUM_FEATURES	<code>TO_CHAR(numeric_expr >=1)</code>	The number of features to extract. The default value is estimated by the algorithm. If the matrix rank is smaller than this number, then fewer features are returned.

 **See Also:**

- [About Model Settings](#)
- [Shared Settings](#)

Example 9-11 Using the oml.esa Class

This example creates an ESA model and uses some of the methods of the `oml.esa` class.

```
import oml
from oml import cursor
import pandas as pd

# Create training data and test data.
dat = oml.push(pd.DataFrame(
    {'COMMENTS':['Aids in Africa: Planning for a long war',
                'Mars rover maneuvers for rim shot',
                'Mars express confirms presence of water at Mars south pole',
                'NASA announces major Mars rover finding',
                'Drug access, Asia threat in focus at AIDS summit',
                'NASA Mars Odyssey THEMIS image: typical crater',
                'Road blocks for Aids'],
     'YEAR':['2017', '2018', '2017', '2017', '2018', '2018', '2018'],
     'ID':[1,2,3,4,5,6,7]})).split(ratio=(0.7,0.3), seed = 1234)
train_dat = dat[0]
test_dat = dat[1]

# Specify settings.
cur = cursor()
cur.execute("Begin ctx_ddl.create_policy('DMDEMO_ESA_POLICY'); End;")
cur.close()

odm_settings = {'odms_text_policy_name': 'DMDEMO_ESA_POLICY',
                '"ODMS_TEXT_MIN_DOCUMENTS"' : 1,
                '"ESAS_MIN_ITEMS"' : 1}

ctx_settings = {'COMMENTS':
                'TEXT(POLICY_NAME:DMDEMO_ESA_POLICY) (TOKEN_TYPE:STEM)' }
```

```
# Create an oml ESA model object.
esa_mod = oml.esa(**odm_settings)

# Fit the ESA model according to the training data and parameter settings.
esa_mod = esa_mod.fit(train_dat, case_id = 'ID',
                      ctx_settings = ctx_settings)

# Show model details.
esa_mod

# Use the model to make predictions on test data.
esa_mod.predict(test_dat,
                supplemental_cols = test_dat[:, ['ID', 'COMMENTS']])

esa_mod.transform(test_dat,
                  supplemental_cols = test_dat[:, ['ID', 'COMMENTS']],
                  topN = 2).sort_values(by = ['ID'])

esa_mod.feature_compare(test_dat,
                       compare_cols = 'COMMENTS',
                       supplemental_cols = ['ID'])

esa_mod.feature_compare(test_dat,
                       compare_cols = ['COMMENTS', 'YEAR'],
                       supplemental_cols = ['ID'])

# Change the setting parameter and refit the model.
new_setting = {'ESAS_VALUE_THRESHOLD': '0.01',
               'ODMS_TEXT_MAX_FEATURES': '2',
               'ESAS_TOPN_FEATURES': '2'}
esa_mod.set_params(**new_setting).fit(train_dat, 'ID', case_id = 'ID',
                                       ctx_settings = ctx_settings)

cur = cursor()
cur.execute("Begin ctx_ddl.drop_policy('DMDEMO_ESA_POLICY'); End;")
cur.close()
```

Listing for This Example

```
>>> import oml
>>> from oml import cursor
>>> import pandas as pd
>>>
>>> # Create training data and test data.
... dat = oml.push(pd.DataFrame(
...     {'COMMENTS':['Aids in Africa: Planning for a long war',
...                 'Mars rover maneuvers for rim shot',
...                 'Mars express confirms presence of water at Mars south pole',
...                 'NASA announces major Mars rover finding',
...                 'Drug access, Asia threat in focus at AIDS summit',
...                 'NASA Mars Odyssey THEMIS image: typical crater',
...                 'Road blocks for Aids'],
...     'YEAR':['2017', '2018', '2017', '2017', '2018', '2018', '2018'],
...     'ID':[1,2,3,4,5,6,7]})).split(ratio=(0.7,0.3), seed = 1234)
>>> train_dat = dat[0]
```



```

>>> test_dat = dat[1]
>>>
>>> # Specify settings.
... cur = cursor()
>>> cur.execute("Begin ctx_ddl.create_policy('DMDEMO_ESA_POLICY'); End;")
>>> cur.close()
>>>
>>> odm_settings = {'odms_text_policy_name': 'DMDEMO_ESA_POLICY',
...                 '"ODMS_TEXT_MIN_DOCUMENTS"': 1,
...                 '"ESAS_MIN_ITEMS"': 1}
>>>
>>> ctx_settings = {'COMMENTS':
...                 'TEXT(POLICY_NAME:DMDEMO_ESA_POLICY) (TOKEN_TYPE:STEM)'}
>>>
>>> # Create an oml ESA model object.
... esa_mod = oml.esa(**odm_settings)
>>>
>>> # Fit the ESA model according to the training data and parameter settings.
... esa_mod = esa_mod.fit(train_dat, case_id = 'ID',
...                        ctx_settings = ctx_settings)
>>>
>>> # Show model details.
... esa_mod

```

Algorithm Name: Explicit Semantic Analysis

Mining Function: FEATURE_EXTRACTION

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_EXPLICIT_SEMANTIC_ANALYS
1	ESAS_MIN_ITEMS	1
2	ESAS_TOPN_FEATURES	1000
3	ESAS_VALUE_THRESHOLD	.00000001
4	ODMS_DETAILS	ODMS_ENABLE
5	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
6	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
7	ODMS_TEXT_MAX_FEATURES	300000
8	ODMS_TEXT_MIN_DOCUMENTS	1
9	ODMS_TEXT_POLICY_NAME	DMDEMO_ESA_POLICY
10	PREP_AUTO	ON

Global Statistics:

	attribute name	attribute value
0	NUM_ROWS	4

Attributes:

COMMENTS
YEAR

Partition: NO

Features:

	FEATURE_ID	ATTRIBUTE_NAME	ATTRIBUTE_VALUE	COEFFICIENT
0	1	COMMENTS.AFRICA	None	0.342997

```

1          1          COMMENTS.AIDS          None          0.171499
2          1          COMMENTS.LONG          None          0.342997
3          1          COMMENTS.PLANNING      None          0.342997
...      ...      ...      ...      ...
24         6          COMMENTS.ODYSSEY      None          0.282843
25         6          COMMENTS.THEMIS       None          0.282843
26         6          COMMENTS.TYPICAL      None          0.282843
27         6          YEAR                   2018         0.707107

>>> # Use the model to make predictions on test data.
...  esa_mod.predict(test_dat,
...                   supplemental_cols = test_dat[:, ['ID', 'COMMENTS']])
   ID          COMMENTS  FEATURE_ID
0   4  NASA announces major Mars rover finding      3
1   6  NASA Mars Odyssey THEMIS image: typical crater  2
2   7          Road blocks for Aids                  5
>>>
>>> esa_mod.transform(test_dat,
...                   supplemental_cols = test_dat[:, ['ID', 'COMMENTS']],
...                   topN = 2).sort_values(by = ['ID'])
   ID          COMMENTS  TOP_1  TOP_1_VAL \
0   4  NASA announces major Mars rover finding      3  0.647065
1   6  NASA Mars Odyssey THEMIS image: typical crater  2  0.766237
2   7          Road blocks for Aids                  5  0.759125

   TOP_2  TOP_2_VAL
0       1  0.590565
1       2  0.616672
2       2  0.632604
>>>
>>> esa_mod.feature_compare(test_dat,
...                          compare_cols = 'COMMENTS',
...                          supplemental_cols = ['ID'])
   ID_A  ID_B  SIMILARITY
0     4    6    0.946469
1     4    7    0.871994
2     6    7    0.954565

>>> esa_mod.feature_compare(test_dat,
...                          compare_cols = ['COMMENTS', 'YEAR'],
...                          supplemental_cols = ['ID'])
   ID_A  ID_B  SIMILARITY
0     4    6    0.467644
1     4    7    0.377144
2     6    7    0.952857

>>> # Change the setting parameter and refit the model.
...  new_setting = {'ESAS_VALUE_THRESHOLD': '0.01',
...                 'ODMS_TEXT_MAX_FEATURES': '2',
...                 'ESAS_TOPN_FEATURES': '2'}
>>> esa_mod.set_params(**new_setting).fit(train_dat, case_id = 'ID',
...                                       ctx_settings = ctx_settings)

```

Algorithm Name: Explicit Semantic Analysis

```
Mining Function: FEATURE_EXTRACTION
```

```
Settings:
```

	setting name	setting value
0	ALGO_NAME	ALGO_EXPLICIT_SEMANTIC_ANALYS
1	ESAS_MIN_ITEMS	1
2	ESAS_TOPN_FEATURES	2
3	ESAS_VALUE_THRESHOLD	0.01
4	ODMS_DETAILS	ODMS_ENABLE
5	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
6	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
7	ODMS_TEXT_MAX_FEATURES	2
8	ODMS_TEXT_MIN_DOCUMENTS	1
9	ODMS_TEXT_POLICY_NAME	DMDEMO_ESA_POLICY
10	PREP_AUTO	ON

```
Global Statistics:
```

	attribute name	attribute value
0	NUM_ROWS	4

```
Attributes:
```

```
COMMENTS  
YEAR
```

```
Partition: NO
```

```
Features:
```

	FEATURE_ID	ATTRIBUTE_NAME	ATTRIBUTE_VALUE	COEFFICIENT
0	1	COMMENTS.AIDS	None	0.707107
1	1	YEAR	2017	0.707107
2	2	COMMENTS.MARS	None	0.707107
3	2	YEAR	2018	0.707107
4	3	COMMENTS.MARS	None	0.707107
5	3	YEAR	2017	0.707107
6	5	COMMENTS.AIDS	None	0.707107
7	5	YEAR	2018	0.707107

```
>>>  
>>> cur = cursor()  
>>> cur.execute("Begin ctx_ddl.drop_policy('DMDEMO_ESA_POLICY'); End;")  
>>> cur.close()
```

9.12 Generalized Linear Model

The `oml.glm` class builds a Generalized Linear Model (GLM) model.

GLM models include and extend the class of linear models. They relax the restrictions on linear models, which are often violated in practice. For example, binary (yes/no or 0/1) responses do not have the same variance across classes.

GLM is a parametric modeling technique. Parametric models make assumptions about the distribution of the data. When the assumptions are met, parametric models can be more efficient than non-parametric models.

The challenge in developing models of this type involves assessing the extent to which the assumptions are met. For this reason, quality diagnostics are key to developing quality parametric models.

In addition to the classical weighted least squares estimation for linear regression and iteratively re-weighted least squares estimation for logistic regression, both solved through Cholesky decomposition and matrix inversion, Oracle Machine Learning GLM provides a conjugate gradient-based optimization algorithm that does not require matrix inversion and is very well suited to high-dimensional data. The choice of algorithm is handled internally and is transparent to the user.

GLM can be used to build classification or regression models as follows:

- **Classification:** Binary logistic regression is the GLM classification algorithm. The algorithm uses the logit link function and the binomial variance function.
- **Regression:** Linear regression is the GLM regression algorithm. The algorithm assumes no target transformation and constant variance over the range of target values.

The `oml.glm` class allows you to build two different types of models. Some arguments apply to classification models only and some to regression models only.

For information on the `oml.glm` class attributes and methods, invoke `help(oml.glm)` or see Oracle Machine Learning for Python API Reference.

Settings for a Generalized Linear Model

The following table lists the settings that apply to GLM models.

Table 9-10 Generalized Linear Model Settings

Setting Name	Setting Value	Description
CLAS_COST_TABLE_NAME	<i>table_name</i>	<p>The name of a table that stores a cost matrix for the algorithm to use in scoring the model. The cost matrix specifies the costs associated with misclassifications.</p> <p>The cost matrix table is user-created. The following are the column requirements for the table.</p> <ul style="list-style-type: none"> • Column Name: ACTUAL_TARGET_VALUE Data Type: Valid target data type • Column Name: PREDICTED_TARGET_VALUE Data Type: Valid target data type • Column Name: COST Data Type: NUMBER
CLAS_WEIGHTS_BALANCED	ON OFF	<p>Indicates whether the algorithm must create a model that balances the target distribution. This setting is most relevant in the presence of rare targets, as balancing the distribution may enable better average accuracy (average of per-class accuracy) instead of overall accuracy (which favors the dominant class). The default value is OFF.</p>

Table 9-10 (Cont.) Generalized Linear Model Settings


Setting Name	Setting Value	Description
CLAS_WEIGHTS_TABLE_NAME	<i>table_name</i>	The name of a table that stores weighting information for individual target values in GLM logistic regression models. The weights are used by the algorithm to bias the model in favor of higher weighted classes. The class weights table is user-created. The following are the column requirements for the table. <ul style="list-style-type: none"> Column Name: TARGET_VALUE Data Type: Valid target data type Column Name: CLASS_WEIGHT Data Type: NUMBER
GLMS_BATCH_ROWS	0 or a positive integer.	Number of rows in a batch used by the SGD solver. The value of this parameter sets the size of the batch for the SGD solver. An input of 0 triggers a data-driven batch size estimate. The default value is 2000.
GLMS_CONF_LEVEL	TO_CHAR(0 < numeric_expr < 1)	The confidence level for coefficient confidence intervals. The default confidence level is 0.95.
GLMS_CONV_TOLERANCE	The range is (0, 1) non-inclusive.	Convergence tolerance setting of the GLM algorithm. The default value is system-determined.
GLMS_FTR_GEN_METHOD	GLMS_FTR_GEN_CUBIC GLMS_FTR_GEN_QUADRATIC	Whether feature generation is cubic or quadratic. When you enable feature generation, the algorithm automatically chooses the most appropriate feature generation method based on the data.
GLMS_FTR_GENERATION	GLMS_FTR_GENERATION_ENABLED GLMS_FTR_GENERATION_DISABLED	Whether or not feature generation is enabled for GLM. By default, feature generation is not enabled.
		 Note: Note: Feature generation can only be enabled when feature selection is also enabled.
GLMS_FTR_SEL_CRIT	GLMS_FTR_SEL_AIC GLMS_FTR_SEL_ALPHA_INV GLMS_FTR_SEL_RIC GLMS_FTR_SEL_SBIC	Feature selection penalty criterion for adding a feature to the model. When feature selection is enabled, the algorithm automatically chooses the penalty criterion based on the data.
GLMS_FTR_SELECTION	GLMS_FTR_SELECTION_DISABLED	Enable or disable feature selection for GLM. By default, feature selection is not enabled.
GLMS_MAX_FEATURES	TO_CHAR(0 < numeric_expr <= 2000)	When feature selection is enabled, this setting specifies the maximum number of features that can be selected for the final model. By default, the algorithm limits the number of features to ensure sufficient memory.
GLMS_NUM_ITERATIONS	A positive integer.	Maximum number of iterations for the GLM algorithm. The default value is system-determined.

Table 9-10 (Cont.) Generalized Linear Model Settings

Setting Name	Setting Value	Description
GLMS_PRUNE_MODEL	GLMS_PRUNE_MODEL_ENABLE GLMS_PRUNE_MODEL_DISABLE	When feature selection is enabled, the algorithm automatically performs pruning based on the data.
GLMS_REFERENCE_CLASS_NAME	<i>target_value</i>	The target value used as the reference class in a binary logistic regression model. Probabilities are produced for the other class. By default, the algorithm chooses the value with the highest prevalence (the most cases) for the reference class.
GLMS_RIDGE_REGRESSION	GLMS_RIDGE_REG_ENABLE GLMS_RIDGE_REG_DISABLE	Enable or disable ridge regression. Ridge applies to both regression and classification machine learning functions. When ridge is enabled, prediction bounds are not produced by the PREDICTION_BOUNDS SQL function.
GLMS_RIDGE_VALUE	TO_CHAR(<i>numeric_expr</i> > 0)	The value of the ridge parameter. Use this setting only when you have configured the algorithm to use ridge regression. If ridge regression is enabled internally by the algorithm, then the ridge parameter is determined by the algorithm.
GLMS_ROW_DIAGNOSTICS	GLMS_ROW_DIAG_ENABLE GLMS_ROW_DIAG_DISABLE	Enable or disable row diagnostics. By default, row diagnostics are disabled.
GLMS_SOLVER	GLMS_SOLVER_CHOL GLMS_SOLVER_LBFGS_ADMIN GLMS_SOLVER_QR GLMS_SOLVER_SGD	Specifies the GLM solver. You cannot select the solver if GLMS_FTR_SELECTION setting is enabled. The default value is system determined. The GLMS_SOLVER_CHOL solver uses Cholesky decomposition. The GLMS_SOLVER_SGD solver uses stochastic gradient descent.
GLMS_SPARSE_SOLVER	GLMS_SPARSE_SOLVER_ENABLE GLMS_SPARSE_SOLVER_DISABLE	Enable or disable the use of a sparse solver if it is available. The default value is GLMS_SPARSE_SOLVER_DISABLE.
ODMS_ROW_WEIGHT_COLUMN_NAME	<i>column_name</i>	The name of a column in the training data that contains a weighting factor for the rows. The column datatype must be NUMBER. You can use row weights as a compact representation of repeated rows, as in the design of experiments where a specific configuration is repeated several times. You can also use row weights to emphasize certain rows during model construction. For example, to bias the model towards rows that are more recent and away from potentially obsolete data.

 **See Also:**

- [About Model Settings](#)
- [Shared Settings](#)

Example 9-12 Using the oml.glm Class

This example demonstrates the use of various methods of the `oml.glm` class. In the listing for this example, some of the output is not shown as indicated by ellipses.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Petal_Width')
train_y = dat[0]['Petal_Width']
test_dat = dat[1]

# Specify settings.
setting = {'GLMS_SOLVER': 'dbms_data_mining.GLMS_SOLVER_QR'}

# Create a GLM model object.
glm_mod = oml.glm("regression", **setting)

# Fit the GLM model according to the training data and parameter
# settings.
glm_mod = glm_mod.fit(train_x, train_y)

# Show the model details.
glm_mod

# Use the model to make predictions on the test data.
glm_mod.predict(test_dat.drop('Petal_Width'),
                supplemental_cols = test_dat[:,
                ['Sepal_Length', 'Sepal_Width',
                 'Petal_Length', 'Species']])

# Return the prediction probability.
glm_mod.predict(test_dat.drop('Petal_Width'),
                supplemental_cols = test_dat[:,
                ['Sepal_Length', 'Sepal_Width',
```

```
        'Petal_Length', 'Species']],
        proba = True)

glm_mod.score(test_dat.drop('Petal_Width'),
              test_dat[:, ['Petal_Width']])

# Change the parameter setting and refit the model.
new_setting = {'GLMS_SOLVER': 'GLMS_SOLVER_SGD'}
glm_mod.set_params(**new_setting).fit(train_x, train_y)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                            'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                          {0: 'setosa', 1: 'versicolor',
...                           2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Petal_Width')
>>> train_y = dat[0]['Petal_Width']
>>> test_dat = dat[1]
>>>
>>> # Specify settings.
... setting = {'GLMS_SOLVER': 'dbms_data_mining.GLMS_SOLVER_QR'}
>>>
>>> # Create a GLM model object.
... glm_mod = oml.glm("regression", **setting)
>>>
>>> # Fit the GLM model according to the training data and parameter
... # settings.
>>> glm_mod = glm_mod.fit(train_x, train_y)
>>>
>>> # Show the model details.
... glm_mod
```

Algorithm Name: Generalized Linear Model

Mining Function: REGRESSION

Target: Petal_Width

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_GENERALIZED_LINEAR_MODEL
1	GLMS_CONF_LEVEL	.95
2	GLMS_FTR_GENERATION	GLMS_FTR_GENERATION_DISABLE
3	GLMS_FTR_SELECTION	GLMS_FTR_SELECTION_DISABLE
4	GLMS_SOLVER	GLMS_SOLVER_QR
5	ODMS_DETAILS	ODMS_ENABLE
6	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
7	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
8	PREP_AUTO	ON

Computed Settings:

	setting name	setting value
0	GLMS_CONV_TOLERANCE	.000005000000000000000004
1	GLMS_NUM_ITERATIONS	30
2	GLMS_RIDGE_REGRESSION	GLMS_RIDGE_REG_ENABLE

Global Statistics:

	attribute name	attribute value
0	ADJUSTED_R_SQUARE	0.949634
1	AIC	-363.888
2	COEFF_VAR	14.6284
3	CONVERGED	YES
4	CORRECTED_TOTAL_DF	103
5	CORRECTED_TOT_SS	58.4565
6	DEPENDENT_MEAN	1.15577
7	ERROR_DF	98
8	ERROR_MEAN_SQUARE	0.028585
9	ERROR_SUM_SQUARES	2.80131
10	F_VALUE	389.405
11	GMSEP	0.030347
12	HOCKING_SP	0.000295
13	J_P	0.030234
14	MODEL_DF	5
15	MODEL_F_P_VALUE	0
16	MODEL_MEAN_SQUARE	11.131
17	MODEL_SUM_SQUARES	55.6552
18	NUM_PARAMS	6
19	NUM_ROWS	104
20	RANK_DEFICIENCY	0
21	ROOT_MEAN_SQ	0.16907
22	R_SQ	0.952079
23	SBIC	-348.021
24	VALID_COVARIANCE_MATRIX	YES

[1 rows x 25 columns]

Attributes:

Petal_Length
Sepal_Length
Sepal_Width
Species

Partition: NO

Coefficients:

	name	level	estimate
0	(Intercept)	None	-0.600603
1	Petal_Length	None	0.239775
2	Sepal_Length	None	-0.078338
3	Sepal_Width	None	0.253996
4	Species	versicolor	0.652420
5	Species	virginica	1.010438

Fit Details:

	name	value
0	ADJUSTED_R_SQUARE	9.496338e-01
1	AIC	-3.638876e+02
2	COEFF_VAR	1.462838e+01
3	CORRECTED_TOTAL_DF	1.030000e+02
...		
21	ROOT_MEAN_SQ	1.690704e-01
22	R_SQ	9.520788e-01
23	SBIC	-3.480213e+02
24	VALID_COVARIANCE_MATRIX	1.000000e+00

Rank:

6

Deviance:

2.801309

AIC:

-364

Null Deviance:

58.456538

DF Residual:

98.0

DF Null:

103.0

Converged:

True

>>>

```
>>> # Use the model to make predictions on the test data.  
... glm_mod.predict(test_dat.drop('Petal_Width'),
```

```

...         supplemental_cols = test_dat[:,
...         ['Sepal_Length', 'Sepal_Width',
...         'Petal_Length', 'Species'])
    Sepal_Length  Sepal_Width  Petal_Length  Species  PREDICTION
0                4.9          3.0          1.4    setosa    0.113215
1                4.9          3.1          1.5    setosa    0.162592
2                4.8          3.4          1.6    setosa    0.270602
3                5.8          4.0          1.2    setosa    0.248752
...         ...           ...           ...     ...     ...
42               6.7          3.3          5.7  virginica  2.89876
43               6.7          3.0          5.2  virginica  1.893790
44               6.5          3.0          5.2  virginica  1.909457
45               5.9          3.0          5.1  virginica  1.932483

>>> # Return the prediction probability.
... glm_mod.predict(test_dat.drop('Petal_Width'),
...                 supplemental_cols = test_dat[:,
...                 ['Sepal_Length', 'Sepal_Width',
...                 'Petal_Length', 'Species']]),
...                 proba = True)
    Sepal_Length  Sepal_Width  Species  PREDICTION
0                4.9          3.0    setosa    0.113215
1                4.9          3.1    setosa    0.162592
2                4.8          3.4    setosa    0.270602
3                5.8          4.0    setosa    0.248752
...         ...           ...     ...     ...
42               6.7          3.3  virginica  2.089876
43               6.7          3.0  virginica  1.893790
44               6.5          3.0  virginica  1.909457
45               5.9          3.0  virginica  1.932483

>>>
>>> glm_mod.score(test_dat.drop('Petal_Width'),
...               test_dat[:, ['Petal_Width']])
0.951252
>>>
>>> # Change the parameter setting and refit the model.
... new_setting = {'GLMS_SOLVER': 'GLMS_SOLVER_SGD'}
>>> glm_mod.set_params(**new_setting).fit(train_x, train_y)

```

Algorithm Name: Generalized Linear Model

Mining Function: REGRESSION

Target: Petal_Width

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_GENERALIZED_LINEAR_MODEL
1	GLMS_CONF_LEVEL	.95
2	GLMS_FTR_GENERATION	GLMS_FTR_GENERATION_DISABLE
3	GLMS_FTR_SELECTION	GLMS_FTR_SELECTION_DISABLE
4	GLMS_SOLVER	GLMS_SOLVER_SGD
5	ODMS_DETAILS	ODMS_ENABLE
6	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
7	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
8	PREP_AUTO	ON

Computed Settings:

	setting name	setting value
0	GLMS_BATCH_ROWS	2000
1	GLMS_CONV_TOLERANCE	.0001
2	GLMS_NUM_ITERATIONS	500
3	GLMS_RIDGE_REGRESSION	GLMS_RIDGE_REG_ENABLE
4	GLMS_RIDGE_VALUE	.01

Global Statistics:

	attribute name	attribute value
0	ADJUSTED_R_SQUARE	0.94175
1	AIC	-348.764
2	COEFF_VAR	15.7316
3	CONVERGED	NO
4	CORRECTED_TOTAL_DF	103
5	CORRECTED_TOT_SS	58.4565
6	DEPENDENT_MEAN	1.15577
7	ERROR_DF	98
8	ERROR_MEAN_SQUARE	0.033059
9	ERROR_SUM_SQUARES	3.23979
10	F_VALUE	324.347
11	GMSEP	0.035097
12	HOCKING_SP	0.000341
13	J_P	0.034966
14	MODEL_DF	5
15	MODEL_F_P_VALUE	0
16	MODEL_MEAN_SQUARE	10.7226
17	MODEL_SUM_SQUARES	53.613
18	NUM_PARAMS	6
19	NUM_ROWS	104
20	RANK_DEFICIENCY	0
21	ROOT_MEAN_SQ	0.181821
22	R_SQ	0.944578
23	SBIC	-332.898
24	VALID_COVARIANCE_MATRIX	NO

[1 rows x 25 columns]

Attributes:

Petal_Length
Sepal_Length
Sepal_Width
Species

Partition: NO

Coefficients:

	name	level	estimate
0	(Intercept)	None	-0.338046
1	Petal_Length	None	0.378658
2	Sepal_Length	None	-0.084440
3	Sepal_Width	None	0.137150
4	Species	versicolor	0.151916
5	Species	virginica	0.337535

Fit Details:

	name	value
0	ADJUSTED_R_SQUARE	9.417502e-01
1	AIC	-3.487639e+02
2	COEFF_VAR	1.573164e+01
3	CORRECTED_TOTAL_DF	1.030000e+02
...
21	ROOT_MEAN_SQ	1.818215e-01
22	R_SQ	9.445778e-01
23	SBIC	-3.328975e+02
24	VALID_COVARIANCE_MATRIX	0.000000e+00

Rank:

6

Deviance:

3.239787

AIC:

-349

Null Deviance:

58.456538

Prior Weights:

1

DF Residual:

98.0

DF Null:

103.0

Converged:

False

9.13 k-Means

The `oml.km` class uses the *k*-Means (KM) algorithm, which is a hierarchical, distance-based clustering algorithm that partitions data into a specified number of clusters.

The algorithm has the following features:

- Several distance functions: Euclidean, Cosine, and Fast Cosine distance functions. The default is Euclidean.

- For each cluster, the algorithm returns the centroid, a histogram for each attribute, and a rule describing the hyperbox that encloses the majority of the data assigned to the cluster. The centroid reports the mode for categorical attributes and the mean and variance for numeric attributes.

For information on the `oml.km` class attributes and methods, invoke `help(oml.km)` or see Oracle Machine Learning for Python API Reference.

Settings for a k-Means Model

The following table lists the settings that apply to KM models.

Table 9-11 k-Means Model Settings

Setting Name	Setting Value	Description
CLUS_NUM_CLUSTERS	<code>TO_CHAR(numeric_expr >= 1)</code>	The maximum number of leaf clusters generated by the algorithm. The algorithm produces the specified number of clusters unless there are fewer distinct data points. The default value is 10.
KMNS_CONV_TOLERANCE	<code>TO_CHAR(0 < numeric_expr < 1)</code>	Minimum Convergence Tolerance for k-Means. The algorithm iterates until the minimum Convergence Tolerance is satisfied or until the maximum number of iterations, specified in <code>KMNS_ITERATIONS</code> , is reached. Decreasing the Convergence Tolerance produces a more accurate solution but may result in longer run times. The default Convergence Tolerance is 0.001.
KMNS_DETAILS	<code>KMNS_DETAILS_ALL</code> <code>KMNS_DETAILS_HIERARCHY</code> <code>KMNS_DETAILS_NONE</code>	Determines the level of cluster detail that is computed during the build. <code>KMNS_DETAILS_ALL</code> : Cluster hierarchy, record counts, descriptive statistics (means, variances, modes, histograms, and rules) are computed. <code>KMNS_DETAILS_HIERARCHY</code> : Cluster hierarchy and cluster record counts are computed. This is the default value. <code>KMNS_DETAILS_NONE</code> : No cluster details are computed. Only the scoring information is persisted.
KMNS_DISTANCE	<code>KMNS_COSINE</code> <code>KMNS_EUCLIDEAN</code>	Distance function for k-Means. The default distance function is <code>KMNS_EUCLIDEAN</code> .
KMNS_ITERATIONS	<code>TO_CHAR(positive_numeric_expr)</code>	Maximum number of iterations for k-Means. The algorithm iterates until either the maximum number of iterations is reached or the minimum Convergence Tolerance, specified in <code>KMNS_CONV_TOLERANCE</code> , is satisfied. The default number of iterations is 20.
KMNS_MIN_PCT_ATTR_SUPP ORT	<code>TO_CHAR(0 <= numeric_expr <= 1)</code>	Minimum percentage of attribute values that must be non-null in order for the attribute to be included in the rule description for the cluster. If the data is sparse or includes many missing values, a minimum support that is too high can cause very short rules or even empty rules. The default minimum support is 0.1.

Table 9-11 (Cont.) k-Means Model Settings

Setting Name	Setting Value	Description
KMNS_NUM_BINS	TO_CHAR(<i>numeric_expr</i> > 0)	Number of bins in the attribute histogram produced by <i>k</i> -Means. The bin boundaries for each attribute are computed globally on the entire training data set. The binning method is equi-width. All attributes have the same number of bins with the exception of attributes with a single value, which have only one bin. The default number of histogram bins is 11.
KMNS_RANDOM_SEED	Non-negative integer	Controls the seed of the random generator used during the <i>k</i> -Means initialization. It must be a non-negative integer value. The default value is 0.
KMNS_SPLIT_CRITERION	KMNS_SIZE KMNS_VARIANCE	Split criterion for <i>k</i> -Means. The split criterion controls the initialization of new <i>k</i> -Means clusters. The algorithm builds a binary tree and adds one new cluster at a time. When the split criterion is based on size, the new cluster is placed in the area where the largest current cluster is located. When the split criterion is based on the variance, the new cluster is placed in the area of the most spread-out cluster. The default split criterion is the KMNS_VARIANCE.

 **See Also:**

- [About Model Settings](#)
- [Shared Settings](#)

Example 9-13 Using the oml.km Class

This example creates a KM model and uses methods of it. In the listing for this example, some of the output is not shown as indicated by ellipses.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
```

```
        oml.drop('IRIS')
    except:
        pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_dat = dat[0]
test_dat = dat[1]

# Specify settings.
setting = {'kmns_iterations': 20}

# Create a KM model object and fit it.
km_mod = oml.km(n_clusters = 3, **setting).fit(train_dat)

# Show model details.
km_mod

# Use the model to make predictions on the test data.
km_mod.predict(test_dat,
               supplemental_cols =
                   test_dat[:, ['Sepal_Length', 'Sepal_Width',
                               'Petal_Length', 'Species']])
km_mod.predict_proba(test_dat,
                    supplemental_cols =
                        test_dat[:, ['Species']]).sort_values(by =
                                                                ['Species', 'PROBABILITY_OF_3'])

km_mod.transform(test_dat)

km_mod.score(test_dat)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                             'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...     oml.drop('IRIS')
... except:
...     pass
```



```

>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_dat = dat[0]
>>> test_dat = dat[1]
>>>
>>> # Specify settings.
... setting = {'kmns_iterations': 20}
>>>
>>> # Create a KM model object and fit it.
... km_mod = omlkm(n_clusters = 3, **setting).fit(train_dat)
>>>
>>> # Show model details.
... km_mod

```

Algorithm Name: K-Means

Mining Function: CLUSTERING

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_KMEANS
1	CLUS_NUM_CLUSTERS	3
2	KMNS_CONV_TOLERANCE	.001
3	KMNS_DETAILS	KMNS_DETAILS_HIERARCHY
4	KMNS_DISTANCE	KMNS_EUCLIDEAN
5	KMNS_ITERATIONS	20
6	KMNS_MIN_PCT_ATTR_SUPPORT	.1
7	KMNS_NUM_BINS	11
8	KMNS_RANDOM_SEED	0
9	KMNS_SPLIT_CRITERION	KMNS_VARIANCE
10	ODMS_DETAILS	ODMS_ENABLE
11	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
12	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
13	PREP_AUTO	ON

Global Statistics:

	attribute name	attribute value
0	CONVERGED	YES
1	NUM_ROWS	104.0

Attributes: Petal_Length

Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Clusters:

CLUSTER_ID	ROW_CNT	PARENT_CLUSTER_ID	TREE_LEVEL	DISPERSION
------------	---------	-------------------	------------	------------

0	1	104	NaN	1	0.986153
1	2	68	1.0	2	1.102147
2	3	36	1.0	2	0.767052
3	4	37	2.0	3	1.015669
4	5	31	2.0	3	1.205363

Taxonomy:

	PARENT_CLUSTER_ID	CHILD_CLUSTER_ID	
0	1	2.0	
1	1	3.0	
2	2	4.0	
3	2	5.0	
4	3	NaN	
5	4	NaN	
6	5	NaN	

Leaf Cluster Counts:

	CLUSTER_ID	CNT
0	3	50
1	4	53
2	5	47

>>>

>>> # Use the model to make predictions on the test data.

```
... km_mod.predict(test_dat, ['Sepal_Length', 'Sepal_Width',
...                          'Petal_Length', 'Species'])
```

	Sepal_Length	Sepal_Width	Petal_Length	Species	CLUSTER_ID
0	4.9	3.0	1.4	setosa	3
1	4.9	3.1	1.5	setosa	3
2	4.8	3.4	1.6	setosa	3
3	5.8	4.0	1.2	setosa	3
...
38	6.4	2.8	5.6	virginica	5
39	6.9	3.1	5.4	virginica	5
40	6.7	3.1	5.6	virginica	5
41	5.8	2.7	5.1	virginica	5

>>>

```
>>> km_mod.predict_proba(test_dat,
...                       supplemental_cols =
...                       test_dat[:, ['Species']]).sort_values(by =
...                       ['Species', 'PROBABILITY_OF_3'])
```

	Species	PROBABILITY_OF_3	PROBABILITY_OF_4	PROBABILITY_OF_5
0	setosa	0.791267	0.208494	0.000240
1	setosa	0.971498	0.028350	0.000152
2	setosa	0.981020	0.018499	0.000481
3	setosa	0.981907	0.017989	0.000104
...
42	virginica	0.000655	0.316671	0.682674
43	virginica	0.001036	0.413744	0.585220
44	virginica	0.001036	0.413744	0.585220
45	virginica	0.002452	0.305021	0.692527

>>>

```
>>> km_mod.transform(test_dat)
```

	CLUSTER_DISTANCE
0	1.050234

```

1          0.859817
2          0.321065
3          1.427080
...          ...
42         0.837757
43         0.479313
44         0.448562
45         1.123587
>>>
>>> km_mod.score(test_dat)
-47.487712

```

9.14 Naive Bayes

The `oml.nb` class creates a Naive Bayes (NB) model for classification.

The Naive Bayes algorithm is based on conditional probabilities. Naive Bayes looks at the historical data and calculates conditional probabilities for the target values by observing the frequency of attribute values and of combinations of attribute values.

Naive Bayes assumes that each predictor is conditionally independent of the others. (Bayes' Theorem requires that the predictors be independent.)

For information on the `oml.nb` class attributes and methods, invoke `help(oml.nb)` or see Oracle Machine Learning for Python API Reference.

Settings for a Naive Bayes Model

The following table lists the settings that apply to NB models.

Table 9-12 Naive Bayes Model Settings

Setting Name	Setting Value	Description
CLAS_COST_TABLE_NAME	<i>table_name</i>	<p>The name of a table that stores a cost matrix for the algorithm to use in building the model. The cost matrix specifies the costs associated with misclassifications.</p> <p>The cost matrix table is user-created. The following are the column requirements for the table.</p> <ul style="list-style-type: none"> Column Name: ACTUAL_TARGET_VALUE Data Type: Valid target data type Column Name: PREDICTED_TARGET_VALUE Data Type: Valid target data type Column Name: COST Data Type: NUMBER
CLAS_MAX_SUP_BINS	<i>2 <= a number <= 2147483647</i>	<p>Specifies the maximum number of bins for each attribute. The default value is 32.</p>
CLAS_PRIORS_TABLE_NAME	<i>table_name</i>	<p>The name of a table that stores prior probabilities to offset differences in distribution between the build data and the scoring data.</p> <p>The priors table is user-created. The following are the column requirements for the table.</p> <ul style="list-style-type: none"> Column Name: TARGET_VALUE Data Type: Valid target data type Column Name: PRIOR_PROBABILITY Data Type: NUMBER

Table 9-12 (Cont.) Naive Bayes Model Settings

Setting Name	Setting Value	Description
CLAS_WEIGHTS_BALANCED	ON OFF	Indicates whether the algorithm must create a model that balances the target distribution. This setting is most relevant in the presence of rare targets, as balancing the distribution may enable better average accuracy (average of per-class accuracy) instead of overall accuracy (which favors the dominant class). The default value is OFF.
NABS_PAIRWISE_THRESHOLD	TO_CHAR(0 <= numeric_expr <= 1)	Value of the pairwise threshold for the NB algorithm. The default value is 0.
NABS_SINGLETON_THRESHOLD	TO_CHAR(0 <= numeric_expr <= 1)	Value of the singleton threshold for the NB algorithm. The default value is 0.

 **See Also:**

- [About Model Settings](#)
- [Shared Settings](#)

Example 9-14 Using the oml.nb Class

This example creates an NB model and uses some of the methods of the `oml.nb` class.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop(table = 'NB_PRIOR_PROBABILITY_DEMO')
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()

train_x = dat[0].drop('Species')
```

```
train_y = dat[0]['Species']
test_dat = dat[1]

# User specified settings.
setting = {'CLAS_WEIGHTS_BALANCED': 'ON'}

# Create an oml NB model object.
nb_mod = oml.nb(**setting)

# Fit the NB model according to the training data and parameter
# settings.
nb_mod = nb_mod.fit(train_x, train_y)

# Show details of the model.
nb_mod

# Create a priors table in the database.
priors = {'setosa': 0.2, 'versicolor': 0.3, 'virginica': 0.5}
priors = oml.create(pd.DataFrame(list(priors.items()),
                                columns = ['TARGET_VALUE',
                                           'PRIOR_PROBABILITY']),
                  table = 'NB_PRIOR_PROBABILITY_DEMO')

# Change the setting parameter and refit the model
# with a user-defined prior table.
new_setting = {'CLAS_WEIGHTS_BALANCED': 'OFF'}
nb_mod = nb_mod.set_params(**new_setting).fit(train_x,
                                              train_y,
                                              priors = priors)

nb_mod

# Use the model to make predictions on test data.
nb_mod.predict(test_dat.drop('Species'),
              supplemental_cols = test_dat[:, ['Sepal_Length',
                                              'Sepal_Width',
                                              'Petal_Length',
                                              'Species']])

# Return the prediction probability.
nb_mod.predict(test_dat.drop('Species'),
              supplemental_cols = test_dat[:, ['Sepal_Length',
                                              'Sepal_Width',
                                              'Species']],
              proba = True)

# Return the top two most influential attributes of the highest
# probability class.
nb_mod.predict(test_dat.drop('Species'),
              supplemental_cols = test_dat[:, ['Sepal_Length',
                                              'Sepal_Width',
                                              'Petal_Length',
                                              'Species']],
              topN_attrs = 2)

# Make predictions and return the probability for each class
# on new data.
```

```
nb_mod.predict_proba(test_dat.drop('Species'),
                    supplemental_cols = test_dat[:,
                    ['Sepal_Length',
                    'Species']]).sort_values(by =
                    ['Sepal_Length',
                    'Species',
                    'PROBABILITY_OF_setosa',
                    'PROBABILITY_OF_versicolor'])

# Make predictions on new data and return the mean accuracy.
nb_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                            'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                          {0: 'setosa', 1: 'versicolor',
...                           2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...     oml.drop(table = 'NB_PRIOR_PROBABILITY_DEMO')
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
>>> dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
>>>
>>> # User specified settings.
... setting = {'CLAS_WEIGHTS_BALANCED': 'ON'}
>>>
>>> # Create an oml NB model object.
... nb_mod = oml.nb(**setting)
>>>
>>> # Fit the NB model according to the training data and parameter
... # settings.
>>> nb_mod = nb_mod.fit(train_x, train_y)
>>>
>>> # Show details of the model.
... nb_mod
```

Algorithm Name: Naive Bayes

Mining Function: CLASSIFICATION

Target: Species

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_NAIVE_BAYES
1	CLAS_WEIGHTS_BALANCED	ON
2	NABS_PAIRWISE_THRESHOLD	0
3	NABS_SINGLETON_THRESHOLD	0
4	ODMS_DETAILS	ODMS_ENABLE
5	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
6	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
7	PREP_AUTO	ON

Global Statistics:

	attribute name	attribute value
0	NUM_ROWS	104

Attributes:

Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

Priors:

	TARGET_NAME	TARGET_VALUE	PRIOR_PROBABILITY	COUNT
0	Species	setosa	0.333333	36
1	Species	versicolor	0.333333	35
2	Species	virginica	0.333333	33

Conditionals:

	TARGET_NAME	TARGET_VALUE	ATTRIBUTE_NAME	ATTRIBUTE_SUBNAME	
0	Species	setosa	Petal_Length	None	(;
1.05]					
1	Species	setosa	Petal_Length	None	(1.05; 1.2]
2	Species	setosa	Petal_Length	None	(1.2; 1.35]
3	Species	setosa	Petal_Length	None	(1.35; 1.45]
...
152	Species	virginica	Sepal_Width	None	(3.25; 3.35]
153	Species	virginica	Sepal_Width	None	(3.35; 3.45]
154	Species	virginica	Sepal_Width	None	(3.55; 3.65]
155	Species	virginica	Sepal_Width	None	(3.75; 3.85]

	CONDITIONAL_PROBABILITY	COUNT
0	0.027778	1
1	0.027778	1

```

2          0.083333      3
3          0.277778     10
...          ...      ...
152         0.030303      1
153         0.060606      2
154         0.030303      1
155         0.060606      2

```

```
[156 rows x 7 columns]
```

```

>>> # Create a priors table in the database.
... priors = {'setosa': 0.2, 'versicolor': 0.3, 'virginica': 0.5}
>>> priors = oml.create(pd.DataFrame(list(priors.items()),
...                                  columns = ['TARGET_VALUE',
...                                           'PRIOR_PROBABILITY']),
...                    table = 'NB_PRIOR_PROBABILITY_DEMO')
>>>
>>> # Change the setting parameter and refit the model
... # with a user-defined prior table.
... new_setting = {'CLAS_WEIGHTS_BALANCED': 'OFF'}
>>> nb_mod = nb_mod.set_params(**new_setting).fit(train_x,
...                                               train_y,
...                                               priors = priors)
>>> nb_mod

```

Algorithm Name: Naive Bayes

Mining Function: CLASSIFICATION

Target: Species

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_NAIVE_BAYES
1	CLAS_PRIORS_TABLE_NAME	"OML_USER"."NB_PRIOR_PROBABILITY_DEMO"
2	CLAS_WEIGHTS_BALANCED	OFF
3	NABS_PAIRWISE_THRESHOLD	0
4	NABS_SINGLETON_THRESHOLD	0
5	ODMS_DETAILS	ODMS_ENABLE
6	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
7	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
8	PREP_AUTO	ON

Global Statistics:

	attribute name	attribute value
0	NUM_ROWS	104

Attributes:

```

Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

```

Partition: NO

Priors:

	TARGET_NAME	TARGET_VALUE	PRIOR_PROBABILITY	COUNT
0	Species	setosa	0.2	36
1	Species	versicolor	0.3	35
2	Species	virginica	0.5	33

Conditionals:

	TARGET_NAME	TARGET_VALUE	ATTRIBUTE_NAME	ATTRIBUTE_SUBNAME	
0	Species	setosa	Petal_Length	None	(; 1.05]
1	Species	setosa	Petal_Length	None	(1.05; 1.2]
2	Species	setosa	Petal_Length	None	(1.2; 1.35]
3	Species	setosa	Petal_Length	None	(1.35; 1.45]
...
152	Species	virginica	Sepal_Width	None	(3.25; 3.35]
153	Species	virginica	Sepal_Width	None	(3.35; 3.45]
154	Species	virginica	Sepal_Width	None	(3.55; 3.65]
155	Species	virginica	Sepal_Width	None	(3.75; 3.85]

	CONDITIONAL_PROBABILITY	COUNT
0	0.027778	1
1	0.027778	1
2	0.083333	3
3	0.277778	10
...
152	0.030303	1
153	0.060606	2
154	0.030303	1
155	0.060606	2

[156 rows x 7 columns]

```
>>> # Use the model to make predictions on test data.
... nb_mod.predict(test_dat.drop('Species'),
...                 supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Petal_Length',
...                                                 'Species']])
...
   Sepal_Length  Sepal_Width  Petal_Length  Species  PREDICTION
0             4.9           3.0           1.4   setosa   setosa
1             4.9           3.1           1.5   setosa   setosa
2             4.8           3.4           1.6   setosa   setosa
3             5.8           4.0           1.2   setosa   setosa
...           ...           ...           ...   ...         ...
42            6.7           3.3           5.7  virginica  virginica
43            6.7           3.0           5.2  virginica  virginica
44            6.5           3.0           5.2  virginica  virginica
45            5.9           3.0           5.1  virginica  virginica
```

```
>>> # Return the prediction probability.
>>> nb_mod.predict(test_dat.drop('Species'),
...                 supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Species']],
...                 proba = True)
```

```

    Sepal_Length Sepal_Width Species PREDICTION PROBABILITY
0          4.9         3.0    setosa    setosa      1.000000
1          4.9         3.1    setosa    setosa      1.000000
2          4.8         3.4    setosa    setosa      1.000000
3          5.8         4.0    setosa    setosa      1.000000
...          ...          ...          ...          ...
42         6.7         3.3  virginica  virginica    1.000000
43         6.7         3.0  virginica  virginica    0.953848
44         6.5         3.0  virginica  virginica    1.000000
45         5.9         3.0  virginica  virginica    0.932334

>>> # Return the top two most influential attributes of the highest
... # probability class.
>>> nb_mod.predict(test_dat.drop('Species'),
...                 supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Petal_Length',
...                                                 'Species']],
...                 topN_attrs = 2)
    Sepal_Length Sepal_Width Petal_Length Species PREDICTION \
0          4.9         3.0         1.4    setosa    setosa
1          4.9         3.1         1.5    setosa    setosa
2          4.8         3.4         1.6    setosa    setosa
3          5.8         4.0         1.2    setosa    setosa
... ..
42         6.7         3.3         5.7  virginica  virginica
43         6.7         3.0         5.2  virginica  virginica
44         6.5         3.0         5.2  virginica  virginica
45         5.9         3.0         5.1  virginica  virginica
TOP_N_ATTRIBUTES
0 <Details algorithm="Naive Bayes" class="setosa...
1 <Details algorithm="Naive Bayes" class="setosa...
2 <Details algorithm="Naive Bayes" class="setosa...
3 <Details algorithm="Naive Bayes" class="setosa...
...
42 <Details algorithm="Naive Bayes" class="virgin...
43 <Details algorithm="Naive Bayes" class="virgin...
44 <Details algorithm="Naive Bayes" class="virgin...
45 <Details algorithm="Naive Bayes" class="virgin...

>>> # Make predictions and return the probability for each class
... # on new data.
>>> nb_mod.predict_proba(test_dat.drop('Species'),
...                       supplemental_cols = test_dat[:,
...                                                 ['Sepal_Length',
...                                                 'Species']]).sort_values(by =
...                                                 ['Sepal_Length',
...                                                 'Species',
...                                                 'PROBABILITY_OF_setosa',
...                                                 'PROBABILITY_OF_versicolor'])
    Sepal_Length Species PROBABILITY_OF_SETOSA \
0          4.4    setosa      1.000000e+00
1          4.4    setosa      1.000000e+00
2          4.5    setosa      1.000000e+00
3          4.8    setosa      1.000000e+00
...          ...          ...

```

```

42          6.7  virginica          1.412132e-13
43          6.9  versicolor        5.295492e-20
44          6.9  virginica          5.295492e-20
45          7.0  versicolor        6.189014e-14

```

```

          PROBABILITY_OF_VERSICOLOR  PROBABILITY_OF_VIRGINICA
0          9.327306e-21              7.868301e-20
1          3.497737e-20              1.032715e-19
2          2.238553e-13              2.360490e-19
3          6.995487e-22              2.950617e-21
...          ...
42          4.741700e-13              1.000000e+00
43          1.778141e-07              9.999998e-01
44          2.963565e-20              1.000000e+00
45          4.156340e-01              5.843660e-01

```

```

>>> # Make predictions on new data and return the mean accuracy.
... nb_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.934783

```

9.15 Neural Network

The `oml.nn` class creates a Neural Network (NN) model for classification and regression.

Neural Network models can be used to capture intricate nonlinear relationships between inputs and outputs or to find patterns in data.

The `oml.nn` class methods build a feed-forward neural network for regression on `oml.DataFrame` data. It supports multiple hidden layers with a specifiable number of nodes. Each layer can have one of several activation functions.

The output layer is a single numeric or binary categorical target. The output layer can have any of the activation functions. It has the linear activation function by default.

Modeling with the `oml.nn` class is well-suited for noisy and complex data such as sensor data. Problems that such data might have are the following:

- Potentially many (numeric) predictors, for example, pixel values
- The target may be discrete-valued, real-valued, or a vector of such values
- Training data may contain errors – robust to noise
- Fast scoring
- Model transparency is not required; models difficult to interpret

Typical steps in Neural Network modeling are the following:

1. Specifying the architecture
2. Preparing the data
3. Building the model
4. Specifying the stopping criteria: iterations, error on a validation set within tolerance
5. Viewing statistical results from the model
6. Improving the model

For information on the `oml.nn` class attributes and methods, invoke `help(oml.nn)` or `help(oml.hist)`, or see Oracle Machine Learning for Python API Reference.

Settings for a Neural Network Model

The following table lists settings for NN models.

Table 9-13 Neural Network Models Settings

Setting Name	Setting Value	Description
CLAS_COST_TABLE_NAME	<i>table_name</i>	<p>The name of a table that stores a cost matrix for the algorithm to use in scoring the model. The cost matrix specifies the costs associated with misclassifications.</p> <p>The cost matrix table is user-created. The following are the column requirements for the table.</p> <ul style="list-style-type: none"> Column Name: ACTUAL_TARGET_VALUE Data Type: Valid target data type Column Name: PREDICTED_TARGET_VALUE Data Type: Valid target data type Column Name: COST Data Type: NUMBER
CLAS_WEIGHTS_BALANCED	ON OFF	<p>Indicates whether the algorithm must create a model that balances the target distribution. This setting is most relevant in the presence of rare targets, as balancing the distribution may enable better average accuracy (average of per-class accuracy) instead of overall accuracy (which favors the dominant class). The default value is OFF.</p>
NNET_ACTIVATIONS	<p>A list of the following strings:</p> <ul style="list-style-type: none"> "NNET_ACTIVATIONS_ARCTAN" "NNET_ACTIVATIONS_BIPOLAR_SIG" "NNET_ACTIVATIONS_LINEAR" "NNET_ACTIVATIONS_LOG_SIG" "NNET_ACTIVATIONS_TANH" 	<p>Defines the activation function for the hidden layers. For example, "NNET_ACTIVATIONS_BIPOLAR_SIG", "NNET_ACTIVATIONS_TANH".</p> <p>Different layers can have different activation functions. The default value is "NNET_ACTIVATIONS_LOG_SIG".</p> <p>The number of activation functions must be consistent with NNET_HIDDEN_LAYERS and NNET_NODES_PER_LAYER.</p>
NNET_HELDASIDE_MAX_FAIL	A positive integer	<p>With NNET_REGULARIZER_HELDASIDE, the training process is stopped early if the network performance on the validation data fails to improve or remains the same for NNET_HELDASIDE_MAX_FAIL epochs in a row.</p> <p>The default value is 6.</p>
NNET_HELDASIDE_RATIO	$0 \leq \text{numeric_expr} \leq 1$	<p>Defines the held ratio for the held-aside method.</p> <p>The default value is 0.25.</p>
NNET_HIDDEN_LAYERS	A non-negative integer	<p>Defines the topology by number of hidden layers.</p> <p>The default value is 1.</p>

Note:

All quotes are single and two single quotes are used to escape a single quote in SQL statements.

Table 9-13 (Cont.) Neural Network Models Settings

Setting Name	Setting Value	Description
NNET_ITERATIONS	A positive integer	Specifies the maximum number of iterations in the Neural Network algorithm. The default value is 200.
NNET_NODES_PER_LAYER	A list of positive integers	Defines the topology by number of nodes per layer. Different layers can have different number of nodes. The value should be a comma separated list non-negative integers. For example, '10, 20, 5'. The setting values must be consistent with NNET_HIDDEN_LAYERS. The default number of nodes per layer is the number of attributes or 50 (if the number of attributes > 50).
NNET_REG_LAMBDA	TO_CHAR(<i>numeric_expr</i> >= 0)	Defines the L2 regularization parameter lambda. This can not be set together with NNET_REGULARIZER_HELDASIDE. The default value is 1.
NNET_REGULARIZER	NNET_REGULARIZER_HELDASIDE NNET_REGULARIZER_L2 NNET_REGULARIZER_NONE	Regularization setting for the Neural Network algorithm. If the total number of training rows is greater than 50000, then the default is NNET_REGULARIZER_HELDASIDE. If the total number of training rows is less than or equal to 50000, then the default is NNET_REGULARIZER_NONE.
NNET_SOLVER	NNET_SOLVER_ADAM NNET_SOLVER_LBFGS	Specifies the method of optimization. The default value is NNET_SOLVER_LBFGS.
NNET_TOLERANCE	TO_CHAR(0 < <i>numeric_expr</i> < 1)	Defines the convergence tolerance setting of the Neural Network algorithm. The default value is 0.000001.
NNET_WEIGHT_LOWER_BOUND	A real number	Specifies the lower bound of the region where weights are randomly initialized. NNET_WEIGHT_LOWER_BOUND and NNET_WEIGHT_UPPER_BOUND must be set together. Setting one and not setting the other raises an error. NNET_WEIGHT_LOWER_BOUND must not be greater than NNET_WEIGHT_UPPER_BOUND. The default value is $-\sqrt{6/(1_nodes+r_nodes)}$. The value of 1_nodes for: <ul style="list-style-type: none"> input layer dense attributes is (1+number of dense attributes) input layer sparse attributes is number of sparse attributes each hidden layer is (1+number of nodes in that hidden layer) The value of r_nodes is the number of nodes in the layer that the weight is connecting to.
NNET_WEIGHT_UPPER_BOUND	A real number	Specifies the upper bound of the region where weights are initialized. It should be set in pairs with NNET_WEIGHT_LOWER_BOUND and its value must not be smaller than the value of NNET_WEIGHT_LOWER_BOUND. If not specified, the values of NNET_WEIGHT_LOWER_BOUND and NNET_WEIGHT_UPPER_BOUND are system determined. The default value is $\sqrt{6/(1_nodes+r_nodes)}$. See NNET_WEIGHT_LOWER_BOUND.

Table 9-13 (Cont.) Neural Network Models Settings

Setting Name	Setting Value	Description
ODMS_RANDOM_SEED	A non-negative integer	Controls the random number seed used by the hash function to generate a random number with uniform distribution. The default values is 0.

 **See Also:**

- [About Model Settings](#)
- [Shared Settings](#)

Example 9-15 Building a Neural Network Model

This example creates an NN model and uses some of the methods of the `oml.nn` class.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Create a Neural Network model object.
nn_mod = oml.nn(nnet_hidden_layers = 1,
                nnet_activations= "'NNET_ACTIVATIONS_LOG_SIG'",
                NNET_NODES_PER_LAYER= '30')

# Fit the NN model according to the training data and parameter
# settings.
```

```
nn_mod = nn_mod.fit(train_x, train_y)

# Show details of the model.
nn_mod

# Use the model to make predictions on test data.
nn_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_Width',
                                               'Petal_Length', 'Species']])

nn_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_Width',
                                               'Species']], proba = True)

nn_mod.predict_proba(test_dat.drop('Species'),
                    supplemental_cols = test_dat[:, ['Sepal_Length',
                                                    'Species']]).sort_values(by = ['Sepal_Length', 'Species',
                                                    'PROBABILITY_OF_setosa', 'PROBABILITY_OF_versicolor'])

nn_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])

# Change the setting parameter and refit the model.
new_setting = {'NNET_NODES_PER_LAYER': '50'}
nn_mod.set_params(**new_setting).fit(train_x, train_y)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                            'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                          {0: 'setosa', 1: 'versicolor',
...                           2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
>>>
```

```

>>> # Create a Neural Network model object.
... nn_mod = oml.nn(nnet_hidden_layers = 1,
...                 nnet_activations= "'NNET_ACTIVATIONS_LOG_SIG'",
...                 NNET_NODES_PER_LAYER= '30')
>>>
>>> # Fit the NN model according to the training data and parameter
... # settings.
... nn_mod = nn_mod.fit(train_x, train_y)
>>>
>>> # Show details of the model.
... nn_mod

```

Algorithm Name: Neural Network

Mining Function: CLASSIFICATION

Target: Species

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_NEURAL_NETWORK
1	CLAS_WEIGHTS_BALANCED	OFF
2	LBFGS_GRADIENT_TOLERANCE	.000000001
3	LBFGS_HISTORY_DEPTH	20
4	LBFGS_SCALE_HESSIAN	LBFGS_SCALE_HESSIAN_ENABLE
5	NNET_ACTIVATIONS	'NNET_ACTIVATIONS_LOG_SIG'
6	NNET_HELDDASIDE_MAX_FAIL	6
7	NNET_HELDDASIDE_RATIO	.25
8	NNET_HIDDEN_LAYERS	1
9	NNET_ITERATIONS	200
10	NNET_NODES_PER_LAYER	30
11	NNET_TOLERANCE	.000001
12	ODMS_DETAILS	ODMS_ENABLE
13	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
14	ODMS_RANDOM_SEED	0
15	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
16	PREP_AUTO	ON

Computed Settings:

	setting name	setting value
0	NNET_REGULARIZER	NNET_REGULARIZER_NONE

Global Statistics:

	attribute name	attribute value
0	CONVERGED	YES
1	ITERATIONS	60.0
2	LOSS_VALUE	0.0
3	NUM_ROWS	102.0

Attributes:

Sepal_Length
Sepal_Width
Petal_Length
Petal_Width

Partition: NO

Topology:

HIDDEN_LAYER_ID	NUM_NODE	ACTIVATION_FUNCTION
0	0	30
		NNET_ACTIVATIONS_LOG_SIG

Weights:

ATTRIBUTE_VALUE	LAYER	IDX_FROM	IDX_TO	ATTRIBUTE_NAME	ATTRIBUTE_SUBNAME
0	0	0.0	0	Petal_Length	None
None	1	0	0.0	1	Petal_Length
None	2	0	0.0	2	Petal_Length
None	3	0	0.0	3	Petal_Length
...
239	1	29.0	2	None	None
None	240	1	NaN	0	None
None	241	1	NaN	1	None
None	242	1	NaN	2	None
None					

	TARGET_VALUE	WEIGHT
0	None	-39.836487
1	None	32.604824
2	None	0.953903
3	None	0.714064
...
239	virginica	-22.650606
240	setosa	2.402457
241	versicolor	7.647615
242	virginica	-9.493982

[243 rows x 8 columns]

```
>>> # Use the model to make predictions on test data.
... nn_mod.predict(test_dat.drop('Species'),
...                 supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_Width',
...                                                 'Petal_Length', 'Species']])
...   Sepal_Length  Sepal_Width  Petal_Length  Species  PREDICTION
0              4.9           3.0           1.4   setosa   setosa
1              4.9           3.1           1.5   setosa   setosa
2              4.8           3.4           1.6   setosa   setosa
3              5.8           4.0           1.2   setosa   setosa
...           ...           ...           ...   ...       ...
44             6.7           3.3           5.7  virginica  virginica
```

```

45         6.7         3.0         5.2  virginica  virginica
46         6.5         3.0         5.2  virginica  virginica
47         5.9         3.0         5.1  virginica  virginica

>>> nn_mod.predict(test_dat.drop('Species'),
...     supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_Width',
...     'Species']], proba = True)
   Sepal_Length  Sepal_Width  Species  PREDICTION  PROBABILITY
0             4.9           3.0   setosa    setosa      1.000000
1             4.9           3.1   setosa    setosa      1.000000
2             4.8           3.4   setosa    setosa      1.000000
3             5.8           4.0   setosa    setosa      1.000000
...           ...           ...       ...       ...       ...
44            6.7           3.3  virginica  virginica    1.000000
45            6.7           3.0  virginica  virginica    1.000000
46            6.5           3.0  virginica  virginica    1.000000
47            5.9           3.0  virginica  virginica    1.000000

>>> nn_mod.predict_proba(test_dat.drop('Species'),
...     supplemental_cols = test_dat[:, ['Sepal_Length',
...     'Species']]).sort_values(by = ['Sepal_Length', 'Species',
...     'PROBABILITY_OF_setosa', 'PROBABILITY_OF_versicolor'])
   Sepal_Length  Species  PROBABILITY_OF_SETOSA \
0             4.4   setosa      1.000000e+00
1             4.4   setosa      1.000000e+00
2             4.5   setosa      1.000000e+00
3             4.8   setosa      1.000000e+00
...           ...       ...
44            6.7  virginica      4.567318e-218
45            6.9  versicolor      3.028266e-177
46            6.9  virginica      1.203417e-215
47            7.0  versicolor      3.382837e-148

   PROBABILITY_OF_VERSICOLOR  PROBABILITY_OF_VIRGINICA
0             3.491272e-67             3.459448e-283
1             8.038930e-58             2.883999e-288
2             5.273544e-64             2.243282e-293
3             1.332150e-78             2.040723e-283
...           ...
44             1.328042e-36             1.000000e+00
45             1.000000e+00             5.063405e-55
46             4.000953e-31             1.000000e+00
47             1.000000e+00             2.593761e-121

>>> nn_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.9375

>>> # Change the setting parameter and refit the model.
... new_setting = {'NNET_NODES_PER_LAYER': '50'}
>>> nn_mod.set_params(**new_setting).fit(train_x, train_y)

```

Algorithm Name: Neural Network

Mining Function: CLASSIFICATION

Target: Species

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_NEURAL_NETWORK
1	CLAS_WEIGHTS_BALANCED	OFF
2	LBFGS_GRADIENT_TOLERANCE	.000000001
3	LBFGS_HISTORY_DEPTH	20
4	LBFGS_SCALE_HESSIAN	LBFGS_SCALE_HESSIAN_ENABLE
5	NNET_ACTIVATIONS	'NNET_ACTIVATIONS_LOG_SIG'
6	NNET_HELDDASIDE_MAX_FAIL	6
7	NNET_HELDDASIDE_RATIO	.25
8	NNET_HIDDEN_LAYERS	1
9	NNET_ITERATIONS	200
10	NNET_NODES_PER_LAYER	50
11	NNET_TOLERANCE	.000001
12	ODMS_DETAILS	ODMS_ENABLE
13	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
14	ODMS_RANDOM_SEED	0
15	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
16	PREP_AUTO	ON

Computed Settings:

	setting name	setting value
0	NNET_REGULARIZER	NNET_REGULARIZER_NONE

Global Statistics:

	attribute name	attribute value
0	CONVERGED	YES
1	ITERATIONS	68.0
2	LOSS_VALUE	0.0
3	NUM_ROWS	102.0

Attributes:

Sepal_Length
Sepal_Width
Petal_Length
Petal_Width

Partition: NO

Topology:

	HIDDEN_LAYER_ID	NUM_NODE	ACTIVATION_FUNCTION
0	0	50	NNET_ACTIVATIONS_LOG_SIG

Weights:

	LAYER	IDX_FROM	IDX_TO	ATTRIBUTE_NAME	ATTRIBUTE_SUBNAME
0	0	0.0	0	Petal_Length	None
1	0	0.0	1	Petal_Length	None
2	0	0.0	2	Petal_Length	None
3	0	0.0	3	Petal_Length	None

```

None
...      ...      ...      ...      ...      ...      ...
399      1      49.0      2      None      None
None
400      1      NaN      0      None      None
None
401      1      NaN      1      None      None
None
402      1      NaN      2      None      None
None

      TARGET_VALUE      WEIGHT
0      None      10.606389
1      None      -37.256485
2      None      -14.263772
3      None      -17.945173
...      ...      ...
399      virginica      -22.179815
400      setosa      -6.452953
401      versicolor      13.186332
402      virginica      -6.973605

[403 rows x 8 columns]

```

9.16 Random Forest

The `oml.rf` class creates a Random Forest (RF) model that provides an ensemble learning technique for classification.

By combining the ideas of bagging and random selection of variables, the Random Forest algorithm produces a collection of decision trees with controlled variance while avoiding overfitting, which is a common problem for decision trees.

For information on the `oml.rf` class attributes and methods, invoke `help(oml.rf)` or see Oracle Machine Learning for Python API Reference.

Settings for a Random Forest Model

The following table lists settings for RF models.

Table 9-14 Random Forest Model Settings

Setting Name	Setting Value	Description
CLAS_COST_TABLE_NAME	<i>table_name</i>	<p>The name of a table that stores a cost matrix for the algorithm to use in scoring the model. The cost matrix specifies the costs associated with misclassifications.</p> <p>The cost matrix table is user-created. The following are the column requirements for the table.</p> <ul style="list-style-type: none"> Column Name: ACTUAL_TARGET_VALUE Data Type: Valid target data type Column Name: PREDICTED_TARGET_VALUE Data Type: Valid target data type Column Name: COST Data Type: NUMBER
CLAS_MAX_SUP_BINS	$2 \leq a \text{ number} \leq 254$	<p>Specifies the maximum number of bins for each attribute.</p> <p>The default value is 32.</p>
CLAS_WEIGHTS_BALANCED	ON OFF	<p>Indicates whether the algorithm must create a model that balances the target distribution. This setting is most relevant in the presence of rare targets, as balancing the distribution may enable better average accuracy (average of per-class accuracy) instead of overall accuracy (which favors the dominant class). The default value is OFF.</p>
ODMS_RANDOM_SEED	A non-negative integer	<p>Controls the random number seed used by the hash function to generate a random number with uniform distribution. The default values is 0.</p>
RFOR_MTRY	$A \text{ number} \geq 0$	<p>Size of the random subset of columns to consider when choosing a split at a node. For each node, the size of the pool remains the same but the specific candidate columns change. The default is half of the columns in the model signature. The special value 0 indicates that the candidate pool includes all columns.</p>
RFOR_NUM_TREES	$1 \leq a \text{ number} \leq 65535$	<p>Number of trees in the forest</p> <p>The default value is 20.</p>
RFOR_SAMPLING_RATIO	$0 < a \text{ fraction} \leq 1$	<p>Fraction of the training data to be randomly sampled for use in the construction of an individual tree. The default is half of the number of rows in the training data.</p>

Table 9-14 (Cont.) Random Forest Model Settings

Setting Name	Setting Value	Description
TREE_IMPURITY_METRIC	TREE_IMPURITY_ENTROPY TREE_IMPURITY_GINI	Tree impurity metric for a decision tree model. Tree algorithms seek the best test question for splitting data at each node. The best splitter and split value are those that result in the largest increase in target value homogeneity (purity) for the entities in the node. Purity is measured in accordance with a metric. Decision trees can use either gini (TREE_IMPURITY_GINI) or entropy (TREE_IMPURITY_ENTROPY) as the purity metric. By default, the algorithm uses TREE_IMPURITY_GINI.
TREE_TERM_MAX_DEPTH	$2 \leq a \text{ number} \leq 100$	Criteria for splits: maximum tree depth (the maximum number of nodes between the root and any leaf node, including the leaf node). The default is 16.
TREE_TERM_MINPCT_NODE	$0 \leq a \text{ number} \leq 10$	The minimum number of training rows in a node expressed as a percentage of the rows in the training data. The default value is 0.05, indicating 0.05%.
TREE_TERM_MINPCT_SPLIT	$0 < a \text{ number} \leq 20$	Minimum number of rows required to consider splitting a node expressed as a percentage of the training rows. The default value is 0.1, indicating 0.1%.
TREE_TERM_MINREC_NODE	$A \text{ number} \geq 0$	Minimum number of rows in a node. The default value is 10.
TREE_TERM_MINREC_SPLIT	$A \text{ number} > 1$	Criteria for splits: minimum number of records in a parent node expressed as a value. No split is attempted if the number of records is below this value. The default value is 20.

 **See Also:**

- [About Model Settings](#)
- [Shared Settings](#)

Example 9-16 Using the oml.rf Class

This example creates an RF model and uses some of the methods of the `oml.rf` class.

```
import oml
import pandas as pd
from sklearn import datasets
```

```
# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                  columns = ['Sepal_Length', 'Sepal_Width',
                             'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                           {0: 'setosa', 1: 'versicolor',
                            2: 'virginica'}[x], iris.target)),
                  columns = ['Species'])

try:
    oml.drop('IRIS')
    oml.drop(table = 'RF_COST')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Create a cost matrix table in the database.
cost_matrix = [['setosa', 'setosa', 0],
               ['setosa', 'virginica', 0.2],
               ['setosa', 'versicolor', 0.8],
               ['virginica', 'virginica', 0],
               ['virginica', 'setosa', 0.5],
               ['virginica', 'versicolor', 0.5],
               ['versicolor', 'versicolor', 0],
               ['versicolor', 'setosa', 0.4],
               ['versicolor', 'virginica', 0.6]]
cost_matrix = \
    oml.create(pd.DataFrame(cost_matrix,
                            columns = ['ACTUAL_TARGET_VALUE',
                                       'PREDICTED_TARGET_VALUE',
                                       'COST']),
              table = 'RF_COST')

# Create an RF model object.
rf_mod = oml.rf(tree_term_max_depth = '2')

# Fit the RF model according to the training data and parameter
# settings.
rf_mod = rf_mod.fit(train_x, train_y, cost_matrix = cost_matrix)

# Show details of the model.
rf_mod

# Use the model to make predictions on the test data.
rf_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
```

```
        'Sepal_Width',
        'Petal_Length',
        'Species']]

# Return the prediction probability.
rf_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
                                                'Sepal_Width',
                                                'Species']],
               proba = True)

# Return the top two most influential attributes of the highest
# probability class.
rf_mod.predict_proba(test_dat.drop('Species'),
                    supplemental_cols = test_dat[:, ['Sepal_Length',
                                                    'Species']],
                    topN = 2).sort_values(by = ['Sepal_Length', 'Species'])

rf_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])

# Reset TREE_TERM_MAX_DEPTH and refit the model.
rf_mod.set_params(tree_term_max_depth = '3').fit(train_x, train_y,
cost_matrix)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                            'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                          {0: 'setosa', 1: 'versicolor',
...                           2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...     oml.drop('IRIS')
...     oml.drop(table = 'RF_COST')
... except:
...     pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
```



```

>>>
>>> # Create a cost matrix table in the database.
... cost_matrix = [['setosa', 'setosa', 0],
...                 ['setosa', 'virginica', 0.2],
...                 ['setosa', 'versicolor', 0.8],
...                 ['virginica', 'virginica', 0],
...                 ['virginica', 'setosa', 0.5],
...                 ['virginica', 'versicolor', 0.5],
...                 ['versicolor', 'versicolor', 0],
...                 ['versicolor', 'setosa', 0.4],
...                 ['versicolor', 'virginica', 0.6]]
>>> cost_matrix = \
...     oml.create(pd.DataFrame(cost_matrix,
...                             columns = ['ACTUAL_TARGET_VALUE',
...                                       'PREDICTED_TARGET_VALUE',
...                                       'COST']),
...               table = 'RF_COST')
>>>
>>> # Create an RF model object.
... rf_mod = oml.rf(tree_term_max_depth = '2')
>>>
>>> # Fit the RF model according to the training data and parameter
... # settings.
>>> rf_mod = rf_mod.fit(train_x, train_y, cost_matrix = cost_matrix)
>>>
>>> # Show details of the model.
... rf_mod

```

Algorithm Name: Random Forest

Mining Function: CLASSIFICATION

Target: Species

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_RANDOM_FOREST
1	CLAS_COST_TABLE_NAME	"OML_USER"."RF_COST"
2	CLAS_MAX_SUP_BINS	32
3	CLAS_WEIGHTS_BALANCED	OFF
4	ODMS_DETAILS	ODMS_ENABLE
5	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
6	ODMS_RANDOM_SEED	0
7	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
8	PREP_AUTO	ON
9	RFOR_NUM_TREES	20
10	RFOR_SAMPLING_RATIO	.5
11	TREE_IMPURITY_METRIC	TREE_IMPURITY_GINI
12	TREE_TERM_MAX_DEPTH	2
13	TREE_TERM_MINPCT_NODE	.05
14	TREE_TERM_MINPCT_SPLIT	.1
15	TREE_TERM_MINREC_NODE	10
16	TREE_TERM_MINREC_SPLIT	20

Computed Settings:

setting name	setting value
--------------	---------------

```
0    RFOR_MTRY          2
```

```
Global Statistics:
```

```
  attribute name  attribute value
0      AVG_DEPTH          2
1  AVG_NODECOUNT          3
2      MAX_DEPTH          2
3  MAX_NODECOUNT          2
4      MIN_DEPTH          2
5  MIN_NODECOUNT          2
6      NUM_ROWS         104
```

```
Attributes:
```

```
Petal_Length
Petal_Width
Sepal_Length
```

```
Partition: NO
```

```
Importance:
```

	ATTRIBUTE_NAME	ATTRIBUTE_SUBNAME	ATTRIBUTE_IMPORTANCE
0	Petal_Length	None	0.329971
1	Petal_Width	None	0.296799
2	Sepal_Length	None	0.037309
3	Sepal_Width	None	0.000000

```
>>> # Use the model to make predictions on the test data.
```

```
... rf_mod.predict(test_dat.drop('Species'),
...                 supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Petal_Length',
...                                                 'Species']])
...
... Sepal_Length  Sepal_Width  Petal_Length  Species  PREDICTION
0              4.9           3.0           1.4    setosa    setosa
1              4.9           3.1           1.5    setosa    setosa
2              4.8           3.4           1.6    setosa    setosa
3              5.8           4.0           1.2    setosa    setosa
...           ...           ...           ...     ...     ...
42             6.7           3.3           5.7  virginica  virginica
43             6.7           3.0           5.2  virginica  virginica
44             6.5           3.0           5.2  virginica  virginica
45             5.9           3.0           5.1  virginica  virginica
```

```
>>> # Return the prediction probability.
```

```
... rf_mod.predict(test_dat.drop('Species'),
...                 supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Species']],
...                 proba = True)
...
... Sepal_Length  Sepal_Width  Species  PREDICTION  PROBABILITY
0              4.9           3.0    setosa    setosa    0.989130
1              4.9           3.1    setosa    setosa    0.989130
2              4.8           3.4    setosa    setosa    0.989130
3              5.8           4.0    setosa    setosa    0.950000
...           ...           ...     ...     ...     ...
```

```

42          6.7          3.3  virginica  virginica  0.501016
43          6.7          3.0  virginica  virginica  0.501016
44          6.5          3.0  virginica  virginica  0.501016
45          5.9          3.0  virginica  virginica  0.501016

>>> # Return the top two most influential attributes of the highest
... # probability class.
>>> rf_mod.predict_proba(test_dat.drop('Species'),
...                       supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                         'Species']],
...                       topN = 2).sort_values(by = ['Sepal_Length', 'Species'])
   Sepal_Length  Species  TOP_1  TOP_1_VAL  TOP_2  TOP_2_VAL
0             4.4   setosa  setosa  0.989130  versicolor  0.010870
1             4.4   setosa  setosa  0.989130  versicolor  0.010870
2             4.5   setosa  setosa  0.989130  versicolor  0.010870
3             4.8   setosa  setosa  0.989130  versicolor  0.010870
...           ...     ...     ...     ...     ...     ...
42          6.7  virginica  virginica  0.501016  versicolor  0.498984
43          6.9  versicolor  virginica  0.501016  versicolor  0.498984
44          6.9  virginica  virginica  0.501016  versicolor  0.498984
45          7.0  versicolor  virginica  0.501016  versicolor  0.498984

>>> rf_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.76087

>>> # Reset TREE_TERM_MAX_DEPTH and refit the model.
... rf_mod.set_params(tree_term_max_depth = '3').fit(train_x, train_y,
cost_matrix)

```

Algorithm Name: Random Forest

Mining Function: CLASSIFICATION

Target: Species

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_RANDOM_FOREST
1	CLAS_COST_TABLE_NAME	"OML_USER"."RF_COST"
2	CLAS_MAX_SUP_BINS	32
3	CLAS_WEIGHTS_BALANCED	OFF
4	ODMS_DETAILS	ODMS_ENABLE
5	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
6	ODMS_RANDOM_SEED	0
7	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
8	PREP_AUTO	ON
9	RFOR_NUM_TREES	20
10	RFOR_SAMPLING_RATIO	.5
11	TREE_IMPURITY_METRIC	TREE_IMPURITY_GINI
12	TREE_TERM_MAX_DEPTH	3
13	TREE_TERM_MINPCT_NODE	.05
14	TREE_TERM_MINPCT_SPLIT	.1
15	TREE_TERM_MINREC_NODE	10
16	TREE_TERM_MINREC_SPLIT	20

Computed Settings:

setting name	setting value
--------------	---------------

```

0    RFOR_MTRY                2

Global Statistics:
  attribute name  attribute value
0      AVG_DEPTH                3
1    AVG_NODECOUNT            5
2      MAX_DEPTH                3
3    MAX_NODECOUNT            6
4      MIN_DEPTH                3
5    MIN_NODECOUNT            4
6      NUM_ROWS                 104

```

```

Attributes:
Petal_Length
Petal_Width
Sepal_Length

```

```
Partition: NO
```

```
Importance:
```

```

  ATTRIBUTE_NAME  ATTRIBUTE_SUBNAME  ATTRIBUTE_IMPORTANCE
0    Petal_Length                None                0.501022
1    Petal_Width                 None                0.568170
2    Sepal_Length                None                0.091617
3    Sepal_Width                 None                0.000000

```

9.17 Singular Value Decomposition

Use the `oml.svd` class to build a model for feature extraction.

The `oml.svd` class creates a model that uses the Singular Value Decomposition (SVD) algorithm for feature extraction. SVD performs orthogonal linear transformations that capture the underlying variance of the data by decomposing a rectangular matrix into three matrices: U, V, and D. Columns of matrix V contain the right singular vectors and columns of matrix U contain the left singular vectors. Matrix D is a diagonal matrix and its singular values reflect the amount of data variance captured by the bases.

The `SVDS_MAX_NUM_FEATURES` constant specifies the maximum number of features supported by SVD. The value of the constant is 2500.

For information on the `oml.svd` class attributes and methods, invoke `help(oml.svd)` or see Oracle Machine Learning for Python API Reference.

Settings for a Singular Value Decomposition Model

Table 9-15 Singular Value Decomposition Model Settings

Setting Name	Setting Value	Description
FEAT_NUM_FEATURES	TO_CHAR(<i>numeric_expr</i> >=1)	The number of features to extract. The default value is estimated by the algorithm. If the matrix rank is smaller than this number, fewer features are returned.

Table 9-15 (Cont.) Singular Value Decomposition Model Settings

Setting Name	Setting Value	Description
SVDS_OVER_SAMPLING	Range [1, 5000].	Configures the number of columns in the sampling matrix used by the Stochastic SVD solver. The number of columns in this matrix is equal to the requested number of features plus the oversampling setting. <code>TSVDS_SOLVER</code> must be set to <code>SVDS_SOLVER_SSVD</code> or <code>SVDS_SOLVER_STEIGEN</code> .
SVDS_POWER_ITERATIONS	Range [0, 20].	Improves the accuracy of the SSVD solver. The default value is 2. <code>SVDS_SOLVER</code> must be set to <code>SVDS_SOLVER_SSVD</code> or <code>SVDS_SOLVER_STEIGEN</code> .
SVDS_RANDOM_SEED	Range [0 - 4,294,967,296]	The random seed value for initializing the sampling matrix used by the Stochastic SVD solver. The default value is 0. <code>SVDS_SOLVER</code> must be set to <code>SVDS_SOLVER_SSVD</code> or <code>SVDS_SOLVER_STEIGEN</code> .
SVDS_SCORING_MODE	SVDS_SCORING_PCA SVDS_SCORING_SVD	Whether to use SVD or PCA scoring for the model. When the build data is scored with SVD, the projections are the same as the U matrix. When the build data is scored with PCA, the projections are the product of the U and D matrices. The default value is <code>SVDS_SCORING_SVD</code> .
SVDS_SOLVER	SVDS_SOLVER_STEIGEN SVDS_SOLVER_SSVD SVDS_SOLVER_TSEIGEN SVDS_SOLVER_TSSVD	Specifies the solver to be used for computing SVD of the data. For PCA, the solver setting indicates the type of SVD solver used to compute the PCA for the data. When this setting is not specified, the solver type selection is data driven. If the number of attributes is greater than 3240, then the default wide solver is used. Otherwise, the default narrow solver is selected. The following are the group of solvers: <ul style="list-style-type: none"> Narrow data solvers: for matrices with up to 11500 attributes (<code>TSEIGEN</code>) or up to 8100 attributes (<code>TSSVD</code>). Wide data solvers: for matrices up to 1 million attributes. For narrow data solvers: <ul style="list-style-type: none"> Tall-Skinny SVD uses QR computation <code>TSVD</code> (<code>SVDS_SOLVER_TSSVD</code>) Tall-Skinny SVD uses eigenvalue computation, <code>TSEIGEN</code> (<code>SVDS_SOLVER_TSEIGEN</code>), which is the default solver for narrow data. For wide data solvers: <ul style="list-style-type: none"> Stochastic SVD uses QR computation <code>SSVD</code> (<code>SVDS_SOLVER_SSVD</code>), is the default solver for wide data solvers. Stochastic SVD uses eigenvalue computations, <code>STEIGEN</code> (<code>SVDS_SOLVER_STEIGEN</code>).
SVDS_TOLERANCE	Range [0, 1]	Defines the minimum value for the eigenvalue of a feature as a share of the first eigenvalue to not prune. Use this setting to prune features. The default value is data driven.

Table 9-15 (Cont.) Singular Value Decomposition Model Settings

Setting Name	Setting Value	Description
SVDS_U_MATRIX_OUTPUT	SVDS_U_MATRIX_ENABLE	Specifies whether to persist the U matrix produced by SVD.
	SVDS_U_MATRIX_DISABLE	The U matrix in SVD has as many rows as the number of rows in the build data. To avoid creating a large model, the U matrix is persisted only when SVDS_U_MATRIX_OUTPUT is enabled. When SVDS_U_MATRIX_OUTPUT is enabled, the build data must include a case ID. If no case ID is present and the U matrix is requested, then an exception is raised. The default value is SVDS_U_MATRIX_DISABLE.

 **See Also:**

- [About Model Settings](#)
- [Shared Settings](#)

Example 9-17 Using the oml.svd Class

This example uses some of the methods of the `oml.svd` class. In the listing for this example, some of the output is not shown as indicated by ellipses.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_dat = dat[0]
test_dat = dat[1]

# Create an SVD model object.
```

```
svd_mod = oml.svd(ODMS_DETAILS = 'ODMS_ENABLE')

# Fit the model according to the training data and parameter
# settings.
svd_mod = svd_mod.fit(train_dat)

# Show the model details.
svd_mod

# Use the model to make predictions on the test data.
svd_mod.predict(test_dat,
                 supplemental_cols = test_dat[:,
                                             ['Sepal_Length',
                                              'Sepal_Width',
                                              'Petal_Length',
                                              'Species']])

# Perform dimensionality reduction and return values for the two
# features that have the highest topN values.
svd_mod.transform(test_dat,
                  supplemental_cols = test_dat[:, ['Sepal_Length']],
                  topN = 2).sort_values(by = ['Sepal_Length',
                                              'TOP_1',
                                              'TOP_1_VAL'])
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                             'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_dat = dat[0]
>>> test_dat = dat[1]
>>>
>>> # Create an SVD model object.
```

```
... svd_mod = oml.svd(ODMS_DETAILS = 'ODMS_ENABLE')
>>>
>>> # Fit the model according to the training data and parameter
... # settings.
>>> svd_mod = svd_mod.fit(train_dat)
>>>
>>> # Show the model details.
... svd_mod
```

Algorithm Name: Singular Value Decomposition

Mining Function: FEATURE_EXTRACTION

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_SINGULAR_VALUE_DECOMP
1	ODMS_DETAILS	ODMS_ENABLE
2	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
3	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
4	PREP_AUTO	ON
5	SVDS_SCORING_MODE	SVDS_SCORING_SVD
6	SVDS_U_MATRIX_OUTPUT	SVDS_U_MATRIX_DISABLE

Computed Settings:

	setting name	setting value
0	FEAT_NUM_FEATURES	8
1	SVDS_SOLVER	SVDS_SOLVER_TSEIGEN
2	SVDS_TOLERANCE	.000000000000024646951146678475

Global Statistics:

	attribute name	attribute value
0	NUM_COMPONENTS	8
1	NUM_ROWS	111
2	SUGGESTED_CUTOFF	1

Attributes:

Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Features:

	FEATURE_ID	ATTRIBUTE_NAME	ATTRIBUTE_VALUE	VALUE
0	1	ID	None	0.996297
1	1	Petal_Length	None	0.046646
2	1	Petal_Width	None	0.015917
3	1	Sepal_Length	None	0.063312
...
60	8	Sepal_Width	None	-0.030620
61	8	Species	setosa	0.431543
62	8	Species	versicolor	0.566418
63	8	Species	virginica	0.699261

[64 rows x 4 columns]

D:

	FEATURE_ID	VALUE
0	1	886.737809
1	2	32.736792
2	3	10.043389
3	4	5.270496
4	5	2.708602
5	6	1.652340
6	7	0.938640
7	8	0.452170

V:

	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'
0	0.001332	0.156581	-0.317375	0.113462	-0.154414	-0.113058	0.799390	
1	0.003692	0.052289	0.316295	0.733040	0.190746	0.022285	-0.046406	
2	0.005267	-0.051498	-0.052111	0.527881	-0.066995	0.046461	-0.469396	
3	0.015917	0.008741	0.263614	0.244811	0.460445	0.767503	0.262966	
4	0.030208	0.550384	-0.358277	0.041807	0.689962	-0.261815	-0.143258	
5	0.046646	0.189325	0.766663	0.326363	0.079611	-0.479070	0.177661	
6	0.063312	0.790864	0.097964	-0.051230	-0.490804	0.312159	-0.131337	
7	0.996297	-0.076079	-0.035940	-0.017429	-0.000960	-0.001908	0.001755	
0	0.431543							
1	0.566418							
2	0.699261							
3	0.005000							
4	-0.030620							
5	-0.016932							
6	-0.052185							
7	-0.001415							

```
>>> # Use the model to make predictions on the test data.
```

```
>>> svd_mod.predict(test_dat,
                    supplemental_cols = test_dat[:,
                    ['Sepal_Length',
                    'Sepal_Width',
                    'Petal_Length',
                    'Species']])
Sepal_Length Sepal_Width Petal_Length Species FEATURE_ID
0            5.0         3.6          1.4  setosa         2
1            5.0         3.4          1.5  setosa         2
2            4.4         2.9          1.4  setosa         8
3            4.9         3.1          1.5  setosa         2
... ..
35           6.9         3.1          5.4  virginica        1
36           5.8         2.7          5.1  virginica        1
37           6.2         3.4          5.4  virginica        5
38           5.9         3.0          5.1  virginica        1
```

```
>>> # Perform dimensionality reduction and return values for the two
... # features that have the highest topN values.
```

```
>>> svd_mod.transform(test_dat,
...     supplemental_cols = test_dat[:, ['Sepal_Length']],
...     topN = 2).sort_values(by = ['Sepal_Length',
...     'TOP_1',
...     'TOP_1_VAL'])
```

	Sepal_Length	TOP_1	TOP_1_VAL	TOP_2	TOP_2_VAL
0	4.4	7	0.153125	3	-0.130778
1	4.4	8	0.171819	2	0.147070
2	4.8	2	0.159324	6	-0.085194
3	4.8	7	0.157187	3	-0.141668
...
35	7.2	6	-0.167688	1	0.142545
36	7.2	7	-0.176290	6	-0.175527
37	7.6	4	0.205779	3	0.141533
38	7.9	8	-0.253194	7	-0.166967

9.18 Support Vector Machine

The `oml.svm` class creates a Support Vector Machine (SVM) model for classification, regression, or anomaly detection.

SVM is a powerful, state-of-the-art algorithm with strong theoretical foundations based on the Vapnik-Chervonenkis theory. SVM has strong regularization properties. Regularization refers to the generalization of the model to new data.

SVM models have a functional form similar to neural networks and radial basis functions, which are both popular machine learning techniques.

SVM can be used to solve the following problems:

- **Classification:** SVM classification is based on decision planes that define decision boundaries. A decision plane is one that separates a set of objects having different class memberships. SVM finds the vectors ("support vectors") that define the separators that give the widest separation of classes.
SVM classification supports both binary and multiclass targets.
- **Regression:** SVM uses an epsilon-insensitive loss function to solve regression problems.
SVM regression tries to find a continuous function such that the maximum number of data points lie within the epsilon-wide insensitivity tube. Predictions falling within epsilon distance of the true target value are not interpreted as errors.
- **Anomaly Detection:** Anomaly detection identifies unusual cases in data that is seemingly homogeneous. Anomaly detection is an important tool for detecting fraud, network intrusion, and other rare events that may have great significance but are hard to find.

Anomaly detection is implemented as one-class SVM classification. An anomaly detection model predicts whether a data point is typical for a given distribution or not.

The `oml.svm` class builds each of these three different types of models. Some arguments apply to classification models only, some to regression models only, and some to anomaly detection models only.

For information on the `oml.svm` class attributes and methods, invoke `help(oml.svm)` or see Oracle Machine Learning for Python API Reference.

Support Vector Machine Model Settings

The following table lists settings for SVM models.

Table 9-16 Support Vector Machine Settings

Setting Name	Setting Value	Description
CLAS_COST_TABLE_NAME	<i>table_name</i>	<p>The name of a table that stores a cost matrix for the algorithm to use in scoring the model. The cost matrix specifies the costs associated with misclassifications.</p> <p>The cost matrix table is user-created. The following are the column requirements for the table.</p> <ul style="list-style-type: none"> Column Name: ACTUAL_TARGET_VALUE Data Type: Valid target data type Column Name: PREDICTED_TARGET_VALUE Data Type: Valid target data type Column Name: COST Data Type: NUMBER
CLAS_WEIGHTS_BALANCED	ON OFF	<p>Indicates whether the algorithm must create a model that balances the target distribution. This setting is most relevant in the presence of rare targets, as balancing the distribution may enable better average accuracy (average of per-class accuracy) instead of overall accuracy (which favors the dominant class). The default value is OFF.</p>
CLAS_WEIGHTS_TABLE_NAME	<i>table_name</i>	<p>The name of a table that stores weighting information for individual target values in GLM logistic regression models. The weights are used by the algorithm to bias the model in favor of higher weighted classes.</p> <p>The class weights table is user-created. The following are the column requirements for the table.</p> <ul style="list-style-type: none"> Column Name: TARGET_VALUE Data Type: Valid target data type Column Name: CLASS_WEIGHT Data Type: NUMBER
SVMS_BATCH_ROWS	Positive integer	<p>Sets the size of the batch for the SGD solver. This setting applies to SVM models with linear kernel. An input of 0 triggers a data driven batch size estimate. The default value is 20000.</p>
SVMS_COMPLEXITY_FACTOR	TO_CHAR(<i>numeric_exp</i> <i>r</i> >0)	<p>Regularization setting that balances the complexity of the model against model robustness to achieve good generalization on new data. SVM uses a data-driven approach to finding the complexity factor.</p> <p>Value of complexity factor for SVM algorithm (both Classification and Regression).</p> <p>Default value estimated from the data by the algorithm.</p>
SVMS_CONV_TOLERANCE	TO_CHAR(<i>numeric_exp</i> <i>r</i> >0)	<p>Convergence tolerance for SVM algorithm.</p> <p>Default is 0.0001.</p>
SVMS_EPSILON	TO_CHAR(<i>numeric_exp</i> <i>r</i> >0)	<p>Regularization setting for regression, similar to complexity factor. Epsilon specifies the allowable residuals, or noise, in the data.</p> <p>Value of epsilon factor for SVM regression.</p> <p>Default is 0.1.</p>
SVMS_KERNEL_FUNCTION	SVMS_GAUSSIAN SVMS_LINEAR	<p>Kernel for Support Vector Machine. Linear or Gaussian.</p> <p>The default value is SVMS_LINEAR.</p>
SVMS_NUM_ITERATIONS	Positive integer	<p>Sets an upper limit on the number of SVM iterations. The default is system determined because it depends on the SVM solver.</p>

Table 9-16 (Cont.) Support Vector Machine Settings

Setting Name	Setting Value	Description
SVMS_NUM_PIVOTS	Range [1; 10000]	Sets an upper limit on the number of pivots used in the Incomplete Cholesky decomposition. It can be set only for non-linear kernels. The default value is 200.
SVMS_OUTLIER_RATE	TO_CHAR($0 < \text{numeric_expr} < 1$)	The desired rate of outliers in the training data. Valid for One-Class SVM models only (Anomaly Detection). The default value is 0.01.
SVMS_REGULARIZER	SVMS_REGULARIZER_L1 SVMS_REGULARIZER_L2	Controls the type of regularization that the SGD SVM solver uses. The setting applies only to linear SVM models. The default value is system determined because it depends on the potential model size.
SVMS_SOLVER	SVMS_SOLVER_SGD (Sub-Gradient Descend) SVMS_SOLVER_IPM (Interior Point Method)	Allows the user to choose the SVM solver. The SGD solver cannot be selected if the kernel is non-linear. The default value is system determined.
SVMS_STD_DEV	TO_CHAR(numeric_exp $r > 0$)	Controls the spread of the Gaussian kernel function. SVM uses a data-driven approach to find a standard deviation value that is on the same scale as distances between typical cases. Value of standard deviation for SVM algorithm. This is applicable only for the Gaussian kernel. The default value is estimated from the data by the algorithm.

 **See Also:**

- [About Model Settings](#)
- [Shared Settings](#)

Example 9-18 Using the oml.svm Class

This example demonstrates the use of various methods of the `oml.svm` class. In the listing for this example, some of the output is not shown as indicated by ellipses.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species']))

try:
```

```
        oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Create an SVM model object.
svm_mod = oml.svm('classification',
                  svms_kernel_function =
                  'dbms_data_mining.svms_linear')

# Fit the SVM Model according to the training data and parameter
# settings.
svm_mod.fit(train_x, train_y)

# Use the model to make predictions on test data.
svm_mod.predict(test_dat.drop('Species'),
                supplemental_cols = test_dat[:, ['Sepal_Length',
                                                'Sepal_Width',
                                                'Petal_Length',
                                                'Species']])

# Return the prediction probability.
svm_mod.predict(test_dat.drop('Species'),
                supplemental_cols = test_dat[:, ['Sepal_Length',
                                                'Sepal_Width',
                                                'Species']],
                proba = True)
svm_mod.predict_proba(test_dat.drop('Species'),
                     supplemental_cols = test_dat[:, ['Sepal_Length',
                                                       'Sepal_Width',
                                                       'Species']],
                     topN = 1).sort_values(by = ['Sepal_Length', 'Sepal_Width'])

svm_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                             'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
```

```

...             {0: 'setosa', 1: 'versicolor',
...             2:'virginica'}[x], iris.target)),
...             columns = ['Species'])
>>>
>>> try:
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
>>>
>>> # Create an SVM model object.
... svm_mod = oml.svm('classification',
...                   svms_kernel_function =
...                   'dbms_data_mining.svms_linear')
>>>
>>> # Fit the SVM model according to the training data and parameter
... # settings.
>>> svm_mod.fit(train_x, train_y)

```

Algorithm Name: Support Vector Machine

Mining Function: CLASSIFICATION

Target: Species

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_SUPPORT_VECTOR_MACHINES
1	CLAS_WEIGHTS_BALANCED	OFF
2	ODMS_DETAILS	ODMS_ENABLE
3	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
4	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
5	PREP_AUTO	ON
6	SVMS_CONV_TOLERANCE	.0001
7	SVMS_KERNEL_FUNCTION	SVMS_LINEAR

Computed Settings:

	setting name	setting value
0	SVMS_COMPLEXITY_FACTOR	10
1	SVMS_NUM_ITERATIONS	30
2	SVMS_SOLVER	SVMS_SOLVER_IPM

Global Statistics:

	attribute name	attribute value
0	CONVERGED	YES
1	ITERATIONS	14
2	NUM_ROWS	104

```
Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
```

```
Partition: NO
```

```
COEFFICIENTS:
```

	TARGET_VALUE	ATTRIBUTE_NAME	ATTRIBUTE_SUBNAME	ATTRIBUTE_VALUE	COEF
0	setosa	Petal_Length	None	None	-0.5809
1	setosa	Petal_Width	None	None	-0.7736
2	setosa	Sepal_Length	None	None	-0.1653
3	setosa	Sepal_Width	None	None	0.5689
4	setosa	None	None	None	-0.7355
5	versicolor	Petal_Length	None	None	1.1304
6	versicolor	Petal_Width	None	None	-0.3323
7	versicolor	Sepal_Length	None	None	-0.8877
8	versicolor	Sepal_Width	None	None	-1.2582
9	versicolor	None	None	None	-0.9091
10	virginica	Petal_Length	None	None	4.6042
11	virginica	Petal_Width	None	None	4.0681
12	virginica	Sepal_Length	None	None	-0.7985
13	virginica	Sepal_Width	None	None	-0.4328
14	virginica	None	None	None	-5.3180

```
>>> # Use the model to make predictions on test data.
... svm_mod.predict(test_dat.drop('Species'),
...                 supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Petal_Length',
...                                                 'Species']])
...
   Sepal_Length  Sepal_Width  Petal_Length  Species  PREDICTION
0             4.9           3.0           1.4   setosa   setosa
1             4.9           3.1           1.5   setosa   setosa
2             4.8           3.4           1.6   setosa   setosa
3             5.8           4.0           1.2   setosa   setosa
...           ...           ...           ...     ...         ...
44            6.7           3.3           5.7  virginica  virginica
45            6.7           3.0           5.2  virginica  virginica
46            6.5           3.0           5.2  virginica  virginica
47            5.9           3.0           5.1  virginica  virginica
```

```
>>> # Return the prediction probability.
... svm_mod.predict(test_dat.drop('Species'),
...                 supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Species']],
...                 proba = True)
...
   Sepal_Length  Sepal_Width  Species  PREDICTION  PROBABILITY
0             4.9           3.0   setosa   setosa      0.761886
1             4.9           3.1   setosa   setosa      0.805510
2             4.8           3.4   setosa   setosa      0.920317
3             5.8           4.0   setosa   setosa      0.998398
...           ...           ...     ...         ...         ...
```

```

44         6.7         3.3  virginica  virginica  0.927706
45         6.7         3.0  virginica  virginica  0.855353
46         6.5         3.0  virginica  virginica  0.799556
47         5.9         3.0  virginica  virginica  0.688024

>>> # Make predictions and return the probability for each class
... # on new data.
>>> svm_mod.predict_proba(test_dat.drop('Species'),
...   supplemental_cols = test_dat[:, ['Sepal_Length',
...   'Sepal_Width',
...   'Species']],
...   topN = 1).sort_values(by = ['Sepal_Length', 'Sepal_Width'])
   Sepal_Length  Sepal_Width  Species  TOP_1  TOP_1_VAL
0             4.4           3.0   setosa   setosa  0.698067
1             4.4           3.2   setosa   setosa  0.815643
2             4.5           2.3   setosa  versicolor  0.605105
3             4.8           3.4   setosa   setosa  0.920317
...           ...           ...     ...     ...     ...
44            6.7           3.3  virginica  virginica  0.927706
45            6.9           3.1  versicolor  versicolor  0.378391
46            6.9           3.1  virginica  virginica  0.881118
47            7.0           3.2  versicolor   setosa  0.586393

>>> svm_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.895833

```

9.19 Non-Negative Matrix Factorization

The `oml.nmf` class creates a Non-Negative Matrix Factorization (NMF) model for feature extraction.

Each feature extracted by NMF is a linear combination of the original attribution set. Each feature has a set of non-negative coefficients, which are a measure of the weight of each attribute on the feature. If the argument `allow.negative.scores` is `TRUE`, then negative coefficients are allowed.

Settings for a Non-Negative Matrix Factorization Models

The following table lists settings that apply to Non-Negative Matrix Factorization models.

Table 9-17 Non-Negative Matrix Factorization Model Settings

Setting Name	Setting Value	Description
NMFS_CONV_TOLERANCE	(0 < numeric_expr <= 0.5)	Convergence tolerance for NMF algorithm Default is 0.05
NMFS_NONNEGATIVE_SCORING	NMFS_NONNEG_SCORING_ENABLE NG NMFS_NONNEG_SCORING_DISABLE	Whether negative numbers should be allowed in scoring results. When set to <code>NMFS_NONNEG_SCORING_ENABLE</code> , negative feature values will be replaced with zeros. When set to <code>NMFS_NONNEG_SCORING_DISABLE</code> , negative feature values will be allowed. Default is <code>NMFS_NONNEG_SCORING_ENABLE</code>

Table 9-17 (Cont.) Non-Negative Matrix Factorization Model Settings

Setting Name	Setting Value	Description
NMFS_NUM_ITERATIONS	(1 <= numeric_expr <=500)	Number of iterations for NMF algorithm Default is 50
NMFS_RANDOM_SEED	(numeric_expr)	Random seed for NMF algorithm. Default is -1.

Example 9-19 Using the oml.nmf Class

This example creates an NMF model and uses some of the methods of the `oml.nmf` class.

```
import oml
import pandas as pd from sklearn import datasets
#For on-premises database follow the below command to connect to the database
oml.connect("<username>","<password>",dsn="dsn")

iris = datasets.load_iris()
x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width',
    'Petal_Length', 'Petal_Width'])
x.insert(0, "ID", range(1, len(x) + 1))
y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor',
    2:'virginica'}[x], iris.target)), columns = ['Species'])

z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

#Create training and test data sets.

train_dat, test_dat = oml.sync(table = "IRIS").split()

#Create a Non-Negative Matrix Factorization model using oml.nmf.

nmf_mod = oml.nmf()

#Fit the model to the training data.

nmf_mod = nmf_mod.fit(train_dat)

#Show the model details.

nmf_mod
#Use the model to make predictions on the test data, returning the
Sepal_Length, Sepal_Width, Petal_Length, and Species columns in the result.

nmf_mod.predict(test_dat, supplemental_cols = test_dat[:, ['Sepal_Length',
'Sepal_Width', 'Petal_Length', 'Species']])

nmf_mod.transform(test_dat, supplemental_cols = test_dat[:,
['Sepal_Length']], topN = 2).sort_values(by = ['Sepal_Length', 'TOP_1',
'TOP_1_VAL'])

#Feature comparison
```

```
nmf_mod.feature_compare(test_dat, compare_cols = ["Sepal_Length",
"Petal_Length"], supplemental_cols = ["Species"])

#Set new parameters and refit the model to produce U matrix output.

new_setting = {'nmfs_conv_tolerance':0.05}
nmf_mod2 = nmf_mod.set_params(**new_setting).fit(train_dat, case_id = "ID")
nmf_mod2
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets

>>> #For on-premises database follow the below command to connect to the
database
>>> oml.connect("<username>","<password>", dsn="<dsn>")

>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width',
'Petal_Length', 'Petal_Width'])
>>> x.insert(0, "ID", range(1, len(x) + 1))
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor',
2:'virginica'}[x], iris.target)), columns = ['Species'])

>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

#Create training and test data sets.

>>> dat = oml.sync(table = "IRIS").split()
>>> train_dat = dat[0]
>>> test_dat = dat[1]

#Create a Non-Negative Matrix Factorization model using oml.nmf.

>>> nmf_mod = oml.nmf()

#Fit the model to the training data.

>>> nmf_mod = nmf_mod.fit(train_dat)

#Show the model details.

>>> nmf_mod

Algorithm Name: Non-Negative Matrix Factorizationx
Mining Function: FEATURE_EXTRACTION

Settings:
          setting name          setting value
0          ALGO_NAME  ALGO_NONNEGATIVE_MATRIX_FACTOR
```

```

1          NMFS_CONV_TOLERANCE                .05
2    NMFS_NONNEGATIVE_SCORING    NMFS_NONNEG_SCORING_ENABLE
3          NMFS_NUM_ITERATIONS                50
4          NMFS_RANDOM_SEED                  -1
5          ODMs_DETAILS                      ODMs_ENABLE
6    ODMs_MISSING_VALUE_TREATMENT    ODMs_MISSING_VALUE_AUTO
7          ODMs_SAMPLING                    ODMs_SAMPLING_DISABLE
8          PREP_AUTO                          ON

```

Computed Settings:

```

          setting name setting value
0    FEAT_NUM_FEATURES                2
1    NMFS_NUM_ITERATIONS                2
2    ODMs_EXPLOSION_MIN_SUPP            1

```

Global Statistics:

```

          attribute name attribute value
0    CONVERGED                          YES
1    CONV_ERROR                          0.0444448
2    ITERATIONS                          2
3    NUM_ROWS                             111
4    SAMPLE_SIZE                          111

```

Attributes:

```

ID
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

```

Partition: NO

H:

	FEATURE_ID	FEATURE_NAME	ATTRIBUTE_NAME	ATTRIBUTE_VALUE	COEFFICIENT
0	1	1	ID	None	0.581551
1	1	1	Petal_Length	None	0.355323
2	1	1	Petal_Width	None	0.158492
3	1	1	Sepal_Length	None	0.656558
4	1	1	Sepal_Width	None	0.424101
5	1	1	Species	setosa	0.089560
6	1	1	Species	versicolor	0.534806
7	1	1	Species	virginica	0.539590
8	2	2	ID	None	0.344647
9	2	2	Petal_Length	None	0.506623
10	2	2	Petal_Width	None	0.650077
11	2	2	Sepal_Length	None	0.170237
12	2	2	Sepal_Width	None	0.248640
13	2	2	Species	setosa	0.249221
14	2	2	Species	versicolor	0.042316
15	2	2	Species	virginica	0.093861

W:

FEATURE_ID	FEATURE_NAME	ATTRIBUTE_NAME	ATTRIBUTE_VALUE	COEFFICIENT
------------	--------------	----------------	-----------------	-------------

```

0      1      1      ID      None      0.288559
1      1      1      Petal_Length      None      -0.062579
2      1      1      Petal_Width      None      -0.370128
3      1      1      Sepal_Length      None      0.502382
4      1      1      Sepal_Width      None      0.212611
5      1      1      Species      versicolor      0.486970
6      1      1      Species      setosa      -0.113835
7      1      1      Species      virginica      0.450038
8      2      2      ID      None      0.119462
9      2      2      Petal_Length      None      0.578697
10     2      2      Petal_Width      None      0.982575
11     2      2      Sepal_Length      None      -0.238993
12     2      2      Sepal_Width      None      0.082511
13     2      2      Species      setosa      0.353453
14     2      2      Species      versicolor      -0.359264
15     2      2      Species      virginica      -0.275074

```

#Use the model to make predictions on the test data, returning the Sepal_Length, Sepal_Width, Petal_Length, and Species columns in the result.

```

>>> nmf_mod.predict(test_dat, supplemental_cols = test_dat[:,
['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Species']])
      Sepal_Length  Sepal_Width  Petal_Length  Species  FEATURE_ID
0          5.0         3.6         1.4      setosa         2
1          5.0         3.4         1.5      setosa         2
2          4.4         2.9         1.4      setosa         2
3          4.9         3.1         1.5      setosa         2
...         ...         ...         ...         ...         ...
35         6.9         3.1         5.4  virginica         2
36         5.8         2.7         5.1  virginica         2
37         6.2         3.4         5.4  virginica         2
38         5.9         3.0         5.1  virginica         2

```

#Transform

```

>>> nmf_mod.transform(test_dat, supplemental_cols = test_dat[:,
['Sepal_Length']], topN = 2).sort_values(by = ['Sepal_Length', 'TOP_1',
'TOP_1_VAL'])
      Sepal_Length  TOP_1  TOP_1_VAL  TOP_2  TOP_2_VAL
0          4.4         2  0.464041     1  0.000000
1          4.4         2  0.482051     1  0.045518
2          4.8         2  0.475169     1  0.083874
3          4.8         2  0.510372     1  0.101880
...         ...         ...         ...         ...         ...
35         7.2         1  0.915012     2  0.850330
36         7.2         1  0.938112     2  0.745207
37         7.6         2  0.980757     1  0.864508
38         7.9         1  1.048287     2  0.947744

```

#Feature comparison

```

>>> nmf_mod.feature_compare(test_dat, compare_cols = ["Sepal_Length",
"Petal_Length"], supplemental_cols = ["Species"])
      Species_A  Species_B  SIMILARITY

```

```

0      setosa      setosa      0.990134
1      setosa      setosa      0.929516
2      setosa      setosa      0.976885
3      setosa      setosa      0.953770
...
737   virginica   virginica   0.849758
738   virginica   virginica   0.944063
739   virginica   virginica   0.983637
740   virginica   virginica   0.958018

```

[741 rows x 3 columns]

#Set new parameters and refit tthe model to produce U matrix output.

```

>>> new_setting = {'nmfs_conv_tolerance':0.05}
>>> nmf_mod2 = nmf_mod.set_params(**new_setting).fit(train_dat, case_id =
"ID")
>>> nmf_mod2

```

Algorithm Name: Non-Negative Matrix Factorizationx

Mining Function: FEATURE_EXTRACTION

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_NONNEGATIVE_MATRIX_FACTOR
1	NMFS_CONV_TOLERANCE	0.05
2	NMFS_NONNEGATIVE_SCORING	NMFS_NONNEG_SCORING_ENABLE
3	NMFS_NUM_ITERATIONS	50
4	NMFS_RANDOM_SEED	-1
5	ODMS_DETAILS	ODMS_ENABLE
6	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
7	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
8	PREP_AUTO	ON

Computed Settings:

	setting name	setting value
0	FEAT_NUM_FEATURES	2
1	NMFS_NUM_ITERATIONS	8
2	ODMS_EXPLOSION_MIN_SUPP	1

Global Statistics:

	attribute name	attribute value
0	CONVERGED	YES
1	CONV_ERROR	0.0277253
2	ITERATIONS	8
3	NUM_ROWS	111
4	SAMPLE_SIZE	111

Attributes:

Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

H:

	FEATURE_ID	FEATURE_NAME	ATTRIBUTE_NAME	ATTRIBUTE_VALUE	COEFFICIENT
0	1	1	Petal_Length	None	9.889792e-02
1	1	1	Petal_Width	None	1.060984e-01
2	1	1	Sepal_Length	None	1.947197e-01
3	1	1	Sepal_Width	None	5.099539e-01
4	1	1	Species	setosa	7.507257e-01
5	1	1	Species	versicolor	5.773815e-03
6	1	1	Species	virginica	8.136382e-02
7	2	2	Petal_Length	None	6.652922e-01
8	2	2	Petal_Width	None	6.571416e-01
9	2	2	Sepal_Length	None	5.702848e-01
10	2	2	Sepal_Width	None	2.420062e-01
11	2	2	Species	setosa	1.643131e-08
12	2	2	Species	versicolor	5.158020e-01
13	2	2	Species	virginica	4.948837e-01

W:

	FEATURE_ID	FEATURE_NAME	ATTRIBUTE_NAME	ATTRIBUTE_VALUE	COEFFICIENT
0	1	1	Petal_Length	None	-0.071259
1	1	1	Petal_Width	None	-0.059774
2	1	1	Sepal_Length	None	0.077608
3	1	1	Sepal_Width	None	0.571981
4	1	1	Species	versicolor	-0.144686
5	1	1	Species	setosa	0.947005
6	1	1	Species	virginica	-0.043170
7	2	2	Petal_Length	None	0.392684
8	2	2	Petal_Width	None	0.385395
9	2	2	Sepal_Length	None	0.304214
10	2	2	Sepal_Width	None	0.003195
11	2	2	Species	setosa	-0.221185
12	2	2	Species	versicolor	0.325338
13	2	2	Species	virginica	0.289804

9.20 Exponential Smoothing Method

The `oml.esm` function uses the Exponential Smoothing Method (ESM) algorithm to create a time series model.

Exponential Smoothing Methods have been widely used in forecasting for over half a century. It has applications at the strategic, tactical, and operation level. For example, at a strategic level, forecasting is used for projecting return on investment, growth and the effect of innovations. At a tactical level, forecasting is used for projecting costs, inventory requirements, and customer satisfaction. At an operational level, forecasting is used for setting targets and predicting quality and conformance with standards.

In its simplest form, Exponential Smoothing is a moving average method with a single parameter that models an exponentially decreasing effect of past levels on future values. With a variety of extensions, Exponential Smoothing covers a broader class of models than other well-known approaches, such as the Box-Jenkins auto-regressive integrated moving average

(ARIMA) approach. Oracle Machine Learning implements Exponential Smoothing using a state-of-the-art state space method that incorporates a single source of error (SSOE) assumption that provides theoretical and performance advantages.

Settings for an ESM model

The following table lists settings for ESM models.

Table 9-18 ESM Model Settings

Setting Name	Setting Value	Description
EXSM_MODEL	It can take value in set {EXSM_SIMPLE, EXSM_SIMPLE_MULT, EXSM_HOLT, EXSM_HOLT_DMP, EXSM_MUL_TRND, EXSM_MULTRD_DMP, EXSM_SEAS_ADD, EXSM_SEAS_MUL, EXSM_HW, EXSM_HW_DMP, EXSM_HW_ADDSEA, EXSM_DHW_ADDSEA, EXSM_HWMT, EXSM_HWMT_DMP}	This setting specifies the model. EXSM_SIMPLE: Simple exponential smoothing model is applied. EXSM_SIMPLE_MULT: Simple exponential smoothing model with multiplicative error is applied. EXSM_HOLT: Holt linear exponential smoothing model is applied. EXSM_HOLT_DMP: Holt linear exponential smoothing model with damped trend is applied. EXSM_MUL_TRND: Exponential smoothing model with multiplicative trend is applied. EXSM_MULTRD_DMP: Exponential smoothing model with multiplicative damped trend is applied. EXSM_SEAS_ADD: Exponential smoothing with additive seasonality, but no trend, is applied. EXSM_SEAS_MUL: Exponential smoothing with multiplicative seasonality, but no trend, is applied. EXSM_HW: Holt-Winters triple exponential smoothing model, additive trend, multiplicative seasonality is applied. EXSM_HW_DMP: Holt-Winters multiplicative exponential smoothing model with damped trend, additive trend, multiplicative seasonality is applied. EXSM_HW_ADDSEA: Holt-Winters additive exponential smoothing model, additive trend, additive seasonality is applied. EXSM_DHW_ADDSEA: Holt-Winters additive exponential smoothing model with damped trend, additive trend, additive seasonality is applied. EXSM_HWMT: Holt-Winters multiplicative exponential smoothing model with multiplicative trend, multiplicative trend, multiplicative seasonality is applied. EXSM_HWMT_DMP: Holt-Winters multiplicative exponential smoothing model with damped multiplicative trend, multiplicative trend, multiplicative seasonality is applied. The default value is EXSM_SIMPLE.

Table 9-18 (Cont.) ESM Model Settings

Setting Name	Setting Value	Description
EXSM_SEASONALITY	positive integer > 1	<p>This setting specifies a positive integer value as the length of seasonal cycle. The value specified must be larger than 1. For example, setting value 4 means that every group of four observations forms a seasonal cycle.</p> <p>This setting is only applicable and must be provided for models with seasonality, otherwise the model throws an error.</p> <p>When EXSM_INTERVAL is not set, this setting applies to the original input time series. When EXSM_INTERVAL is set, this setting applies to the accumulated time series.</p>
EXSM_INTERVAL	It can take value in set {EXSM_INTERVAL_YEAR, EXSM_INTERVAL_QTR, EXSM_INTERVAL_MONTH, EXSM_INTERVAL_WEEK, EXSM_INTERVAL_DAY, EXSM_INTERVAL_HOUR, EXSM_INTERVAL_MIN, EXSM_INTERVAL_SEC}	<p>This setting only applies and must be provided when the time column (<i>case_id</i> column) has datetime type. It specifies the spacing interval of the accumulated equally spaced time series.</p> <p>The model throws an error if the time column of input table is of datetime type and setting EXSM_INTERVAL is not provided.</p> <p>The model throws an error if the time column of input table is of oracle number type and setting EXSM_INTERVAL is provided.</p>
EXSM_ACCUMULATE	It can take value in set {EXSM_ACCU_TOTAL, EXSM_ACCU_STD, EXSM_ACCU_MAX, EXSM_ACCU_MIN, EXSM_ACCU_AVG, EXSM_ACCU_MEDIAN, EXSM_ACCU_COUNT}	<p>This setting only applies and must be provided when the time column has datetime type. It specifies how to generate the value of the accumulated time series from the input time series.</p>

Table 9-18 (Cont.) ESM Model Settings

Setting Name	Setting Value	Description
EXSM_SETMISSING	It can also specify an option taking value in set {EXSM_MISS_MIN, EXSM_MISS_MAX, EXSM_MISS_AVG, EXSM_MISS_MEDIAN, EXSM_MISS_LAST, EXSM_MISS_FIRST, EXSM_MISS_PREV, EXSM_MISS_NEXT, EXSM_MISS_AUTO}.	<p>This setting specifies how to handle missing values, which may come from input data and/or the accumulation process of time series. You can specify either a number or an option. If a number is specified, all the missing values are set to that number.</p> <p>EXSM_MISS_MIN: Replaces missing value with minimum of the accumulated time series.</p> <p>EXSM_MISS_MAX: Replaces missing value with maximum of the accumulated time series.</p> <p>EXSM_MISS_AVG: Replaces missing value with average of the accumulated time series.</p> <p>EXSM_MISS_MEDIAN: Replaces missing value with median of the accumulated time series.</p> <p>EXSM_MISS_LAST: Replaces missing value with last non-missing value of the accumulated time series.</p> <p>EXSM_MISS_FIRST: Replaces missing value with first non-missing value of the accumulated time series.</p> <p>EXSM_MISS_PREV: Replaces missing value with the previous non-missing value of the accumulated time series.</p> <p>EXSM_MISS_NEXT: Replaces missing value with the next non-missing value of the accumulated time series.</p> <p>EXSM_MISS_AUTO: EXSM model treats the input data as an irregular (non-uniformly spaced) time series. If this setting is not provided, EXSM_MISS_AUTO is the default value. In such a case, the model treats the input time series as irregular time series, viewing missing values as gaps.</p>
EXSM_PREDICTION_STEP	It must be set to a number between 1-30.	<p>This setting specifies how many steps ahead the predictions are to be made.</p> <p>If it is not set, the default value is 1: the model gives one-step-ahead prediction. A value greater than 30 results in an error.</p>
EXSM_CONFIDENCE_LEVEL	It must be a number between 0 and 1, exclusive.	<p>This setting specifies the desired confidence level for prediction.</p> <p>The lower and upper bounds of the specified confidence interval is reported. If this setting is not specified, the default confidence level is 95%.</p>

Table 9-18 (Cont.) ESM Model Settings

Setting Name	Setting Value	Description
EXSM_OPT_CRITERION	It takes value in set {EXSM_OPT_CRIT_LIK, EXSM_OPT_CRIT_MSE, EXSM_OPT_CRIT_AMSE, EXSM_OPT_CRIT_SIG, EXSM_OPT_CRIT_MAE}.	This setting specifies the desired optimization criterion. The optimization criterion is useful as a diagnostic for comparing models' fit to the same data. EXSM_OPT_CRIT_LIK: Minus twice the log-likelihood of a model. EXSM_OPT_CRIT_MSE: Mean square error of a model. EXSM_OPT_CRIT_AMSE: Average mean square error over user-specified time window. EXSM_OPT_CRIT_SIG: Model's standard deviation of residuals. EXSM_OPT_CRIT_MAE: Mean absolute error of a model. The default value is EXSM_OPT_CRIT_LIK.
EXSM_NMSE	positive integer	This setting specifies the length of the window used in computing the error metric average mean square error (AMSE).

Example 9-20 Using the oml.esm Class

This example creates an ESM model and uses some of the methods of the `oml.esm` class.

```
import oml
import pandas as pd

df = pd.DataFrame({'EVENT': ['A', 'B', 'C', 'D'],
                  'START': ['2021-10-04 13:29:00', '2021-10-07 12:30:00',
                           '2021-10-15 04:20:00', '2021-10-18 15:45:03'],
                  'END':   ['2021-10-08 11:29:06', '2021-10-15 10:30:07',
                           '2021-10-29 05:50:15', '2021-10-22 15:40:03']})

df['START'] = pd.to_datetime(df['START'])
df['END'] = pd.to_datetime(df['END'])
df['DURATION'] = df['END'] - df['START']
df['HOURS'] = df['DURATION'] / pd.Timedelta(hours=1)
df['MINUTES'] = df['DURATION'] / pd.Timedelta(minutes=1)
#For on-premises database follow the below command to connect to the database#
oml.connect("<username>", "<password>", dsn="<dsn>")
dat = oml.create(df, table='DF')
train_x = dat[:, 1]
train_y = dat[:, 4]

setting = {'EXSM_INTERVAL': 'EXSM_INTERVAL_DAY'}
esm_mod = oml.esm(**setting).fit(train_x, train_y, time_seq = 'START')

esm_mod
train_x = dat[:, 4]
train_y = dat[:, 5]
esm_mod = oml.esm().fit(train_x, train_y, time_seq = 'HOURS')

esm_mod
```

Listing for This Example

Create pandas DataFrame with start and end dates for an event. Convert start and end date columns to datetime, and create new columns that contain timedelta between the start and end dates. Convert timedelta into total number of hours and convert timedelta into total number of minutes.

```
>>> import oml
>>> import pandas as pd

>>> df = pd.DataFrame({'EVENT': ['A', 'B', 'C', 'D'],
                      'START': ['2021-10-04 13:29:00', '2021-10-07 12:30:00',
                                '2021-10-15 04:20:00', '2021-10-18 15:45:03'],
                      'END':   ['2021-10-08 11:29:06', '2021-10-15 10:30:07',
                                '2021-10-29 05:50:15', '2021-10-22 15:40:03']})

>>> df['START'] = pd.to_datetime(df['START'])
>>> df['END'] = pd.to_datetime(df['END'])
>>> df['DURATION'] = df['END'] - df['START']
>>> df['HOURS'] = df['DURATION'] / pd.Timedelta(hours=1)
>>> df['MINUTES'] = df['DURATION'] / pd.Timedelta(minutes=1)

>>> #For on-premises database follow the below command to connect to the
database#
>>> oml.connect("<username>", "<password>", dsn="<dsn>")
>>> dat = oml.create(df, table='DF')
```

Using Datetime type

```
>>> train_x = dat[:, 1]
>>> train_y = dat[:, 4]

>>> setting = {'EXSM_INTERVAL': 'EXSM_INTERVAL_DAY'}
>>> esm_mod = oml.esm(**setting).fit(train_x, train_y, time_seq = 'START')

>>> esm_mod
```

Algorithm Name: Exponential Smoothing

Mining Function: TIME_SERIES

Target: HOURS

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_EXPONENTIAL_SMOOTHING
1	EXSM_ACCUMULATE	EXSM_ACCU_TOTAL
2	EXSM_CONFIDENCE_LEVEL	.95
3	EXSM_INTERVAL	EXSM_INTERVAL_DAY
4	EXSM_NMSE	3
5	EXSM_OPTIMIZATION_CRIT	EXSM_OPT_CRIT_LIK
6	EXSM_PREDICTION_STEP	1
7	EXSM_SETMISSING	EXSM_MISS_AUTO
8	ODMS_BOXCOX	ODMS_BOXCOX_ENABLE
9	ODMS_DETAILS	ODMS_ENABLE

```

10 ODMS_MISSING_VALUE_TREATMENT    ODMS_MISSING_VALUE_AUTO
11                ODMS_SAMPLING      ODMS_SAMPLING_DISABLE
12                PREP_AUTO           ON

```

Computed Settings:

```

    setting name setting value
0  EXSM_MODEL    EXSM_SIMPLE

```

Global Statistics:

```

    attribute name attribute value
0  -2 LOG-LIKELIHOOD      -21.1618
1                AIC       48.3236
2                AICC      None
3                ALPHA     0.000100034
4                ALPHA DISC 0.9999
5                AMSE      12175.3
6                BIC       46.4825
7                CONVERGED  YES
8                INITIAL ALPHA 0.000100034
9                INITIAL LEVEL 179.353
10               MAE       84.403
11               MSE      9843.9
12               NUM_ROWS   4
13               SIGMA     140.313
14               STD       140.313

```

Attributes:

Partition: NO

Prediction:

	TIME_SEQ	VALUE	PREDICTION	LOWER	UPPER
0	2021-10-04	94.001667	179.352705	NaN	NaN
1	2021-10-07	190.001944	179.344167	NaN	NaN
2	2021-10-15	337.504167	179.345233	NaN	NaN
3	2021-10-18	95.916667	179.361069	NaN	NaN
4	2021-10-19	NaN	179.352712	-95.656158	454.361582

Using Float type

```

>>> train_x = dat[:, 4]
>>> train_y = dat[:, 5]
>>> esm_mod = oml.esm().fit(train_x, train_y, time_seq = 'HOURS')

```

```

>>> esm_mod

```

Algorithm Name: Exponential Smoothing

Mining Function: TIME_SERIES

Target: MINUTES

Settings:

```

    setting name                setting value

```

```

0          ALGO_NAME      ALGO_EXPONENTIAL_SMOOTHING
1      EXSM_CONFIDENCE_LEVEL      .95
2          EXSM_NMSE      3
3      EXSM_OPTIMIZATION_CRIT      EXSM_OPT_CRIT_LIK
4      EXSM_PREDICTION_STEP      1
5          EXSM_SETMISSING      EXSM_MISS_AUTO
6          ODMS_BOXCOX      ODMS_BOXCOX_ENABLE
7          ODMS_DETAILS      ODMS_ENABLE
8      ODMS_MISSING_VALUE_TREATMENT      ODMS_MISSING_VALUE_AUTO
9          ODMS_SAMPLING      ODMS_SAMPLING_DISABLE
10         PREP_AUTO      ON

```

Computed Settings:

```

      setting name setting value
0      EXSM_MODEL      EXSM_HOLT

```

Global Statistics:

```

      attribute name attribute value
0      -2 LOG-LIKELIHOOD      4.47424
1          AIC      1.05153
2          AICC      None
3          ALPHA      0.000104161
4          AMSE      0.0190133
5          BETA      0.000104153
6          BIC      -2.017
7      CONVERGED      YES
8      INITIAL_LEVEL      8.00977
9      INITIAL_TREND      0.452033
10         LAMBDA      4.08563e-05
11         MAE      1175.53
12         MSE      0.0266914
13         NUM_ROWS      4
14         SIGMA      0.188649
15         STD      0.188649

```

Attributes:

Partition: NO

Prediction:

```

      TIME_SEQ      VALUE      PREDICTION      LOWER      UPPER
0          94      5640.100000      4807.666451      NaN      NaN
1          95      5755.000000      7554.329741      NaN      NaN
2         190      11400.116667      11869.239245      NaN      NaN
3         337      20250.250000      18649.004898      NaN      NaN
4         338      NaN      29301.840039      19894.31833      41663.104953

```

9.21 XGBoost

The `oml.xgb` class supports the in-database scalable gradient tree boosting algorithm for both classification, regression specifications, ranking models, and survival models. It makes available the open source gradient boosting framework. It prepares the categorical encoding and missing value replacement from the OML infrastructure, calls the in-database XGBoost,

builds and persists a model as a first-class database model object, and supports using the model for prediction.

You can use `oml.xgb` as a stand-alone predictor or incorporate it into real-world production pipelines for a wide range of problems such as ad click-through rate prediction, hazard risk prediction, web text classification, and so on.

The `oml.xgb` algorithm takes three types of parameters: general parameters, booster parameters, and task parameters. You set the parameters through the model settings. The algorithm supports most of the settings of the open source XGBoost project. For more information on the supported settings, see [XGBoost parameters](#).

Through `oml.xgb`, OML4Py supports a number of different classification and regression specifications, ranking models, and survival models. Binary and multi-class models are supported under the classification machine learning technique while regression, ranking, count, and survival are supported under the regression machine learning technique.

`oml.xgb` also supports partitioned models and internalizes the data preparation.

Settings for an XGBoost model

The following table lists settings that apply to XGBoost models.

Table 9-19 XGBoost Model Settings

Setting Name	Setting Value	Description
<code>booster</code>	A string that is one of the following: <ul style="list-style-type: none"> <code>dart</code> <code>gblinear</code> <code>gbtree</code> 	The booster to use: <ul style="list-style-type: none"> <code>dart</code> <code>gblinear</code> <code>gbtree</code> <p>The <code>dart</code> and <code>gbtree</code> boosters use tree-based models whereas <code>gblinear</code> uses linear functions.</p> <p>The default value is <code>gbtree</code>.</p>
<code>num_round</code>	A non-negative integer.	The number of rounds for boosting.
		The default value is 10.

For more information on the booster settings, see [XGBoost parameters](#)

Example 9-21 Using the `oml.xgb` Class

This example creates an XGB model and uses some of the methods of the `oml.xgb` class.

```
#Load the iris data from sklearn and combine the target and predictors into a
single DataFrame, which matches the form of a database table.
Use the oml.create function to load this Pandas DataFrame into the databae,
which creates a persistent table and returns a proxy object that you assign
to z.#

import oml
from sklearn import datasets
import pandas as pd

iris = datasets.load_iris()
x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width',
'Sepal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor',
```

```

2:'virginica'}[x], iris.target)), columns = ['Species'])

#For on-premises database follow the below command to connect to the database#
oml.connect("<username>","<password>", dsn="<dsn>")
z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

#Create training data and test data.#

dat = oml.sync(table = "IRIS").split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

#Classification Example:#

#Create an XGBoost model object.#

setting = {'xgboost_max_depth': '3',
...         'xgboost_eta': '1',
...         'xgboost_num_round': '10'}

xgb_mod = oml.xgb('classification', **setting)

#Fit the XGBoost model to the training data.#

xgb_mod.fit(train_x, train_y)
#Use the model to make predictions on the test data and return the prediction
probabilities for each category in Species.#
xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:,
['Sepal_Length', 'Sepal_Width', 'Species']], proba = True).sort_values(by =
['Sepal_Length', 'Sepal_Width'])

```

	Sepal_Length	Sepal_Width	Species	TOP_1	TOP_1_VAL
0	4.4	3.0	setosa	setosa	0.993619
1	4.4	3.2	setosa	setosa	0.993619
2	4.5	2.3	setosa	setosa	0.942128
3	4.8	3.4	setosa	setosa	0.993619
...
42	6.7	3.3	virginica	virginica	0.996170
43	6.9	3.1	versicolor	versicolor	0.925217
44	6.9	3.1	virginica	virginica	0.996170
45	7.0	3.2	versicolor	versicolor	0.990586

```

#Create training data and test data.#

dat = oml.sync(table = "IRIS").split()

train_x = dat[0].drop('Sepal_Length')
train_y = dat[0]['Sepal_Length']
test_dat = dat[1]

#Create an XGBoost model object.#

setting = {'xgboost_booster': 'gblinear'}
xgb_mod = oml.xgb('regression', **setting)

#Fit the XGBoost Model according to the training data and parameter settings.#

```

```

xgb_mod.fit(train_x, train_y)
xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:,
['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Species']]) # doctest:
+NORMALIZE_WHITESPACE, +ELLIPSIS
#Create an XGBoost model object.#

setting = {'xgboost_objective': 'rank:pairwise',
...         'xgboost_max_depth': '3',
...         'xgboost_eta': '0.1',
...         'xgboost_gamma': '1.0',
...         'xgboost_num_round': '4'}

xgb_mod = oml.xgb('regression', **setting)

#Fit the XGBoost Model according to the training data and parameter settings.#

xgb_mod.fit(train_x, train_y)
#Use the model to make predictions on the test data, returning the
Sepal_Length, Sepal_Width, Petal_Length, and Species columns in the result.#

xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:,
['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Species']])

```

Listing for This Example

#Load the iris data from sklearn and combine the target and predictors into a single DataFrame, which matches the form of a database table. Use the oml.create function to load this Pandas DataFrame into the database, which creates a persistent table and returns a proxy object that you assign to z.#

```

>>> import oml
>>> from sklearn import datasets
>>> import pandas as pd

>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width',
'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor',
2:'virginica'}[x], iris.target)), columns = ['Species'])

>>> #For on-premises database follow the below command to connect to the
database#
>>> oml.connect("<username>", "<password>", dsn="<dsn>")
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

#Create training data and test data.#

>>> dat = oml.sync(table = "IRIS").split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]

```



```

#Classification Example:#

#Create an XGBoost model object.#

>>> setting = {'xgboost_max_depth': '3',
...           'xgboost_eta': '1',
...           'xgboost_num_round': '10'}

>>> xgb_mod = oml.xgb('classification', **setting)

#Fit the XGBoost model to the training data.#

>>> xgb_mod.fit(train_x, train_y)

Algorithm Name: XGBOOST
Mining Function: CLASSIFICATION
Target: Species

Settings:
           setting name           setting value
0           ALGO_NAME             ALGO_XGBOOST
1     CLAS_WEIGHTS_BALANCED             OFF
2           ODMS_DETAILS             ODMS_ENABLE
3     ODMS_MISSING_VALUE_TREATMENT     ODMS_MISSING_VALUE_AUTO
4           ODMS_SAMPLING             ODMS_SAMPLING_DISABLE
5           PREP_AUTO                 ON
6           booster                  gbtree
7           eta                       1
8           max_depth                 3
9           ntree_limit                0
10          num_round                 10
11          objective                 multi:softprob

Global Statistics:
  attribute name attribute value
0     NUM_ROWS             104
1     mlogloss             0.024858

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

ATTRIBUTE IMPORTANCE:

  PNAME ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE   GAIN   COVER
\
0  None   Petal_Length           None           None  0.743941
0.560554
1  None   Petal_Width           None           None  0.162191
0.245400
2  None   Sepal_Length           None           None  0.003738
0.044741

```

```
3 None Sepal_Width None None 0.090129
0.149306
```

```
FREQUENCY
0 0.447761
1 0.268657
2 0.119403
3 0.164179
```

```
#Use the model to make predictions on the test data and return the prediction
probabilities for each category in Species.#
```

```
>>> xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:,
['Sepal_Length', 'Sepal_Width', 'Species']], proba = True).sort_values(by =
['Sepal_Length', 'Sepal_Width'])
```

	Sepal_Length	Sepal_Width	Species	TOP_1	TOP_1_VAL
0	4.4	3.0	setosa	setosa	0.993619
1	4.4	3.2	setosa	setosa	0.993619
2	4.5	2.3	setosa	setosa	0.942128
3	4.8	3.4	setosa	setosa	0.993619
...
42	6.7	3.3	virginica	virginica	0.996170
43	6.9	3.1	versicolor	versicolor	0.925217
44	6.9	3.1	virginica	virginica	0.996170
45	7.0	3.2	versicolor	versicolor	0.990586

```
#Regression Example:#
```

```
#Create training data and test data.#
```

```
>>> dat = oml.sync(table = "IRIS").split()
```

```
>>> train_x = dat[0].drop('Sepal_Length')
```

```
>>> train_y = dat[0]['Sepal_Length']
```

```
>>> test_dat = dat[1]
```

```
#Create an XGBoost model object.#
```

```
>>> setting = {'xgboost_booster': 'gblinear'}
```

```
>>> xgb_mod = oml.xgb('regression', **setting)
```

```
#Fit the XGBoost Model according to the training data and parameter settings.#
```

```
>>> xgb_mod.fit(train_x, train_y)
```

```
Algorithm Name: XGBOOST
Mining Function: REGRESSION
Target: Sepal_Length
```

```
Settings:
```

	setting name	setting value
0	ALGO_NAME	ALGO_XGBOOST
1	ODMS_DETAILS	ODMS_ENABLE
2	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
3	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE

```

4          PREP_AUTO                ON
5          booster                   gblinear
6          ntree_limit                0
7          num_round                  10

```

Computed Settings:

```

          setting name setting value
0  ODSM_EXPLOSION_MIN_SUPP          1

```

Global Statistics:

```

          attribute name attribute value
0          NUM_ROWS                104
1          rmse                     0.364149

```

Attributes:

```

Petal_Length
Petal_Width
Sepal_Width
Species

```

Partition: NO

ATTRIBUTE IMPORTANCE:

	PNAME	ATTRIBUTE_NAME	ATTRIBUTE_SUBNAME	ATTRIBUTE_VALUE	WEIGHT	CLASS
0	None	Petal_Length	None	None	0.335183	0
1	None	Petal_Width	None	None	0.368738	0
2	None	Sepal_Width	None	None	0.249208	0
3	None	Species	None	versicolor	-0.197582	0
4	None	Species	None	virginica	-0.170522	0

```

>>> xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:,
['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Species']]) # doctest:

```

```

+NORMALIZE_WHITESPACE, +ELLIPSIS

```

	Sepal_Length	Sepal_Width	Petal_Length	Species	PREDICTION
0	4.9	3.0	1.4	setosa	4.797075
1	4.9	3.1	1.5	setosa	4.818641
2	4.8	3.4	1.6	setosa	4.963796
3	5.8	4.0	1.2	setosa	4.979247
...
42	6.7	3.3	5.7	virginica	6.990700
43	6.7	3.0	5.2	virginica	6.674599
44	6.5	3.0	5.2	virginica	6.563977
45	5.9	3.0	5.1	virginica	6.456711

```

#Ranking Example:#

```

```

#Create an XGBoost model object.#

```

```

>>> setting = {'xgboost_objective': 'rank:pairwise',
...           'xgboost_max_depth': '3',
...           'xgboost_eta': '0.1',
...           'xgboost_gamma': '1.0',
...           'xgboost_num_round': '4'}

```

```

>>> xgb_mod = oml.xgb('regression', **setting)

#Fit the XGBoost Model according to the training data and parameter settings.#

>>> xgb_mod.fit(train_x, train_y)
Algorithm Name: XGBOOST
Mining Function: REGRESSION
Target: Sepal_Length

Settings:
           setting name           setting value
0           ALGO_NAME             ALGO_XGBOOST
1           ODMS_DETAILS          ODMS_ENABLE
2 ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
3           ODMS_SAMPLING        ODMS_SAMPLING_DISABLE
4           PREP_AUTO             ON
5           booster              gbtree
6           eta                   0.1
7           gamma                 1.0
8           max_depth            3
9           ntree_limit          0
10          num_round            4
11          objective            rank:pairwise

Computed Settings:
           setting name setting value
0 ODMS_EXPLOSION_MIN_SUPP      1

Global Statistics:
  attribute name  attribute value
0      NUM_ROWS      104
1      map           1

Attributes:
Petal_Length
Petal_Width
Sepal_Width
Species

Partition: NO

ATTRIBUTE IMPORTANCE:

  PNAME ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE      GAIN      COVER
\
0 None   Petal_Length          None           None  0.873855
0.677624
1 None   Petal_Width           None           None  0.083504
0.184802
2 None   Sepal_Width           None           None  0.042641
0.137574

  FREQUENCY
0  0.500000
1  0.285714
2  0.214286

```

```
#Use the model to make predictions on the test data, returning the  
Sepal_Length, Sepal_Width, Petal_Length, and Species columns in the result.#
```

```
>>> xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:,  
['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Species']])
```

	Sepal_Length	Sepal_Width	Petal_Length	Species	PREDICTION
0	4.9	3.0	1.4	setosa	0.243485
1	4.9	3.1	1.5	setosa	0.243485
2	4.8	3.4	1.6	setosa	0.243485
3	5.8	4.0	1.2	setosa	0.310980
...
42	6.7	3.3	5.7	virginica	0.771761
43	6.7	3.0	5.2	virginica	0.728637
44	6.5	3.0	5.2	virginica	0.728637
45	5.9	3.0	5.1	virginica	0.674835

10

Automated Machine Learning

Use the automated algorithm selection, feature selection, and hyperparameter tuning of Automated Machine Learning to accelerate the machine learning modeling process.

Automated Machine Learning in OML4Py is described in the following topics:

- [About Automated Machine Learning](#)
Automated Machine Learning (AutoML) provides built-in data science expertise about data analytics and modeling that you can employ to build machine learning models.
- [Algorithm Selection](#)
The `oml.automl.AlgorithmSelection` class uses the characteristics of the data set and the task to rank algorithms from the set of supported Oracle Machine Learning algorithms.
- [Feature Selection](#)
The `oml.automl.FeatureSelection` class identifies the most relevant feature subsets for a training data set and an Oracle Machine Learning algorithm.
- [Model Tuning](#)
The `oml.automl.ModelTuning` class tunes the hyperparameters for the specified classification or regression algorithm and training data.
- [Model Selection](#)
The `oml.automl.ModelSelection` class automatically selects an Oracle Machine Learning algorithm according to the selected score metric and then tunes that algorithm.

10.1 About Automated Machine Learning

Automated Machine Learning (AutoML) provides built-in data science expertise about data analytics and modeling that you can employ to build machine learning models.

Any modeling problem for a specified data set and prediction task involves a sequence of data cleansing and preprocessing, algorithm selection, and model tuning tasks. Each of these steps require data science expertise to help guide the process to an efficient final model. Automated Machine Learning (AutoML) automates this process with its built-in data science expertise.

OML4Py has the following AutoML capabilities:

- Automated algorithm selection that selects the appropriate algorithm from the supported machine learning algorithms
- Automated feature selection that reduces the size of the original feature set to speed up model training and tuning, while possibly also increasing model quality
- Automated tuning of model hyperparameters, which selects the model with the highest score metric from among several metrics as selected by the user

AutoML performs those common modeling tasks automatically, with less effort and potentially better results. It also leverages in-database algorithm parallel processing and scalability to minimize runtime and produce high-quality results.



Note:

As the `fit` method of the machine learning classes does, the AutoML functions `reduce`, `select`, and `tune` provide a `case_id` parameter that you can use to achieve repeatable data sampling and data shuffling during model building.

The AutoML functionality is also available in a no-code user interface alongside OML Notebooks on Oracle Autonomous Database. For more information, see [Oracle Machine Learning AutoML User Interface](#).

Automated Machine Learning Classes and Algorithms

The Automated Machine Learning classes are the following.

Class	Description
<code>oml.automl.AlgorithmSelection</code>	Using only the characteristics of the data set and the task, automatically selects the best algorithms from the set of supported Oracle Machine Learning algorithms. Supports classification and regression functions.
<code>oml.automl.FeatureSelection</code>	Uses meta-learning to quickly identify the most relevant feature subsets given a training data set and an Oracle Machine Learning algorithm. Supports classification and regression functions.
<code>oml.automl.ModelTuning</code>	Uses a highly parallel, asynchronous gradient-based hyperparameter optimization algorithm to tune the algorithm hyperparameters. Supports classification and regression functions.
<code>oml.automl.ModelSelection</code>	Selects the best Oracle Machine Learning algorithm and then tunes that algorithm. Supports classification and regression functions.

The Oracle Machine Learning algorithms supported by AutoML are the following:

Table 10-1 Machine Learning Algorithms Supported by AutoML

Algorithm Abbreviation	Algorithm Name
<code>dt</code>	Decision Tree
<code>glm</code>	Generalized Linear Model
<code>glm_ridge</code>	Generalized Linear Model with ridge regression
<code>nb</code>	Naive Bayes
<code>nn</code>	Neural Network
<code>rf</code>	Random Forest
<code>svm_gaussian</code>	Support Vector Machine with Gaussian kernel
<code>svm_linear</code>	Support Vector Machine with linear kernel

Classification and Regression Metrics

The following tables list the scoring metrics supported by AutoML.

Table 10-2 Binary and Multiclass Classification Metrics

Metric	Description, Scikit-learn Equivalent, and Formula
accuracy	<p>Calculates the rate of correct classification of the target.</p> <pre>sklearn.metrics.accuracy_score(y_true, y_pred, normalize=True, sample_weight=None)</pre> <p>Formula: $(tp + tn) / \text{samples}$</p>
f1_macro	<p>Calculates the f-score or f-measure, which is a weighted average of the precision and recall. The f1_macro takes the unweighted average of per-class scores.</p> <pre>sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='macro', sample_weight=None)</pre> <p>Formula: $2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$</p>
f1_micro	<p>Calculates the f-score or f-measure with micro-averaging in which true positives, false positives, and false negatives are counted globally.</p> <pre>sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='micro', sample_weight=None)</pre> <p>Formula: $2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$</p>
f1_weighted	<p>Calculates the f-score or f-measure with weighted averaging of per-class scores based on support (the fraction of true samples per class). Accounts for imbalanced classes.</p> <pre>sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='weighted', sample_weight=None)</pre> <p>Formula: $2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$</p>
precision_macro	<p>Calculates the ability of the classifier to not label a sample incorrectly. The precision_macro takes the unweighted average of per-class scores.</p> <pre>sklearn.metrics.precision_score(y_true, y_pred, labels=None, pos_label=1, average='macro', sample_weight=None)</pre> <p>Formula: $tp / (tp + fp)$</p>
precision_micro	<p>Calculates the ability of the classifier to not label a sample incorrectly. Uses micro-averaging in which true positives, false positives, and false negatives are counted globally.</p> <pre>sklearn.metrics.precision_score(y_true, y_pred, labels=None, pos_label=1, average='micro', sample_weight=None)</pre> <p>Formula: $tp / (tp + fp)$</p>

Table 10-2 (Cont.) Binary and Multiclass Classification Metrics

Metric	Description, Scikit-learn Equivalent, and Formula
precision_weighted	<p>Calculates the ability of the classifier to not label a sample incorrectly. Uses weighted averaging of per-class scores based on support (the fraction of true samples per class). Accounts for imbalanced classes.</p> <pre>sklearn.metrics.precision_score(y_true, y_pred, labels=None, pos_label=1, average='weighted', sample_weight=None)</pre> <p>Formula: $tp / (tp + fp)$</p>
recall_macro	<p>Calculates the ability of the classifier to correctly label each class. The recall_macro takes the unweighted average of per-class scores.</p> <pre>sklearn.metrics.recall_score(y_true, y_pred, labels=None, pos_label=1, average='macro', sample_weight=None)</pre> <p>Formula: $tp / (tp + fn)$</p>
recall_micro	<p>Calculates the ability of the classifier to correctly label each class with micro-averaging in which the true positives, false positives, and false negatives are counted globally.</p> <pre>sklearn.metrics.recall_score(y_true, y_pred, labels=None, pos_label=1, average='micro', sample_weight=None)</pre> <p>Formula: $tp / (tp + fn)$</p>
recall_weighted	<p>Calculates the ability of the classifier to correctly label each class with weighted averaging of per-class scores based on support (the fraction of true samples per class). Accounts for imbalanced classes.</p> <pre>sklearn.metrics.recall_score(y_true, y_pred, labels=None, pos_label=1, average='weighted', sample_weight=None)</pre> <p>Formula: $tp / (tp + fn)$</p>

See Also: [Scikit-learn classification metrics](#)

Table 10-3 Binary Classification Metrics Only

Metric	Description, Scikit-learn Equivalent, and Formula
f1	<p>Calculates the f-score or f-measure, which is a weighted average of the precision and recall. This metric by default requires a positive target to be encoded as 1 to function as expected.</p> <pre>sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None)</pre> <p>Formula: $2 * (precision * recall) / (precision + recall)$</p>

Table 10-3 (Cont.) Binary Classification Metrics Only

Metric	Description, Scikit-learn Equivalent, and Formula
precision	<p>Calculates the ability of the classifier to not label a sample positive (1) that is actually negative (0).</p> <pre>sklearn.metrics.precision_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None)</pre> <p>Formula: $tp / (tp + fp)$</p>
recall	<p>Calculates the ability of the classifier to label all positive (1) samples correctly.</p> <pre>sklearn.metrics.recall_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None)</pre> <p>Formula: $tp / (tp + fn)$</p>
roc_auc	<p>Calculates the Area Under the Receiver Operating Characteristic Curve (roc_auc) from prediction scores.</p> <pre>sklearn.metrics.roc_auc_score(y_true, y_pred, normalize=True, sample_weight=None)</pre> <p>See also the definition of receiver operation characteristic.</p>

Table 10-4 Regression Metrics

Metric	Description, Scikit-learn Equivalent, and Formula
r2	<p>Calculates the coefficient of determination (R squared).</p> <pre>sklearn.metrics.r2_score(y_true, y_pred, sample_weight=None, multioutput='uniform_average')</pre> <p>See also the definition of coefficient of determination.</p>
neg_mean_absolute_error	<p>Calculates the mean of the absolute difference of predicted and true targets (MAE).</p> <pre>sklearn.metrics.mean_absolute_error(y_true, y_pred, sample_weight=None, multioutput='uniform_average')</pre> <p>Formula:</p> $-\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)$

Table 10-4 (Cont.) Regression Metrics

Metric	Description, Scikit-learn Equivalent, and Formula
<code>neg_mean_squared_error</code>	<p>Calculates the mean of the squared difference of predicted and true targets.</p> <pre>-1.0 * sklearn.metrics.mean_squared_error(y_true, y_pred, sample_weight=None, multioutput='uniform_average')</pre> <p>Formula:</p> $-\frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$
<code>neg_mean_squared_log_error</code>	<p>Calculates the mean of the difference in the natural log of predicted and true targets.</p> <pre>sklearn.metrics.mean_squared_log_error(y_true, y_pred, sample_weight=None, multioutput='uniform_average')</pre> <p>Formula:</p> $-\frac{1}{n} \sum_{i=1}^n (\log(Y_i) - \log(\hat{Y}_i))^2$
<code>neg_median_absolute_error</code>	<p>Calculates the median of the absolute difference between predicted and true targets.</p> <pre>sklearn.metrics.median_absolute_error(y_true, y_pred)</pre> <p>Formula:</p> $-\text{Med}(\{Y_i - \hat{Y}_i, 0 \leq i < n\})$

See Also: [Scikit-learn regression metrics](#)

10.2 Algorithm Selection

The `oml.automl.AlgorithmSelection` class uses the characteristics of the data set and the task to rank algorithms from the set of supported Oracle Machine Learning algorithms.

Selecting the best Oracle Machine Learning algorithm for a data set and a prediction task is non-trivial. No single algorithm works best for all modeling problems. The `oml.automl.AlgorithmSelection` class ranks the candidate algorithms according to how likely each is to produce a quality model. This is achieved by using Oracle advanced meta-learning intelligence learned from a repertoire of data sets with the goal of avoiding exhaustive searches, thereby reducing overall compute time and costs.

The `oml.automl.AlgorithmSelection` class supports classification and regression algorithms. To use the class, you specify a data set and the number of algorithms you want to evaluate.

The `select` method of the class returns a sorted list of the top algorithms and their predicted rank (from best to worst).

For information on the parameters and methods of the class, invoke `help(oml.automl.AlgorithmSelection)` or see Oracle Machine Learning for Python API Reference.

Example 10-1 Using the `oml.automl.AlgorithmSelection` Class

This example creates an `oml.automl.AlgorithmSelection` object and then displays the algorithm rankings with their corresponding score metric. You may select the top entry or choose a different model depending on the needs of your particular business problem.

```
import oml
from oml import automl
import pandas as pd
from sklearn import datasets

# Load the breast cancer data set.
bc = datasets.load_breast_cancer()
bc_data = bc.data.astype(float)
X = pd.DataFrame(bc_data, columns = bc.feature_names)
y = pd.DataFrame(bc.target, columns = ['TARGET'])

# Create the database table BreastCancer.
oml_df = oml.create(pd.concat([X, y], axis=1),
                  table = 'BreastCancer')

# Split the data set into training and test data.
train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']

# Create an automated algorithm selection object with fl_macro as
# the score_metric argument.
asel = automl.AlgorithmSelection(mining_function='classification',
                                score_metric='fl_macro', parallel=4)

# Run algorithm selection to get the top k predicted algorithms and
# their ranking without tuning.
algo_ranking = asel.select(X, y, k=3)

# Show the selected and tuned model.
[(m, "{:.2f}".format(s)) for m,s in algo_ranking]

# Drop the database table.
oml.drop('BreastCancer')
```

Listing for This Example

```
>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the breast cancer data set.
... bc = datasets.load_breast_cancer()
>>> bc_data = bc.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns = bc.feature_names)
```

```
>>> y = pd.DataFrame(bc.target, columns = ['TARGET'])
>>>
>>> # Create the database table BreastCancer.
>>> oml_df = oml.create(pd.concat([X, y], axis=1),
...                     table = 'BreastCancer')
>>>
>>> # Split the data set into training and test data.
... train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
>>>
>>> # Create an automated algorithm selection object with fl_macro as
... # the score_metric argument.
... asel = automl.AlgorithmSelection(mining_function='classification',
...                                  score_metric='fl_macro', parallel=4)
>>>
>>> # Run algorithm selection to get the top k predicted algorithms and
... # their ranking without tuning.
... algo_ranking = asel.select(X, y, k=3)
>>>
>>> # Show the selected and tuned model.
>>> [(m, "{:.2f}".format(s)) for m,s in algo_ranking]
[('svm_gaussian', '0.97'), ('glm_ridge', '0.96'), ('nn', '0.96')]
>>>
>>> # Drop the database table.
... oml.drop('BreastCancer')
```

10.3 Feature Selection

The `oml.automl.FeatureSelection` class identifies the most relevant feature subsets for a training data set and an Oracle Machine Learning algorithm.

In a data analytics application, feature selection is a critical data preprocessing step that has a high impact on both runtime and model performance. The `oml.automl.FeatureSelection` class automatically selects the most relevant features for a data set and model. It internally uses several feature-ranking algorithms to identify the best feature subset that reduces model training time without compromising model performance. Oracle advanced meta-learning techniques quickly prune the search space of this feature selection optimization.

The `oml.automl.FeatureSelection` class supports classification and regression algorithms. To use the `oml.automl.FeatureSelection` class, you specify a data set and the Oracle Machine Learning algorithm on which to perform the feature reduction.

For information on the parameters and methods of the class, invoke `help(oml.automl.FeatureSelection)` or see Oracle Machine Learning for Python API Reference.

Example 10-2 Using the `oml.automl.FeatureSelection` Class

This example uses the `oml.automl.FeatureSelection` class. The example builds a model on the full data set and computes predictive accuracy. It performs automated feature selection, filters the columns according to the determined set, and rebuilds the model. It then recomputes predictive accuracy.

```
import oml
from oml import automl
```

```
import pandas as pd
import numpy as np
from sklearn import datasets

# Load the digits data set into the database.
digits = datasets.load_digits()
X = pd.DataFrame(digits.data,
                 columns = ['pixel{}'.format(i) for i
                           in range(digits.data.shape[1])])
y = pd.DataFrame(digits.target, columns = ['digit'])
oml_df = oml.create(pd.concat([X, y], axis=1), table = 'DIGITS')

# Split the data set into train and test.
train, test = oml_df.split(ratio=(0.8, 0.2),
                          seed = 1234, strata_cols='digit')
X_train, y_train = train.drop('digit'), train['digit']
X_test, y_test = test.drop('digit'), test['digit']

# Default model performance before feature selection.
mod = oml.svm(mining_function='classification').fit(X_train,
                                                  y_train)
"{:.2}".format(mod.score(X_test, y_test))

# Create an automated feature selection object with accuracy
# as the score_metric.
fs = automl.FeatureSelection(mining_function='classification',
                             score_metric='accuracy', parallel=4)

# Get the reduced feature subset on the train data set.
subset = fs.reduce('svm_linear', X_train, y_train)
 "{} features reduced to {}".format(len(X_train.columns),
                                   len(subset))

# Use the subset to select the features and create a model on the
# new reduced data set.
X_new = X_train[:,subset]
X_test_new = X_test[:,subset]
mod = oml.svm(mining_function='classification').fit(X_new, y_train)
"{:.2} with {:.1f}x feature reduction".format(
    mod.score(X_test_new, y_test),
    len(X_train.columns)/len(X_new.columns))

# Drop the DIGITS table.
oml.drop('DIGITS')

# For reproducible results, add a case_id column with unique row
# identifiers.
row_id = pd.DataFrame(np.arange(digits.data.shape[0]),
                     columns = ['CASE_ID'])
oml_df_cid = oml.create(pd.concat([row_id, X, y], axis=1),
                       table = 'DIGITS_CID')

train, test = oml_df_cid.split(ratio=(0.8, 0.2), seed = 1234,
                              hash_cols='CASE_ID',
                              strata_cols='digit')
X_train, y_train = train.drop('digit'), train['digit']
```

```

X_test, y_test = test.drop('digit'), test['digit']

# Provide the case_id column name to the feature selection
# reduce function.
subset = fs.reduce('svm_linear', X_train,
                  y_train, case_id='CASE_ID')
("{} features reduced to {} with case_id".format(
    len(X_train.columns)-1,
    len(subset)))

# Drop the tables created in the example.
oml.drop('DIGITS')
oml.drop('DIGITS_CID')

```

Listing for This Example

```

>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets
>>>
>>> # Load the digits data set into the database.
... digits = datasets.load_digits()
>>> X = pd.DataFrame(digits.data,
...                  columns = ['pixel{}'.format(i) for i
...                             in range(digits.data.shape[1])])
>>> y = pd.DataFrame(digits.target, columns = ['digit'])
>>> oml_df = oml.create(pd.concat([X, y], axis=1), table = 'DIGITS')
>>>
>>> # Split the data set into train and test.
... train, test = oml_df.split(ratio=(0.8, 0.2),
...                             seed = 1234, strata_cols='digit')
>>> X_train, y_train = train.drop('digit'), train['digit']
>>> X_test, y_test = test.drop('digit'), test['digit']
>>>
>>> # Default model performance before feature selection.
... mod = oml.svm(mining_function='classification').fit(X_train,
...                                                    y_train)
>>> "{:.2}".format(mod.score(X_test, y_test))
'0.92'
>>>
>>> # Create an automated feature selection object with accuracy
... # as the score_metric.
... fs = automl.FeatureSelection(mining_function='classification',
...                               score_metric='accuracy', parallel=4)
>>> # Get the reduced feature subset on the train data set.
... subset = fs.reduce('svm_linear', X_train, y_train)
>>>("{} features reduced to {}".format(len(X_train.columns),
...                                   len(subset)))
'64 features reduced to 41'
>>>
>>> # Use the subset to select the features and create a model on the
... # new reduced data set.
... X_new = X_train[:,subset]

```

```

>>> X_test_new = X_test[:,subset]
>>> mod = oml.svm(mining_function='classification').fit(X_new, y_train)
>>> "{:.2} with {:.1f}x feature reduction".format(
...     mod.score(X_test_new, y_test),
...     len(X_train.columns)/len(X_new.columns))
'0.92 with 1.6x feature reduction'
>>>
>>> # Drop the DIGITS table.
... oml.drop('DIGITS')
>>>
>>> # For reproducible results, add a case_id column with unique row
... # identifiers.
>>> row_id = pd.DataFrame(np.arange(digits.data.shape[0]),
...                       columns = ['CASE_ID'])
>>> oml_df_cid = oml.create(pd.concat([row_id, X, y], axis=1),
...                         table = 'DIGITS_CID')

>>> train, test = oml_df_cid.split(ratio=(0.8, 0.2), seed = 1234,
...                               hash_cols='CASE_ID',
...                               strata_cols='digit')
>>> X_train, y_train = train.drop('digit'), train['digit']
>>> X_test, y_test = test.drop('digit'), test['digit']
>>>
>>> # Provide the case_id column name to the feature selection
... # reduce function.
>>> subset = fs.reduce('svm_linear', X_train,
...                   y_train, case_id='CASE_ID')
... "{} features reduced to {} with case_id".format(
...     len(X_train.columns)-1,
...     len(subset))
'64 features reduced to 45 with case_id'
>>>
>>> # Drop the tables created in the example.
... oml.drop('DIGITS')
>>> oml.drop('DIGITS_CID')

```

10.4 Model Tuning

The `oml.automl.ModelTuning` class tunes the hyperparameters for the specified classification or regression algorithm and training data.

Model tuning is a laborious machine learning task that relies heavily on data scientist expertise. With limited user input, the `oml.automl.ModelTuning` class automates this process using a highly-parallel, asynchronous gradient-based hyperparameter optimization algorithm to tune the hyperparameters of an Oracle Machine Learning algorithm.

The `oml.automl.ModelTuning` class supports classification and regression algorithms. To use the `oml.automl.ModelTuning` class, you specify a data set and an algorithm to obtain a tuned model and its corresponding hyperparameters. An advanced user can provide a customized hyperparameter search space and a non-default scoring metric to this black box optimizer.

For a partitioned model, if you pass in the column to partition on in the `param_space` argument of the `tune` method, `oml.automl.ModelTuning` tunes the partitioned model's hyperparameters.

For information on the parameters and methods of the class, invoke `help(oml.automl.ModelTuning)` or see Oracle Machine Learning for Python API Reference.

Example 10-3 Using the `oml.automl.ModelTuning` Class

This example creates an `oml.automl.ModelTuning` object.

```
import oml
from oml import automl
import pandas as pd
from sklearn import datasets

# Load the breast cancer data set.
bc = datasets.load_breast_cancer()
bc_data = bc.data.astype(float)
X = pd.DataFrame(bc_data, columns = bc.feature_names)
y = pd.DataFrame(bc.target, columns = ['TARGET'])

# Create the database table BreastCancer.
oml_df = oml.create(pd.concat([X, y], axis=1),
                   table = 'BreastCancer')

# Split the data set into training and test data.
train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']

# Start an automated model tuning run with a Decision Tree model.
at = automl.ModelTuning(mining_function='classification',
                       parallel=4)
results = at.tune('dt', X, y, score_metric='accuracy')

# Show the tuned model details.
tuned_model = results['best_model']
tuned_model

# Show the best tuned model train score and the
# corresponding hyperparameters.
score, params = results['all_evals'][0]
"{:.2}".format(score), [{"{}:{}".format(k, params[k])
                        for k in sorted(params)}]

# Use the tuned model to get the score on the test set.
"{:.2}".format(tuned_model.score(X_test, y_test))

# An example invocation of model tuning with user-defined
# search ranges for selected hyperparameters on a new tuning
# metric (f1_macro).
search_space = {
    'RFOR_SAMPLING_RATIO': {'type': 'continuous',
                           'range': [0.01, 0.5]},
    'RFOR_NUM_TREES': {'type': 'discrete',
                      'range': [50, 100]},
    'TREE_IMPURITY_METRIC': {'type': 'categorical',
                             'range': ['TREE_IMPURITY_ENTROPY',
                                       'TREE_IMPURITY_GINI']},
}
results = at.tune('rf', X, y, score_metric='f1_macro',
                 param_space=search_space)
score, params = results['all_evals'][0]
```

```
("{:2}".format(score), [{"{}:{}".format(k, params[k])
    for k in sorted(params)])

# Some hyperparameter search ranges need to be defined based on the
# training data set sizes (for example, the number of samples and
# features). You can use placeholders specific to the data set,
# such as $nr_features and $nr_samples, as the search ranges.
search_space = {'RFOR_MTRY': {'type': 'discrete',
                              'range': [1, '$nr_features/2']}}

results = at.tune('rf', X, y,
                 score_metric='f1_macro', param_space=search_space)
score, params = results['all_evals'][0]
("{:2}".format(score), [{"{}:{}".format(k, params[k])
    for k in sorted(params)])

# Drop the database table.
oml.drop('BreastCancer')
```

Listing for This Example

```
>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the breast cancer data set.
... bc = datasets.load_breast_cancer()
>>> bc_data = bc.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns = bc.feature_names)
>>> y = pd.DataFrame(bc.target, columns = ['TARGET'])
>>>
>>> # Create the database table BreastCancer.
>>> oml_df = oml.create(pd.concat([X, y], axis=1),
...                    table = 'BreastCancer')
>>>
>>> # Split the data set into training and test data.
... train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
>>>
>>> # Start an automated model tuning run with a Decision Tree model.
... at = automl.ModelTuning(mining_function='classification',
...                         parallel=4)
>>> results = at.tune('dt', X, y, score_metric='accuracy')
>>>
>>> # Show the tuned model details.
... tuned_model = results['best_model']
>>> tuned_model
```

Algorithm Name: Decision Tree

Mining Function: CLASSIFICATION

Target: TARGET

```
Settings:
           setting name           setting value
0           ALGO_NAME             ALGO_DECISION_TREE
1           CLAS_MAX_SUP_BINS     32
2           CLAS_WEIGHTS_BALANCED OFF
3           ODMS_DETAILS         ODMS_DISABLE
4           ODMS_MISSING_VALUE_TREATMENT ODMS_MISSING_VALUE_AUTO
5           ODMS_SAMPLING        ODMS_SAMPLING_DISABLE
6           PREP_AUTO             ON
7           TREE_IMPURITY_METRIC  TREE_IMPURITY_GINI
8           TREE_TERM_MAX_DEPTH  8
9           TREE_TERM_MINPCT_NODE 3.34
10          TREE_TERM_MINPCT_SPLIT 0.1
11          TREE_TERM_MINREC_NODE  10
12          TREE_TERM_MINREC_SPLIT 20
```

```
Attributes:
mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension
```

```
Partition: NO
```

```
>>>
>>> # Show the best tuned model train score and the
... # corresponding hyperparameters.
... score, params = results['all_evals'][0]
>>> "{:.2}".format(score), [{"{}:{}".format(k, params[k])
...   for k in sorted(params)]
```

```

('0.92', ['CLAS_MAX_SUP_BINS:32', 'TREE_IMPURITY_METRIC:TREE_IMPURITY_GINI',
'TREE_TERM_MAX_DEPTH:7', 'TREE_TERM_MINPCT_NODE:0.05',
'TREE_TERM_MINPCT_SPLIT:0.1'])
>>>
>>> # Use the tuned model to get the score on the test set.
... "{:.2}".format(tuned_model.score(X_test, y_test))
'0.92
>>>
>>> # An example invocation of model tuning with user-defined
... # search ranges for selected hyperparameters on a new tuning
... # metric (fl_macro).
... search_space = {
...     'RFOR_SAMPLING_RATIO': {'type': 'continuous',
...                             'range': [0.01, 0.5]},
...     'RFOR_NUM_TREES': {'type': 'discrete',
...                        'range': [50, 100]},
...     'TREE_IMPURITY_METRIC': {'type': 'categorical',
...                              'range': ['TREE_IMPURITY_ENTROPY',
...                                       'TREE_IMPURITY_GINI']},}
>>> results = at.tune('rf', X, y, score_metric='fl_macro',
>>>                  param_space=search_space)
>>> score, params = results['all_evals'][0]
>>> ("{: .2}".format(score), [{"{}:{}".format(k, params[k])
...     for k in sorted(params)]])
('0.92', ['RFOR_NUM_TREES:53', 'RFOR_SAMPLING_RATIO:0.4999951',
'TREE_IMPURITY_METRIC:TREE_IMPURITY_ENTROPY'])
>>>
>>> # Some hyperparameter search ranges need to be defined based on the
... # training data set sizes (for example, the number of samples and
... # features). You can use placeholders specific to the data set,
... # such as $nr_features and $nr_samples, as the search ranges.
... search_space = {'RFOR_MTRY': {'type': 'discrete',
...                               'range': [1, '$nr_features/2']}}
>>> results = at.tune('rf', X, y,
...                   score_metric='fl_macro', param_space=search_space)
>>> score, params = results['all_evals'][0]
>>> ("{: .2}".format(score), [{"{}:{}".format(k, params[k])
...     for k in sorted(params)]])
('0.93', ['RFOR_MTRY:10'])
>>>
>>> # Drop the database table.
... oml.drop('BreastCancer')

```

10.5 Model Selection

The `oml.automl.ModelSelection` class automatically selects an Oracle Machine Learning algorithm according to the selected score metric and then tunes that algorithm.

The `oml.automl.ModelSelection` class supports classification and regression algorithms. To use the `oml.automl.ModelSelection` class, you specify a data set and the number of algorithms you want to tune.

The `select` method of the class returns the best model out of the models considered.

For information on the parameters and methods of the class, invoke `help(oml.automl.ModelSelection)` or see Oracle Machine Learning for Python API Reference.

Example 10-4 Using the `oml.automl.ModelSelection` Class

This example creates an `oml.automl.ModelSelection` object and then uses the object to select and tune the best model.

```
import oml
from oml import automl
import pandas as pd
from sklearn import datasets

# Load the breast cancer data set.
bc = datasets.load_breast_cancer()
bc_data = bc.data.astype(float)
X = pd.DataFrame(bc_data, columns = bc.feature_names)
y = pd.DataFrame(bc.target, columns = ['TARGET'])

# Create the database table BreastCancer.
oml_df = oml.create(pd.concat([X, y], axis=1),
                   table = 'BreastCancer')

# Split the data set into training and test data.
train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']

# Create an automated model selection object with f1_macro as the
# score_metric argument.
ms = automl.ModelSelection(mining_function='classification',
                           score_metric='f1_macro', parallel=4)

# Run model selection to get the top (k=1) predicted algorithm
# (defaults to the tuned model).
select_model = ms.select(X, y, k=1)

# Show the selected and tuned model.
select_model

# Score on the selected and tuned model.
"{:.2}".format(select_model.score(X_test, y_test))

# Drop the database table.
oml.drop('BreastCancer')
```

Listing for This Example

```
>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the breast cancer data set.
... bc = datasets.load_breast_cancer()
```

```
>>> bc_data = bc.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns = bc.feature_names)
>>> y = pd.DataFrame(bc.target, columns = ['TARGET'])
>>>
>>> # Create the database table BreastCancer.
>>> oml_df = oml.create(pd.concat([X, y], axis=1),
...                     table = 'BreastCancer')
>>>
>>> # Split the data set into training and test data.
... train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
>>>
>>> # Create an automated model selection object with fl_macro as the
... # score_metric argument.
... ms = automl.ModelSelection(mining_function='classification',
...                             score_metric='fl_macro', parallel=4)
>>>
>>> # Run the model selection to get the top (k=1) predicted algorithm
... # (defaults to the tuned model).
... select_model = ms.select(X, y, k=1)
>>>
>>> # Show the selected and tuned model.
... select_model
```

Algorithm Name: Support Vector Machine

Mining Function: CLASSIFICATION

Target: TARGET

Settings:

	setting name	setting value
0	ALGO_NAME	ALGO_SUPPORT_VECTOR_MACHINES
1	CLAS_WEIGHTS_BALANCED	OFF
2	ODMS_DETAILS	ODMS_DISABLE
3	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
4	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
5	PREP_AUTO	ON
6	SVMS_COMPLEXITY_FACTOR	10
7	SVMS_CONV_TOLERANCE	.0001
8	SVMS_KERNEL_FUNCTION	SVMS_GAUSSIAN
9	SVMS_NUM_PIVOTS	...
10	SVMS_STD_DEV	5.3999999999999995

Attributes:

area error
compactness error
concave points error
concavity error
fractal dimension error
mean area
mean compactness
mean concave points
mean concavity
mean fractal dimension

```
mean perimeter
mean radius
mean smoothness
mean symmetry
mean texture
perimeter error
radius error
smoothness error
symmetry error
texture error
worst area
worst compactness
worst concave points
worst concavity
worst fractal dimension
worst perimeter
worst radius
worst smoothness
worst symmetry
worst texture
Partition: NO

>>>
>>> # Score on the selected and tuned model.
... "{:.2}".format(select_model.score(X_test, y_test))
'0.99'
>>>
>>> # Drop the database table.
... oml.drop('BreastCancer')
```

Convert Pretrained Models to ONNX Format

OML4Py enables the use of text transformers from Hugging Face by converting them into ONNX format models. OML4Py also adds the necessary tokenization and post-processing. The resulting ONNX pipeline is then imported into the database and can be used to generate embeddings for AI Vector Search.



Note:

This feature will **only** work on OML4Py client. It is not supported on the OML4Py server.

If you do not have a pretrained embedding model in ONNX-format to generate embeddings for your data, Oracle offers a Python utility package that downloads pretrained models from an external source, converts the model to ONNX format augmented with pre-processing and post-processing steps, and imports the resulting ONNX-format model into Oracle Database. Use the `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure or OML4Py's `export2db()` function to import the file as a mining model.. Then leverage the in-database ONNX Runtime with the ONNX model to produce vector embeddings.

At a high level, the Python utility package performs the following tasks:

- Downloads the pretrained model from external source to your system
- Augments the model with pre-processing and post-processing steps and creates a new ONNX model
- Validates the augmented ONNX model
- Loads into the database as a mining model or optionally exports to a file

The Python utility can take any of the models in the preconfigured list as input. Alternatively, you can use the built-in template that contains common configurations for certain groups of models such as text-based models. To understand what a preconfigured list, what is a built-in template is, and how to use them, read further.

Limitations

This table describes the limitations of the Python utility package.



Note:

This feature is available with the OML4Py client only.

Parameter	Description
Transformer Model Type	Currently supported only for text transformers.
Model Size	Model size should be less than 1GB. Quantization can help reduce the size.

Parameter	Description
Tokenizers	Must be either BERT, GPT2, SENTENCEPIECE, or ROBERTA.

Preconfigured List of Models

Preconfigured list of models are common models from external resource repositories that are provided with the Python utility. The preconfigured models have an existing specification. Users can create their own specification using the text template as a starting point. To get a list of all model names in the preconfigured list, you can use the `show_preconfigured` function.

Templates

The Python utility package provides built-in text template for you to configure the pretrained models with pre-processing and post-processing operations. The template has a default specification for the pretrained models. This specification can be changed or augmented to create custom configurations. The text template uses Mean Pooling and Normalization as post-processing operations by default.

The Python utility package provides the following classes:

- `EmbeddingModelConfig`
- `EmbeddingModel`

To learn more about the Python classes, their properties, and to configure the properties, see [Python Classes to Convert Pretrained Models to ONNX Models](#).

To use the Python utility, ensure that you have the following:

- OML4Py Client running on Linux X64 for On-Premises Databases
- Python 3.12 (the earlier versions are not compatible)

1. Start Python in your work directory.

```
python3
```

```
Python 3.12.2 | (main, Feb 27 2024, 17:35:02) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

2. On the OML4Py client, load the Python classes:

```
from oml.utils import EmbeddingModel, EmbeddingModelConfig
```

3. You can get a list of all preconfigured models by running the following:

```
EmbeddingModelConfig.show_preconfigured()
```

4. To get a list of available templates:

```
EmbeddingModelConfig.show_templates()
```

5. Choose from:

- Generate an ONNX file from the preconfigured model "sentence-transformers/all-MiniLM-L6-v2":

```
#generate from preconfigured model "sentence-transformers/all-MiniLM-L6-v2"
em = EmbeddingModel(model_name="sentence-transformers/all-MiniLM-L6-v2")
em.export2file("your_preconfig_file_name",output_dir=".")
```

- Generate an ONNX model from the preconfigured model "sentence-transformers/all-MiniLM-L6-v2" in the database:

```
#generate from preconfigured model "sentence-transformers/all-MiniLM-L6-v2"
em = EmbeddingModel(model_name="sentence-transformers/all-MiniLM-L6-v2")
em.export2db("your_preconfig_model_name")
```

- Generate an ONNX file using the provided text template:

```
#generate using the "text" template
config = EmbeddingModelConfig.from_template("text",max_seq_length=512)
em = EmbeddingModel(model_name="intfloat/e5-small-v2",config=config)
em.export2file("your_template_file_name",output_dir=".")
```

Let's understand the code:

```
from oml.utils import EmbeddingModel, EmbeddingModelConfig
```

This line imports two classes, `EmbeddingModel` and `EmbeddingModelConfig`.

In the preconfigured models first example:

- `em = EmbeddingModel(model_name="sentence-transformers/all-MiniLM-L6-v2")` creates an instance of the `EmbeddingModel` class, loading a pretrained model specified by the `model_name` parameter. `em` is the embedding model object. `sentence-transformers/all-MiniLM-L6-v2` is the model name for computing sentence embeddings. This is the model name under Hugging Face. Oracle supports models from Hugging Face.
- The `export2file` command creates an ONNX format model with a user-specified model name in the database. `your_preconfig_file_name` is a user defined ONNX model file name.
- `output_dir="."` specifies the output directory where the file will be saved. The `"."` denotes the current directory (that is, the directory from which the script is running).

In the preconfigured models second example:

- `em = EmbeddingModel(model_name="sentence-transformers/all-MiniLM-L6-v2")` creates an instance of the `EmbeddingModel` class, loading a pretrained model specified by the `model_name` parameter. `em` is the embedding model object. `sentence-transformers/all-MiniLM-L6-v2` is the model name for computing sentence embeddings. This is the model name under Hugging Face. Oracle supports models from Hugging Face.
- The `export2db` command creates an ONNX format model with a user defined model name in the database. `your_preconfig_model_name` is a user defined ONNX model name.

In the template example:

- `config = EmbeddingModelConfig.from_template("text", max_seq_length=512):` This line creates a configuration object for an embedding model using a method called `from_template`. The `"text"` argument indicates the name of the template. The `max_seq_length=512` parameter specifies the maximum length of input to the model as number of tokens. There is no default value. Specify this value for models that are not preconfigured.
- `em = EmbeddingModel(model_name="intfloat/e5-small-v2", config=config)` initializes an `EmbeddingModel` instance with a specific model and the previously defined configuration. The `model_name="intfloat/e5-small-v2"` argument specifies the name or identifier of the pretrained model to be loaded.
- The `export2file` command creates an ONNX format model with a user defined model name in the database. `your_template_file_name` is a user defined ONNX model name.
- `output_dir="."` specifies the output directory where the file will be saved. The `"."` denotes the current directory (that is, the directory from which the script is running).

 **Note:**

- The model size is limited to 1 gigabyte. For models larger than 400MB, Oracle recommends quantization.

Quantization reduces the model size by converting model weights from high-precision representation to low-precision format. The quantization option converts the weights to INT8. The smaller model size enables you to cache the model in shared memory further improving the performance.

- The `.onnx` file is created with opset version 17 and ir version 8. For more information about these version numbers, see <https://onnxruntime.ai/docs/reference/compatibility.html#onnx-opset-support>.

6. Exit Python.

```
exit()
```

7. Inspect if the converted models are present in your directory.

 **Note:**

ONNX files are only created when `export2file` is used. If `export2db` is used, no local ONNX files will be generated and the model will be saved in the database.

List all files in the current directory that have the extension `onnx`.

```
ls -ltr *.onnx
```

The Python utility package validates the embedding text model before you can run them using ONNX Runtime. Oracle supports ONNX embedding models that conform to `string` as input and `float32 [<vector dimension>]` as output.

If the input or output of the model doesn't conform to the description, you receive an error during the import.

8. Verify the models is in the database:

At your operating system prompt, start SQL*Plus, connect to it :

```
sqlplus OML_USER/password@pdbname_medium;
```

```
SQL*Plus: Release 23.0.0.0.0 - Production on Wed May 1 15:33:29 2024
Version 23.4.0.24.05
```

```
Copyright (c) 1982, 2024, Oracle. All rights reserved.
```

```
Last Successful login time: Wed May 01 2024 15:27:06 -07:00
```

```
Connected to:
```

```
Oracle Database 23ai Enterprise Edition Release 23.0.0.0.0 - Production
Version 23.4.0.24.05
```

To list all the ONNX model is in the database:

```
SQL> select MODEL_NAME, ALGORITHM from user_mining_models WHERE
MODEL_NAME='YOUR_PRECONFIG_MODEL_NAME';
```

```
MODEL_NAME ALGORITHM
-----
YOUR_PRECONFIG_MODEL_NAME ONNX
```

DBMS_VECTOR.LOAD_ONNX_MODEL is only needed if `export2file` was used to save the ONNX model file to the local system instead of using `export2db` to save the model in the database. The DBMS_VECTOR.LOAD_ONNX_MODEL imports the ONNX format model into the Oracle Database to leverage the in-database ONNX Runtime to produce vector embeddings using the VECTOR_EMBEDDING SQL operator.

- [Convert Pretrained Models to ONNX Model: End-to-End Instructions](#)
This section provides end-to-end instructions from installing the OML4Py client to downloading a pretrained embedding model in ONNX-format using the Python utility package offered by Oracle.
- [Python Classes to Convert Pretrained Models to ONNX Models](#)
Explore the functions and attributes of the `EmbeddingModelConfig` class and `EmbeddingModel` class within Python. These classes are designed to configure pretrained embedding models.

 **See Also:**

- *Oracle Database SQL Language Reference* for information about the `VECTOR_EMBEDDING` SQL function
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `IMPORT_ONNX_MODEL` procedure
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `LOAD_ONNX_MODEL` procedure
- *Oracle Machine Learning for SQL Concepts* for more information about importing pretrained embedding models in ONNX format and generating vector embeddings
- <https://onnx.ai/onnx/intro/> for ONNX documentation

11.1 Convert Pretrained Models to ONNX Model: End-to-End Instructions

This section provides end-to-end instructions from installing the OML4Py client to downloading a pretrained embedding model in ONNX-format using the Python utility package offered by Oracle.

These instructions assume you have configured your Oracle Linux 8 repo in `/etc/yum.repos.d`, configured a Wallet if using an Autonomous Database, and set up a proxy if needed.

1. Install Python:

```
sudo yum install libffi-devel openssl openssl-devel tk-devel xz-devel zlib-
devel bzip2-devel readline-devel libuuid-devel ncurses-devel libaio
mkdir -p $HOME/python
wget https://www.python.org/ftp/python/3.12.0/Python-3.12.0.tgz
tar -xvzf Python-3.12.0.tgz --strip-components=1 -C /home/$USER/python
cd $HOME/python
./configure --prefix=$HOME/python
make clean; make
make altinstall
```

2. Set variables PYTHONHOME, PATH, and LD_LIBRARY_PATH:

```
export PYTHONHOME=$HOME/python
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
```

3. Create symlink for python3 and pip3:

```
cd $HOME/python/bin
ln -s python3.12 python3
ln -s pip3.12 pip3
```

4. Install Oracle Instant client if you will be exporting embedded models to the database from Python. If you will be exporting to a file, skip steps 4 and 5 and see the note under environment variables in step 6:

```
cd $HOME
wget https://download.oracle.com/otn_software/linux/instantclient/2340000/
instantclient-basic-linux.x64-23.4.0.24.05.zip
unzip instantclient-basic-linux.x64-23.4.0.24.05.zip
```

5. Set variable `LD_LIBRARY_PATH`:

```
export LD_LIBRARY_PATH=$HOME/instantclient_23_4:$LD_LIBRARY_PATH
```

6. Create an environment file, for example, `env.sh`, that defines the Python and Oracle Instant client environment variables and source these environment variables before each OML4Py client session. Alternatively, add the environment variable definitions to `.bashrc` so they are defined when the user logs into their Linux machine.

```
# Environment variables for Python
export PYTHONHOME=$HOME/python
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
```

Note:

Environment variable for Oracle Instant Client - only if the Oracle Instant Client is installed for exporting models to the database. `export LD_LIBRARY_PATH=$HOME/instantclient_23_4:$LD_LIBRARY_PATH`.

7. Create a file named `requirements.txt` that contains the required third-party packages listed below.

```
--extra-index-url https://download.pytorch.org/whl/cpu
pandas==2.1.1
setuptools==68.0.0
scipy==1.12.0
matplotlib==3.8.4
oracledb==2.0.1
scikit-learn==1.4.1post1
numpy==1.26.4
onnxruntime==1.17.0
onnxruntime-extensions==0.10.1
onnx==1.16.0
torch==2.2.0+cpu
transformers==4.38.1
sentencepiece==0.2.0
```

8. Upgrade `pip3` and install the packages listed in `requirements.txt`.

```
pip3 install --upgrade pip
pip3 install -r requirements.txt
```

9. Install OML4Py client. Download OML4Py 2.0 client from [OML4Py download page](#) and upload it to the Linux machine.

```
unzip oml4py-client-linux-x86_64-2.0.zip
pip3 install client/oml-2.0-cp312-cp312-linux_x86_64.whl
```

10. Get a list of all preconfigured models. Start Python and import EmbeddingModelConfig from oml.utils.

```
python3

from oml.utils import EmbeddingModelConfig

EmbeddingModelConfig.show_preconfigured()

['sentence-transformers/all-mpnet-base-v2', 'sentence-transformers/all-
MiniLM-L6-v2',
  'sentence-transformers/multi-qa-MiniLM-L6-cos-v1', 'ProsusAI/
finbert',
  'medicalai/ClinicalBERT', 'sentence-transformers/distiluse-base-
multilingual-cased-v2',
  'sentence-transformers/all-MiniLM-L12-v2', 'BAAI/bge-small-en-v1.5',
'BAAI/bge-base-en-v1.5',
  'taylorAI/bge-micro-v2', 'intfloat/e5-small-v2', 'intfloat/e5-base-
v2', 'prajjwal/bert-tiny',
  'thenlper/gte-base', 'thenlper/gte-small', 'TaylorAI/gte-tiny',
'infgrad/stella-base-en-v2',
  'sentence-transformers/paraphrase-multilingual-mpnet-base-v2',
  'intfloat/multilingual-e5-base', 'intfloat/multilingual-e5-small',
'sentence-transformers/stsb-xlm-r-multilingual']
```

11. Choose from:

- To generate an ONNX file that you can manually upload to the database using the `DBMS_VECTOR.LOAD_ONNX_MODEL`, refer to step 3 of SQL Quick Start and skip steps 12 and 13.
- To upload the model directly into the database, skip this step and proceed to step 12.

Export a preconfigured embedding model to a local file. Import EmbeddingModel from oml.utils.

```
from oml.utils import EmbeddingModel

# Export to file
em = EmbeddingModel(model_name="sentence-transformers/all-MiniLM-L6-v2")
em.export2file("all-MiniLM-L6-v2", output_dir=".")
```

12. Export a preconfigured embedding model to the database. If using a database connection to update to match your credentials and database environment.

 **Note:**

To ensure step 12 works properly, complete steps 4 and 5 first.

```
# Import oml library and EmbeddingModel from oml.utils
import oml
from oml.utils import EmbeddingModel

# Set embedded mode to false for Oracle Database on premises. This is not
supported or required for Oracle Autonomous Database.
oml.core.methods.__embed__ = False

# Create a database connection.

# Oracle Database on-premises
oml.connect("<user>", "<password>", port=<port number> host="<hostname>",
service_name="<service name>")

# Oracle Autonomous Database
oml.connect(user="<user>", password="<password>", dsn="myadb_low")

em = EmbeddingModel(model_name="sentence-transformers/all-MiniLM-L6-v2")
em.export2db("ALL_MINILM_L6")
```

13. Verify the model exists using SQL:

```
sqlplus $USER/pass@PDBNAME;

select model_name, algorithm, mining_function from user_mining_models
where model_name='ALL_MINILM_L6';
```

MODEL_NAME	ALGORITHM	MINING_FUNCTION
ALL_MINILM_L6	ONNX	EMBEDDING

11.2 Python Classes to Convert Pretrained Models to ONNX Models

Explore the functions and attributes of the `EmbeddingModelConfig` class and `EmbeddingModel` class within Python. These classes are designed to configure pretrained embedding models.

EmbeddingModelConfig

The `EmbeddingModelConfig` class contains the properties required for the package to perform downloading, exporting, augmenting, validation, and storing of an ONNX model. The class provides access to configuration properties using the dot operator. As a convenience, well-known configurations are provided as templates.

Parameters

This table describes the functions and properties of the `EmbeddingModelConfig` class.

Functions	Parameter Type	Returns	Description
<code>from_template(name, **kwargs)</code>	<ul style="list-style-type: none"> <code>name (String)</code>: The name of the template <code>**kwargs</code>: template properties to override or add 	Instance of <code>EmbeddingModelConfig</code>	A static function that creates an <code>EmbeddingModelConfig</code> object based on a predefined template given by the <code>name</code> parameter. You can use named arguments to override the template properties.
<code>show_templates()</code>	NA	List of existing templates	A static function that returns a list of existing templates by name.
<code>show_preconfigured()</code>	<ul style="list-style-type: none"> <code>include_properties (bool, optional)</code>: A flag indicating whether properties should be included in the results. Defaults to <code>False</code> so only names will be included by default. <code>model_name (str, optional)</code>: A model name to filter by when including properties. This argument will be ignored if <code>include_properties</code> is <code>False</code>. Otherwise only the properties of this model will be included in the results. 	A list of preconfigured model names or properties.	Shows a list of preconfigured model names, or properties. By default, this function returns a list of names only. If the properties are required, pass the <code>include_properties</code> parameter as <code>True</code> . The returned list will contain a single dict where each key of the dict is the name of a preconfigured model and the value is the property set for that model. Finally, if only a single set of properties for a specific model is required, pass the name of the model in the <code>model_name</code> parameter (the <code>include_properties</code> parameter should also be <code>True</code>). This will return a list of a single dict with the properties for the specified model.

Template Properties

The text template has configuration properties shown below:

```
"do_lower_case": true,
"post_processors": [{"name": "Pooling", "type": "mean"}, {"name": "Normalize"}]
```



Note:

All other properties in the Properties table will take the default values. Any property without a default value must be provided when creating the `EmbeddingModelConfig` instance.

Properties

This table shows all properties that can be configured. preconfigured models already have these properties set to specific values. Templates will use the default values unless a user overrides it when using the `from_template` function on `EmbeddingModelConfig`.

Property	Description
<code>post_processors</code>	An array of <code>post_processors</code> that will be loaded after the model is loaded or initialized. The list of known and supported <code>post_processors</code> is provided later in this section. Templates may define a list of <code>post_processors</code> for the types of models they support. Otherwise, an empty array is the default.
<code>max_seq_length</code>	This property is applicable for text-based models only. The maximum length of input to the model as number of tokens. There is no default value. Specify this value for models that are not preconfigured.
<code>do_lower_case</code>	Specifies whether or not to lowercase the input when tokenizing. The default value is <code>True</code> .
<code>quantize_model</code>	Perform quantization on the model. This could greatly reduce the size of the model as well as speed up the process. It may however result in different results for the embedding vector (against the original model) and possibly small reduction in accuracy. The default value is <code>False</code> .
<code>distance_metrics</code>	An array of names of suitable distance metrics for the model. The names must be name of distance metrics used for Oracle vector distance operator. Only used when exporting the model to the database. Supported list is ["EUCLIDEAN", "COSINE", "MANHATTAN", "HAMMING", "DOT", "EUCLIDEAN_SQUARED"]. The default value is an empty array.
<code>languages</code>	A array of language (Abbreviation) supported in the Database. Only used when exporting the model to the database. For a supported list of languages, see Languages. The default value is an empty array.
<code>use_float16</code>	Specifies whether or not to convert the exported onnx model to float16. The default value is <code>False</code> .

Properties of post_processors

This table describes the built-in `post_processors` and their configuration parameters.

post_processor	Parameters	Description
Pooling	<ul style="list-style-type: none"> <code>name</code>: Pooling. <code>type</code>: Valid values should be <code>mean(Default)</code>, <code>max</code>, <code>cls</code> 	The Pooling <code>post_processor</code> summarizes the output of the transformer model into a fixed-length vector.

post_processor	Parameters	Description
Normalize	<ul style="list-style-type: none"> name: Specify Normalize 	The Normalize post_processor bounds the vector values to a range using L2 normalization.
Dense	<ul style="list-style-type: none"> name: Dense in_features: Input feature size out_features: Output feature size bias: Whether to learn an additive bias. The default value is True. activation_function: Activation function of the dense layer. Currently only supports Tanh as the activation function. 	Applies transformation to the incoming data.

Example: Configure post_processors

In this example, you override the post_processors in the sentence-transformers template with a Max Pooling post_processor followed by Normalization.

```
config = EmbeddingModelConfig.from_template("text")
config.post_processors = [{"name": "Pooling", "type": "max"},
{"name": "Normalize"}]
```

EmbeddingModel

Use the EmbeddingModel class to convert transformer models to the ONNX format with post_processing steps embedded into the final model.

Parameters

This table describes the signature and properties of the EmbeddingModel class.

Functions	Parameters	Description
EmbeddingModel(model_name, configuration=None, settings={})	<ul style="list-style-type: none"> model_name: The name of the model to be used. For example, medicalai/ClinicalBERT configuration: An initialized EmbeddingModelConfig object. This parameter must be specified when using a template. If not specified, the model will be assumed to be a preconfigured model. settings: A dictionary of various settings that are global and control various operations such as logging levels and locations for files. 	Creates a new instance of the EmbeddingModel class.

Settings

The settings object is a dictionary passed to the `EmbeddingModel` class. It provides global properties for the `EmbeddingModel` class that are used for non-model-specific operations, such as logging.

Property	Default Value	Description
<code>cache_dir</code>	<code>\$HOME/.cache/OML</code>	The base directory used for downloads. Model files will be downloaded from the repository to directories relative to the <code>cache_dir</code> . If the <code>cache_dir</code> does not exist at time of execution, it will be created.
<code>logging_level</code>	<code>ERROR</code>	The level for logging. Valid values are ['DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL'].
<code>force_download</code>	<code>False</code>	Forces download of model files instead of reloading from cache.

 **Note:**

This log level is also applied globally to all python packages and is also mapped to the ONNX Runtime libraries.

Property	Default Value	Description
ignore_checksum_error	False	Ignores any errors caused by mismatch in checksums when using preconfigured models.

Functions

This table describes the function and properties of the `EmbeddingModel` class.

Function	Parameters	Description
<code>export2file(export_name,output_dir=None)</code>	<ul style="list-style-type: none"> <code>export_name(string)</code>: The name of the file. The file will be saved with the file extension <code>.onnx</code> <code>output_dir(string)</code>: An optional output directory. If not specified the file will be saved to the current directory 	Exports the model to a file.
<code>export2db(export_name)</code>	<ul style="list-style-type: none"> <code>export_name(string)</code>: The name that will be used for the mining model object. This name must be compliant with existing rules for object names in the database. 	Exports the model to the database.

Example: Preconfigured Model

This example illustrates the preconfigured embedding model that comes with the Python package. You can use this model without any additional configurations.

```
"sentence-transformers/distiluse-base-multilingual-cased-v2": {
    "max_seq_length": 128,
    "do_lower_case": false,
    "post_processors": [{"name": "Pooling", "type": "mean"}],
    {"name": "Dense", "in_features": 768, "out_features": 512, "bias": true,
    "activation_function": "Tanh"}],
    "quantize_model": true,
    "distance_metrics": ["COSINE"],
    "languages": ["ar", "bg", "ca", "cs", "dk", "d", "us", "el", "et",
    "fa", "sf", "f", "frc", "gu", "iw", "hi", "hr", "hu", "hy", "in", "i", "ja",
    "ko", "lt", "lv", "mk", "mr", "ms", "n", "nl", "pl", "pt", "ptb", "ro", "ru",
    "sk", "sl", "sq", "lsr", "s", "th", "tr", "uk", "ur", "vn", "zhs", "zht"]
}
```

Embedded Python Execution

Embedded Python Execution is a feature of Oracle Machine Learning for Python that allows you to invoke user-defined Python functions directly in an Oracle database instance.

Embedded Python Execution is described in the following topics:

- [About Embedded Python Execution](#)
- [Python API for Embedded Python Execution](#)
- [REST API for Embedded Python Execution](#)
- [SQL API for Embedded Python Execution with Autonomous Database](#)

Embedded Python Execution is available on:

- Oracle Autonomous Database, where pre-installed Python packages can be used, via Python, REST and SQL APIs.
- Oracle Database on premises, ExaCS, ExaC@C, DBCS, and Oracle Database deployed in a compute instance, where the user can custom install third-party packages to use with EPE, via Python and SQL APIs.
- [About Embedded Python Execution](#)
With Embedded Python Execution, you can invoke user-defined Python functions in Python engines spawned and managed by the Oracle database instance.
- [Parallelism with OML4Py Embedded Python Execution](#)
OML4Py embedded Python execution allows users to invoke user-defined functions from Python, SQL, and REST interfaces using Python engines spawned and controlled by the Oracle Autonomous Database environment.
- [Datastores Supporting Embedded Python Execution](#)
OML4Py includes the datastore views supporting Embedded Python Execution. You can use these datastore views with the Embedded Python Execution APIs to work with the datastores and their contents.
- [Script repository for user-defined Python functions supporting EPE](#)
OML4Py includes the script repository views supporting Embedded Python Execution. You can use these script repository views with the Embedded Python Execution APIs to work with the script repository and their contents.
- [Python API for Embedded Python Execution](#)
You can invoke user-defined Python functions directly in an Oracle database instance by using Embedded Python Execution functions.
- [SQL API for Embedded Python Execution with On-premises Database](#)
SQL API for Embedded Python Execution with On-premises Database has SQL interfaces for Embedded Python Execution and for datastore and script repository management.
- [SQL API for Embedded Python Execution with Autonomous Database](#)
The SQL API for Embedded Python Execution with Autonomous Database provides SQL interfaces for setting authorization tokens, managing access control list (ACL) privileges, executing Python scripts, and synchronously and asynchronously running jobs.

12.1 About Embedded Python Execution

With Embedded Python Execution, you can invoke user-defined Python functions in Python engines spawned and managed by the Oracle database instance.

Embedded Python Execution is available in Oracle Autonomous Database.

In Oracle Autonomous Database, you can use:

- In Oracle Autonomous Database, you can use:
 - An OML Notebooks Python interpreter session (see [Use the Python Interpreter in a Notebook Paragraph](#))
 - REST API for Embedded Python Execution
 - [SQL API for Embedded Python Execution with Autonomous Database](#)
- In an on-premises Oracle Database, you can use:
 - [Python API for Embedded Python Execution](#)
 - [SQL API for Embedded Python Execution with On-premises Database](#)
- [Comparison of the Embedded Python Execution APIs](#)
The table below compares the four Embedded Python Execution APIs.

12.1.1 Comparison of the Embedded Python Execution APIs

The table below compares the four Embedded Python Execution APIs.

The APIs are:

- Embedded Python Execution API
- REST API for Embedded Python Execution (for use with Oracle Autonomous Database)
- SQL API for Embedded Python Execution with Oracle Autonomous Database.

The APIs share many functions, but they differ in some ways because of the different environments. For example, the APIs available for Autonomous Database provide an API for operating in a web environment.

The procedures and functions are part of the `PYQSYS` and `SYS` schemas.

Category	Python API for Embedded Python Execution	REST API for Embedded Python Execution	SQL APIs for Embedded Python Execution
Embedded Python Execution function	<code>oml.do_eval</code> function See Run a User-Defined Python Function .	POST <code>/api/py-scripts/v1/do-eval/{scriptName}</code> See Run a Python Function . POST <code>/api/py-scripts/v1/do-eval/{scriptName}/{ownerName}</code> See Run a Python Function with Script Owner Specified .	pyqEval Function (Autonomous Database) (Autonomous Database)

Category	Python API for Embedded Python Execution	REST API for Embedded Python Execution	SQL APIs for Embedded Python Execution
Embedded Python Execution function	<p><code>oml.table_apply</code> function</p> <p>See Run a User-Defined Python Function on the Specified Data.</p>	<p>POST <code>/api/py-scripts/v1/table-apply/{scriptName}</code></p> <p>See Run a Python Function on Specified Data.</p> <p>POST <code>/api/py-scripts/v1/table-apply/{scriptName}/{ownerName}</code></p> <p>See Run a Python Function on Specified Data with Script Owner Specified.</p>	<p>pyqTableEval Function (Autonomous Database)</p>
Embedded Python Execution function	<p><code>oml.group_apply</code> function</p> <p>See Run a Python Function on Data Grouped By Column Values.</p>	<p>POST <code>/api/py-scripts/v1/group-apply/{scriptName}</code></p> <p>See Run a Python Function on Grouped Data.</p> <p>POST <code>/api/py-scripts/v1/group-apply/{scriptName}/{ownerName}</code></p> <p>See Run a Python Function on Grouped Data with Script Owner Specified.</p>	<p>pyqGroupEval Function (Autonomous Database)</p>
Embedded Python Execution function	<p><code>oml.row_apply</code> function</p> <p>See Run a User-Defined Python Function on Sets of Rows.</p>	<p>POST <code>/api/py-scripts/v1/row-apply/{scriptName}</code></p> <p>See Run a Python Function on Chunks of Rows.</p> <p>POST <code>/api/py-scripts/v1/row-apply/{scriptName}/{ownerName}</code></p> <p>See Run a Python Function on Chunks of Rows with Script Owner Specified.</p>	<p>pyqRowEval Function (Autonomous Database)</p>
Embedded Python Execution function	<p><code>oml.index_apply</code> function</p> <p>See Run a User-Defined Python Function Multiple Times.</p>	<p>POST <code>/api/py-scripts/v1/index-apply/{scriptName}</code></p> <p>See Run a Python Function Multiple Times.</p> <p>POST <code>/api/py-scripts/v1/index-apply/{scriptName}/{ownerName}</code></p> <p>See Run a Python Function Multiple Times with Script Owner Specified.</p>	<p>pyqIndexEval Function (Autonomous Database)</p>
Job status API	NA	<p>GET <code>/api/py-scripts/v1/jobs/{jobId}</code></p> <p>See Retrieve Asynchronous Job Status.</p>	<p>pyqJobStatus Function (Autonomous Database)</p>
Job result API	NA	<p>GET <code>/api/py-scripts/v1/jobs/{jobId}/result</code></p> <p>See Retrieve Asynchronous Job Result.</p>	<p>pyqJobResult Function (Autonomous Database)</p>

Category	Python API for Embedded Python Execution	REST API for Embedded Python Execution	SQL APIs for Embedded Python Execution
Script repository	<code>oml.script.dir</code> function See List Available User-Defined Python Functions .	GET <code>/api/py-scripts/v1/scripts</code> See List Scripts .	List the scripts by querying the ALL_PYQ_SCRIPTS View and the USER_PYQ_SCRIPTS View .
Script repository	<code>oml.script.create</code> function See Create and Store a User-Defined Python Function .	NA	pyqScriptCreate Procedure (Autonomous Database) (Autonomous Database)
Script repository	<code>oml.script.drop</code> function See Drop a User-Defined Python Function from the Repository .	NA	pyqScriptDrop Procedure (Autonomous Database) (Autonomous Database)
Script repository	<code>oml.script.load</code> function See Load a User-Defined Python Function .	NA	NA (Scripts are loaded in the SQL APIs when the function is called.)
Script repository and datastore	<code>oml.grant</code> function See About the Script Repository .	NA	pyqGrant procedure (Oracle Autonomous Database)
Script repository and datastore	<code>oml.revoke</code> function See About the Script Repository .	NA	pyqRevoke procedure (Autonomous Database)
Authorization - Access Control Lists	NA	NA	pyqAppendHostACE Procedure (Autonomous Database)
Authorization - Access Control Lists	NA	NA	pyqRemoveHostACE Procedure (Autonomous Database)
Authorization - Access Control Lists	NA	NA	pyqGetHostACE Function (Autonomous Database)
Authorization - Tokens	NA	See Authenticate .	<ul style="list-style-type: none"> pyqSetAuthToken Procedure (Autonomous Database) NA (on-premises database)
Authorization - Tokens	NA	See Authenticate .	pyqIsTokenSet Function (Autonomous Database)

 **Note:**

An output limit exists on the length function for REST API and SQL APIs for embedded Python execution. A query on the length function with a length of more than 5000 will result in an error with error code 1024 and the error message "Output exceeds maximum length 5000". The limit is set on the `len()` result of the returning python object. For example, `len()` of a `pandas.DataFrame` is the number of rows, `len()` of a list is the length of the list, etc. If `pandas.DataFrame` is returned, it cannot have more than 5000 rows. If a list is returned, it should not contain more than 5000 items. This limit can be extended by updating the `OML_OUTPUT_SZLIMIT` in a `%script` paragraph:

```
%script  
  
EXEC sys.pyqconfigset('OML_OUTPUT_SZLIMIT', '8000')
```

12.2 Parallelism with OML4Py Embedded Python Execution

OML4Py embedded Python execution allows users to invoke user-defined functions from Python, SQL, and REST interfaces using Python engines spawned and controlled by the Oracle Autonomous Database environment.

The user-defined functions can be invoked in a data-parallel and task-parallel manner with multiple Python engines, with output formats including structured data, XML, JSON, and PNG images.

Oracle Autonomous Database provides different service levels to manage the load on the system by controlling the degree of parallelism jobs can use:

- LOW - the default, with maximum 2 degrees of parallelism
- MEDIUM - maximum of 4 degrees of parallelism, and allows greater concurrency for job processing
- HIGH - maximum of 8 degrees of parallelism but significantly limits the number of concurrent jobs

Parallelism applies to:

- `oml.row_apply`, `oml.group_apply`, and `oml.index_apply` using the Python API for embedded Python execution
- `pyqRowEval`, `pyqGroupEval`, and `*pyqIndexEval` using the SQL API for embedded Python execution
- `row-apply`, `group-apply`, `index-apply` using the REST API for embedded Python execution

 **Note:**

`pyqIndexEval` is available on Oracle Autonomous Database only.

Setting Parallelism Using Embedded Python Execution

For the ADB Python API for Embedded Python Execution:

The `parallel` parameter specifies the preferred degree of parallelism to use in the embedded Python execution job. The value may be one of the following:

- A positive integer greater than or equal to 1 for a specific degree of parallelism
- `False`, `None`, or `0` for no parallelism
- `True` for the default data parallelism

Setting the argument `parallel=True` corresponds to service level defined in the notebook interpreter. The argument `parallel=x` is limited by the service level. For instance, the maximum number of parallel engines allowed by the MEDIUM service level is 4, therefore selecting `parallel=6` effectively results in `parallel=4`.

For the ADB SQL API for Embedded Python Execution:

The argument `oml_parallel_flag` and `oml_service_level` are used together to enable data-parallelism and task-parallelism. For more information see [Special Control Arguments \(Autonomous Database\)](#).

For the ADB REST API for Embedded Python Execution:

When executing a REST API Embedded Python Execution function, the `service` argument allows you to select the Autonomous Database service level to be used. For example, the `parallelFlag` is set to `true` in order to use database parallelism along with the MEDIUM service.

```
-d '{"parallelFlag":true,"service":"MEDIUM"}
```

For more information see [Specify a Service Level](#).

12.3 Datastores Supporting Embedded Python Execution

OML4Py includes the datastore views supporting Embedded Python Execution. You can use these datastore views with the Embedded Python Execution APIs to work with the datastores and their contents.

View	Description
ALL_PYQ_DATASTORES View	Contains information about the datastores available to the current user.
ALL_PYQ_DATASTORE_CONTENTS View	Contains information about the objects in the datastores available to the current user.
USER_PYQ_DATASTORES View	Contains information about the datastores owned by the current user.

- [ALL_PYQ_DATASTORE_CONTENTS View](#)
The `ALL_PYQ_DATASTORE_CONTENTS` view contains information about the contents of datastores that are available to the current user.
- [ALL_PYQ_DATASTORES View](#)
The `ALL_PYQ_DATASTORES` view contains information about the datastores that are available to the current user.

- [USER_PYQ_DATASTORES View](#)
 The `USER_PYQ_DATASTORES` view contains information about the datastores that are owned by the current user.

12.3.1 ALL_PYQ_DATASTORE_CONTENTS View

The `ALL_PYQ_DATASTORE_CONTENTS` view contains information about the contents of datastores that are available to the current user.

Column	Datatype	Null	Description
DSOWNER	VARCHAR2(128)	NULL permitted	The owner of the datastore.
DSNAME	VARCHAR2(128)	NULL permitted	The name of the datastore.
OBJNAME	VARCHAR2(128)	NULL permitted	The name of an object in the datastore.
CLASS	VARCHAR2(128)	NULL permitted	The class of a Python object in the datastore.
OBJSIZE	NUMBER	NULL permitted	The size of an object in the datastore.
LENGTH	NUMBER	NULL permitted	The length of an object in the datastore. The length is 1 for all objects unless the object is a list, dict, pandas.DataFrame, or oml.DataFrame, in which case it is equal to len(obj).
NROW	NUMBER	NULL permitted	The number of rows of an object in the datastore. The number is 1 for all objects except for pandas.DataFrame and oml.DataFrame objects, in which case it is equal to len(df).
NCOL	NUMBER	NULL permitted	The number of columns of an object in the datastore. The number is len(obj) if the object is a list or dict, len(obj.columns) if the object is a pandas.DataFrame or oml.DataFrame, and 1 otherwise.

Example 12-1 Selecting from the ALL_PYQ_DATASTORE_CONTENTS View

This example selects all columns from the `ALL_PYQ_DATASTORE_CONTENTS` view. For the creation of the datastores in this example, see [Example 7-14](#).

```
SELECT * FROM ALL_PYQ_DATASTORE_CONTENTS
```

DSOWNER	DSNAME	OBJNAME	CLASS	OBJSIZE	LENGTH
NROW	NCOL				
-----	-----	-----	-----	-----	-----
-----	-----				
OML_USER	ds_pydata	oml_boston	oml.DataFrame	1073	506
506	14				
OML_USER	ds_pydata	oml_diabetes	oml.DataFrame	964	442
442	11				
OML_USER	ds_pydata	wine	Bunch	24177	5
1	5				
OML_USER	ds_pymodel	regr1	LinearRegression	706	1

```

1      1
OML_USER ds_pymodel   regr2      oml.glm      5664      1
1      1
OML_USER ds_wine_data oml_wine    oml.DataFrame 1410      178
178     14
  
```

12.3.2 ALL_PYQ_DATASTORES View

The ALL_PYQ_DATASTORES view contains information about the datastores that are available to the current user.

Column	Datatype	Null	Description
DSOWNER	VARCHAR2 (256)	NULL permitted	The owner of the datastore.
DSNAME	VARCHAR2 (128)	NULL permitted	The name of the datastore.
NOBJ	NUMBER	NULL permitted	The number of objects in the datastore.
DSSIZE	NUMBER	NULL permitted	The size of the datastore.
CDATE	DATE	NULL permitted	The date on which the datastore was created.
DESCRIPTION	VARCHAR2 (2000)	NULL permitted	A description of the datastore.
GRANTABLE	VARCHAR2 (1)	NULL permitted	Whether or not the read privilege to the datastore may be granted. The value in this column is either T for True or F for False.

Example 12-2 Selecting from the ALL_PYQ_DATASTORES View

This example selects all columns from the ALL_PYQ_DATASTORES view. It then selects only the DSNAME and GRANTABLE columns from the view. For the creation of the datastores in these examples, see [Example 7-14](#).

```
SELECT * FROM ALL_PYQ_DATASTORES;
```

```

DSOWNER   DSNAME           NOBJ   DSSIZE   CDATE        DESCRIPTION      G
-----
OML_USER  ds_pydata        3      26214   18-MAY-19   python datasets  F
OML_USER  ds_pymodel       2      6370    18-MAY-19   wine dataset     T
OML_USER  ds_wine_data     1      1410    18-MAY-19   wine dataset     F
  
```

This example selects only the DSNAME and GRANTABLE columns from the view.

```
SELECT DSNAME, GRANTABLE FROM ALL_PYQ_DATASTORES;
```

```

DSNAME      G
-----
ds_pydata   F
  
```

```
ds_pymodel    T
ds_wine_data  F
```

12.3.3 USER_PYQ_DATASTORES View

The USER_PYQ_DATASTORES view contains information about the datastores that are owned by the current user.

Column	Datatype	Null	Description
DSNAME	VARCHAR2 (128)	NULL permitted	The name of the datastore.
NOBJ	NUMBER	NULL permitted	The number of objects in the datastore.
DSSIZE	NUMBER	NULL permitted	The size of the datastore.
CDATE	DATE	NULL permitted	The date on which the datastore was created.
DESCRIPTION	VARCHAR2 (2000)	NULL permitted	A description of the datastore.
GRANTABLE	VARCHAR2 (1)	NULL permitted	Whether or not the read privilege to the datastore may be granted. The value in this column is either T for True or F for False.

Example 12-3 Selecting from the USER_PYQ_DATASTORES View

This example selects all columns from the USER_PYQ_DATASTORES view. For the creation of the datastores in these examples, see [Example 7-14](#).

```
SELECT * FROM USER_PYQ_DATASTORES;
```

```
DSNAME          NOBJ    DSSIZE    CDATE          DESCRIPTION      G
-----          -
ds_wine_data    1       1410     18-MAY-19     wine dataset     F
ds_pydata       3       26214    18-MAY-19     python datasets  F
ds_pymodel      2       6370     18-MAY-19
```

This example selects only the DSNAME and GRANTABLE columns from the view.

```
SELECT DSNAME, GRANTABLE FROM USER_PYQ_DATASTORES;
```

```
DSNAME          G
-----          -
ds_wine_data    F
ds_pydata       F
ds_pymodel      T
```

12.4 Script repository for user-defined Python functions supporting EPE

OML4Py includes the script repository views supporting Embedded Python Execution. You can use these script repository views with the Embedded Python Execution APIs to work with the script repository and their contents.

View	Description
ALL_PYQ_SCRIPTS View	Describes the scripts that are available to the current user.
USER_PYQ_SCRIPTS View	Describes the user-defined Python functions in the script repository that are owned by the current user.

- [ALL_PYQ_SCRIPTS View](#)
The `ALL_PYQ_SCRIPTS` view contains information about the user-defined Python functions in the OML4Py script repository that are available to the current user.
- [USER_PYQ_SCRIPTS View](#)
This view contains information about the user-defined Python functions in the OML4Py script repository that are owned by the current user.

12.4.1 ALL_PYQ_SCRIPTS View

The `ALL_PYQ_SCRIPTS` view contains information about the user-defined Python functions in the OML4Py script repository that are available to the current user.

Column	Datatype	Null	Description
OWNER	VARCHAR2(256)	NULL permitted	The owner of the user-defined Python function.
NAME	VARCHAR2(128)	NULL permitted	The name of the user-defined Python function.
SCRIPT	CLOB	NULL permitted	The user-defined Python function.

Example 12-4 Selecting from the ALL_PYQ_SCRIPTS View

This example selects the owner and the name of the user-defined Python function from the `ALL_PYQ_SCRIPTS` view.

```
SELECT owner, name FROM ALL_PYQ_SCRIPTS;
```

```
OWNER      NAME
-----  -
OML_USER   create_iris_table
OML_USER   tmpqfun2
PYQSYS     tmpqfun2
```

This example selects the name of the user-defined Python function and the function definition from the view.

```
SELECT name, script FROM ALL_PYQ_SCRIPTS WHERE name = 'create_iris_table';
```

```
NAME                SCRIPT
-----
-----
create_iris_table   "def create_iris_table():  from sklearn.datasets import
load_iris ...
```

12.4.2 USER_PYQ_SCRIPTS View

This view contains information about the user-defined Python functions in the OML4Py script repository that are owned by the current user.

Column	Datatype	Null	Description
NAME	VARCHAR2(128)	NOT NULL	The name of the user-defined Python function.
SCRIPT	CLOB	NULL permitted	The user-defined Python function.

Example 12-5 Selecting from the USER_PYQ_SCRIPTS View

This example selects all columns from USER_PYQ_SCRIPTS.

```
SELECT * FROM USER_PYQ_SCRIPTS;
```

```
NAME                SCRIPT
-----
-----
create_iris_table   "def create_iris_table():  from sklearn.datasets import
load_iris ...
tmpqfun2            "def return_frame():      import numpy as np        import
pickle ...
```

12.5 Python API for Embedded Python Execution

You can invoke user-defined Python functions directly in an Oracle database instance by using Embedded Python Execution functions.

- [About Python API for Embedded Python Execution](#)
- [Run a User-Defined Python Function](#)
Use the `oml.do_eval` function to run a user-defined input function that explicitly retrieves data or for which external data is not required.
- [Run a User-Defined Python Function on the Specified Data](#)
Use the `oml.table_apply` function to run a Python function on data that you specify with the `data` parameter.

- [Run a Python Function on Data Grouped By Column Values](#)
Use the `oml.group_apply` function to group the values in a database table by one or more columns and then run a user-defined Python function on each group.
- [Run a User-Defined Python Function on Sets of Rows](#)
Use the `oml.row_apply` function to chunk data into sets of rows and then run a user-defined Python function on each chunk.
- [Run a User-Defined Python Function Multiple Times](#)
Use the `oml.index_apply` function to run a Python function multiple times in Python engines spawned by the database environment.
- [Save and Manage User-Defined Python Functions in the Script Repository](#)
The OML4Py script repository stores user-defined Python functions for use with Embedded Python Execution functions.

12.5.1 About Python API for Embedded Python Execution

You may choose to run your functions in a data-parallel or task-parallel manner in one or more of these Python engines. In data-parallel processing, the data is partitioned and the same user-defined Python function of each data subset is invoked using one or more Python engines. In task-parallel processing, a user-defined function is invoked multiple times in one or more Python engines with a unique index passed in as an argument; for example, you may use task parallelism for Monte Carlo simulations in which you use the index to set a random seed.

The following table lists the Python functions for Embedded Python Execution.

Function	Description
<code>oml.do_eval</code>	Runs a user-defined Python function in a Python engine spawned and managed by the database environment.
<code>oml.group_apply</code>	Partitions a database table by the values in one or more columns and runs the provided user-defined Python function on each partition.
<code>oml.index_apply</code>	Runs a Python function multiple times, passing in a unique index of the invocation to the user-defined function.
<code>oml.row_apply</code>	Partitions a database table into sets of rows and runs the provided user-defined Python function on the data in each set.
<code>oml.table_apply</code>	Runs a Python function on data in the database as a single <code>pandas.DataFrame</code> in a single Python engine.

About Special Control Arguments

Special control arguments control what happens before or after the running of the function that you pass to an Embedded Python Execution function. You specify a special control argument with the `**kwargs` parameter of a function such as `oml.do_eval`. The control arguments are not passed to the function specified by the `func` argument of that function.

Table 12-1 Special Control Arguments

Argument	Description
<code>oml_input_type</code>	<p>Identifies the type of input data object that you are supplying to the <code>func</code> argument.</p> <p>The input types are the following:</p> <ul style="list-style-type: none"> • <code>pandas.DataFrame</code> • <code>numpy.recarray</code> • 'default' (the default value) <p>If all columns are numeric, then default type is a 2-dimensional <code>numpy.ndarray</code> of type <code>numpy.float64</code>. Otherwise, the default type is a <code>pandas.DataFrame</code>.</p>
<code>oml_na_omit</code>	<p>Controls the handling of missing values in the input data. If you specify <code>oml_na_omit = True</code>, then rows that contain missing values are removed from the input data. If all of the rows contain missing values, then the input data is an empty <code>oml.DataFrame</code>. The default value is <code>False</code>.</p>

About Output

When a user-defined Python function runs in OML4Py, by default it returns the Python objects returned by the function. Also, OML4Py captures all `matplotlib.figure.Figure` objects created by the user-defined Python function and converts them into PNG format.

If `graphics = True`, the Embedded Python Execution functions return `oml.embed.data_image._DataImage` objects. The `oml.embed.data_image._DataImage` class contains Python objects and PNG images. Calling the method `__repr__()` displays the PNG images and prints out the Python object. By default, `.dat` returns the Python object that the user-defined Python function returned; `.img` returns a list containing PNG image data for each figure.

About the Script Repository

Embedded Python Execution includes the ability to create and store user-defined Python functions in the OML4Py script repository, grant or revoke the read privilege to a user-defined Python function, list the available user-defined Python functions, load user-defined Python functions into the Python environment, or drop a user-defined Python function from the script repository.

Along with whatever other actions a user-defined Python function performs, it can also create, retrieve, and modify Python objects that are stored in OML4Py datastores.

In Embedded Python Execution, a user-defined Python function runs in one or more Python engines spawned and managed by the database environment. The engines are dynamically started and managed by the database. From the same user-defined Python function you can get structured data and PNG images.

You can make the user-defined Python function either private or global. A global function is available to any user. A private function is available only to the owner or to users to whom the owner of the function has granted the read privilege.

12.5.2 Run a User-Defined Python Function

Use the `oml.do_eval` function to run a user-defined input function that explicitly retrieves data or for which external data is not required.

The `oml.do_eval` function runs a user-defined Python function in a Python engine spawned and managed by the database environment.

The syntax of the `oml.do_eval` function is the following:

```
oml.do_eval(func, func_owner=None, graphics=False, **kwargs)
```

The `func` argument is the function to run. It may be one of the following:

- A Python function
- A string that is the name of a user-defined Python function in the OML4Py script repository
- A string that defines a Python function
- An `oml.script.script.Callable` object returned by the `oml.script.load` function

The optional `func_owner` argument is a string or `None` (the default) that specifies the owner of the registered user-defined Python function when argument `func` is a registered user-defined Python function name.

The `graphics` argument is a boolean that specifies whether to look for images. The default value is `False`.

With the `**kwargs` parameter, you can pass additional arguments to the `func` function. Special control arguments, which start with `oml_`, are not passed to the function specified by `func`, but instead control what happens before or after the running of the function.

The `oml.do_eval` function returns a Python object or an `oml.embed.data_image._DataImage`. If no image is rendered in the user-defined Python function, `oml.do_eval` returns whatever Python object is returned by the function. Otherwise, it returns an `oml.embed.data_image._DataImage` object.

Example 12-6 Using the `oml.do_eval` Function

This example defines a Python function that returns a Pandas DataFrame with the columns `ID` and `RES`. It then passes that function to the `oml.do_eval` function.

```
import pandas as pd
import oml

def return_df(num, scale):
    import pandas as pd
    id = list(range(0, int(num)))
    res = [i/scale for i in id]
    return pd.DataFrame({"ID":id, "RES":res})

res = oml.do_eval(func=return_df, scale = 100, num = 10)
type(res)

res
```

Listing for This Example

```
>>> import pandas as pd
>>> import oml
>>>
>>> def return_df(num, scale):
...     import pandas as pd
...     id = list(range(0, int(num)))
...     res = [i/scale for i in id]
...     return pd.DataFrame({"ID":id, "RES":res})
...
>>>
>>> res = oml.do_eval(func=return_df, scale = 100, num = 10)
>>> type(res)
<class 'pandas.core.frame.DataFrame'>
>>>
>>> res
   ID  RES
0  0.0  0.00
1  1.0  0.01
2  2.0  0.02
3  3.0  0.03
4  4.0  0.04
5  5.0  0.05
6  6.0  0.06
7  7.0  0.07
8  8.0  0.08
9  9.0  0.09
```

12.5.3 Run a User-Defined Python Function on the Specified Data

Use the `oml.table_apply` function to run a Python function on data that you specify with the `data` parameter.

The `oml.table_apply` function runs a user-defined Python function in a Python engine spawned and managed by the database environment. With the `func` parameter, you can supply a Python function or you can specify the name of a user-defined Python function in the OML4Py script repository.

The syntax of the function is the following:

```
oml.table_apply(data, func, func_owner=None, graphics=False, **kwargs)
```

The `data` argument is an `oml.DataFrame` that contains the data that the `func` function operates on.

The `func` argument is the function to run. It may be one of the following:

- A Python function
- A string that is the name of a user-defined Python function in the OML4Py script repository
- A string that defines a Python function
- An `oml.script.script.Callable` object returned by the `oml.script.load` function

The optional `func_owner` argument is a string or `None` (the default) that specifies the owner of the registered user-defined Python function when argument `func` is a registered user-defined Python function name.

The `graphics` argument is a boolean that specifies whether to look for images. The default value is `False`.

With the `**kwargs` parameter, you can pass additional arguments to the `func` function. Special control arguments, which start with `oml_`, are not passed to the function specified by `func`, but instead control what happens before or after the execution of the function.

The `oml.table_apply` function returns a Python object or an `oml.embed.data_image._DataImage`. If no image is rendered in the user-defined Python function, `oml.table_apply` returns whatever Python object is returned by the function. Otherwise, it returns an `oml.embed.data_image._DataImage` object.

Example 12-7 Using the `oml.table_apply` Function

This example builds a regression model using in-memory data, and then uses the `oml.table_apply` function to predict using the model on the first 10 rows of the IRIS table.

```
import oml
import pandas as pd
from sklearn import datasets
from sklearn import linear_model

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()

x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

# Drop the IRIS database table if it exists.
try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Build a regression model using in-memory data.
iris = oml_iris.pull()
regr = linear_model.LinearRegression()
regr.fit(iris[['Sepal_Width', 'Petal_Length', 'Petal_Width']],
         iris[['Sepal_Length']])
regr.coef_

# Use oml.table_apply to predict using the model on the first 10
# rows of the IRIS table.
def predict(dat, regr):
    import pandas as pd
```

```
    pred = regr.predict(dat[['Sepal_Width', 'Petal_Length',
                            'Petal_Width']])
    return pd.concat([dat,pd.DataFrame(pred)], axis=1)

res = oml.table_apply(data=oml_iris.head(n=10),
                     func=predict, regr=regr)

res
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>> from sklearn import linear_model
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>>
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> # Drop the IRIS database table if it exists.
... try:
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create the IRIS database table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Build a regression model using in-memory data.
... iris = oml_iris.pull()
>>> regr = linear_model.LinearRegression()
>>> regr.fit(iris[['Sepal_Width', 'Petal_Length', 'Petal_Width']],
...         iris[['Sepal_Length']])
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                 normalize=False)
>>> regr.coef_
array([[ 0.65083716,  0.70913196, -0.55648266]])
>>>
>>> # Use oml.table_apply to predict using the model on the first 10
... # rows of the IRIS table.
... def predict(dat, regr):
...     import pandas as pd
...     pred = regr.predict(dat[['Sepal_Width', 'Petal_Length',
...                             'Petal_Width']])
...     return pd.concat([dat,pd.DataFrame(pred)], axis=1)
...
>>> res = oml.table_apply(data=oml_iris.head(n=10),
...                       func=predict, regr=regr)
```

```

>>> res   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
0         4.6           3.6           1             0.2
1         5.1           2.5           3             1.1
2         6.0           2.2           4             1.0
3         5.8           2.6           4             1.2
4         5.5           2.3           4             1.3
5         5.5           2.5           4             1.3
6         6.1           2.8           4             1.3
7         5.7           2.5           5             2.0
8         6.0           2.2           5             1.5
9         6.3           2.5           5             1.9

      Species      0
0      setosa  4.796847
1  versicolor  4.998355
2  versicolor  5.567884
3  versicolor  5.716923
4  versicolor  5.466023
5  versicolor  5.596191
6   virginica  5.791442
7   virginica  5.915785
8   virginica  5.998775
9   virginica  5.971433

```

12.5.4 Run a Python Function on Data Grouped By Column Values

Use the `oml.group_apply` function to group the values in a database table by one or more columns and then run a user-defined Python function on each group.

The `oml.group_apply` function runs a user-defined Python function in a Python engine spawned and managed by the database environment. The `oml.group_apply` function passes the `oml.DataFrame` specified by the `data` argument to the user-defined `func` function as its first argument. The `index` argument to `oml.group_apply` specifies the columns of the `oml.DataFrame` by which the database groups the data for processing by the user-defined Python function. The `oml.group_apply` function can use data-parallel execution, in which one or more Python engines perform the same Python function on different groups of data.

The syntax of the function is the following.

```
oml.group_apply(data, index, func, func_owner=None, parallel=None,
orderby=None, graphics=False, **kwargs)
```

The `data` argument is an `oml.DataFrame` that contains the in-database data that the `func` function operates on.

The `index` argument is an `oml.DataFrame` object, the columns of which are used to group the data before sending it to the `func` function.

The `func` argument is the function to run. It may be one of the following:

- A Python function
- A string that is the name of a user-defined Python function in the OML4Py script repository
- A string that defines a Python function
- An `oml.script.script.Callable` object returned by the `oml.script.load` function

The optional `func_owner` argument is a string or `None` (the default) that specifies the owner of the registered user-defined Python function when argument `func` is a registered user-defined Python function name.

The `parallel` argument is a boolean, an `int`, or `None` (the default) that specifies the preferred degree of parallelism to use in the Embedded Python Execution job. The value may be one of the following:

- A positive integer greater than or equal to 1 for a specific degree of parallelism
- `False`, `None`, or 0 for no parallelism
- `True` for the default data parallelism

The optional `orderby` argument is an `oml.DataFrame`, `oml.Float`, or `oml.String` that specifies the ordering of the group partitions.

The `graphics` argument is a boolean that specifies whether to look for images. The default value is `False`.

With the `**kwargs` parameter, you can pass additional arguments to the `func` function. Special control arguments, which start with `oml_`, are not passed to the function specified by `func`, but instead control what happens before or after the running of the function.

The `oml.group_apply` function returns a dict of Python objects or a dict of `oml.embed.data_image._DataImage` objects. If no image is rendered in the user-defined Python function, `oml.group_apply` returns a dict of Python object returned by the function. Otherwise, it returns a dict of `oml.embed.data_image._DataImage` objects.

Example 12-8 Using the `oml.group_apply` Function

This example defines some functions and calls `oml.group_apply` for each function.

```
import pandas as pd
from sklearn import datasets
import oml

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()

x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

# Drop the IRIS database table if it exists.
try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Define a function that counts the number of rows and returns a
# dataframe with the species and the count.
```



```

def group_count(dat):
    import pandas as pd
    return pd.DataFrame([(dat["Species"][0], dat.shape[0]),\
                        columns = ["Species", "COUNT"])

# Select the Species column to use as the index argument.
index = oml.DataFrame(oml_iris['Species'])

# Group the data by the Species column and run the user-defined
# function for each species.
res = oml.group_apply(oml_iris, index, func=group_count,
                    oml_input_type="pandas.DataFrame")

res

# Define a function that builds a linear regression model, with
# Petal_Width as the feature and Petal_Length as the target value,
# and that returns the model after fitting the values.
def build_lm(dat):
    from sklearn import linear_model
    lm = linear_model.LinearRegression()
    X = dat[["Petal_Width"]]
    y = dat[["Petal_Length"]]
    lm.fit(X, y)
    return lm

# Run the model for each species and return an objectList in
# dict format with a model for each species.
mod = oml.group_apply(oml_iris[:,["Petal_Length", "Petal_Width",
                                "Species"]], index, func=build_lm)

# The output is a dict of key-value pairs for each species and model.
type(mod)

# Sort dict by the key species.
{k: mod[k] for k in sorted(mod.keys())}

```

Listing for This Example

```

>>> import pandas as pd
>>> from sklearn import datasets
>>> import oml
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>>
>>> x = pd.DataFrame(iris.data,
...                 columns = ['Sepal_Length', 'Sepal_Width',
...                           'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2: 'virginica'}[x], iris.target)),
...                 columns = ['Species'])
>>>
>>> # Drop the IRIS database table if it exists.
... try:

```

```
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create the IRIS database table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Define a function that counts the number of rows and returns a
... # dataframe with the species and the count.
... def group_count(dat):
...     import pandas as pd
...     return pd.DataFrame([(dat["Species"][0], dat.shape[0]),\
...                           columns = ["Species", "COUNT"])
...
>>> # Select the Species column to use as the index argument.
... index = oml.DataFrame(oml_iris['Species'])
>>>
>>> # Group the data by the Species column and run the user-defined
... # function for each species.
... res = oml.group_apply(oml_iris, index, func=group_count,
...                        oml_input_type="pandas.DataFrame")
>>> res
{'setosa':  Species  COUNT
0 setosa      50, 'versicolor':  Species  COUNT
0 versicolor  50, 'virginica':   Species  COUNT
0 virginica   50}
>>>
>>> # Define a function that builds a linear regression model, with
... # Petal_Width as the feature and Petal_Length as the target value,
... # and that returns the model after fitting the values.
... def build_lm(dat):
...     from sklearn import linear_model
...     lm = linear_model.LinearRegression()
...     X = dat[["Petal_Width"]]
...     y = dat[["Petal_Length"]]
...     lm.fit(X, y)
...     return lm
...
>>> # Run the model for each species and return an objectList in
... # dict format with a model for each species.
... mod = oml.group_apply(oml_iris[:,["Petal_Length", "Petal_Width",
...                                     "Species"]], index, func=build_lm)
>>>
>>> # The output is a dict of key-value pairs for each species and model.
... type(mod)
<class 'dict'>
>>>
>>> # Sort dict by the key species.
... {k: mod[k] for k in sorted(mod.keys())}
{'setosa': LinearRegression(copy_X=True, fit_intercept=True,
n_jobs=None, normalize=False), 'versicolor': LinearRegression(copy_X=True,
fit_intercept=True, n_jobs=None, normalize=False), 'virginica':
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)}
```

12.5.5 Run a User-Defined Python Function on Sets of Rows

Use the `oml.row_apply` function to chunk data into sets of rows and then run a user-defined Python function on each chunk.

The `oml.row_apply` function passes the `oml.DataFrame` specified by the `data` argument as the first argument to the user-defined `func` Python function. The `rows` argument specifies the maximum number of rows of the `oml.DataFrame` to assign to each chunk. The last chunk of rows may have fewer rows than the number specified.

The `oml.row_apply` function runs the Python function in a database-spawned Python engine. The function can use data-parallel execution, in which one or more Python engines perform the same Python function on different chunks of the data.

The syntax of the function is the following.

```
oml.row_apply(data, func, func_owner=None, rows=1, parallel=None,
graphics=False, **kwargs)
```

The `data` argument is an `oml.DataFrame` that contains the data that the `func` function operates on.

The `func` argument is the function to run. It may be one of the following:

- A Python function
- A string that is the name of a user-defined Python function in the OML4Py script repository
- A string that defines a Python function
- An `oml.script.script.Callable` object returned by the `oml.script.load` function

The optional `func_owner` argument is a string or `None` (the default) that specifies the owner of the registered user-defined Python function when argument `func` is a registered user-defined Python function name.

The `rows` argument is an `int` that specifies the maximum number of rows to include in each chunk.

The `parallel` argument is a boolean, an `int`, or `None` (the default) that specifies the preferred degree of parallelism to use in the Embedded Python Execution job. The value may be one of the following:

- A positive integer greater than or equal to 1 for a specific degree of parallelism
- `False`, `None`, or 0 for no parallelism
- `True` for the default data parallelism

The `graphics` argument is a boolean that specifies whether to look for images. The default value is `True`.

With the `**kwargs` parameter, you can pass additional arguments to the `func` function. Special control arguments, which start with `oml_`, are not passed to the function specified by `func`, but instead control what happens before or after the running of the function.

The `oml.row_apply` function returns a `pandas.DataFrame` or a list of `oml.embed.data_image._DataImage` objects. If no image is rendered in the user-defined

Python function, `oml.row_apply` returns a `pandas.DataFrame`. Otherwise, it returns a list of `oml.embed.data_image._DataImage` objects.

Example 12-9 Using the `oml.row_apply` Function

This example creates the `x` and `y` variables using the iris data set. It then creates the persistent database table `IRIS` and the `oml.DataFrame` object `oml_iris` as a proxy for the table.

The example builds a regression model based on iris data. It defines a function that predicts the `Petal_Width` values based on the `Sepal_Length`, `Sepal_Width`, and `Petal_Length` columns of the input data. It then concatenates the `Species` column, the `Petal_Width` column, and the predicted `Petal_Width` as the object to return. Finally, the example calls the `oml.row_apply` function to apply the `make_pred()` function on each 4-row chunk of the input data.

```
import oml
import pandas as pd
from sklearn import datasets
from sklearn import linear_model

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length', 'Sepal_Width',
                           'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2: 'virginica'}[x], iris.target)),
                 columns = ['Species'])

# Drop the IRIS database table if it exists.
try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Build a regression model to predict Petal_Width using in-memory
# data.
iris = oml_iris.pull()
regr = linear_model.LinearRegression()
regr.fit(iris[['Sepal_Length', 'Sepal_Width', 'Petal_Length']],
         iris[['Petal_Width']])
regr.coef_

# Define a Python function.
def make_pred(dat, regr):
    import pandas as pd
    import numpy as np
    pred = regr.predict(dat[['Sepal_Length',
                            'Sepal_Width',
                            'Petal_Length']])
    return pd.concat([dat[['Species', 'Petal_Width']],
                    pd.DataFrame(pred,
                                columns=['Pred_Petal_Width']),
                    axis=1)
```

```
input_data = oml_iris.split(ratio=(0.9, 0.1), strata_cols='Species')[1]
input_data.crosstab(index = 'Species').sort_values('Species')

res = oml.row_apply(input_data, rows=4, func=make_pred,
                    regr=regr, parallel=2)

type(res)
res
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>> from sklearn import linear_model
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length', 'Sepal_Width',
...                             'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                            {0: 'setosa', 1: 'versicolor',
...                             2: 'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> # Drop the IRIS database table if it exists.
... try:
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
>>> oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Build a regression model to predict Petal_Width using in-memory
... # data.
... iris = oml_iris.pull()
>>> regr = linear_model.LinearRegression()
>>> regr.fit(iris[['Sepal_Length', 'Sepal_Width', 'Petal_Length']],
...         iris[['Petal_Width']])
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
>>> regr.coef_
array([[ -0.20726607,  0.22282854,  0.52408311]])
>>>
>>> # Define a Python function.
... def make_pred(dat, regr):
...     import pandas as pd
...     import numpy as np
...     pred = regr.predict(dat[['Sepal_Length',
...                               'Sepal_Width',
...                               'Petal_Length']])
...     return pd.concat([dat[['Species', 'Petal_Width']],
...                       pd.DataFrame(pred,
```

```

...             columns=['Pred_Petal_Width']]),
...             axis=1)
>>>
>>> input_data = oml_iris.split(ratio=(0.9, 0.1), strata_cols='Species')[1]
>>> input_data.crosstab(index = 'Species').sort_values('Species')
   SPECIES  count
0    setosa     7
1  versicolor  8
2   virginica   4
>>> res = oml.row_apply(input_data, rows=4, func=make_pred, regr=regr,
...                     columns=['Species',
...                               'Petal_Width',
...                               'Pred_Petal_Width'])
>>> res = oml.row_apply(input_data, rows=4, func=make_pred,
...                     regr=regr, parallel=2)
>>> type(res)
<class 'pandas.core.frame.DataFrame'>
>>> res
   Species  Petal_Width  Pred_Petal_Width
0    setosa         0.4         0.344846
1    setosa         0.3         0.335509
2    setosa         0.2         0.294117
3    setosa         0.2         0.220982
4    setosa         0.2         0.080937
5  versicolor         1.5         1.504615
6  versicolor         1.3         1.560570
7  versicolor         1.0         1.008352
8  versicolor         1.3         1.131905
9  versicolor         1.3         1.215622
10 versicolor         1.3         1.272388
11 virginica         1.8         1.623561
12 virginica         1.8         1.878132

```

12.5.6 Run a User-Defined Python Function Multiple Times

Use the `oml.index_apply` function to run a Python function multiple times in Python engines spawned by the database environment.

The syntax of the function is the following:

```
oml.index_apply(times, func, func_owner=None, parallel=None, graphics=False,
**kwargs)
```

The `times` argument is an `int` that specifies the number of times to run the `func` function.

The `func` argument is the function to run. It may be one of the following:

- A Python function
- A string that is the name of a user-defined Python function in the OML4Py script repository
- A string that defines a Python function
- An `oml.script.script.Callable` object returned by the `oml.script.load` function

The optional `func_owner` argument is a string or `None` (the default) that specifies the owner of the registered user-defined Python function when argument `func` is a registered user-defined Python function name.

The `parallel` argument is a boolean, an `int`, or `None` (the default) that specifies the preferred degree of parallelism to use in the Embedded Python Execution job. The value may be one of the following:

- A positive integer greater than or equal to 1 for a specific degree of parallelism
- `False`, `None`, or 0 for no parallelism
- `True` for the default data parallelism

The `graphics` argument is a boolean that specifies whether to look for images. The default value is `True`.

With the `**kwargs` parameter, you can pass additional arguments to the `func` function. Special control arguments, which start with `oml_`, are not passed to the function specified by `func`, but instead control what happens before or after the running of the function.

The `oml.index_apply` function returns a list of Python objects or a list of `oml.embed.data_image._DataImage` objects. If no image is rendered in the user-defined Python function, `oml.index_apply` returns a list of the Python objects returned by the user-defined Python function. Otherwise, it returns a list of `oml.embed.data_image._DataImage` objects.

Example 12-10 Using the `oml.index_apply` Function

This example defines a function that returns the mean of a set of random numbers the specified number of times.

```
import oml
import pandas as pd

def compute_random_mean(index):
    import numpy as np
    import scipy
    from statistics import mean
    np.random.seed(index)
    res = np.random.random((100,1))*10
    return mean(res[1])
res = oml.index_apply(times=10, func=compute_random_mean)
type(res)
res
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>>
>>> def compute_random_mean(index):
...     import numpy as np
...     import scipy
...     from statistics import mean
...     np.random.seed(index)
...     res = np.random.random((100,1))*10
...     return mean(res[1])
```

```
...
>>> res = oml.index_apply(times=10, func=compute_random_mean)
>>> type(res)
<class 'list'>
>>> res
[7.203244934421581, 0.25926231827891333, 7.081478226181048,
5.4723224917572235, 8.707323061773764, 3.3197980530117723,
7.7991879224011464, 9.68540662820932, 5.018745921487388,
0.207519493594015]
```

12.5.7 Save and Manage User-Defined Python Functions in the Script Repository

The OML4Py script repository stores user-defined Python functions for use with Embedded Python Execution functions.



Note:

The user-defined Python functions can be used outside of Embedded Python Execution. You can store functions and reload them back into notebooks or other user-defined functions.

The script repository is a component of the Embedded Python Execution functionality.

The following topics describe the script repository and the Python functions for managing user-defined Python functions:

- [About the Script Repository](#)
- [Create and Store a User-Defined Python Function](#)
- [List Available User-Defined Python Functions](#)
- [Load a User-Defined Python Function](#)
- [Drop a User-Defined Python Function from the Repository](#)
- [About the Script Repository](#)
Use these functions to store, manage, and use user-defined Python functions in the script repository.
- [Create and Store a User-Defined Python Function](#)
Use the `oml.script.create` function to add a user-defined Python function to the script repository.
- [List Available User-Defined Python Functions](#)
Use the `oml.script.dir` function to list the user-defined Python functions in the OML4Py script repository.
- [Load a User-Defined Python Function](#)
Use the `oml.script.load` function to load a user-defined Python function from the script repository into a Python session.
- [Drop a User-Defined Python Function from the Repository](#)
Use the `oml.script.drop` function to remove a user-defined Python function from the script repository.

12.5.7.1 About the Script Repository

Use these functions to store, manage, and use user-defined Python functions in the script repository.

The following table lists the Python functions for the script repository.

Function	Description
<code>oml.script.create</code>	Registers a single user-defined Python function in the script repository.
<code>oml.script.dir</code>	Lists the user-defined Python functions present in the script repository.
<code>oml.script.drop</code>	Drops a user-defined Python function from the script repository.
<code>oml.script.load</code>	Loads a user-defined Python function from the script repository into a Python session.

The following table lists the Python functions for managing access to user-defined Python functions in the script repository, and to datastores and datastore objects.

Function	Description
<code>oml.grant</code>	Grants read privilege permission to another user to a datastore or user-defined Python function owned by the current user.
<code>oml.revoke</code>	Revokes the read privilege permission that was granted to another user to a datastore or user-defined Python function owned by the current user.

12.5.7.2 Create and Store a User-Defined Python Function

Use the `oml.script.create` function to add a user-defined Python function to the script repository.

With the `oml.script.create` function, you can store a single user-defined Python function in the OML4Py script repository. You can then specify the user-defined Python function as the `func` argument to the Embedded Python Execution functions `oml.do_eval`, `oml.group_apply`, `oml.index_apply`, `oml.row_apply`, and `oml.table_apply`.

You can make the user-defined Python function either private or global. A private user-defined Python function is available only to the owner, unless the owner grants the read privilege to other users. A global user-defined Python function is available to any user.

The syntax of `oml.script.create` is the following:

```
oml.script.create(name, func, is_global=False, overwrite=False)
```

The `name` argument is a string that specifies a name for the user-defined Python function in the Python script repository.

The `func` argument is the Python function to run. The argument can be a Python function or a string that contains the definition of a Python function. You must specify a string in an interactive session if `readline` cannot get the command history.

The `is_global` argument is a boolean that specifies whether to create a global user-defined Python function. The default value is `False`, which indicates that the user-defined Python function is a private function available only to the current session user. When `is_global` is

`True`, it specifies that the function is global and every user has the read privilege and the execute privilege to it.

The `overwrite` argument is a boolean that specifies whether to overwrite the user-defined Python function if it already exists. The default value is `False`.

Example 12-11 Using the `oml.script.create` Function

This example stores two user-defined Python functions in the script repository. It then lists the contents of the script repository using different arguments to the `oml.script.dir` function.

Load the iris dataset as a pandas dataframe from the seaborn library. Use the `oml.create` function to create the IRIS database table and the proxy object for the table.

```
%python

from sklearn import datasets
import pandas as pd
import oml

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()

# Create objects containing data for the user-defined functions to use.
x = pd.DataFrame(iris.data,
                 columns =
                 ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor', 2:'virginica'}[x],
                          iris.target))),
                 columns = ['Species'])

# Create the IRIS database table and the proxy object for the table.

try:
    oml.drop(table="IRIS")
except:
    pass

oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create an user-defined function `build_lm1` and use `oml.script.create` function to store it in the OML4Py script repository. The parameter `"build_lm1"` is a string that specifies the name of the user-defined function. The parameter `func=build_lm1` is the Python function to run. Run the user-defined Python function in embedded Python execution.

```
%python

# Define a function.

build_lm1 = '''def build_lm1(dat):
    from sklearn import linear_model
    regr = linear_model.LinearRegression()
    import pandas as pd
    dat = pd.get_dummies(dat, drop_first=True)
    X = dat[["Sepal_Width", "Petal_Length", "Petal_Width",
```

```
"Species_versicolor", "Species_virginica"]
y = dat[["Sepal_Length"]]
regr.fit(X, y)
return regr'''

# Create a private user-defined Python function.
oml.script.create("build_lm1", func=build_lm1, overwrite=True)

# Run the user-defined Python function in embedded Python execution
res = oml.table_apply(oml_iris, func="build_lm1",
oml_input_type="pandas.DataFrame")

res
res.coef_
```

The output is the following:

```
array([[ 0.49588894,  0.82924391, -0.31515517, -0.72356196, -1.02349781]])
```

Define another user-defined function `build_lm2`, store the function as a global script in the OML4Py script repository. Run the user-defined Python function in embedded Python execution.

```
%python

# Define another function

build_lm2 = '''def build_lm2(dat):
    from sklearn import linear_model
    regr = linear_model.LinearRegression()
    X = dat[["Petal_Width"]]
    y = dat[["Petal_Length"]]
    regr.fit(X, y)
    return regr'''

# Save the function as a global script to the script repository, overwriting
any existing function with the same name.
oml.script.create("build_lm2", func=build_lm2, is_global=True,
overwrite=True)

res = oml.table_apply(oml_iris, func="build_lm2",
oml_input_type="pandas.DataFrame")
res
```

The output is the following:

```
LinearRegression()
```

List the user-defined Python functions in the script repository available to the current user only.

```
%python

oml.script.dir()
```

The output is similar to the following:

```
          name ...          date
0      build_lm1 ... 2022-12-15 19:02:44
1      build_mod ... 2022-12-12 23:02:31
2      myFitMultiple ... 2022-12-14 22:30:43
3      sample_iris_table ... 2022-12-14 22:21:24

[4 rows x 4 columns]
```

List all of the user-defined Python functions available to the current user.

```
%python
oml.script.dir(sctype='all')
```

The output is similar to the following:

```
          owner ...          date
0      PYQSYS ... 2022-02-11 06:06:44
1      PYQSYS ... 2022-10-19 16:59:50
2      PYQSYS ... 2022-10-19 16:59:52
3      PYQSYS ... 2022-10-19 16:59:53
```

List the user-defined Python functions available to all users.

```
%python
oml.script.dir(sctype='global')
```

The output is similar to the following:

```
          name ...          date
0          GLBLM ... 2022-02-11 06:06:44
1      RandomRedDots ... 2022-10-19 16:59:50
2      RandomRedDots2 ... 2022-10-19 16:59:52
3      RandomRedDots3 ... 2022-10-19 16:59:53
4          TEST ... 2021-08-13 17:37:02
5          TEST4 ... 2021-08-13 17:42:49
6      TEST_FUN ... 2021-08-13 22:38:54
```

12.5.7.3 List Available User-Defined Python Functions

Use the `oml.script.dir` function to list the user-defined Python functions in the OML4Py script repository.

The syntax of the `oml.script.dir` function is the following:

```
oml.script.dir(name=None, regex_match=False, sctype='user')
```

The `name` argument is a string that specifies the name of a user-defined Python function or a regular expression to match to the names of user-defined Python functions in the script repository. When `name` is `None`, this function returns the type of user-defined Python functions specified by argument `sctype`.

The `regex_match` argument is a boolean that indicates whether argument `name` is a regular expression to match. The default value is `False`.

The `sctype` argument is a string that specifies the type of user-defined Python function to list. The value may be one of the following.

- `user`, to specify the user-defined Python functions available to the current user only.
- `grant`, to specify the user-defined Python functions to which the read and execute privilege have been granted by the current user to other users.
- `granted`, to specify the user-defined Python functions to which the read and execute privilege have been granted by other users to the current user.
- `global`, to specify all of the global user-defined Python functions created by the current user.
- `all`, to specify all of the user-defined Python functions available to the current user.

The `oml.script.dir` function returns a `pandas.DataFrame` that contains the columns `NAME` and `SCRIPT` and, optionally, the columns `OWNER` and `GRANTEE`.

Example 12-12 Using the `oml.script.dir` Function

This example lists the contents of the script repository using different arguments to the `oml.script.dir` function. For the creation of the user-defined Python functions, see [Example 12-11](#).

```
import oml

# List the user-defined Python functions in the script
# repository available to the current user only.
oml.script.dir()

# List all of the user-defined Python functions available
# to the current user.
oml.script.dir(sctype='all')

# List the user-defined Python functions available to all users.
oml.script.dir(sctype='global')

# List the user-defined Python functions that contain the letters
# BL and that are available to all users.
oml.script.dir(name="BL", regex_match=True, sctype='all')
```

Listing for This Example

```
>>> import oml
>>>
>>> # List the user-defined Python functions in the script
... # repository available to the current user only.
... oml.script.dir()
      NAME                               SCRIPT
0  MYLM  def build_lm1(dat):\n    from sklearn import l...
```

```

>>>
>>> # List all of the user-defined Python functions available
... to the current user.
... oml.script.dir(sctype='all')
      OWNER      NAME                                SCRIPT
0  PYQSYS  GLBLM  def build_lm2(dat):\n    from sklearn import l...
1  OML_USER MYLM   def build_lm1(dat):\n    from sklearn import l...
>>>
>>> # List the user-defined Python functions available to all users.
>>> oml.script.dir(sctype='global')
      NAME                                SCRIPT
0  GLBLM  def build_lm2(dat):\n    from sklearn import l...
>>>
>>> # List the user-defined Python functions that contain the letters
... # BL and that are available to all users.
... oml.script.dir(name="BL", regex_match=True, sctype='all')
      OWNER      NAME                                SCRIPT
0  PYQSYS  GLBLM  def build_lm2(dat):\n    from sklearn import l...

```

12.5.7.4 Load a User-Defined Python Function

Use the `oml.script.load` function to load a user-defined Python function from the script repository into a Python session.

The syntax of the function is the following:

```
oml.script.load(name, owner=None)
```

The `name` argument is a string that specifies the name of the user-defined Python function to load from the OML4Py script repository.

The optional `owner` argument is a string that specifies the owner of the user-defined Python function or `None` (the default). If `owner=None`, then this function finds and loads the user-defined Python function that matches `name` in the following order:

1. A user-defined Python function that the current user created.
2. A global user-defined Python function that was created by another user.

The `oml.script.load` function returns an `oml.script.script.Callable` object that references the named user-defined Python function.

Example 12-13 Using the `oml.script.load` Function

This example loads user-defined Python functions from the script repository and pulls them to the local Python session. For the creation of the user-defined Python functions, see [Example 12-11](#).

```

import oml

# Load the MYLM and GLBLM user-defined Python functions.
MYLM = oml.script.load(name="MYLM")
GMYLM = oml.script.load(name="GLBLM")

# Pull the models to the local Python session.

```

```
MYLM(oml_iris.pull()).coef_  
GMYLM(oml_iris.pull())
```

Listing for This Example

```
>>> import oml  
>>>  
>>> # Load the MYLM and GLBLM user-defined Python functions.  
>>> MYLM = oml.script.load(name="MYLM")  
>>> GMYLM = oml.script.load(name="GLBLM")  
>>>  
>>> # Pull the models to the local Python session.  
... MYLM(oml_iris.pull()).coef_  
array([[ 0.49588894,  0.82924391, -0.31515517, -0.72356196, -1.02349781]])  
>>> GMYLM(oml_iris.pull())  
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1,  
                 normalize=False)
```

12.5.7.5 Drop a User-Defined Python Function from the Repository

Use the `oml.script.drop` function to remove a user-defined Python function from the script repository.

The `oml.script.drop` function drops a user-defined Python function from the OML4Py script repository.

The syntax of the function is the following:

```
oml.script.drop(name, is_global=False, silent=False)
```

The `name` argument is a string that specifies the name of the user-defined Python function in the script repository.

The `is_global` argument is a boolean that specifies whether the user-defined Python function to drop is a global or a private user-defined Python function. The default value is `False`, which indicates a private user-defined Python function.

The `silent` argument is a boolean that specifies whether to display an error message when `oml.script.drop` encounters an error in dropping the specified user-defined Python function. The default value is `False`.

Example 12-14 Using the `oml.script.drop` Function

This example drops user-defined Python functions the MYLM private user-defined Python function and the GLBLM global user-defined Python function from the script repository. For the creation of the user-defined Python functions, see [Example 12-11](#).

```
import oml  
  
# List the available user-defined Python functions.  
oml.script.dir(sctype="all")  
  
# Drop the private user-defined Python function.  
oml.script.drop("MYLM")
```

```
# Drop the global user-defined Python function.
oml.script.drop("GLBLM", is_global=True)

# List the available user-defined Python functions again.
oml.script.dir(sctype="all")
```

Listing for This Example

```
>>> import oml
>>>
>>> # List the available user-defined Python functions.
... oml.script.dir(sctype="all")
      OWNER  NAME                                SCRIPT
0  PYQSYS  GLBLM  def build_lm2(dat):\n  from sklearn import lin...
1  OML_USER  MYLM  def build_lm1(dat):\n  from sklearn import lin...
>>>
>>> # Drop the private user-defined Python function.
... oml.script.drop("MYLM")
>>>
>>> # Drop the global user-defined Python function.
... oml.script.drop("GLBLM", is_global=True)
>>>
>>> # List the available user-defined Python functions again.
... oml.script.dir(sctype="all")
Empty DataFrame
Columns: [OWNER, NAME, SCRIPT]
Index: []
```

12.6 SQL API for Embedded Python Execution with On-premises Database

SQL API for Embedded Python Execution with On-premises Database has SQL interfaces for Embedded Python Execution and for datastore and script repository management.

The following topics describe the OML4Py SQL interfaces for Embedded Python Execution.

- [About the SQL API for Embedded Python Execution with On-Premises Database](#)
- [pyqEval Function \(On-Premises Database\)](#)
- [pyqTableEval Function \(On-Premises Database\)](#)
- [pyqRowEval Function \(On-Premises Database\)](#)
- [pyqGroupEval Function \(On-Premises Database\)](#)
- [pyqGrant Function \(On-Premises Database\)](#)
- [pyqRevoke Function \(On-Premises Database\)](#)
- [pyqScriptCreate Procedure \(On-Premises Database\)](#)
- [pyqScriptDrop Procedure \(On-Premises Database\)](#)
- [About the SQL API for Embedded Python Execution with On-Premises Database](#)
With the SQL API, you can run user-defined Python functions in one or more separate Python engines in an Oracle database environment, manage user-defined Python

functions in the OML4Py script repository, and control access to and get information about datastores and about user-defined Python functions in the script repository.

- [pyqEval Function \(On-Premises Database\)](#)
This topic describes the `pyqEval` function when used in an on-premises Oracle Database. The `pyqEval` function runs a user-defined Python function that explicitly retrieves data or for which external data is to be automatically loaded for the function.
- [pyqTableEval Function \(On-Premises Database\)](#)
This topic describes the `pyqTableEval` function when used in an on-premises Oracle Database. The `pyqTableEval` function runs a user-defined Python function on data from an Oracle Database table.
- [pyqRowEval Function \(On-Premises Database\)](#)
This topic describes the `pyqRowEval` function when used in an on-premises Oracle Database. The `pyqRowEval` function chunks data into sets of rows and then runs a user-defined Python function on each chunk.
- [pyqGroupEval Function \(On-Premises Database\)](#)
This topic describes the `pyqGroupEval` function when used in an on-premises Oracle Database. The `pyqGroupEval` function groups data by one or more columns and runs a user-defined Python function on each group.
- [pyqGrant Function \(On-Premises Database\)](#)
This topic describes the `pyqGrant` function when used in an on-premises Oracle Database.
- [pyqRevoke Function \(On-Premises Database\)](#)
This topic describes the `pyqRevoke` function when used in an on-premises Oracle Database.
- [pyqScriptCreate Procedure \(On-Premises Database\)](#)
This topic describes the `pyqScriptCreate` procedure in an on-premises Oracle Database. The `pyqScriptCreate` procedure creates a user-defined Python function and adds it to the OML4Py script repository.
- [pyqScriptDrop Procedure \(On-Premises Database\)](#)
This topic describes the `pyqScriptDrop` procedure in an on-premises Oracle Database. The `pyqScriptDrop` procedure removes a user-defined Python function from the OML4Py script repository.

12.6.1 About the SQL API for Embedded Python Execution with On-Premises Database

With the SQL API, you can run user-defined Python functions in one or more separate Python engines in an Oracle database environment, manage user-defined Python functions in the OML4Py script repository, and control access to and get information about datastores and about user-defined Python functions in the script repository.

OML4Py provides the following types of SQL functions and procedures.

- SQL table functions for running user-defined Python functions in one or more database-spawned and managed Python engines; the user-defined Python functions may reference Python objects in OML4Py datastores and use third-party packages installed with the database server machine Python engines..
- PL/SQL procedures for creating and dropping user-defined Python functions in the OML4Py script repository.

- PL/SQL procedures for granting and revoking the read privilege to datastores and the datastore objects in them, and to user-defined Python functions in the OML4Py script repository.

The following table lists the SQL functions for Embedded Python Execution and the PL/SQL procedures for managing datastores and user-defined Python functions.

Function or Procedure	Description
<code>pyqEval</code> function	Runs a user-defined Python function on the data passed in.
<code>pyqGroupEval</code> function	Groups data by one or more columns and runs a user-defined Python function on each group.
<code>pyqTableEval</code> function	Runs a user-defined Python function on data in the database.
<code>pyqRowEval</code> function	Runs the specified number of rows in each invocation of the user-defined Python function in parallel processes.
<code>pyqGrant</code> procedure	Grants the read privilege to another user to a user-defined Python function owned by the current user.
<code>pyqRevoke</code> procedure	Revokes the read privilege that was granted to another user to a user-defined Python function owned by the current user.
<code>pyqScriptCreate</code> procedure	Creates a user-defined Python function in the script repository.
<code>pyqScriptDrop</code> procedure	Drops a user-defined Python function from the script repository.

12.6.2 pyqEval Function (On-Premises Database)

This topic describes the `pyqEval` function when used in an on-premises Oracle Database. The `pyqEval` function runs a user-defined Python function that explicitly retrieves data or for which external data is to be automatically loaded for the function.

You can pass arguments to the Python function with the `PAR_LST` parameter.

The `pyqEval` function does not automatically receive any data from the database. The Python function generates the data that it uses or it explicitly retrieves it from a data source such as Oracle Database, other databases, or flat files.

The Python function can return a `boolean`, a `dict`, a `float`, an `int`, a `list`, a `str`, a `tuple` or a `pandas.DataFrame` object. You define the form of the returned value with the `OUT_FMT` parameter.

Syntax

```
pyqEval (
    par_lst VARCHAR2,
    out_fmt VARCHAR2,
    scr_name VARCHAR2,
    scr_owner VARCHAR2 DEFAULT NULL)
```

Parameters

Parameter	Description
PAR_LST	<p>A JSON string that contains additional parameters to pass to the user-defined Python function specified by the <code>SCR_NAME</code> parameter. Special control arguments, which start with <code>oml_</code>, are not passed to the function specified by <code>SCR_NAME</code>, but instead control what happens before or after the invocation of the function.</p> <p>For example, to specify the input data type as <code>pandas.DataFrame</code>, use:</p> <pre>'{"oml_input_type":"pandas.DataFrame"}'</pre>
OUT_FMT	<p>The format of the output returned by the function. It can be one of the following:</p> <ul style="list-style-type: none"> • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. • The name of a table or view to use as a prototype. If using a table or view owned by another user, use the format <code><owner name>.<table/view name></code>. You must have read access to the specified table or view. • The string <code>'XML'</code>, which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string <code>'PNG'</code>, which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation.
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.

Returns

Function `pyqEval` returns a table that has the structure specified by the `OUT_FMT` parameter value.

Example 12-15 Using the `pyqEval` Function

This example defines Python functions and stores them in the OML4Py script repository. It invokes the `pyqEval` function on the user-defined Python functions.

In a PL/SQL block, create an unnamed Python function that is stored in script repository with the name `pyqFun1`.

```
BEGIN
    sys.pyqScriptCreate('pyqFun1', 'func = lambda: "Hello World from a
```

```
lambda!'' ,
                                FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
/
```

Invoke the `pyqEval` function, which runs the user-defined Python function and returns the results as XML.

```
SELECT name, value
FROM table(pyqEval(
            NULL,
            'XML',
            'pyqFun1'));
```

The output is the following.

```
NAME  VALUE
-----
      <root><str>Hello World from a lambda!</str></root>
```

Drop the user-defined Python function.

```
BEGIN
  sys.pyqScriptDrop('pyqFun1');
END;
/
```

Define a Python function that returns a `numpy.ndarray` that is stored in script repository with the name `pyqFun2`.

```
BEGIN
  sys.pyqScriptCreate('pyqFun2',
    'def return_frame():
      import numpy as np
      import pickle
      z = np.array([y for y in zip([str(x)+"demo" for x in range(10)],
                                  [float(x)/10 for x in range(10)],
                                  [x for x in range(10)],
                                  [bool(x%2) for x in range(10)],
                                  [pickle.dumps(x) for x in range(10)],
                                  ["test"+str(x**2) for x in range(10)]]),
                  dtype=[("a", "U10"), ("b", "f8"), ("c", "i4"),
                          ("d", "?"), ("e", "S20"), ("f", "O")])

      return z');
END;
/
```

Invoke the `pyqEval` function, which runs the `pyqFun2` user-defined Python function.

```
SELECT *
FROM table(pyqEval(
```

```

NULL,
'{"A":"varchar2(10)", "B":"number",
  "C":"number", "D":"number",
  "E":"raw(10)", "F": "varchar2(10)" }',
'pyqFun2'));

```

The output is the following.

A	B	C	D E	F
0demo	0	0	0 80034B002E	test0
1demo	1.0E-001	1	1 80034B012E	test1
2demo	2.0E-001	2	0 80034B022E	test4
3demo	3.0E-001	3	1 80034B032E	test9
4demo	4.0E-001	4	0 80034B042E	test16
5demo	5.0E-001	5	1 80034B052E	test25
6demo	6.0E-001	6	0 80034B062E	test36
7demo	7.0E-001	7	1 80034B072E	test49
8demo	8.0E-001	8	0 80034B082E	test64
9demo	9.0E-001	9	1 80034B092E	test81

10 rows selected.

Drop the user-defined Python function.

```

BEGIN
  sys.pyqScriptDrop('pyqFun2');
END;
/

```

12.6.3 pyqTableEval Function (On-Premises Database)

This topic describes the `pyqTableEval` function when used in an on-premises Oracle Database. The `pyqTableEval` function runs a user-defined Python function on data from an Oracle Database table.

You pass data to the Python function with the `INP_NAM` parameter. You can pass arguments to the Python function with the `PAR_LST` parameter.

The Python function can return a boolean, a dict, a float, an int, a list, a str, a tuple or a `pandas.DataFrame` object. You define the form of the returned value with the `OUT_FMT` parameter.

Syntax

```

pyqTableEval(
  inp_nam VARCHAR2,
  par_lst VARCHAR2,
  out_fmt VARCHAR2,
  scr_name VARCHAR2,
  scr_owner VARCHAR2 DEFAULT NULL)

```

Parameters

Parameter	Description
INP_NAM	The name of a table or view that specifies the data to pass to the Python function specified by the SCR_NAME parameter. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view.
PAR_LST	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the SCR_NAME parameter. Special control arguments, which start with oml_, are not passed to the function specified by SCR_NAME, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as pandas.DataFrame, use: <pre>'{"oml_input_type":"pandas.DataFrame"}'</pre>
OUT_FMT	The format of the output returned by the function. It can be one of the following: <ul style="list-style-type: none"> • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. • The name of a table or view to use as a prototype. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view. • The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation.
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.

Returns

Function `pyqTableEval` returns a table that has the structure specified by the `OUT_FMT` parameter value.

Example 12-16 Using the pyqTableEval Function

This example stores a user-defined Python function in the OML4Py script repository with the name `create_iris_table`. It uses the function to create a database table as the result of a `pyqEval` function invocation. It creates another user-defined Python function that fits a linear regression model to the input data and saves the model in the OML4Py datastore. The example runs a SQL `SELECT` statement that invokes the `pyqTableEval` function, which invokes the function stored in the script repository with the name `myLinearRegressionModel`.

In a PL/SQL block, define the Python function `create_iris_table` and store in the script repository with the name `create_iris_table`, overwriting any existing user-defined Python function stored in the script repository with the same name.

The `create_iris_table` function imports and loads the iris data set, creates two `pandas.DataFrame` objects, and then returns the concatenation of those objects.

```
BEGIN
  sys.pyqScriptCreate('create_iris_table',
    'def create_iris_table():
      from sklearn.datasets import load_iris
      import pandas as pd
      iris = load_iris()
      x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
        "Sepal_Width", "Petal_Length", "Petal_Width"])
      y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
        2: "virginica"}[x], iris.target)),\
        columns = ["Species"])
      return pd.concat([y, x], axis=1)',
    FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
/
CREATE TABLE IRIS AS
(SELECT * FROM pyqEval(
  NULL,
  '{"Species":"VARCHAR2(10)","Sepal_Length":"number",
  "Sepal_Width":"number","Petal_Length":"number",
  "Petal_Width":"number"}',
  'create_iris_table'
));
```

Define the Python function `fit_model` and store it with the name `myLinearRegressionModel` as a private function in the script repository, overwriting any existing user-defined Python function stored with that name.

The `fit_model` function fits a regression model to the input data `dat` and then saves the fitted model as an object specified by the `modelName` argument to the datastore specified by the `datastoreName` argument. The `fit_model` function returns the fitted model in a string format.

By default, Python objects are saved to a new datastore with the specified `datastoreName`. To save an object to an existing datastore, either set the `overwrite` or `append` argument to `True` in the `oml.ds.save` invocation.

```
BEGIN
  sys.pyqScriptCreate('myLinearRegressionModel',
    'def fit_model(dat, modelName, datastoreName):
      import oml
```

```

        from sklearn import linear_model
        regr = linear_model.LinearRegression()
        regr.fit(dat.loc[:, ["Sepal_Length", "Sepal_Width", \
                            "Petal_Length"]], dat.loc[:, ["Petal_Width"]])
        oml.ds.save(objs={modelName:regr}, name=datastoreName,
                    overwrite=True)
        return str(regr)',
        FALSE, TRUE);
END;
/

```

Run a `SELECT` statement that invokes the `pyqTableEval` function. The `INP_NAM` parameter of the `pyqTableEval` function specifies the IRIS table as the data to pass to the Python function. The `PAR_LST` parameter specifies the names of the model and datastore to pass to the Python function, and specifies the `oml_connect` control argument to establish an OML4Py connection to the database during the invocation of the user-defined Python function. The `OUT_FMT` parameter specifies returning the value in XML format and the `SCR_NAME` parameter specifies the `myLinearRegressionModel` function in the script repository as the Python function to invoke. The XML output is a CLOB; you can call `set long [length]` to get more output.

```

SELECT *
FROM table(pyqTableEval(
    'IRIS',
    '{"modelName":"linregr",
     "datastoreName":"pymodel",
     "oml_connect":1}',
    'XML',
    'myLinearRegressionModel'));

```

The output is the following:

```

NAME  VALUE
-----
      <root><str>LinearRegression()</str></root>

```

12.6.4 pyqRowEval Function (On-Premises Database)

This topic describes the `pyqRowEval` function when used in an on-premises Oracle Database. The `pyqRowEval` function chunks data into sets of rows and then runs a user-defined Python function on each chunk.

The `pyqRowEval` function passes the data specified by the `INP_NAM` parameter to the Python function specified by the `SCR_NAME` parameter. You can pass arguments to the Python function with the `PAR_LST` parameter.

The `ROW_NUM` parameter specifies the maximum number of rows to pass to each invocation of the Python function. The last set of rows may have fewer rows than the number specified.

The Python function can return a boolean, a dict, a float, an int, a list, a str, a tuple or a `pandas.DataFrame` object. You may define the form of the returned value with the `OUT_FMT` parameter.

Syntax

```
pyqRowEval (
    inp_nam VARCHAR2,
    par_lst VARCHAR2,
    out_fmt VARCHAR2,
    row_num NUMBER,
    scr_name VARCHAR2,
    scr_owner VARCHAR2 DEFAULT NULL)
```

Parameters

Parameter	Description
INP_NAM	The name of a table or view that specifies the data to pass to the Python function specified by the SCR_NAME parameter. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view.
PAR_LST	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the SCR_NAME parameter. Special control arguments, which start with oml_, are not passed to the function specified by SCR_NAME, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as pandas.DataFrame, use: '{"oml_input_type":"pandas.DataFrame"}'
OUT_FMT	The format of the output returned by the function. It can be one of the following: <ul style="list-style-type: none"> • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. • The name of a table or view to use as a prototype. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view. • The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation.
ROW_NUM	The number of rows to include in each invocation of the Python function.

Parameter	Description
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.

Returns

Function `pyqRowEval` returns a table that has the structure specified by the `OUT_FMT` parameter value.

Example 12-17 Using the `pyqRowEval` Function

This example loads the Python model `linreg` to predict row chunks of sample iris data. The model is created and saved in the datastore `pymodel` in [Example 12-16](#).

The example defines a Python function and stores it in the OML4Py script repository. It uses the user-defined Python function to create a database table as the result of the `pyqEval` function. It defines a Python function that runs a prediction function on a model loaded from the OML4Py datastore. It then invokes the `pyqTableEval` function to invoke the function on chunks of rows from the database table.

In a PL/SQL block, define the function `sample_iris_table` and store it in the script repository. The function loads the iris data set, creates two `pandas.DataFrame` objects, and then returns a sample of the concatenation of those objects.

```
BEGIN
  sys.pyqScriptCreate('sample_iris_table',
    'def sample_iris_table(size):
      from sklearn.datasets import load_iris
      import pandas as pd
      iris = load_iris()
      x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
          "Sepal_Width","Petal_Length","Petal_Width"])
      y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
          2: "virginica"}[x], iris.target)),\
          columns = ["Species"])
      return pd.concat([y, x], axis=1).sample(int(size))',
    FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
/
```

Create the `SAMPLE_IRIS` table in the database as the result of a `SELECT` statement, which invokes the `pyqEval` function on the `sample_iris_table` user-defined Python function saved in the script repository with the same name. The `sample_iris_table` function returns an iris data sample of size `size`.

```
CREATE TABLE sample_iris AS
SELECT *
FROM TABLE(pyqEval(
    '{"size":20}',
    '{"Species":"varchar2(10)","Sepal_Length":"number",
    "Sepal_Width":"number","Petal_Length":"number",
```

```
"Petal_Width":"number"}',
'sample_iris_table'));
```

Define the Python function `predict_model` and store it with the name `linregrPredict` in the script repository. The function predicts the data in `dat` with the Python model specified by the `modelName` argument, which is loaded from the datastore specified by the `datastoreName` argument. The predictions are finally concatenated and returned with `dat` as the object that the function returns.

```
BEGIN
  sys.pyqScriptCreate('linregrPredict',
    'def predict_model(dat, modelName, datastoreName):
      import oml
      import pandas as pd
      objs = oml.ds.load(name=datastoreName, to_globals=False)
      pred = objs[modelName].predict(dat[["Sepal_Length", "Sepal_Width", \
                                          "Petal_Length"]])
      return pd.concat([dat, pd.DataFrame(pred, \
                                          columns=["Pred_Petal_Width"])], axis=1)',
    FALSE, TRUE);
END;
/
```

Run a `SELECT` statement that invokes the `pyqRowEval` function, which runs the specified Python function on each chunk of rows in the specified data set.

The `INP_NAM` argument specifies the data in the `SAMPLE_IRIS` table to pass to the Python function.

The `PAR_LST` argument specifies connecting to the OML4Py server with the special control argument `oml_connect`, passing the input data as a `pandas.DataFrame` with the special control argument `oml_input_type`, along with values for the function arguments `modelName` and `datastoreName`.

In the `OUT_FMT` argument, the JSON string specifies the column names and data types of the table returned by `pyqRowEval`.

The `ROW_NUM` argument specifies that five rows are included in each invocation of the function specified by `SCR_NAME`.

The `SCR_NAME` parameter specifies `linregrPredict`, which is the name in the script repository of the user-defined Python function to invoke.

```
SELECT *
FROM table(pyqRowEval(
  'SAMPLE_IRIS',
  '{"oml_connect":1,"oml_input_type":"pandas.DataFrame",
   "modelName":"linregr", "datastoreName":"pymodel"}',
  '{"Species":"varchar2(10)", "Sepal_Length":"number",
   "Sepal_Width":"number", "Petal_Length":"number",
   "Petal_Width":"number", "Pred_Petal_Width":"number"}',
  5,
  'linregrPredict'));
```

The output is the following:

Species	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Pred_Petal_Width
versicolor	5.4	3	4.5	1.5	1.66731546068336
versicolor	6	3.4	4.5	1.6	1.63208723397328
setosa	5.5	4.2	1.4	0.2	
0.289325450127603					
virginica	6.4	3.1	5.5	1.8	2.00641535609046
versicolor	6.1	2.8	4.7	1.2	1.58248012323666
setosa	5.4	3.7	1.5	0.2	
0.251046097050724					
virginica	7.2	3	5.8	1.6	1.97554457713195
versicolor	6.2	2.2	4.5	1.5	1.32323976658868
setosa	4.8	3.1	1.6	0.2	
0.294116926466465					
virginica	6.7	3.3	5.7	2.5	2.0936178656911
virginica	7.2	3.6	6.1	2.5	2.26646663788204
setosa	5	3.6	1.4	0.2	
0.259261360689759					
virginica	6.3	3.4	5.6	2.4	2.14639883810232
virginica	6.1	3	4.9	1.8	1.73186245496453
versicolor	6.1	2.9	4.7	1.4	1.60476297762276
versicolor	5.7	2.8	4.5	1.3	1.56056992978395
virginica	6.4	2.7	5.3	1.9	1.8124673155904
setosa	5	3.5	1.3	0.3	
0.184570194825823					
versicolor	5.6	2.7	4.2	1.3	1.40178874834007
setosa	4.5	2.3	1.3	0.3	
0.0208089790714202					

12.6.5 pyqGroupEval Function (On-Premises Database)

This topic describes the `pyqGroupEval` function when used in an on-premises Oracle Database. The `pyqGroupEval` function groups data by one or more columns and runs a user-defined Python function on each group.

The `pyqGroupEval` function runs the user-defined Python function specified by the `SCR_NAME` parameter. Pass data to the Python function with the `INP_NAM` parameter. Pass arguments to the Python function with the `PAR_LST` parameter. Specify one or more grouping columns with the `GRP_COL` parameter.

The Python function can return a boolean, a dict, a float, an int, a list, a str, a tuple or a `pandas.DataFrame` object. Define the form of the returned value with the `OUT_FMT` parameter.

Syntax

```
pyqGroupEval (
    inp_nam VARCHAR2,
    par_lst VARCHAR2,
    out_fmt VARCHAR2,
    grp_col VARCHAR2,
```

```
scr_name VARCHAR2,  
scr_owner VARCHAR2 DEFAULT NULL)
```

Parameters

Parameter	Description
INP_NAM	The name of a table or view that specifies the data to pass to the Python function specified by the SCR_NAME parameter. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view.
PAR_LST	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the SCR_NAME parameter. Special control arguments, which start with oml_, are not passed to the function specified by SCR_NAME, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as pandas.DataFrame, use: <pre>'{"oml_input_type":"pandas.DataFrame"}'</pre>
OUT_FMT	The format of the output returned by the function. It can be one of the following: <ul style="list-style-type: none"> • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. • The name of a table or view to use as a prototype. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view. • The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation.
GRP_COL	The names of the grouping columns by which to partition the data. Use commas to separate multiple columns. For example, to group by GENDER and YEAR: <pre>"GENDER, YEAR"</pre>
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.

Returns

Function `pyqGroupEval` returns a table that has the structure specified by the `OUT_FMT` parameter value.

Example 12-18 Using the `pyqGroupEval` Function

This example defines the Python function `create_iris_table` and stores it with the name `create_iris_table` in the OML4Py script repository. It then invokes `pyqEval`, which invokes the user-defined Python function and creates the IRIS database table. The example creates the package `irisPkg` and uses that package in specifying the data cursor to pass to the `irisGroupEval` function, which is a user-defined `pyqGroupEval` function. It defines another Python function, `group_count` and stores it in the script repository with the name `mygroupcount`. The example then invokes the `irisGroupEval` function and passes it the Python function saved with the name `mygroupcount`.

In a PL/SQL block, define the Python function `create_iris_table` and store in the script repository with the name `create_iris_table`.

```
BEGIN
  sys.pyqScriptCreate('create_iris_table',
    'def create_iris_table():
      from sklearn.datasets import load_iris
      import pandas as pd
      iris = load_iris()
      x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
        "Sepal_Width","Petal_Length","Petal_Width"])
      y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1:"versicolor",\
        2:"virginica"}[x], iris.target)),\
        columns = ["Species"])
      return pd.concat([y, x], axis=1)');
END;
/
```

Invoke the `pyqEval` function to create the database table `IRIS`, using the Python function stored with the name `create_iris_table` in the script repository.

```
CREATE TABLE IRIS AS
(SELECT * FROM pyqEval(
  NULL,
  '{"Species":"VARCHAR2(10)","Sepal_Length":"number",
  "Sepal_Width":"number","Petal_Length":"number",
  "Petal_Width":"number"}',
  'create_iris_table'
));
```

Define the Python function `group_count` and store it with the name `mygroupcount` in the script repository. The function returns a `pandas.DataFrame` generated on each group of data `dat`.

```
BEGIN
  sys.pyqScriptCreate('mygroupcount',
    'def group_count(dat):
      import pandas as pd
      return pd.DataFrame([(dat["Species"][0], dat.shape[0])],\
```

```

        columns = ["Species", "CNT"]) ');
END;
/

```

Issue a query that invokes the `pyqGroupEval` function. In the function, the `INP_NAM` argument specifies the data in the IRIS table to pass to the function.

The `PAR_LST` argument specifies the special control argument `oml_input_type`.

The `OUT_FMT` argument specifies a JSON string that contains the column names and data types of the table returned by `pyqGroupEval`.

The `GRP_COL` parameter specifies the column to group by.

The `SCR_NAME` parameter specifies the user-defined Python function stored with the name `mygroupcount` in the script repository.

```

SELECT *
FROM table(
  pyqGroupEval(
    'IRIS',
    '{"oml_input_type":"pandas.DataFrame"}',
    '{"Species":"varchar2(10)", "CNT":"number"}',
    'Species',
    'mygroupcount');

```

The output is the following.

Species	CNT
setosa	50
versicolor	50
virginica	50

12.6.6 pyqGrant Function (On-Premises Database)

This topic describes the `pyqGrant` function when used in an on-premises Oracle Database.

The `pyqGrant` function grants read privilege access to an OML4Py datastore or to a script in the OML4Py script repository.

Syntax

```

pyqGrant (
  V_NAME          VARCHAR2    IN
  V_TYPE          VARCHAR2    IN
  V_USER          VARCHAR2    IN      DEFAULT)

```

Parameters

Parameter	Description
<code>V_NAME</code>	The name of an OML4Py datastore or a script in the OML4Py script repository.
<code>V_TYPE</code>	For a datastore, the type is <code>datastore</code> ; for script the type is <code>pyqScript</code> .

Parameter	Description
V_USER	The name of the user to whom to grant access.

Example 12-19 Granting Read Access to a script

```
-- Grant read privilege access to Scott.
BEGIN
  pyqGrant('pyqFun1', 'pyqscript', 'SCOTT');
END;
/
```

Example 12-20 Granting Read Access to a datastore

```
-- Grant read privilege access to datastore ds1 to SCOTT.
BEGIN
  pyqGrant('ds1', 'datastore', 'SCOTT');
END;
/
```

Example 12-21 Granting Read Access to a Script to all Users

```
-- Grant read privilege access to script RandomRedDots to all users.
BEGIN
  pyqGrant('pyqFun1', 'pyqscript', NULL);
END;
/
```

Example 12-22 Granting Read Access to a datastore to all Users

```
-- Grant read privilege access to datastore ds1 to all users.
BEGIN
  pyqGrant('ds1', 'datastore', NULL);
END;
/
```

12.6.7 pyqRevoke Function (On-Premises Database)

This topic describes the `pyqRevoke` function when used in an on-premises Oracle Database.

The `pyqRevoke` function revokes read privilege access to an OML4Py datastore or to a script in the OML4Py script repository.

Syntax

```
pyqRevoke (
  V_NAME          VARCHAR2    IN
  V_TYPE          VARCHAR2    IN
  V_USER          VARCHAR2    IN      DEFAULT)
```


Parameters

Parameter	Description
V_NAME	The name of an OML4Py datastore or a script in the OML4Py script repository.
V_TYPE	For a datastore, the type is <code>datastore</code> ; for script the type is <code>pyqScript</code> .
V_USER	The name of the user from whom to revoke access.

Example 12-23 Revoking Read Access to a script

```
-- Revoke read privilege access to script pyqFun1 from SCOTT.
BEGIN
  pyqRevoke('pyqFun1', 'pyqscript', 'SCOTT');
END;
/
```

Example 12-24 Revoking Read Access to a datastore

```
-- Revoke read privilege access to datastore ds1 from SCOTT.
BEGIN
  pyqRevoke('ds1', 'datastore', 'SCOTT');
END;
/
```

Example 12-25 Revoking Read Access to a script from all Users

```
-- Revoke read privilege access to script pyqFun1 from all users.
BEGIN
  pyqRevoke('pyqFun1', 'pyqscript', NULL);
END;
/
```

Example 12-26 Revoking Read Access to a datastore from all Users

```
-- Revoke read privilege access to datastore ds1 from all users.
BEGIN
  pyqRevoke('ds1', 'datastore', NULL);
END;
/
```

12.6.8 pyqScriptCreate Procedure (On-Premises Database)

This topic describes the `pyqScriptCreate` procedure in an on-premises Oracle Database. The `pyqScriptCreate` procedure creates a user-defined Python function and adds it to the OML4Py script repository.

To create a user-defined Python function, you must have the PYQADMIN database role.

Syntax

```
sys.pyqScriptCreate (
  V_NAME          VARCHAR2    IN
```

V_SCRIPT	CLOB	IN	
V_GLOBAL	BOOLEAN	IN	DEFAULT
V_OVERWRITE	BOOLEAN	IN	DEFAULT)

Parameter	Description
V_NAME	A name for the user-defined Python function in the OML4Py script repository.
V_SCRIPT	The definition of the Python function.
V_GLOBAL	TRUE specifies that the user-defined Python function is public; FALSE specifies that the user-defined Python function is private.
V_OVERWRITE	If the script repository already has a user-defined Python function with the same name as V_NAME, then TRUE replaces the content of that user-defined Python function with V_SCRIPT and FALSE does not replace it.

Example 12-27 Using the pyqScriptCreate Procedure

This example creates a private user-defined Python function named `pyqFun2` in the OML4Py script repository.

```
BEGIN
  sys.pyqScriptCreate('pyqFun2',
    'def return_frame():
      import numpy as np
      import pickle
      z = np.array([y for y in zip([str(x)+"demo" for x in range(10)],
        [float(x)/10 for x in range(10)],
        [x for x in range(10)],
        [bool(x%2) for x in range(10)],
        [pickle.dumps(x) for x in range(10)],
        ["test"+str(x**2) for x in range(10)]]),
        dtype=[("a", "U10"), ("b", "f8"), ("c", "i4"), ("d", "?"),
        ("e", "S20"), ("f", "O")])
      return z');
END;
/
```

This example creates a global user-defined Python function named `pyqFun2` in the script repository and overwrites any existing user-defined Python function of the same name.

```
BEGIN
  sys.pyqScriptCreate('pyqFun2',
    'def return_frame():
      import numpy as np
      import pickle
      z = np.array([y for y in zip([str(x)+"demo" for x in range(10)],
        [float(x)/10 for x in range(10)],
        [x for x in range(10)],
        [bool(x%2) for x in range(10)],
        [pickle.dumps(x) for x in range(10)],
        ["test"+str(x**2) for x in range(10)]]),
        dtype=[("a", "U10"), ("b", "f8"), ("c", "i4"), ("d", "?"),
        ("e", "S20"), ("f", "O")])
      return z',
    TRUE, -- Make the user-defined Python function global.
    TRUE); -- Overwrite any global user-defined Python function
```

```

-- with the same name.
END;
/

```

This example creates a private user-defined Python function named `create_iris_table` in the script repository.

```

BEGIN
  sys.pyqScriptCreate('create_iris_table',
    'def create_iris_table():
      from sklearn.datasets import load_iris
      import pandas as pd
      iris = load_iris()
      x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
        "Sepal_Width","Petal_Length","Petal_Width"])
      y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
        2: "virginica"}[x], iris.target)),\
        columns = ["Species"])
      return pd.concat([y, x], axis=1)');
END;
/

```

Display the user-defined Python functions owned by the current user.

```
SELECT * from USER_PYQ_SCRIPTS;
```

```

NAME                SCRIPT
-----
create_iris_table  def create_iris_table():      from sklearn.datasets
import load_iris ...
pyqFun2            def return_frame():          import numpy as np      import
pickle             ...

```

Display the user-defined Python functions available to the current user.

```
SELECT * from ALL_PYQ_SCRIPTS;
```

```

OWNER    NAME                SCRIPT
-----
OML_USER  create_iris_table  "def create_iris_table(): from
sklearn.datasets import load_iris ...
OML_USER  pyqFun2            "def return_frame(): import numpy as np
import pickle ...
PYQSYS    pyqFun2            "def return_frame(): import numpy as np
import pickle ...

```

12.6.9 pyqScriptDrop Procedure (On-Premises Database)

This topic describes the `pyqScriptDrop` procedure in an on-premises Oracle Database. The `pyqScriptDrop` procedure removes a user-defined Python function from the OML4Py script repository.

To drop a user-defined Python function, you must have the `PYQADMIN` database role.

Syntax

```
sys.pyqScriptDrop (
    V_NAME          VARCHAR2      IN
    V_GLOBAL        BOOLEAN       IN      DEFAULT
    V_SILENT        BOOLEAN       IN      DEFAULT)
```

Parameter	Description
<code>V_NAME</code>	A name for the user-defined Python function in the OML4Py script repository.
<code>V_GLOBAL</code>	A <code>BOOLEAN</code> that specifies whether the user-defined Python function to drop is a global or a private user-defined Python function. The default value is <code>FALSE</code> , which indicates a private user-defined Python function. <code>TRUE</code> specifies that the user-defined Python function is public.
<code>V_SILENT</code>	A <code>BOOLEAN</code> that specifies whether to display an error message when <code>sys.pyqScriptDrop</code> encounters an error in dropping the specified user-defined Python function. The default value is <code>FALSE</code> .

Example 12-28 Using the `sys.pyqScriptDrop` Procedure

For the creation of the user-defined Python functions dropped in these examples, see [Example 12-27](#).

This example drops the private user-defined Python function `pyqFun2` from the script repository.

```
BEGIN
    sys.pyqScriptDrop('pyqFun2');
END;
/
```

This example drops the global user-defined Python function `pyqFun2` from the script repository.

```
BEGIN
    sys.pyqScriptDrop('pyqFun2', TRUE);
END;
/
```

12.7 SQL API for Embedded Python Execution with Autonomous Database

The SQL API for Embedded Python Execution with Autonomous Database provides SQL interfaces for setting authorization tokens, managing access control list (ACL) privileges, executing Python scripts, and synchronously and asynchronously running jobs.

The following topics describe the SQL API.

- [Access and Authorization Procedures and Functions](#)
- [Embedded Python Execution Functions \(Autonomous Database\)](#)
- [oml_async_flag Argument](#)
- [Special Control Arguments \(Autonomous Database\)](#)
- [Output Formats \(Autonomous Database\)](#)
- [Access and Authorization Procedures and Functions](#)
Use the network access control lists (ACL) API to control access by users to external network services and resources from the database. Use the token store API to persist the authorization token issued by a cloud host so it can be used with subsequent SQL calls.
- [Embedded Python Execution Functions \(Autonomous Database\)](#)
The SQL API for Embedded Python Execution with Autonomous Database functions are described in the following topics.
- [Asynchronous Jobs \(Autonomous Database\)](#)
When a function is run asynchronously, it's run as a job which can be tracked by using the `pyqJobStatus` and `pyqJobResult` functions.
- [Special Control Arguments \(Autonomous Database\)](#)
Use the `PAR_LST` parameter to specify special control arguments and additional arguments to be passed into the Python script.
- [Output Formats \(Autonomous Database\)](#)
The `OUT_FMT` parameter controls the format of output returned by the table functions `pyqEval`, `pyqGroupEval`, `pyqIndexEval`, `pyqRowEval`, `pyqTableEval`, and `pyqJobResult`.

12.7.1 Access and Authorization Procedures and Functions

Use the network access control lists (ACL) API to control access by users to external network services and resources from the database. Use the token store API to persist the authorization token issued by a cloud host so it can be used with subsequent SQL calls.

Use the following to manage ACL privileges. An `ADMIN` user is required.

- [pyqAppendHostACE Procedure](#)
- [pyqGetHostACE Function](#)
- [pyqRemoveHostACE Procedure](#)

Use the following to manage authorization tokens:

- [pyqSetAuthToken Procedure](#)
- [pyqIsTokenSet Function](#)

Workflow

The typical workflow for using the SQL API for Embedded Python Execution with Autonomous Database is:

1. Connect to PDB as the `ADMIN` user, and add a normal user `OMLUSER` to the ACL list of the cloud host of which the root domain is `adb.us-region-1.oraclecloudapps.com`:

```
exec pyqAppendHostAce('OMLUSER','adb.us-region-1.oraclecloudapps.com');
```

2. The OML Rest URLs can be obtained from the Oracle Autonomous Database that is provisioned.
 - a. Sign into your [Oracle Cloud Infrastructure](#) account. You will need your OCI user name and password.
 - b. Click the hamburger menu and select Autonomous Database instance that is provisioned. For more information on provisioning an Autonomous Database, see: [Provision an Oracle Autonomous Database](#).
 - c. Click **Service Console** and then click **Development**.
 - d. Scroll down to **Oracle Machine Learning RESTful Services** tile and click **Copy** to obtain the following URLs for:
 - Obtaining the REST authentication token for REST APIs provided by OML:

```
<oml-cloud-service-location-url>/omlusers/
```

The URL `<oml-cloud-service-location-url>` includes the tenancy ID, location, and database name. For example, `https://qtraya2braestch-omldb.adb.us-sanjose-1.oraclecloudapps.com`.

In this example,

- `qtraya2braestch` is the tenancy ID
 - `omldb` is the database name
 - `us-sanjose-1` is the datacenter region
 - `oraclecloudapps.com` is the root domain
3. The Oracle Machine Learning REST API uses tokens to authenticate an Oracle Machine Learning user. To authenticate and obtain an access token, send a POST request to the Oracle Machine Learning User Management Cloud Service REST endpoint `/oauth2/v1/token` with your OML username and password.

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json'
-d '{"grant_type":"password", "username":"'${username}'", "password":"'${password}'"}'
"<oml-cloud-service-location-url>/omlusers/api/oauth2/v1/token"
```

The example uses the following values:

- `username` is the OML username.
- `password` is the OML user password.

- `oml-cloud-service-location-url` is a variable containing the REST server portion of the Oracle Machine Learning User Management Cloud Service instance URL that includes the tenancy ID, database name, and the location name. You can obtain the `omlserver` URL from the Development tab in the Service Console of your Oracle Autonomous Database instance.

 **Note:**

When a token expires, all calls to the OML Services REST endpoints will return a message stating that the token has expired along with the HTTP error:
HTTP/1.1 401 Unauthorized

4. Connect to PDB as `OMLUSER`, set the access token, and run `pyqIndexEval`:

```
exec pyqSetAuthToken('<access token>');
select *
  from table(pyqIndexEval(
    par_lst => NULL,
    out_fmt => '{"ID":"number", "RES":"varchar2(3)"}',
    times_num => 3,
    scr_name => 'idx_ret_df'));
```

```

ID RES
----- ---
  1 a
  2 b
  3 c
```

3 rows selected.

- [pyqAppendHostACE Procedure](#)
The `pyqAppendHostACE` procedure appends an access control entry (ACE) to the access control list (ACL) of the cloud host. The ACL controls access to the cloud host from the database, and the ACE specifies the connect privilege granted to the specified user name.
- [pyqGetHostACE Function](#)
The `pyqGetHostACE` function gets the existing host access control entry (ACE) for the specified user. An exception is raised if the host ACE doesn't exist for the specified user.
- [pyqRemoveHostACE Procedure](#)
- [pyqSetAuthToken Procedure](#)
The `pyqSetAuthToken` procedure sets the access token in the token store.
- [pyqIsTokenSet Function](#)
The `pyqIsTokenSet` function returns whether the authorization token is set or not.

12.7.1.1 pyqAppendHostACE Procedure

The `pyqAppendHostACE` procedure appends an access control entry (ACE) to the access control list (ACL) of the cloud host. The ACL controls access to the cloud host from the database, and the ACE specifies the connect privilege granted to the specified user name.

Syntax

```
PROCEDURE SYS.pyqAppendHostACE(  
    username IN VARCHAR2,  
    host_root_domain IN VARCHAR2  
)
```

Parameter

username - Database user to whom the connect privilege to the cloud host is granted.

host_root_domain - Root domain of the cloud host. For example, if the URL is `https://qtraya2braestch-omldb.adb.us-sanjose-1.oraclecloudapps.com`, the root domain of the cloud host is: `adb.us-sanjose-1.oraclecloudapps.com`.

Example

```
exec pyqAppendHostAce('OMLUSER', 'adb.us-region-1.oraclecloudapps.com');
```



Note:

OML username is case sensitive

12.7.1.2 pyqGetHostACE Function

The `pyqGetHostACE` function gets the existing host access control entry (ACE) for the specified user. An exception is raised if the host ACE doesn't exist for the specified user.

Syntax

```
FUNCTION sys.pyqGetHostACE(  
    p_username IN VARCHAR2  
)
```

Parameter

p_username - Database user to look for the host ACE.

Example

If user `OMLUSER` has access to the cloud host, i.e., `ibuw1q4mjakeils-omlrgpy1.adb.us-region-1.oraclecloudapps.com`, the ADMIN user can run the following to check the user's privileges:

```
SQL> set serveroutput on  
DECLARE
```



```
hostname VARCHAR2(4000);
BEGIN
  hostname := pyqGetHostACE('OMLUSER');
  DBMS_OUTPUT.put_line ('hostname: ' || hostname);
END;
/
SQL> hostname: ibuw1q4mjqkeils-omlrgpy1.adb.us-region-1.oraclecloudapps.com
PL/SQL procedure successfully completed.
```

12.7.1.3 pyqRemoveHostACE Procedure

The `pyqRemoveHostACE` procedure removes the existing host access control entry (ACE) from the specified `username`. If an access token was set for the cloud host, the token is also removed. An exception is raised if the host ACE does not exist.

Syntax

```
PROCEDURE SYS.pyqRemoveHostACE(
  username IN VARCHAR2
)
```

Parameter

`username` - Database user from whom the connect privilege to the cloud host is revoked.

12.7.1.4 pyqSetAuthToken Procedure

The `pyqSetAuthToken` procedure sets the access token in the token store.

Syntax

```
PROCEDURE SYS.pyqSetAuthToken(
  access_token IN VARCHAR2
)
```

12.7.1.5 pyqIsTokenSet Function

The `pyqIsTokenSet` function returns whether the authorization token is set or not.

Syntax

```
FUNCTION SYS.pyqIsTokenSet() RETURN BOOLEAN
```

Example

The following example shows how to use the `pyqSetAuthToken` procedure and the `pyqIsTokenSet` function.

```
DECLARE
  is_set BOOLEAN;
BEGIN
  pyqSetAuthToken('<access token>');

```

```

is_set := pyqIsTokenSet();
IF (is_set) THEN
    DBMS_OUTPUT.put_line ('token is set');
END IF;
END;
/

```

12.7.2 Embedded Python Execution Functions (Autonomous Database)

The SQL API for Embedded Python Execution with Autonomous Database functions are described in the following topics.

Topics

- [pyqListEnvs Function \(Autonomous Database\)](#)
- [pyqEval Function \(Autonomous Database\)](#)
- [pyqTableEval Function \(Autonomous Database\)](#)
- [pyqRowEval Function \(Autonomous Database\)](#)
- [pyqGroupEval Function \(Autonomous Database\)](#)
- [pyqIndexEval Function \(Autonomous Database\)](#)
- [pyqGrant Function \(Autonomous Database\)](#)
- [pyqRevoke Function \(Autonomous Database\)](#)
- [pyqScriptCreate Procedure \(Autonomous Database\)](#)
- [pyqScriptDrop Procedure \(Autonomous Database\)](#)
- [pyqListEnvs Function \(Autonomous Database\)](#)
The function `pyqListEnvs` when used in Oracle Autonomous Database, lists the environments saved in an Object Storage.
- [pyqEval Function \(Autonomous Database\)](#)
The function `pyqEval`, when used in Oracle Autonomous Database, calls a user-defined Python function. Users can pass arguments to the user-defined Python function.
- [pyqTableEval Function \(Autonomous Database\)](#)
The function `pyqTableEval` function when used in Oracle Autonomous Database, runs a user-defined Python function on data from an Oracle Database table.
- [pyqRowEval Function \(Autonomous Database\)](#)
The function `pyqRowEval` when used in Oracle Autonomous Database, chunks data into sets of rows and then runs a user-defined Python function on each chunk.
- [pyqGroupEval Function \(Autonomous Database\)](#)
The function `pyqGroupEval` when used in Oracle Autonomous Database, groups data by one or more columns and runs a user-defined Python function on each group.
- [pyqIndexEval Function \(Autonomous Database\)](#)
The function `pyqIndexEval` when used in Oracle Autonomous Database, runs a user-defined Python function multiple times as required in the Python engines spawned by the database environment.
- [pyqGrant Function \(Autonomous Database\)](#)
This topic describes the `pyqGrant` function when used in Oracle Autonomous Database.

- [pyqRevoke Function \(Autonomous Database\)](#)
This topic describes the `pyqRevoke` function when used in Oracle Autonomous Database.
- [pyqScriptCreate Procedure \(Autonomous Database\)](#)
This topic describes the `pyqScriptCreate` procedure in Oracle Autonomous Database. Use the `pyqScriptCreate` procedure to create a user-defined Python function and add it to the OML4Py script repository.
- [pyqScriptDrop Procedure \(Autonomous Database\)](#)
This topic describes the `pyqScriptDrop` procedure in Oracle Autonomous Database. Use the `pyqScriptDrop` procedure to remove a user-defined Python function from the OML4Py script repository.

12.7.2.1 pyqListEnvs Function (Autonomous Database)

The function `pyqListEnvs` when used in Oracle Autonomous Database, lists the environments saved in an Object Storage.

Syntax

```
FUNCTION PYQSYS.pyqListEnvs
RETURN SYS.AnyDataSet
```

Example

Issue a query that calls the `pyqListEnvs` function and lists the environments present.

```
select * from table(pyqListEnvs());
```

The output is similar to the following:

```
NAME
-----
--
VALUE
-----
--
{"envs":[{"size":" 1.7 GiB","name":"sbenv","description":"Conda environment
with seaborn","number_of_installed_packages":78,"tags":"appli
cation":"OML4PY"}]}
```

12.7.2.2 pyqEval Function (Autonomous Database)

The function `pyqEval`, when used in Oracle Autonomous Database, calls a user-defined Python function. Users can pass arguments to the user-defined Python function.

The function `pyqEval` does not automatically load the data. Within the user-defined Python function, the user may explicitly access and/or retrieve data using the transparency layer or an ROracle database connection.

Syntax

```
FUNCTION PYQSYS.pyqEval (
    PAR_LST  VARCHAR2,
```

```

OUT_FMT    VARCHAR2,
SCR_NAME   VARCHAR2,
SCR_OWNER  VARCHAR2 DEFAULT NULL,
ENV_NAME   VARCHAR2 DEFAULT NULL
)
RETURN SYS.AnyDataSet

```

Parameters

Parameter	Description
PAR_LST	<p>A JSON string that contains additional parameters to pass to the user-defined Python function specified by the <code>SCR_NAME</code> parameter. Special control arguments, which start with <code>oml_</code>, are not passed to the function specified by <code>SCR_NAME</code>, but instead control what happens before or after the invocation of the function.</p> <p>For example, to specify the input data type as <code>pandas.DataFrame</code>, use:</p> <pre>'{"oml_input_type":"pandas.DataFrame"}'</pre> <p>See also: Special Control Arguments (Autonomous Database).</p>
OUT_FMT	<p>The format of the output returned by the function. It can be one of the following:</p> <ul style="list-style-type: none"> • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. The Python function must return a <code>pandas.DataFrame</code>, a <code>numpy.ndarray</code>, a tuple, or a list of tuples. • The string <code>'JSON'</code>, which specifies that the table returned contains a CLOB that is a JSON string. • The string <code>'XML'</code>, which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string <code>'PNG'</code>, which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation. <p>See also: Output Formats (Autonomous Database).</p>
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is <code>NULL</code> . If <code>NULL</code> , will search for the Python script in the user's script repository.
ENV_NAME	The name of the conda environment that should be used when running the named user-defined Python function.

This example defines a Python function and stores it in the OML4Py script repository. It calls the `pyqEval` function on the user-defined Python functions.

In a PL/SQL block, create a Python function that is stored in script repository with the name `pyqFun1`.

```

begin
  sys.pyqScriptCreate('pyqFun1',
    'def fun_tab():
      import pandas as pd
      names = ["demo_"+str(i) for i in range(10)]
      ids = [x for x in range(10)]
      floats = [float(x)/10 for x in range(10)]
      d = {'ID': ids, 'NAME': names, 'FLOAT': floats}
      scores_table = pd.DataFrame(d)

```

```

        return scores_table
',FALSE,TRUE); -- V_GLOBAL, V_OVERWRITE
end;
/

```

Next, call the `pyqEval` function, which runs the user-defined Python function.

The `PAR_LST` argument specifies using `LOW` service level with the special control argument `oml_service_level`.

In the `OUT_FMT` argument, the string `'JSON'`, specifies that the table returned contains a CLOB that is a JSON string.

The `SCR_NAME` parameter specifies the `pyqFun1` function in the script repository as the Python function to call.

The JSON output is a CLOB. You can call `set long [length]` to get more output.

```

set long 500
select *
  from table(pyqEval(
    par_lst => '{"oml_service_level":"LOW"}',
    out_fmt => 'JSON',
    scr_name => 'pyqFun1'));

```

The output is the following.

```

NAME
-----
VALUE
-----
[{"FLOAT":0,"ID":0,"NAME":"demo_0"}, {"FLOAT":0.1,"ID":1,"NAME":"demo_1"}, {"FLOAT":0.2,"ID":2,"NAME":"demo_2"}, {"FLOAT":0.3,"ID":3,"NAME":"demo_3"}, {"FLOAT":0.4,"ID":4,"NAME":"demo_4"}, {"FLOAT":0.5,"ID":5,"NAME":"demo_5"}, {"FLOAT":0.6,"ID":6,"NAME":"demo_6"}, {"FLOAT":0.7,"ID":7,"NAME":"demo_7"}, {"FLOAT":0.8,"ID":8,"NAME":"demo_8"}, {"FLOAT":0.9,"ID":9,"NAME":"demo_9"}]

1 row selected.

```

Issue another query that invokes the same `pyqFun1` script. The `OUT_FMT` argument specifies a JSON string that contains the column names and data types of the structured table output.

```

select *
  from table(pyqEval(
    par_lst => '{"oml_service_level":"LOW"}',
    out_fmt => '{"ID":"number", "NAME":"VARCHAR2(8)", "FLOAT":"binary_double"}',
    scr_name => 'pyqFun1'));

```

The output is the following:

```

ID NAME FLOAT
0 demo_0 0.0

```

```

1 demo_1 0.1
2 demo_2 0.2
3 demo_3 0.3
4 demo_4 0.4
5 demo_5 0.5
6 demo_6 0.6
7 demo_7 0.7
8 demo_8 0.8
9 demo_9 0.9

```

10 rows selected.

Use the following code to create the "seaborn" environment based on Python version 3.10 and upload the environment to the object storage owned by the Pluggable Database (PDB).



Note:

Admin privilege is required to create and manage the Conda environments.

```

create -n seaborn python=3.10 seaborn
upload seaborn --overwrite --description 'Python package for seaborn' -t
python 3.10 -t
      application OML4PY

```

The data visualization library 'seaborn' is installed in the environment.

Use the following code to create the script 'test_seaborn_noimp':

```

begin
  sys.pyqScriptCreate('test_seaborn_noimp',
    'def fun_tab():
      import seaborn as sns
      import matplotlib.pyplot as plt
      import numpy as np
      import pandas as pd
      data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]],
size=2000)
      data = pd.DataFrame(data, columns=["x", "y"])
      sns.displot(data["x"])
      plt.title("Dist plot")
      plt.show()
      return "hello world" ',FALSE,TRUE); -- V_GLOBAL, V_OVERWRITE
end;
/

```

This example calls the pyqEval function, which runs the specified Python script.

The PAR_LST argument specifies capturing images rendered in the script with the special control argument oml_graphics_flag.

In the OUT_FMT arguments, the string 'PNG', specifies returning a table with BLOB containing the images generated by the Python function.

The `SCR_NAME` parameter specifies the 'test_seaborn_noinp' script in the script repository as the Python function to call.

The `ENV_NAME` parameter specifies 'seaborn', which is the Conda environment to run the Python function.

```
select *
  from table(pyqEval(
    par_lst => '{"oml_graphics_flag":true}',
    out_fmt => 'PNG',
    scr_name => 'test_seaborn_noinp',
    scr_owner => NULL,
    env_name => 'seaborn'
  ));
```

The output is the following.

```
NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--
          1
"hello world"

NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--
Lineplot
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58
74536F667477617265004D6174706C6F746C69622076657273696F6E332E332E332C2068747470
73
```

```

NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C000000097048597300000F6100000F6101
A8
3FA7690000682C49444154789CEDDD797C5355FE3FFE579236E9BEB7E942DBB414286B0B2D9482
0A
4AC7023A82A2022E2C83B801A37674147F0A2EDFCF1415114719D119293AC280CC208EC8A050D9
84

NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--

```

In a PL/SQL block, define the Python function `create_iris_table` and store in the script repository with the name `create_iris_table`, overwriting any existing user-defined Python function stored in the script repository with the same name.

The `create_iris_table` function imports and loads the iris data set, creates two `pandas.DataFrame` objects, and then returns the concatenation of those objects.

```

BEGIN
    sys.pyqScriptCreate('create_iris_table',
        'def create_iris_table():
            from sklearn.datasets import load_iris
            import pandas as pd
            iris = load_iris()
            x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
                "Sepal_Width", "Petal_Length", "Petal_Width"])
            y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
                2: "virginica"}[x], iris.target)),\
                columns = ["Species"])
            return pd.concat([y, x], axis=1)',

```



```

        FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
    END;
/
CREATE TABLE IRIS AS
(SELECT * FROM pyqEval(
    NULL,
    '{"Species":"VARCHAR2(10)","Sepal_Length":"number",
     "Sepal_Width":"number","Petal_Length":"number",
     "Petal_Width":"number"}',
    'create_iris_table'
));

```

12.7.2.3 pyqTableEval Function (Autonomous Database)

The function `pyqTableEval` function when used in Oracle Autonomous Database, runs a user-defined Python function on data from an Oracle Database table.

Pass data to the user-defined Python function from the table name specified in the `INP_NAM` parameter. Pass arguments to the user-defined Python function with the `PAR_LST` parameter.

The user-defined Python function can return a `boolean`, a `dict`, a `float`, an `int`, a `list`, a `str`, a `tuple` or a `pandas.DataFrame` object. You define the form of the returned value with the `OUT_FMT` parameter.

Syntax

```

FUNCTION PYQSYS.pyqTableEval(
    INP_NAM    VARCHAR2,
    PAR_LST    VARCHAR2,
    OUT_FMT    VARCHAR2,
    SCR_NAME   VARCHAR2,
    SCR_OWNER  VARCHAR2 DEFAULT NULL,
    ENV_NAME   VARCHAR2 DEFAULT NULL
)
RETURN SYS.AnyDataSet

```

Parameters

Parameter	Description
<code>INP_NAM</code>	The name of a table or view that specifies the data to pass to the Python function specified by the <code>SCR_NAME</code> parameter. If using a table or view owned by another user, use the format <code><owner name>.<table/view name></code> . You must have read access to the specified table or view.
<code>PAR_LST</code>	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the <code>SCR_NAME</code> parameter. Special control arguments, which start with <code>oml_</code> , are not passed to the function specified by <code>SCR_NAME</code> , but instead control what happens before or after the invocation of the function. For example, to specify the input data type as <code>pandas.DataFrame</code> , use: <code>'{"oml_input_type":"pandas.DataFrame"}'</code> See also: Special Control Arguments (Autonomous Database) .

Parameter	Description
OUT_FMT	<p>The format of the output returned by the function. It can be one of the following:</p> <ul style="list-style-type: none"> • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. The Python function must return a <code>pandas.DataFrame</code>, a <code>numpy.ndarray</code>, a tuple, or a list of tuples. • The string 'JSON', which specifies that the table returned contains a CLOB that is a JSON string. • The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation. <p>See also: Output Formats (Autonomous Database).</p>
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.
ENV_NAME	The name of the conda environment that should be used when running the named user-defined Python function.

Example

Define the Python function `fit_model` and store it with the name `myLinearRegressionModel` as a private function in the script repository, overwriting any existing user-defined Python function stored with that name.

The `fit_model` function fits a regression model to the input data `dat` and then saves the fitted model as an object specified by the `modelName` argument to the datastore specified by the `datastoreName` argument. The `fit_model` function returns the fitted model in a string format.

By default, Python objects are saved to a new datastore with the specified `datastoreName`. To save an object to an existing datastore, either set the `overwrite` or `append` argument to `True` in the `oml.ds.save` invocation.

```
BEGIN
  sys.pyqScriptCreate('myLinearRegressionModel',
    'def fit_model(dat, modelName, datastoreName):
      import oml
      from sklearn import linear_model
      regr = linear_model.LinearRegression()
      regr.fit(dat.loc[:, ["Sepal_Length", "Sepal_Width", \
                          "Petal_Length"]],
              dat.loc[:, ["Petal_Width"]])
      oml.ds.save(objs={modelName:regr}, name=datastoreName,
                  overwrite=True)
      return str(regr)',
    FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
/
```

Use the following code to create the 'test_seaborn_inp' script:

```
begin sys.pyqScriptCreate('test_seaborn_inp',
    'def fun_tab(dat):
        import seaborn as sns
        import matplotlib.pyplot as plt
        sns.lineplot(x="Sepal_Length", y="Sepal_Width", data=dat)
        plt.title("Iris plot")
        plt.show()
        return "hello world"
    ',FALSE,TRUE); -- V_GLOBAL, V_OVERWRITE
end;
/
```

This example calls the pyqTableEval function, which runs the specified Python function on the specified data set.

The INP_NAM argument specifies the data in the IRIS table to pass to the Python function.

The PAR_LST argument specifies capturing images rendered in the script with the special control argument oml_graphics_flag.

The OUT_FMT arguments specifies returning a table with BLOB containing the images generated by the Python function.

The SCR_NAME parameter specifies the 'test_seaborn_inp' script, which is the name in the script repository of the user-defined Python function to invoke.

The ENV_NAME parameter specifies 'seaborn', which is a Conda environment created in pyqEval Function (Autonomous Database) .

```
select *
  from table(pyqTableEval(
    inp_nam => 'IRIS',
    par_lst => '{"oml_graphics_flag":true}',
    out_fmt => 'PNG',
    scr_name => 'test_seaborn_inp',
    scr_owner => NULL,
    env_name => 'seaborn'
  ));
```

The output is the following.

```
NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
```

```
--  
  
      1  
"hello world"  
  
NAME  
-----  
--  
      ID  
-----  
VALUE  
-----  
--  
TITLE  
-----  
--  
IMAGE  
-----  
--  
Iris plot  
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445  
58  
74536F667477617265004D6174706C6F746C69622076657273696F6E332E332E332C2068747470  
73  
  
NAME  
-----  
--  
      ID  
-----  
VALUE  
-----  
--  
TITLE  
-----  
--  
IMAGE  
-----  
--  
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C000000097048597300000F6100000F6101  
A8  
3FA7690000B9BC49444154789CECDD797CDC759D3FF0D7F79A2B9399DC499BA44DEF527A41B9CA  
61  
3945C042576559500AABAC2BE2AE8ABA520459442CBA80E0FA0304517057B60A0B28972C22E5A6  
F4  
  
NAME  
-----  
--  
      ID  
-----  
VALUE  
-----  
--  
TITLE  
-----
```

```
--
IMAGE
-----
--
```

This example uses the IRIS table created in the example shown in `pyqEval` Function (Autonomous Database). Run a `SELECT` statement that invokes the `pyqTableEval` function. The `INP_NAM` parameter of the `pyqTableEval` function specifies the IRIS table as the data to pass to the Python function. The `PAR_LST` parameter specifies the names of the model and datastore to pass to the Python function. The `OUT_FMT` parameter specifies returning the value in XML format and the `SCR_NAME` parameter specifies the `myLinearRegressionModel` function in the script repository as the Python function to invoke. The XML output is a CLOB; you can call `set long [length]` to get more output.

```
SELECT *
FROM table(pyqTableEval(
    inp_nam => 'IRIS',
    par_lst => '{"modelName":"linregr",
              "datastoreName":"pymodel"}',
    out_fmt => 'XML',
    scr_name => 'myLinearRegressionModel'));
```

The output is the following:

```
NAME
-----
--
VALUE
-----
--
<root><str>LinearRegression()</str></root>
1 row selected.
```

12.7.2.4 pyqRowEval Function (Autonomous Database)

The function `pyqRowEval` when used in Oracle Autonomous Database, chunks data into sets of rows and then runs a user-defined Python function on each chunk.

The `ROW_NUM` parameter specifies the maximum number of rows to pass to each invocation of the user-defined Python function. The last set of rows may have fewer rows than the number specified.

The user-defined Python function can return a boolean, a dict, a float, an int, a list, a str, a tuple or a `pandas.DataFrame` object. You can define the form of the returned value with the `OUT_FMT` parameter.

Syntax

```
FUNCTION PYQSYS.pyqRowEval(
    INP_NAM    VARCHAR2,
    PAR_LST    VARCHAR2,
    OUT_FMT    VARCHAR2,
    ROW_NUM    NUMBER,
```

```

SCR_NAME  VARCHAR2,
SCR_OWNER VARCHAR2 DEFAULT NULL,
ENV_NAME  VARCHAR2 DEFAULT NULL
)
RETURN SYS.AnyDataSet

```

Parameters

Parameter	Description
INP_NAM	The name of a table or view that specifies the data to pass to the Python function specified by the SCR_NAME parameter. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view.
PAR_LST	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the SCR_NAME parameter. Special control arguments, which start with oml_, are not passed to the function specified by SCR_NAME, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as pandas.DataFrame, use: <code>'{"oml_input_type":"pandas.DataFrame"}'</code> See also: Special Control Arguments (Autonomous Database) .
OUT_FMT	The format of the output returned by the function. It can be one of the following: <ul style="list-style-type: none"> • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. The Python function must return a pandas.DataFrame, a numpy.ndarray, a tuple, or a list of tuples. • The string 'JSON', which specifies that the table returned contains a CLOB that is a JSON string. • The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation. See also: Output Formats (Autonomous Database) .
ROW_NUM	The number of rows in a chunk. The Python script is executed in each chunk.
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.
ENV_NAME	The name of the conda environment that should be used when running the named user-defined Python function.

Example

This example calls the pyqRowEval function, which runs the specified Python script on each chunk of rows in the specified data set.

The INP_NAM argument specifies the data in the IRIS table to pass to the Python function.

The PAR_LST argument specifies capturing images rendered in the script with the special control argument oml_graphics_flag.

The `OUT_FMT` arguments specifies returning a table with BLOB containing the images generated by the Python function.

The `ROW_NUM` argument specifies that 50 rows are included in each invocation of the function specified by `SCR_NAME`.

The `SCR_NAME` parameter specifies the 'test_seaborn_inp' script, which is created in `pyqTableEval` Function (Autonomous Database).

The `ENV_NAME` parameter specifies 'seaborn', which is a Conda environment created in `pyqEval` Function (Autonomous Database) .

```
select *
  from table(pyqRowEval(
    inp_nam => 'IRIS',
    par_lst => '{"oml_graphics_flag":true}',
    out_fmt => 'PNG',
    row_num => 50,
    scr_name => 'test_seaborn_inp',
    scr_owner => NULL,
    env_name => 'seaborn'
  ));
```

The output is the following.

```
NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--
CHUNK_1
          1
"hello world"

NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
```

```
--  
Iris plot  
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445  
58  
74536F667477617265004D6174706C6F746C69622076657273696F6E332E332E332C2068747470  
73
```

NAME

--
ID

VALUE

--
TITLE

--
IMAGE

--
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C000000097048597300000F6100000F6101
A8
3FA7690000812549444154789CEDDD7774D5F5FD3FF0E767DC99BD13C82081B0041450101C888A
0A
54455B57ADA3167F75B46AF5EBB7DA6FADB5D6E2AAA3B52A6A155A6B69B56A97685DE042050105
8A

NAME

--
ID

VALUE

--
TITLE

--
IMAGE

--
CHUNK_2
1

NAME

--
ID

VALUE

--
TITLE

```

--
IMAGE
-----
--
"hello world"
Iris plot
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58

NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--
74536F667477617265004D6174706C6F746C69622076657273696F6E332E332E332C2068747470
73
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C000000097048597300000F6100000F6101
A8
3FA7690000ABB149444154789CECDD79985C65993EFEFB2C55A7BABBBA7A4D6F49670F21210B10
02

NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--
84C5B093804846C5AF8E4C4445470467181CD1388A0A32114401C71FA88802A318070750190920
92

CHUNK_3

NAME
-----
--
          ID
-----
VALUE
-----

```

```

--
TITLE
-----
--
IMAGE
-----
--
      1
"hello world"
Iris plot

NAME
-----
--
      ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58
74536F667477617265004D6174706C6F746C69622076657273696F6E332E332E332C2068747470
73
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C000000097048597300000F6100000F6101
A8

NAME
-----
--
      ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--
3FA76900008C7149444154789CEDDD77945BD5D536F0E7AA774D2F9EE25EB14D316D3060209862
E2
50120204B009900487040229E004028418432881242FA663E7230402015E9AF14B33C5543730B6
71

```

Example

This example loads the Python model `linreg` to predict row chunks of sample iris data. The model is created and saved in the datastore `pymodel`, which is shown in the example for [pyqTableEval Function \(Autonomous Database\)](#).

The example defines a Python function and stores it in the OML4Py script repository. It uses the user-defined Python function to create a database table as the result of the `pyqEval` function. It defines a Python function that runs a prediction function on a model loaded from the OML4Py datastore. It then invokes the `pyqTableEval` function to invoke the function on chunks of rows from the database table.

In a PL/SQL block, define the function `sample_iris_table` and store it in the script repository. The function loads the iris data set, creates two `pandas.DataFrame` objects, and then returns a sample of the concatenation of those objects.

```
BEGIN
  sys.pyqScriptCreate('sample_iris_table',
    'def sample_iris_table(size):
      from sklearn.datasets import load_iris
      import pandas as pd
      iris = load_iris()
      x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
        "Sepal_Width","Petal_Length","Petal_Width"])
      y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
        2: "virginica"}[x], iris.target)),\
        columns = ["Species"])
      return pd.concat([y, x], axis=1).sample(int(size)),
        FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
/
```

Create the `SAMPLE_IRIS` table in the database as the result of a `SELECT` statement, which invokes the `pyqEval` function on the `sample_iris_table` user-defined Python function saved in the script repository with the same name. The `sample_iris_table` function returns an iris data sample of size `size`.

```
CREATE TABLE sample_iris AS
SELECT *
FROM TABLE(pyqEval(
  '{"size":20}',
  '{"Species":"varchar2(10)","Sepal_Length":"number",
  "Sepal_Width":"number","Petal_Length":"number",
  "Petal_Width":"number"}',
  'sample_iris_table'));
```

Define the Python function `predict_model` and store it with the name `linregrPredict` in the script repository. The function predicts the data in `dat` with the Python model specified by the `modelName` argument, which is loaded from the datastore specified by the `datastoreName` argument. The function also plots the actual petal width values with the predicted values. The predictions are finally concatenated and returned with `dat` as the object that the function returns.

```
BEGIN
  sys.pyqScriptCreate('linregrPredict',
    'def predict_model(dat, modelName, datastoreName):
      import oml
      import pandas as pd
      objs = oml.ds.load(name=datastoreName, to_globals=False)
      pred = objs[modelName].predict(dat[["Sepal_Length"],\
```

```

        "Sepal_Width", "Petal_Length"]])
        return pd.concat([dat, pd.DataFrame(pred, \
            columns=["Pred_Petal_Width"])], axis=1)',
        FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
/

```

Run a `SELECT` statement that invokes the `pyqRowEval` function, which runs the specified Python function on each chunk of rows in the specified data set.

The `INP_NAM` argument specifies the data in the `SAMPLE_IRIS` table to pass to the Python function.

The `PAR_LST` argument specifies passing the input data as a pandas. `DataFrame` with the special control argument `oml_input_type`, along with values for the function arguments `modelName` and `datastoreName`.

In the `OUT_FMT` argument, the JSON string specifies the column names and data types of the structured table output.

The `ROW_NUM` argument specifies that five rows are included in each invocation of the function specified by `SCR_NAME`.

The `SCR_NAME` parameter specifies `linregrPredict`, which is the name in the script repository of the user-defined Python function to invoke.

```

SELECT *
FROM table(pyqRowEval(
    inp_nam => 'SAMPLE_IRIS',
    par_lst => '{"oml_input_type":"pandas.DataFrame",
               "modelName":"linregr", "datastoreName":"pymodel"}',
    out_fmt => '{"Species":"varchar2(12)", "Petal_Length":"number",
               "Pred_Petal_Width":"number"}',
    row_num => 5,
    scr_name => 'linregrPredict'));

```

The output is the following.

Species	Petal_Length	Pred_Petal_Width
setosa	1.2	0.0653133202
versicolor	4.5	1.632087234
setosa	1.3	0.2420812759
setosa	1.9	0.5181904241
setosa	1.4	0.2162518989
setosa	1.4	0.1732424372
setosa	1.5	0.2510460971
setosa	1.3	0.1907951829
versicolor	3.9	1.1999981051
versicolor	4.2	1.4017887483
versicolor	4	1.2332360562
versicolor	4.8	1.765473067
virginica	5.6	2.0095892178
versicolor	4.7	1.5824801232
Species	Petal_Length	Pred_Petal_Width
virginica	5.4	2.0623088225

```

versicolor          4.7          1.6524411804
virginica            5.6          1.9919751044
virginica            5.8          2.1206308288
virginica            5.1          1.7983383572
versicolor          4.4          1.3677441077

```

20 rows selected.

Run a `SELECT` statement that invokes the `pyqRowEval` function and return the XML output. Each invocation of script `linregrPredict` is applied to 10 rows of data in the `SAMPLE_IRIS` table. The XML output is a CLOB; you can call `set long [length]` to get more output.

```

set long 300
SELECT *
  FROM table(pyqRowEval(
         inp_nam => 'SAMPLE_IRIS',
         par_lst => '{"oml_input_type":"pandas.DataFrame",
                   "modelName":"linregr", "datastoreName":"pymodel",
"oml_parallel_flag":true", "oml_service_level":"MEDIUM"}',
         out_fmt => 'XML',
         row_num => 10,
         scr_name => 'linregrPredict'));

```

The output is the following:

```

NAME VALUE
      <root><pandas_dataFrame><ROW-pandas_dataFrame><Species>setosa</
Species><Sepal_Length>5</Sepal_Length><Sepal_Width>3.2</
Sepal_Width><Petal_Length>1.2</Petal_Length><Petal_Width>0.2</
Petal_Width><Pred_Petal_Width>0.0653133201897007</Pred_Petal_Width></ROW-
pandas_dataFrame><ROW-pandas_dataFrame><Species>

```

12.7.2.5 pyqGroupEval Function (Autonomous Database)

The function `pyqGroupEval` when used in Oracle Autonomous Database, groups data by one or more columns and runs a user-defined Python function on each group.

The user-defined Python function can return a boolean, a dict, a float, an int, a list, a str, a tuple or a `pandas.DataFrame` object. Define the form of the returned value with the `OUT_FMT` parameter.

Syntax

```

FUNCTION PYQSYS.pyqGroupEval (
  INP_NAM   VARCHAR2,
  PAR_LST   VARCHAR2,
  OUT_FMT   VARCHAR2,
  GRP_COL   VARCHAR2,
  ORD_COL   VARCHAR2,
  SCR_NAME  VARCHAR2,
  SCR_OWNER VARCHAR2 DEFAULT NULL,
  ENV_NAME  VARCHAR2 DEFAULT NULL
)

```

```
)  
RETURN SYS.AnyDataSet
```

Parameters

Parameter	Description
INP_NAM	The name of a table or view that specifies the data to pass to the Python function specified by the SCR_NAME parameter. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view.
PAR_LST	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the SCR_NAME parameter. Special control arguments, which start with oml_, are not passed to the function specified by SCR_NAME, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as pandas.DataFrame, use: '{"oml_input_type":"pandas.DataFrame"}' See also: Special Control Arguments (Autonomous Database) .
OUT_FMT	The format of the output returned by the function. It can be one of the following: <ul style="list-style-type: none"> A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. The Python function must return a pandas.DataFrame, a numpy.ndarray, a tuple, or a list of tuples. The string 'JSON', which specifies that the table returned contains a CLOB that is a JSON string. The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation. See also: Output Formats (Autonomous Database) .
GRP_COL	The names of the grouping columns by which to partition the data. Use commas to separate multiple columns. For example, to group by GENDER and YEAR: "GENDER, YEAR"
ORD_COL	Comma-separated column names to order the input data. For example to order by GENDER: "GENDER" If specified, the input data will first be ordered by the ORD_COL columns and then grouped by the GRP_COL columns.
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.
ENV_NAME	The name of the conda environment that should be used when running the named user-defined Python function.

Example

This example calls the pyqGroupEval function, which runs the specified Python script on each partition of data in the specified data set.

The INP_NAM argument specifies the data in the IRIS table to pass to the Python function.

The `PAR_LST` argument specifies capturing images rendered in the script with the special control argument `oml_graphics_flag`.

The `OUT_FMT` arguments specifies returning a table with BLOB containing the images generated by the Python function.

The `GRP_COL` argument specifies to group the specified data by the 'Species' column.

The `SCR_NAME` parameter specifies the 'test_seaborn_inp' script, which is created in `pyqTableEval` Function (Autonomous Database).

The `ENV_NAME` parameter specifies 'seaborn', which is a Conda environment created in `pyqEval` Function (Autonomous Database) .

```
select *
  from table(pyqGroupEval(
    inp_nam => 'IRIS',
    par_lst => '{"oml_graphics_flag":true}',
    out_fmt => 'PNG',
    grp_col => 'Species',
    ord_col => NULL,
    scr_name => 'test_seaborn_inp',
    scr_owner => NULL,
    env_name => 'seaborn'
  ));
```

The output is the following.

```
NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--
GROUP_setosa
          1
"hello world"

NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
```

```
--  
IMAGE  
-----  
--  
Iris plot  
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445  
58  
74536F667477617265004D6174706C6F746C69622076657273696F6E332E332E332C2068747470  
73  
  
NAME  
-----  
--  
          ID  
-----  
VALUE  
-----  
--  
TITLE  
-----  
--  
IMAGE  
-----  
--  
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C000000097048597300000F6100000F6101  
A8  
3FA76900006AC649444154789CEDDD797854E5DD3EF0FBCC9EC92CD9F7B02624EC088A804B4440  
05  
AAD2AAB5D4166DD5F7ADDAB75A972AD60D375C706B2D2E588BED5B6A5FFD556AA91B5551145490  
45  
  
NAME  
-----  
--  
          ID  
-----  
VALUE  
-----  
--  
TITLE  
-----  
--  
IMAGE  
-----  
--  
  
GROUP_versicolor  
      1  
  
NAME  
-----  
--  
          ID  
-----  
VALUE  
-----
```



```

--
TITLE
-----
--
IMAGE
-----
--
"hello world"
Iris plot
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58

NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--
74536F667477617265004D6174706C6F746C69622076657273696F6E332E332E332C2068747470
73
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C000000097048597300000F6100000F6101
A8
3FA76900009E9149444154789CECDD77785CE5993FFCEF2973CE548D7AB124F76E6C03A69966AA
C1

NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--
B0244E42360B01872CC96F43CC2659922C980D9B42884902A9FB420229904D1CD22064E90EC1A6
19

GROUP_virginica

NAME
-----
--
          ID

```

```
-----  
VALUE  
-----  
--  
TITLE  
-----  
--  
IMAGE  
-----  
--  
      1  
"hello world"  
Iris plot  
  
NAME  
-----  
--  
      ID  
-----  
VALUE  
-----  
--  
TITLE  
-----  
--  
IMAGE  
-----  
--  
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445  
58  
74536F667477617265004D6174706C6F746C69622076657273696F6E332E332E332C2068747470  
73  
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C000000097048597300000F6100000F6101  
A8  
  
NAME  
-----  
--  
      ID  
-----  
VALUE  
-----  
--  
TITLE  
-----  
--  
IMAGE  
-----  
--  
3FA7690000838E49444154789CEDDD797854E5D906F07BF6996496EC7B2010F64D14C1065450C1  
05  
8A625B6B2D0A56ED82B8B756B12AA245D4AAD5B61FA8B8606BA9AD56DC91E2120465DFF73D0442  
42
```

Example

This example uses the IRIS table created in the example shown in [pyqEval Function \(Autonomous Database\)](#).

Define the Python function `group_count` and store it with the name `mygroupcount` in the script repository. The function returns a `pandas.DataFrame` generated on each group of data `dat`. The function also plots the sepal length with the petal length values on each group.

```
BEGIN
  sys.pyqScriptCreate('mygroupcount',
    'def group_count(dat):
      import pandas as pd
      import matplotlib.pyplot as plt
      plt.plot(dat[["Sepal_Length"]], dat[["Petal_Length"]], ".")
      plt.xlabel("Sepal Length")
      plt.ylabel("Petal Length")
      plt.title("{}".format(dat["Species"][0]))
      return pd.DataFrame([(dat["Species"][0], dat.shape[0])],\
        columns = ["Species", "CNT"]) ',
    FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
/
```

Issue a query that invokes the `pyqGroupEval` function. In the function, the `INP_NAM` argument specifies the data in the IRIS table to pass to the function.

The `PAR_LST` argument specifies the special control argument `oml_input_type`.

The `OUT_FMT` argument specifies a JSON string that contains the column names and data types of the table returned by `pyqGroupEval`.

The `GRP_COL` parameter specifies the column to group by.

The `SCR_NAME` parameter specifies the user-defined Python function stored with the name `mygroupcount` in the script repository.

```
SELECT *
  FROM table(
    pyqGroupEval(
      inp_nam => 'IRIS',
      par_lst => '{"oml_input_type":"pandas.DataFrame"}',
      out_fmt => '{"Species":"varchar2(10)", "CNT":"number"}',
      grp_col => 'Species',
      ord_col => NULL,
      scr_name => 'mygroupcount');
```

The output is the following:

```
Species CNT
-----
virginica 50
setosa 50
versicolor 50
3 rows selected.
```

Run the same script with IRIS data and return the XML output. The `PAR_LST` argument specifies the special control argument `oml_graphics_flag` to capture images rendered in the script. Both structured data and images are included in the XML output. The XML output is a CLOB; you can call `set long [length]` to get more output.

```
set long 300
SELECT *
  FROM table(
    pyqGroupEval(
      inp_nam => 'IRIS',
      par_lst => '{"oml_input_type":"pandas.DataFrame",
"oml_graphics_flag":true, "oml_parallel_flag":true",
"oml_service_level":"MEDIUM"}',
      out_fmt => 'XML',
      grp_col => 'Species',
      ord_col => NULL,
      scr_name => 'mygroupcount');
```

The output is the following.

```
NAME VALUE
virginica <root><Py-data><pandas_dataframe><ROW-
pandas_dataframe><Species>virginica</Species><CNT>50</CNT></ROW-
pandas_dataframe></pandas_dataframe></Py-data><images><image><!
[CDATA[ivBORw0KGgoAAAANSUHEUgAAoAAAAHgCAYAAAA10dzkAAAAOXRFWHRTb2Z0d2FyZQBNYXR
wbG90bGliIHZlcnNpb24zLjMu
setosa <root><Py-data><pandas_dataframe><ROW-
pandas_dataframe><Species>setosa</Species><CNT>50</CNT></ROW-
pandas_dataframe></pandas_dataframe></Py-data><images><image><!
[CDATA[ivBORw0KGgoAAAANSUHEUgAAoAAAAHgCAYAAAA10dzkAAAAOXRFWHRTb2Z0d2FyZQBNYXR
wbG90bGliIHZlcnNpb24zLjMuMyw
versicolor <root><Py-data><pandas_dataframe><ROW-
pandas_dataframe><Species>versicolor</Species><CNT>50</CNT></ROW-
pandas_dataframe></pandas_dataframe></Py-data><images><image><!
[CDATA[ivBORw0KGgoAAAANSUHEUgAAoAAAAHgCAYAAAA10dzkAAAAOXRFWHRTb2Z0d2FyZQBNYXR
wbG90bGliIHZlcnNpb24zLjM
```

Run the same script with IRIS data and get the PNG output. The `PAR_LST` argument specifies the special control argument `oml_graphics_flag` to capture images.

```
column name format a7
column value format a15
column title format a16
column image format a15
SELECT *
  FROM table(
    pyqGroupEval(
      inp_nam => 'IRIS',
      par_lst => '{"oml_input_type":"pandas.DataFrame",
"oml_graphics_flag":true}',
      out_fmt => 'PNG',
```

```
grp_col => 'Species',
ord_col => NULL,
scr_name => 'mygroupcount');
```

The output is the following:

NAME	ID	VALUE	TITLE	IMAGE
GROUP_s etosa	1	[{"Species": "setosa", "CNT": 50}]	setosa	89504E470D0A1A0 A000000D494844 520000028000000 1E0080600000035 D1DCE4000000397 4455874536F6674 77617265004D617 4706C6F746C6962 2076657273696F6 E332E332E332C20 6874747073

NAME	ID	VALUE	TITLE	IMAGE
GROUP_v ersicolor	1	[{"Species": "versicolor", "CNT": 50}]	versicolor	89504E470D0A1A0 A000000D494844 520000028000000 1E0080600000035 D1DCE4000000397 4455874536F6674 77617265004D617 4706C6F746C6962 2076657273696F6 E332E332E332C20

NAME	ID	VALUE	TITLE	IMAGE
GROUP_v irginica	1	[{"Species": "virginica", "CNT": 50}]	virginica	89504E470D0A1A0 A000000D494844 520000028000000 1E0080600000035 D1DCE4000000397 4455874536F6674 77617265004D617 4706C6F746C6962 2076657273696F6

12.7.2.6 pyqIndexEval Function (Autonomous Database)

The function `pyqIndexEval` when used in Oracle Autonomous Database, runs a user-defined Python function multiple times as required in the Python engines spawned by the database environment.

Syntax

```
FUNCTION PYQSYS.pyqIndexEval(  
    PAR_LST VARCHAR2,  
    OUT_FMT VARCHAR2,  
    TIMES_NUM NUMBER,  
    SCR_NAME VARCHAR2,  
    SCR_OWNER VARCHAR2 DEFAULT NULL,  
    ENV_NAME VARCHAR2 DEFAULT NULL  
)  
RETURN SYS.AnyDataSet
```

Parameters

Parameter	Description
PAR_LST	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the SCR_NAME parameter. Special control arguments, which start with om1_, are not passed to the function specified by

Parameter	Description
	<p>SCR_NAME, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as pandas.DataFrame, use: <code>'{"input_type": "pandas.DataFrame"}'</code></p> <p>See also: Special Arguments</p>

Parameter	Description
	(Autonomous Database).

Parameter	Description
OUT_FMT	<p>The format of the output returned by the function. It can be one of the following:</p> <ul style="list-style-type: none"><li data-bbox="1409 888 1466 1911">• A JSON string that specifies the column name

Parameter	Description
	e s a n d d a t a t y p e s o f t h e t a b l e r e t u r n e d b y t h e f u n c t i o n . A n y i m a g e d a

Parameter	Description
	t a i s d i s c a r d e d . T h e P y t h o n f u n c t i o n m u s t r e t u r n a p a n d a s . D a t a F r

Parameter	Description
	a m e , a n u m p y . n d a r r a y , a t t u p l e , O r a c l e l i s t o f t u p l e s . • T h e s t r i n g , J S

Parameter	Description
	ON, which specifies that the table returned contains a CLOB that is

Parameter	Description
	s a J S O N s t r i n g . • The string, XML, which specifies that the table

Parameter	Description
	returned content as a CLOB that is an XML string. The XML can contain a blob

Parameter	Description
	h s t r u c t u r e d d a t a a n d i m a g e s , w i t h s t r u c t u r e d o r s e m i - s t r u c t u r e d

Parameter	Description
	Python object first, followed by the image or image generated by

Parameter	Description
	the Python function. • The string, PNG, which specifies that the data

Parameter	Description
	blob returned contains a BLOB that has the image originally generated

Parameter	Description
	by the Python function .images are returned as a base 64 encoding of

Parameter	Description
TIMES_NUM	<p>the PING representation of data in .</p> <p>See also: Output Formats (Autonomous Database).</p> <p>The number of times to execute the Python script.</p>

Parameter	Description
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.

Parameter	Description
ENV_NAME	The name of the conda environment that should be used when running the named user-defined Python function.

Example

This example defines a Python function to use with Conda environment.

Use the following code to create the 'test_seaborn_idx' script:

```
begin
  sys.pyqScriptCreate('test_seaborn_idx',
    'def fun_tab(idx):
      import seaborn as sns
      import matplotlib.pyplot as plt
      import numpy as np
      import pandas as pd
      data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]], size=2000)
      data = pd.DataFrame(data, columns=["x", "y"])
      sns.displot(data["x"])
      plt.title("Title {}".format(idx))
      plt.show()
      return idx
    ', FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
end;
/
```

This example calls the pyqIndexEval function, which runs the specified Python function multiple times.

The `PAR_LST` argument specifies capturing images rendered in the script with the special control argument `oml_graphics_flag`.

The `OUT_FMT` arguments specifies returning a table with BLOB containing the images generated by the Python function.

The `TIMES_NUM` argument specifies to run the specified script 2 times.

The `SCR_NAME` parameter specifies the 'test_seaborn_idx' script as the Python function to invoke.

The `ENV_NAME` parameter specifies 'seaborn', which is a Conda environment created in `pyqEval` Function (Autonomous Database) .

```
select *
  from table(pyqIndexEval(
    par_lst => '{"oml_graphics_flag":true}',
    out_fmt => 'PNG',
    times_num => 2,
    scr_name => 'test_seaborn_idx',
    scr_owner => NULL,
    env_name => 'seaborn'
  ));
```

The output is the following.

```
NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--
TIME_1          1
1
NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
```

```
-----  
--  
Title 1  
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445  
58  
74536F667477617265004D6174706C6F746C69622076657273696F6E332E332E332C2068747470  
73  
  
NAME  
-----  
--  
          ID  
-----  
VALUE  
-----  
--  
TITLE  
-----  
--  
IMAGE  
-----  
--  
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C000000097048597300000F6100000F6101  
A8  
3FA7690000666749444154789CEDDD797C5355FE3FFE579236E9BEB7495BBA52A0AC2D1428C505  
94  
7E2DA0A3082AA0332083B80C30424747F1A720CE5254441C65649C11706340E68338A283426551  
28  
  
NAME  
-----  
--  
          ID  
-----  
VALUE  
-----  
--  
TITLE  
-----  
--  
IMAGE  
-----  
--  
  
TIME_2  
      1  
  
NAME  
-----  
--  
          ID  
-----  
VALUE  
-----  
--  
TITLE
```

```

-----
--
IMAGE
-----
--
2
Title 2
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58

NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--
74536F667477617265004D6174706C6F746C69622076657273696F6E332E332E332C2068747470
73
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C000000097048597300000F6100000F6101
A8
3FA7690000687649444154789CEDDD79785355FE3FF0F74DDAA47BBA6FD0D2859DB216A8451495
7E

NAME
-----
--
          ID
-----
VALUE
-----
--
TITLE
-----
--
IMAGE
-----
--
2DC2A8082A220E8A88CA80A3561DADBF19709919500171D491D1914505451CF7A55A2A204BD95A
CA

```

Example

Define the Python function `fit_lm` and store it with the name `myFitMultiple` in the script repository. The function returns a `pandas.DataFrame` containing the index and prediction score of the fitted model on the data sampled from `scikit-learn`'s IRIS dataset.

```

begin
  sys.pyqScriptCreate('myFitMultiple',

```

```

def fit_lm(i, sample_size):
    from sklearn import linear_model
    from sklearn.datasets import load_iris
    import pandas as pd

    import random
    random.seed(10)

    iris = load_iris()
    x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
        "Sepal_Width","Petal_Length","Petal_Width"])
    y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
        2: "virginica"}[x], iris.target)),\
        columns = ["Species"])
    dat = pd.concat([y, x], axis=1).sample(sample_size)
    regr = linear_model.LinearRegression()
    regr.fit(x.loc[:, ["Sepal_Length", "Sepal_Width", \
        "Petal_Length"]],
        x.loc[:,["Petal_Width"]])
    sc = regr.score(dat.loc[:, ["Sepal_Length", "Sepal_Width", \
        "Petal_Length"]],
        dat.loc[:,["Petal_Width"]])
    return pd.DataFrame([[i,sc]],columns=["id","score"])
',FALSE,TRUE); -- V_GLOBAL, V_OVERWRITE
end;
/

```

Issue a query that invokes the `pyqIndexEval` function. In the function, the `PAR_LST` argument specifies the function argument `sample_size`. The `OUT_FMT` argument specifies a JSON string that contains the column names and data types of the table returned by `pyqIndexEval`. The `TIMES_NUM` parameter specifies the number of times to execute the script. The `SCR_NAME` parameter specifies the user-defined Python function stored with the name `myFitMultiple` in the script repository.

```

select *
  from table(pyqIndexEval(
    par_lst => '{"sample_size":80,
              "oml_parallel_flag":true,
"oml_service_level":"MEDIUM"}',
    out_fmt => '{"id":"number","score":"number"}',
    times_num => 3,
    scr_name => 'myFitMultiple'));

```

The output is the following:

```

      id score
-----
      1 .943550631
      2 .927836941
      3 .937196049
3 rows selected.

```

12.7.2.7 pyqGrant Function (Autonomous Database)

This topic describes the `pyqGrant` function when used in Oracle Autonomous Database.

The `pyqGrant` function grants read privilege access to an OML4Py datastore or to a script in the OML4Py script repository.

Syntax

```
pyqGrant (
    V_NAME          VARCHAR2      IN
    V_TYPE          VARCHAR2      IN
    V_USER          VARCHAR2      IN      DEFAULT)
```

Parameters

Parameter	Description
V_NAME	The name of an OML4Py datastore or a script in the OML4Py script repository.
V_TYPE	For a datastore, the type is <code>datastore</code> ; for script the type is <code>pyqScript</code> .
V_USER	The name of the user to whom to grant access.

Example 12-29 Granting Read Access to a script

```
-- Grant read privilege access to Scott.
BEGIN
    pyqGrant('pyqFun1', 'pyqscript', 'SCOTT');
END;
/
```

Example 12-30 Granting Read Access to a datastore

```
-- Grant read privilege access to datastore ds1 to SCOTT.
BEGIN
    pyqGrant('ds1', 'datastore', 'SCOTT');
END;
/
```

Example 12-31 Granting Read Access to a Script to all Users

```
-- Grant read privilege access to script RandomRedDots to all users.
BEGIN
    pyqGrant('pyqFun1', 'pyqscript', NULL);
END;
/
```

Example 12-32 Granting Read Access to a datastore to all Users

```
-- Grant read privilege access to datastore ds1 to all users.
BEGIN
    pyqGrant('ds1', 'datastore', NULL);
```

```
END;  
/
```

12.7.2.8 pyqRevoke Function (Autonomous Database)

This topic describes the `pyqRevoke` function when used in Oracle Autonomous Database.

The `pyqRevoke` function revokes read privilege access to an OML4Py datastore or to a script in the OML4Py script repository.

Syntax

```
pyqRevoke (  
    V_NAME          VARCHAR2    IN  
    V_TYPE          VARCHAR2    IN  
    V_USER          VARCHAR2    IN    DEFAULT)
```

Parameters

Parameter	Description
V_NAME	The name of an OML4Py datastore or a script in the OML4Py script repository.
V_TYPE	For a datastore, the type is <code>datastore</code> ; for script the type is <code>pyqScript</code> .
V_USER	The name of the user from whom to revoke access.

Example 12-33 Revoking Read Access to a script

```
-- Revoke read privilege access to script pyqFun1 from SCOTT.  
BEGIN  
    pyqRevoke('pyqFun1', 'pyqscript', 'SCOTT');  
END;  
/
```

Example 12-34 Revoking Read Access to a datastore

```
-- Revoke read privilege access to datastore dsl from SCOTT.  
BEGIN  
    pyqRevoke('dsl', 'datastore', 'SCOTT');  
END;  
/
```

Example 12-35 Revoking Read Access to a script from all Users

```
-- Revoke read privilege access to script pyqFun1 from all users.  
BEGIN  
    pyqRevoke('pyqFun1', 'pyqscript', NULL);  
END;  
/
```

Example 12-36 Revoking Read Access to a datastore from all Users

```
-- Revoke read privilege access to datastore dsl from all users.
BEGIN
  pyqRevoke('dsl', 'datastore', NULL);
END;
/
```

12.7.2.9 pyqScriptCreate Procedure (Autonomous Database)

This topic describes the `pyqScriptCreate` procedure in Oracle Autonomous Database. Use the `pyqScriptCreate` procedure to create a user-defined Python function and add it to the OML4Py script repository.

Syntax

```
sys.pyqScriptCreate (
  V_NAME          VARCHAR2      IN
  V_SCRIPT        CLOB          IN
  V_GLOBAL        BOOLEAN       IN      DEFAULT
  V_OVERWRITE     BOOLEAN       IN      DEFAULT)
```

Parameter	Description
V_NAME	A name for the user-defined Python function in the OML4Py script repository.
V_SCRIPT	The definition of the Python function.
V_GLOBAL	TRUE specifies that the user-defined Python function is public; FALSE specifies that the user-defined Python function is private.
V_OVERWRITE	If the script repository already has a user-defined Python function with the same name as V_NAME, then TRUE replaces the content of that user-defined Python function with V_SCRIPT and FALSE does not replace it.

Example 12-37 Using the pyqScriptCreate Procedure

This example creates a private user-defined Python function named `pyqFun2` in the OML4Py script repository.

```
BEGIN
  sys.pyqScriptCreate('pyqFun2',
    'def return_frame():
      import numpy as np
      import pickle
      z = np.array([y for y in zip([str(x)+"demo" for x in range(10)],
        [float(x)/10 for x in range(10)],
        [x for x in range(10)],
        [bool(x%2) for x in range(10)],
        [pickle.dumps(x) for x in range(10)],
        ["test"+str(x**2) for x in range(10)]]),
        dtype=[("a", "U10"), ("b", "f8"), ("c", "i4"), ("d", "?"),
        ("e", "S20"), ("f", "O")]
      return z');
END;
/
```

This example creates a global user-defined Python function named `pyqFun2` in the script repository and overwrites any existing user-defined Python function of the same name.

```
BEGIN
  sys.pyqScriptCreate('pyqFun2',
    'def return_frame():
      import numpy as np
      import pickle
      z = np.array([y for y in zip([str(x)+"demo" for x in range(10)],
        [float(x)/10 for x in range(10)],
        [x for x in range(10)],
        [bool(x%2) for x in range(10)],
        [pickle.dumps(x) for x in range(10)],
        ["test"+str(x**2) for x in range(10)]]),
        dtype=[("a", "U10"), ("b", "f8"), ("c", "i4"), ("d", "?"),
        ("e", "S20"), ("f", "O")])
      return z',
    TRUE, -- Make the user-defined Python function global.
    TRUE); -- Overwrite any global user-defined Python function
           -- with the same name.
END;
/
```

This example creates a private user-defined Python function named `create_iris_table` in the script repository.

```
BEGIN
  sys.pyqScriptCreate('create_iris_table',
    'def create_iris_table():
      from sklearn.datasets import load_iris
      import pandas as pd
      iris = load_iris()
      x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
        "Sepal_Width", "Petal_Length", "Petal_Width"])
      y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
        2: "virginica"}[x], iris.target)),\
        columns = ["Species"])
      return pd.concat([y, x], axis=1)');
END;
/
```

Display the user-defined Python functions owned by the current user.

```
SELECT * from USER_PYQ_SCRIPTS;
```

```
NAME          SCRIPT
-----
-----
create_iris_table  def create_iris_table():      from sklearn.datasets
import load_iris ...
pyqFun2          def return_frame():          import numpy as np      import
pickle          ...
```


Display the user-defined Python functions available to the current user.

```
SELECT * from ALL_PYQ_SCRIPTS;
```

```
OWNER      NAME                SCRIPT
-----  -
-----
OML_USER   create_iris_table  "def create_iris_table(): from
sklearn.datasets import load_iris ...
OML_USER   pyqFun2            "def return_frame(): import numpy as np
import pickle      ...
PYQSYS     pyqFun2            "def return_frame(): import numpy as np
import pickle      ...
```

12.7.2.10 pyqScriptDrop Procedure (Autonomous Database)

This topic describes the `pyqScriptDrop` procedure in Oracle Autonomous Database. Use the `pyqScriptDrop` procedure to remove a user-defined Python function from the OML4Py script repository.

Syntax

```
sys.pyqScriptDrop (
    V_NAME          VARCHAR2      IN
    V_GLOBAL        BOOLEAN        IN      DEFAULT
    V_SILENT        BOOLEAN        IN      DEFAULT)
```

Parameter	Description
V_NAME	A name for the user-defined Python function in the OML4Py script repository.
V_GLOBAL	A BOOLEAN that specifies whether the user-defined Python function to drop is a global or a private user-defined Python function. The default value is FALSE, which indicates a private user-defined Python function. TRUE specifies that the user-defined Python function is public.
V_SILENT	A BOOLEAN that specifies whether to display an error message when <code>sys.pyqScriptDrop</code> encounters an error in dropping the specified user-defined Python function. The default value is FALSE.

Example 12-38 Using the sys.pyqScriptDrop Procedure

This example drops the private user-defined Python function `pyqFun2` from the script repository.

```
BEGIN
    sys.pyqScriptDrop('pyqFun2');
END;
/
```

This example drops the global user-defined Python function `pyqFun2` from the script repository.

```
BEGIN
    sys.pyqScriptDrop('pyqFun2', TRUE);
```

```
END;  
/
```

12.7.3 Asynchronous Jobs (Autonomous Database)

When a function is run asynchronously, it's run as a job which can be tracked by using the `pyqJobStatus` and `pyqJobResult` functions.

Topics:

- [oml_async_flag Argument](#)
- [pyqJobStatus Function](#)
- [pyqJobResult Function](#)
- [Asynchronous Job Example](#)
- [oml_async_flag Argument](#)
The special control argument `oml_async_flag` determines if a job is run synchronously or asynchronously. The default value is `false`.
- [pyqJobStatus Function](#)
Use the `pyqJobStatus` function to look up the status of an asynchronous job. If the job is pending, it returns `job is still running`. If the job is completed, the function returns a URL.
- [pyqJobResult Function](#)
Use the `pyqJobResult` function to return the job result.
- [Asynchronous Job Example](#)
The following examples shows how to submit asynchronous jobs with non-XML output and with XML output.

12.7.3.1 oml_async_flag Argument

The special control argument `oml_async_flag` determines if a job is run synchronously or asynchronously. The default value is `false`.

Set the `oml_async_flag` Argument

- To run a function in synchronous mode, set `oml_async_flag` to `false`.
In synchronous mode, the SQL API waits for the HTTP call to finish and returns when the HTTP response is ready.
By default, `pyq*Eval` functions are executed synchronously. The default connection timeout limit is 60 seconds. Synchronous mode is used if `oml_async_flag` is not set or if it's set to `false`.
- To run a function in asynchronous mode, set `oml_async_flag` to `true`.
In asynchronous mode, the SQL API returns a URL directly after the asynchronous job is submitted to the web server. The URL contains a job ID, which can be used to fetch the job status and result in subsequent SQL calls.

Submit Asynchronous Job Example

This example uses the table `GRADE`, created as follows:

```
CREATE TABLE GRADE (
NAME VARCHAR2(30),
GENDER VARCHAR2(1),
STATUS NUMBER(10),
YEAR NUMBER(10),
SECTION VARCHAR2(1),
SCORE NUMBER(10),
FINALGRADE NUMBER(10)
);

insert into GRADE values('Abbott', 'F', 2, 97, 'A', 90, 87);
insert into GRADE values('Branford', 'M', 1, 98, 'A', 92, 97);
insert into GRADE values('Crandell', 'M', 2, 98, 'B', 81, 71);
insert into GRADE values('Dennison', 'M', 1, 97, 'A', 85, 72);
insert into GRADE values('Edgar', 'F', 1, 98, 'B', 89, 80);
insert into GRADE values('Faust', 'M', 1, 97, 'B', 78, 73);
insert into GRADE values('Greeley', 'F', 2, 97, 'A', 82, 91);
insert into GRADE values('Hart', 'F', 1, 98, 'B', 84, 80);
insert into GRADE values('Isley', 'M', 2, 97, 'A', 88, 86);
insert into GRADE values('Jasper', 'M', 1, 97, 'B', 91, 83);
```

In the following code, the Python function `score_diff` is defined and stored with the name `computeGradeDiff` as a private function in the script repository. The function returns a `pandas.DataFrame` after assigning a new `DIFF` column by computing the difference between the `SCORE` and `FINALGRADE` column of the input data.

```
begin
  sys.pyqScriptCreate('computeGradeDiff','def score_diff(dat):
    import numpy as np
    import pandas as pd
    df = dat.assign(DIFF=dat.SCORE-dat.FINALGRADE)
    return df
  ');
end;
/
```

Run the saved `computeGradeDiff` script as follows:

```
select *
  from table(pyqTableEval(
    inp_nam => 'GRADE',
    par_lst => '{"oml_async_flag":true}',
    out_fmt => NULL,
    scr_name => 'computeGradeDiff',
    scr_owner => NULL
  ));
```

The `VALUE` column of the result contains a URL containing the job ID of the asynchronous job:

```

NAME
-----
--
VALUE
-----
--
https://<oml-cloud-service-location-url>/oml/api/py-scripts/v1/jobs/<job id>
1 row selected.

```

12.7.3.2 pyqJobStatus Function

Use the `pyqJobStatus` function to look up the status of an asynchronous job. If the job is pending, it returns `job is still running`. If the job is completed, the function returns a URL.

Syntax

```

FUNCTION PYQSYS.pyqJobStatus(
  job_id      VARCHAR2
)
RETURN PYQSYS.pyqClobSet

```

Parameters

Parameter	Description
<code>job_id</code>	The ID of the asynchronous job.

Example

The following example shows a `pyqJobStatus` call and its output.

```

SQL> select * from pyqJobStatus(
      job_id => '<job id>'
);

NAME
-----
--
VALUE
-----
--
https://<oml-cloud-service-location-url>/oml/api/py-scripts/v1/jobs/<job id>/
result
1 row selected.

```

12.7.3.3 pyqJobResult Function

Use the `pyqJobResult` function to return the job result.

Syntax

```
FUNCTION PYQSYS.pyqJobResult(
  job_id      VARCHAR2,
  out_fmt     VARCHAR2 DEFAULT 'JSON'
)
RETURN SYS.AnyDataSet
```

Parameters

Parameter	Description
job_id	The ID of the asynchronous job.
out_fmt	The format of the job result. It can be one of the following: <ul style="list-style-type: none"> A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. The Python function must return a pandas.DataFrame, a numpy.ndarray, a tuple, or a list of tuples. The string 'JSON', which specifies that the table returned contains a CLOB that is a JSON string. The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation.

Example

The following example shows a pyqJobResult call and its output.

```
SQL> select * from pyqJobResult(
  job_id => '<job id>',
  out_fmt =>
  '{"NAME":"varchar2(7)","SCORE":"number","FINALGRADE":"number","DIFF":"number"}'
);
```

NAME	SCORE	FINALGRADE	DIFF
Abbott	90	87	3
Branford	92	97	-5
Crandell	81	71	10
Dennison	85	72	13
Edgar	89	80	9
Faust	78	73	5
Greeley	82	91	-9
Hart	84	80	4
Isley	88	86	2
Jasper	91	83	8

10 rows selected.

12.7.3.4 Asynchronous Job Example

The following examples shows how to submit asynchronous jobs with non-XML output and with XML output.

Non-XML Output

When submitting asynchronous jobs, for JSON, PNG and relational outputs, set the `OUT_FMT` argument to `NULL` when submitting the job. When fetching the job result, specify `OUT_FMT` in the `pyqJobResult` call.

This example uses the IRIS table created in the example shown in the [pyqTableEval Function \(Autonomous Database\)](#) topic and the `linregrPredict` script created in the example shown in the [pyqRowEval Function \(Autonomous Database\)](#) topic.

Issue a `pyqGroupEval` function call to submit an asynchronous job. In the function, the `INP_NAM` argument specifies the data in the IRIS table to pass to the function.

The `PAR_LST` argument specifies submitting the job asynchronously with the special control argument `oml_async_flag`, capturing the images rendered in the script with the special control argument `oml_graphics_flag`, passing the input data as a `pandas.DataFrame` with the special control argument `oml_input_type`, along with values for the function arguments `modelName` and `datastoreName`.

The `OUT_FMT` argument is `NULL`.

The `GRP_COL` parameter specifies the column to group by.

The `SCR_NAME` parameter specifies the user-defined Python function stored with the name `linregrPredict` in the script repository.

The asynchronous call returns a job status URL in CLOB, you can call `set long [length]` to get the full URL.

```
set long 150
  select *
  from table(pyqGroupEval(
    inp_nam => 'IRIS',
    par_lst => '{"oml_input_type":"pandas.DataFrame",
              "oml_async_flag":true, "oml_graphics_flag":true,
              "modelName":"linregr", "datastoreName":"pymodel"}',
    out_fmt => NULL,
    grp_col => 'Species',
    ord_col => NULL,
    scr_name => 'linregrPredict',
    scr_owner => NULL
  ));
```

The output is the following:

```
NAME
-----
--
VALUE
-----
--
```

```
https://<oml-cloud-service-location-url>/oml/api/py-scripts/v1/jobs/<job id>
```

1 row selected.

Run a `SELECT` statement that invokes the `pyqJobStatus` function, which returns a resource URL containing the job ID when the job result is ready.

```
select * from pyqJobStatus(
job_id => '<job id>');
```

The output is the following when the job is still pending.

```
NAME
-----
VALUE
-----
job is still running
1 row selected.
```

The output is the following when the job finishes.

```
NAME
-----
--
VALUE
-----
--

https://<oml-cloud-service-location-url>/oml/api/py-scripts/v1/jobs/<job id>/
result

1 row selected.
```

Run a `SELECT` statement that invokes the `pyqJobResult` function.

In the `OUT_FMT` argument, the string `'PNG'` specifies to include both return value and images (titles and image bytes) in the result.

```
column name format a7
column value format a15
column title format a16
column image format a15
select * from pyqJobResult(
    job_id => '<job id>',
    out_fmt => 'PNG'
);
```

The output is the following.

NAME	ID	VALUE	TITLE	IMAGE
GROUP_s etosa	1	[{"Species": "se tosa", "Sepal_Le tal Width	Prediction of Pe	6956424F5277304 B47676F41414141

```

ngth":4.6,"Sepa
l_Width":3.6,"P
etal_Length":1.
0,"Petal_Width"
:0.2,"Pred_Peta
l_Width":0.1325
345443},{ "Speci
es":"setosa","S
4E5355684555674
141416F41414141
486743415941414
1413130647A6B41
41414142484E435
356514943416749
6641686B6941414
141416C7753466C
7A4141415059514
141443245427144
2B6E61514141414
468305256683055
32396D644864686
36D554162574630
634778766447787
0596942325A584A
7A615739754D793
4784C6A49734947

GROUP_v      1 [{"Species":"ve Prediction of Pe 6956424F5277304
ersicol      rsicolor","Sepa tal Width  B47676F414141414
or           l_Length":5.1,"        4E5355684555674
              Sepal_Width":2.    141416F414141414
              5,"Petal_Length     486743415941414
              ":3.0,"Petal_Wi  1413130647A6B41
              dth":1.1,"Pred_   41414142484E435
              Petal_Width":0. 356514943416749
              8319563387},{ "S    6641686B6941414
              pecies":"versic  141416C7753466C
              7A4141415059514
              141443245427144
              2B6E61514141414
              468305256683055
              32396D644864686
              36D554162574630
              634778766447787
              0596942325A584A
              7A615739754D793
              4784C6A49734947

GROUP_v      1 [{"Species":"vi Prediction of Pe 6956424F5277304
irginic      rginica","Sepal tal Width  B47676F414141414
a           _Length":5.7,"S    4E5355684555674
              epal_Width":2.5     141416F414141414
              ,"Petal_Length"    486743415941414
              :5.0,"Petal_Wid  1413130647A6B41
              th":2.0,"Pred_P     41414142484E435
              etal_Width":1.7   356514943416749
              55762924},{ "Spe    6641686B6941414
              cies":"virginic  141416C7753466C
              7A4141415059514
              141443245427144
              2B6E61514141414
              468305256683055
              32396D644864686
              36D554162574630

```



```
634778766447787
0596942325A584A
7A615739754D793
4784C6A49734947
```

3 rows selected.

XML Output

If XML output is expected from the asynchronous job, set the `OUT_FMT` argument to 'XML' when submitting the job and fetching the job result.

This example uses the script `myFitMultiple` created in the example shown in the [pyqIndexEval Function \(Autonomous Database\)](#) topic.

Issue a `pyqIndexEval` function call to submit an asynchronous job. In the function, the `PAR_LST` argument specifies submitting the job asynchronously with the special control argument `oml_async_flag`, along with values for the function arguments `sample_size`.

The asynchronous call returns a job status URL in CLOB, you can call `set long [length]` to get the full URL.

```
set long 150 select *
  from table(pyqIndexEval(
    par_lst => '{"sample_size":80,"oml_async_flag":true}',
    out_fmt => 'XML',
    times_num => 3,
    scr_name => 'myFitMultiple',
    scr_owner => NULL
  ));
```

The output is the following.

```
NAME
-----
--
VALUE
-----
--
https://<oml-cloud-service-location-url>/oml/api/py-scripts/v1/jobs/<job id>
```

1 row selected.

Run a `SELECT` statement that invokes the `pyqJobStatus` function, which returns a resource URL containing the job id when the job result is ready.

```
select * from pyqJobStatus(
  job_id => '<job id>'
);
```

The output is the following when the job is still pending.

```

NAME
-----
VALUE
-----
job is still running
1 row selected.

```

The output is the following when the job result is ready.

```

NAME
-----
--
VALUE
-----
--
https://<oml-cloud-service-location-url>/oml/api/py-scripts/v1/jobs/<job id>/
result

1 row selected.

```

Run a `SELECT` statement that invokes the `pyqJobResult` function.

In the `OUT_FMT` argument, the string `'XML'` specifies that the table returned contains a CLOB that is an XML string.

```

select * from pyqJobResult(
    job_id => '<job id>',
    out_fmt => 'XML'
);

```

The output is the following.

```

NAME
-----
VALUE
-----
1
<root><pandas_dataframe><ROW-pandas_dataframe><id>1</id><score>0.94355
0631313753</score></ROW-pandas_dataframe></pandas_dataframe></root>

2
<root><pandas_dataframe><ROW-pandas_dataframe><id>2</id><score>0.92783
6941437123</score></ROW-pandas_dataframe></pandas_dataframe></root>

3
<root><pandas_dataframe><ROW-pandas_dataframe><id>3</id><score>0.93719
6049031545</score></ROW-pandas_dataframe></pandas_dataframe></root>

3 rows selected.

```

12.7.4 Special Control Arguments (Autonomous Database)

Use the `PAR_LST` parameter to specify special control arguments and additional arguments to be passed into the Python script.

Argument	Syntax and Description
<code>oml_input_type</code>	<p>Syntax <code>oml_input_type</code>: 'pandas.DataFrame', 'numpy.ndarray', or 'default' (default)</p> <p>Description Specifies the type of object to construct from data in the Autonomous Database. By default, a two-dimensional <code>numpy.ndarray</code> of type <code>numpy.float64</code> is constructed if all columns are numeric. Otherwise, a <code>pandas.DataFrame</code> is constructed.</p>
<code>oml_na_omit</code>	<p>Syntax <code>oml_na_omit</code>: <code>bool</code>, <code>false</code> (default)</p> <p>Description Determines if rows with any missing values will be omitted from the table to be evaluated. If <code>true</code>, omit all rows with missing values from the table. If <code>false</code>, do not omit rows with missing values from the table.</p>
<code>oml_async_flag</code>	<p>Syntax <code>oml_async_flag</code>: <code>bool</code>, <code>false</code> (default)</p> <p>Description If <code>true</code>, the job will be submitted asynchronously. If <code>false</code>, the job will be executed in synchronous mode.</p>
<code>oml_graphics_flag</code>	<p>Syntax <code>oml_graphics_flag</code>: <code>bool</code>, <code>false</code> (default)</p> <p>Description If <code>true</code>, the server will capture images rendered in the Python script. If <code>false</code>, the server will not capture images rendered in the Python script.</p>
<code>oml_parallel_flag</code>	<p>Syntax <code>oml_parallel_flag</code>: <code>bool</code>, <code>false</code> (default)</p> <p>Description If <code>true</code>, the Python script will be run with data parallelism. Data parallelism is only applicable to <code>pyqRowEval</code>, <code>pyqGroupEval</code>, and <code>pyqIndexEval</code>. If <code>false</code>, the Python script will not be run with data parallelism.</p>
<code>oml_service_level</code>	<p>Syntax <code>oml_service_level</code>: <code>string</code>, allowed values: 'LOW' (default), 'MEDIUM', 'HIGH'</p> <p>Description Controls the different levels of performance and concurrency in Autonomous Database.</p>

Examples

- Input data is `pandas.DataFrame`:

```
par_lst => '{"oml_input_type":"pandas.DataFrame"}'
```

- Drop rows with missing values from input data:

```
par_lst => '{"oml_na_omit":true}'
```

- Submit a job in asynchronous mode:

```
par_lst => '{"oml_async_flag":true}'
```

- Use MEDIUM service level:

```
par_lst => '{"oml_service_level":"MEDIUM"}'
```

12.7.5 Output Formats (Autonomous Database)

The `OUT_FMT` parameter controls the format of output returned by the table functions `pyqEval`, `pyqGroupEval`, `pyqIndexEval`, `pyqRowEval`, `pyqTableEval`, and `pyqJobResult`.

The output formats are:

- [JSON](#)
- [Relational](#)
- [XML](#)
- [PNG](#)
- [Asynchronous Mode Output](#)

JSON

When `OUT_FMT` is set to `JSON`, the table functions return a table containing a CLOB that is a JSON string.

The following example invokes the `pyqEval` function on the `'pyqFun1'` created in the `pyqEval` function section.

```
SQL> select *
      from table(pyqEval(
                par_lst => '{"oml_service_level":"MEDIUM"}',
                out_fmt => 'JSON',
                scr_name => 'pyqFun1'));
```

NAME

VALUE

```
-----
[{"FLOAT":0,"ID":0,"NAME":"demo_0"}, {"FLOAT":0.1,"ID":1,"NAME":"demo_1"}, {"FLOAT":0.2,"ID":2,"NAME":"demo_2"}, {"FLOAT":0.3,"ID":3,"NAME":"demo_3"}, {"FLOAT":0.4,"ID":4,"NAME":"demo_4"}, {"FLOAT":0.5,"ID":5,"NAME":"demo_5"}, {"FLOAT":0.6,"ID":6,"NAME":"demo_6"}, {"FLOAT":0.7,"ID":7,"NAME":"demo_7"}, {"FLOAT":0.8,"ID":8,"NAME":"demo_8"}, {"FLOAT":0.9,"ID":9,"NAME":"demo_9"}]
```

1 row selected.

Relational

When `OUT_FMT` is specified with a JSON string where column names are mapped to column types, the table functions return the response by reshaping it into table columns.

For example, if `OUT_FMT` is specified with `{"NAME":"varchar2(7)", "DIFF":"number"}`, the output should contain a `NAME` column of type `VARCHAR2(7)` and a `DIFF` column of type `NUMBER`. The following example uses the table `GRADE` and the script `'computeGradeDiff'` (created in [Asynchronous Jobs \(Autonomous Database\)](#)) and invokes the `computeGradeDiff` function:

```
SQL> select *
      from table(pyqTableEval(
            inp_nam => 'GRADE',
            par_lst => '{"oml_input_type":"pandas.DataFrame"}',
            out_fmt => '{"NAME":"varchar2(7)","DIFF":"number"}',
            scr_name => 'computeGradeDiff'));
```

```
NAME DIFF
-----
Abbott    3
Branfor  -5
Crandel  10
Denniso  13
Edgar     9
Faust     5
Greeley  -9
Hart      4
Isley     2
Jasper    8
```

10 rows selected.

XML

When `OUT_FMT` is specified with `XML`, the table functions return the response in a table with fixed columns. The output consists of two columns. The `NAME` column contains the name of the row. The `NAME` column value is `NULL` for `pyqEval`, `pyqTableEval`, `pyqRowEval` function returns. For `pyqGroupEval`, `pyqIndexEval`, the `NAME` column value is the group/index name. The `VALUE` column contains the XML string.

The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation. To include images in the XML string, the special control argument `oml_graphics_flag` must be set to `true`.

In the following code, the python function `gen_two_images` is defined and stored with name `plotTwoImages` in the script repository. The function renders two subplots with random dots in red and blue color and returns the number of columns of the input data.

```
begin
  sys.pyqScriptCreate('plotTwoImages','def gen_two_images (dat):
    import numpy as np
    import matplotlib.pyplot as plt
    np.random.seed(22)
```

```

fig = plt.figure(1);
fig2 = plt.figure(2);
ax = fig.add_subplot(111);
ax.set_title("Random red dots")
ax2 = fig2.add_subplot(111);
ax2.set_title("Random blue dots")
ax.plot(range(100), np.random.normal(size=100), marker = "o",
        color = "red", markersize = 2)
ax2.plot(range(100,0,-1), marker = "o", color = "blue",
markersize = 2)
return dat.shape[1]
',FALSE,TRUE);
end;
/

```

The following example shows the XML output of a `pyqRowEval` function call where both structured data and images are included in the result:

```

SQL> select *
      from table (pyqRowEval (
                inp_nam => 'GRADE',
                par_lst => '{"oml_graphics_flag":true}',
                out_fmt => 'XML',
                row_num => 5,
                scr_name => 'plotTwoImages'
            ));

```

```

NAME
-----
--
VALUE
-----
1
<root><Py-data><int>7</int></Py-data><images><image><![CDATA[iVBORw0KGgoAAAANSUheUgAAAoAAAAHgCAYAAAA1odzKAAA
ABHNCSVQICAgIFahkiAAAAA1wSFlzAAAPYQAAD2EBqD+naQAAADh0RVh0U29mdHdhcmUAb
WF0cGxvdGxpYiB2ZXJzaW9uMy4xLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy81i6FKAAA
gAE1EQVR4nOydeZwcVb32n549k0xCSMhGEohhEZFNUEBE0UUIYOACG4gFxFwVgZqldf3s
lz1xYuKLBe3i7LcNyhctoxsviCJoAQFNAKCLITQyCQbZJMZqb
2
<root><Py-data><int>7</int></Py-data><images><image><![CDATA[iVBORw0KGgoAAAANSUheUgAAAoAAAAHgCAYAAAA1odzKAAA
ABHNCSVQICAgIFahkiAAAAA1wSFlzAAAPYQAAD2EBqD+naQAAADh0RVh0U29mdHdhcmUAb
WF0cGxvdGxpYiB2ZXJzaW9uMy4xLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy81i6FKAAA
gAE1EQVR4nOydeZwcVb32n549k0xCSMhGEohhEZFNUEBE0UUIYOACG4gFxFwVgZqldf3s
lz1xYuKLBe3i7LcNyhctoxsviCJoAQFNAKCLITQyCQbZJMZqb
2 rows selected

```

PNG

When `OUT_FMT` is specified with `PNG`, the table functions return the response in a table with fixed columns (including an image bytes column). When calling the SQL API, you must set the

special control argument `oml_graphics_flag` to `true` so that the web server can capture images rendered in the executed script.

The PNG output consists of four columns. The `NAME` column contains the name of the row. The `NAME` column value is `NULL` for `pyqEval` and `pyqTableEval` function returns. For `pyqRowEval`, `pyqGroupEval`, `pyqIndexEval`, the `NAME` column value is the chunk/group/index name. The `ID` column indicates the ID of the image. The `VALUE` column contains the return value of the executed script. The `TITLE` column contains the titles of the rendered PNG images. The `IMAGE` column is a `BLOB` column containing the bytes of the PNG images rendered by the executed script.

The following example shows the PNG output of a `pyqRowEval` function call.

```
SQL> column name format a7
column valueformat a5
column title format a16
column image format a15
select *
from table(pyqRowEval(
  inp_nam => 'GRADE',
  par_lst => '{"oml_graphics_flag":true}',
  out_fmt => 'PNG',row_num => 5,
  scr_name => 'plotTwoImages',
  scr_owner =>NULL
));
```

NAME	ID	VALUE	TITLE	IMAGE
CHUNK_1	1	7	Random red dots	6956424F5277304 B47676F41414141 4E5355684555674 141416F41414141 486743415941414 1413130647A6B41 41414142484E435 356514943416749 6641686B6941414 141416C7753466C 7A41414150
CHUNK_1	2	7	Random blue dots	6956424F5277304 B47676F41414141 4E5355684555674 141416F41414141 486743415941414 1413130647A6B41 41414142484E435 356514943416749 6641686B6941414 141416C7753466C 7A41414150
CHUNK_2	1	7	Random red dots	6956424F5277304 B47676F41414141 4E5355684555674

```

141416F41414141
486743415941414
1413130647A6B41
41414142484E435
356514943416749
6641686B6941414
141416C7753466C
7A41414150

CHUNK_2          2 7      Random blue dots 6956424F5277304
B47676F41414141
4E5355684555674
141416F41414141
486743415941414
1413130647A6B41
41414142484E435
356514943416749
6641686B6941414
141416C7753466C
7A41414150

```

4 rows selected.

Asynchronous Mode Output

When you set `oml_async_flag` to `true` to run an asynchronous job, set `OUT_FMT` to `NULL` for jobs that return non-XML results, or set it to `XML` for jobs that return XML results, as described below.

See also [oml_async_flag Argument](#).

Asynchronous Mode: Non-XML Output

When submitting asynchronous jobs, for JSON, PNG, and relational outputs, set `OUT_FMT` to `NULL` when submitting the job. When fetching the job result, specify `OUT_FMT` in the `pyqJobResult` call.

The following example shows how to get the JSON output from an asynchronous `pyqIndexEval` function call:

```

SQL> select *
      from table(pyqGroupEval(
            inp_nam => 'GRADE',
            par_lst => '{"oml_async_flag":true, "oml_graphics_flag":true}',
            out_fmt => NULL,
            grp_col => 'GENDER',
            ord_col => NULL,
            scr_name => 'inp_twoimgs',
            scr_owner => NULL
      ));

```

```

NAME
-----
VALUE
-----

```



```
https://<host name>/oml/tenants/<tenant name>/databases/<database
name>/api/py-scripts/v1/jobs/<job id>
```

1 row selected.

```
SQL> select * from pyqJobStatus(
      job_id => '<job id>');
```

NAME

VALUE

```
https://<host name>/oml/tenants/<tenant name>/databases/<database
name>/api/py-scripts/v1/jobs/<job id>/result
```

1 row selected.

```
SQL> column name format a7
      column value format a5
      column title format a16
      column image format a15
select * from pyqJobResult(
      job_id => '<job id>',
      out_fmt => 'PNG'
);
```

NAME	ID	VALUE	TITLE	IMAGE
GROUP_F	1	7	Random red dots	6956424F5277304 B47676F41414141 4E5355684555674 141416F41414141 486743415941414 1413130647A6B41 41414142484E435 356514943416749 6641686B6941414 141416C7753466C 7A4141415059514 141443245427144 2B6E61514141414 468305256683055 32396D644864686 36D554162574630 634778766447787 0596942325A584A 7A615739754D793 4784C6A49734947
GROUP_F	2	7	Random blue dots	6956424F5277304 B47676F41414141

```

4E5355684555674
141416F41414141
486743415941414
1413130647A6B41
41414142484E435
356514943416749
6641686B6941414
141416C7753466C
7A4141415059514
141443245427144
2B6E61514141414
468305256683055
32396D644864686
36D554162574630
634778766447787
0596942325A584A
7A615739754D793
4784C6A49734947

GROUP_M          1 7      Random red dots 6956424F5277304
B47676F41414141
4E5355684555674
141416F41414141
486743415941414
1413130647A6B41
41414142484E435
356514943416749
6641686B6941414
141416C7753466C
7A4141415059514
141443245427144
2B6E61514141414
468305256683855
32396D644864686
36D554162574630
634778766447787
0596942325A584A
7A615739754D793
4784C6A49734947

GROUP_M          2 7      Random blue dots 6956424F5277304
B47676F414141414
4E5355684555674
141416F41414141
486743415941414
1413130647A6B41
41414142484E435
356514943416749
6641686B6941414
141416C7753466C
7A4141415059514
141443245427144
2B6E61514141414
468305256683055
32396D644864686
36D554162574630

```

```
634778766447787
0596942325A584A
7A615739754D793
4784C6A49734947
```

4 rows selected

Asynchronous Mode: XML Output

If XML output is expected from the asynchronous job, you must set `OUT_FMT` to `XML` when submitting the job and fetching the job result.

The following example shows how to get the XML output from an asynchronous `pyqIndexEval` function call.

```
SQL> select *
      from table(pyqIndexEval(
          par_lst => '{"oml_async_flag":true}',
          out_fmt => 'XML',
          times_num => 3,
          scr_name => 'idx_ret_df',
          scr_owner => NULL
      ));
```

```
NAME
-----
--
VALUE
-----
--
https://<host name>/oml/tenants/<tenant name>/databases/<database
name>/api/py-scripts/v1/jobs/<job id>
```

1 row selected.

```
SQL> select * from pyqJobStatus(
      job_id => '<job id>'
    );
```

```
2
NAME
-----
--
VALUE
-----
--
https://<host name>/oml/tenants/<tenant name>/databases/<database
name>/api/py-scripts/v1/jobs/<job id>/result
```

1 row selected.

```
SQL> select * from pyqJobResult(
      job_id => '<job id>',
      out_fmt => 'XML'
);
```

```
  2    3    4
NAME
```

```
-----
--
VALUE
-----
```

```
--
1
<root><pandas_dataFrame><ROW-
pandas_dataFrame><ID>1</ID><RES>a</RES></ROW-pandas
_dataFrame></pandas_dataFrame></root>
```

```
2
<root><pandas_dataFrame><ROW-
pandas_dataFrame><ID>2</ID><RES>b</RES></ROW-pandas
_dataFrame></pandas_dataFrame></ro
```

```
3
<root><pandas_dataFrame><ROW-
pandas_dataFrame><ID>3</ID><RES>c</RES></ROW-pandas
_dataFrame></pandas_dataFrame></root>
```

3 rows selected

13

Administrative Tasks for Oracle Machine Learning for Python

If you find that your Python process is consuming too many of your machine's resources, or causing your machine to crash, you can get information about, or set limits for, the resources Python is using.

The Python system and process utilities library `psutil` is a cross-platform library for retrieving information on running processes and system utilization, such as CPU, memory, disks, network, and sensors, in Python. It is useful for system monitoring, profiling, limiting process resources, and the management of running processes.

The function `psutil.Process.rlimit` gets or sets process resource limits. In `psutil`, process resource limits are `constants` with names beginning with `psutil.RLIMIT_`. Each resource is controlled by a soft limit and hard limit tuple.

For example, `psutil.RLIMIT_AS` represents the maximum size (in bytes) of the virtual memory (address space) used by the process. The default limit of `psutil.RLIMIT_AS` can be `-1` (`psutil.RLIMIT_INFINITY`). You can lower the resource limit of `psutil.RLIMIT_AS` to prevent your Python program from loading too much data into memory, as shown in the following example.

Example 13-1 Resource Control with `psutil.RLIMIT_AS`

```
import psutil
import numpy as np

# Get the current OS process.
p = psutil.Process()

# Get a list of available resources.
[attr for attr in dir(psutil) if attr[:7] == 'RLIMIT_']

# Display the Virtual Memory Size of the current process.
p.memory_info().vms

# Get the process resource limit RLIMIT_AS.
soft, hard = p.rlimit(psutil.RLIMIT_AS)
print('Original resource limits of RLIMIT_AS (soft/hard): {}/{}'.format(soft,
hard))

# Check the constant used to represent the limit for an unlimited resource.
psutil.RLIMIT_INFINITY

# Set resource RLIMIT_AS (soft, hard) limit to (1GB, 2GB).
p.rlimit(psutil.RLIMIT_AS, (pow(1024,3)*1, pow(1024,3)*2))

# Get the current resource limit of RLIMIT_AS.
cur_soft, cur_hard = p.rlimit(psutil.RLIMIT_AS)
print('Current resource limits of RLIMIT_AS (soft/hard): {}/
```

```

{}'.format(cur_soft, cur_hard))

# Define a list of sizes to be allocated in MB (megabytes).
sz = [5, 10, 20]

# Define a megabyte variable in bytes.
MB = 1024*1024

# Allocate an increasing amount of data.
for val in sz:
    stmt = "Allocate %s MB " % val
    try:
        print("virtual memory: %d MB" % int(p.memory_info().vms/MB))
        m = np.arange(val*MB/8, dtype="u8")
        print(stmt + " Success.")
    except:
        print(stmt + " Fail.")
        raise

# Delete the allocated variable.
del m

# Raise the soft limit of RLIMIT_AS to 2GB.
p.rlimit(psutil.RLIMIT_AS, (pow(1024,3)*2, pow(1024,3)*2))

# Get the current resource limit of RLIMIT_AS.
cur_soft, cur_hard = p.rlimit(psutil.RLIMIT_AS)
print('Current resource limits of RLIMIT_AS (soft/hard): {}/
{}'.format(cur_soft, cur_hard))

# Retry: allocate an increasing amount of data.
for val in sz:
    stmt = "Allocate %s MB " % val
    try:
        print("virtual memory: %d MB" % int(p.memory_info().vms/MB))
        m = np.arange(val*MB/8, dtype="u8")
        print(stmt + " Success.")
    except:
        print(stmt + " Fail.")
        raise

```

Listing for This Example

```

>>> import psutil
>>> import numpy as np
>>>
>>> # Get the current OS process.
... p = psutil.Process()
>>>
>>> # Get a list of available resources.
... [attr for attr in dir(psutil) if attr[:7] == 'RLIMIT_']
['RLIMIT_AS', 'RLIMIT_CORE', 'RLIMIT_CPU', 'RLIMIT_DATA',
 'RLIMIT_FSIZE', 'RLIMIT_LOCKS', 'RLIMIT_MEMLOCK', 'RLIMIT_MSGQUEUE',
 'RLIMIT_NICE', 'RLIMIT_NOFILE', 'RLIMIT_NPROC', 'RLIMIT_RSS',
 'RLIMIT_RTPRIO', 'RLIMIT_RTTIME', 'RLIMIT_SIGPENDING', 'RLIMIT_STACK']

```

```

>>>
>>> # Display the Virtual Memory Size of the current process.
... p.memory_info().vms
413175808
>>>
>>> # Get the process resource limit RLIMIT_AS.
... soft, hard = p.rlimit(psutil.RLIMIT_AS)
>>> print('Original resource limits of RLIMIT_AS (soft/hard): {}/
{}'.format(soft, hard))
Original resource limits of RLIMIT_AS (soft/hard): -1/-1
>>>
>>> # Check the constant used to represent the limit for an unlimited
resource.
... psutil.RLIM_INFINITY
-1
>>>
>>> # Set the resource RLIMIT_AS (soft, hard) limit to (1GB, 2GB).
... p.rlimit(psutil.RLIMIT_AS, (pow(1024,3)*1, pow(1024,3)*2))
>>>
>>> # Get the current resource limit of RLIMIT_AS.
... cur_soft, cur_hard = p.rlimit(psutil.RLIMIT_AS)
>>> print('Current resource limits of RLIMIT_AS (soft/hard): {}/
{}'.format(cur_soft, cur_hard))
Current resource limits of RLIMIT_AS (soft/hard): 1073741824/2147483648
>>>
>>> # Define a list of sizes to be allocated in MB (megabytes).
... sz = [100, 200, 500, 1000]
>>>
>>> # Define a megabyte variable in bytes.
... MB = 1024*1024
>>>
>>> # Allocate an increasing amount of data.
... for val in sz:
...     stmt = "Allocate %s MB " % val
...     try:
...         print("virtual memory: %d MB" % int(p.memory_info().vms/MB))
...         m = np.arange(val*MB/8, dtype="u8")
...         print(stmt + " Success.")
...     except:
...         print(stmt + " Fail.")
...         raise
...
virtual memory: 394 MB
Allocate 100 MB Success.
virtual memory: 494 MB
Allocate 200 MB Success.
virtual memory: 594 MB
Allocate 500 MB Fail.
Traceback (most recent call last):
  File "<stdin>", line 6, in <module>
MemoryError
>>>
>>> # Delete the allocated variable.
... del m
>>>
>>> # Raise the soft limit of RLIMIT_AS to 2GB.

```

```
... p.rlimit(psutil.RLIMIT_AS, (pow(1024,3)*2, pow(1024,3)*2))
>>>
>>> # Get the current resource limit of RLIMIT_AS.
... cur_soft, cur_hard = p.rlimit(psutil.RLIMIT_AS)
>>> print('Current resource limits of RLIMIT_AS (soft/hard): {}/
{}'.format(cur_soft, cur_hard))
Current resource limits of RLIMIT_AS (soft/hard): 2147483648/2147483648
>>>
>>> # Retry: allocate an increasing amount of data.
... for val in sz:
...     stmt = "Allocate %s MB " % val
...     try:
...         print("virtual memory: %d MB" % int(p.memory_info().vms/MB))
...         m = np.arange(val*MB/8, dtype="u8")
...         print(stmt + " Success.")
...     except:
...         print(stmt + " Fail.")
...         raise
...
virtual memory: 458 MB
Allocate 100 MB Success.
virtual memory: 558 MB
Allocate 200 MB Success.
virtual memory: 658 MB
Allocate 500 MB Success.
virtual memory: 958 MB
Allocate 1000 MB Success.
```


Glossary

Index

Numerics

3rd party package, [6-13](#)

3rd party packages, [6-9](#)

A

ADMIN, [6-9](#)

algorithm selection class, [10-6](#)

algorithms

Apriori, [9-21](#)

attribute importance, [9-18](#)

Automated Machine Learning, [10-1](#)

Automatic Data Preparation, [9-11](#)

automatically selecting, [10-15](#)

Decision Tree, [9-27](#)

Expectation Maximization, [9-34](#)

Explicit Semantic Analysis, [9-48](#)

Exponential Smoothing, [9-112](#)

Generalized Linear Model, [9-53](#)

k-Means, [9-63](#)

machine learning, [9-2](#)

Minimum Description Length, [9-18](#)

Naive Bayes, [9-69](#)

Neural Network, [9-77](#)

Non-Negative Matrix Factorization, [9-106](#)

Random Forest, [9-86](#)

settings common to all, [9-5](#)

Singular Value Decomposition, [9-94](#)

Support Vector Machine, [9-100](#)

XGBoost, [9-119](#)

ALL_PYQ_DATASTORE_CONTENTS view, [12-7](#)

ALL_PYQ_DATASTORES view, [12-8](#)

ALL_PYQ_SCRIPTS view, [12-10](#)

anomaly detection models, [9-100](#)

Apriori algorithm, [9-21](#)

attribute importance, [9-18](#)

auto model search

automated model search, [1-3](#)

automatic model search, [1-3](#)

Automated Machine Learning

about, [10-1](#)

Automatic Data Preparation algorithm, [9-11](#)

Automatic Machine Learning

connection parameter, [7-2](#)

Autonomous Database, [7-1](#)

C

classes

Automated Machine Learning, [10-1](#)

GlobalFeatureImportance, [9-12](#)

machine learning, [9-2](#)

oml.ai, [9-18](#)

oml.ar, [9-21](#)

oml.automl.AlgorithmSelection, [10-6](#)

oml.automl.FeatureSelection, [10-8](#)

oml.automl.ModelSelection, [10-15](#)

oml.automl.ModelTuning, [10-11](#)

oml.dt, [9-11](#), [9-27](#)

oml.em, [9-34](#)

oml.esa, [9-48](#)

oml.esm, [9-112](#)

oml.glm, [9-53](#)

oml.graphics, [8-40](#)

oml.km, [9-63](#)

oml.nb, [9-69](#)

oml.nn, [9-77](#)

oml.rf, [9-86](#)

oml.svd, [9-94](#)

oml.svm, [9-100](#)

oml.xgb, [9-119](#)

classification algorithm, [9-86](#)

classification and regression algorithm, [9-119](#)

classification and regression models, [9-119](#)

classification models, [9-11](#), [9-27](#), [9-53](#), [9-69](#), [9-77](#), [9-86](#), [9-100](#)

client

installing for Linux for Autonomous Database, [3-1](#)

installing for Linux on-premises, [4-18](#)

Clustering algorithm, [9-112](#)

clustering models, [9-34](#), [9-48](#), [9-63](#)

Clustering models, [9-112](#)

conda environment, [6-9](#)

connection

creating a on-premises database, [7-4](#)

functions, [7-2](#)

control arguments, [12-12](#)

convert Python to SQL, [2-4](#)

creating

proxy objects, [7-13](#), [7-15](#)

cx_Oracle package, [7-2](#)

`cx_Oracle.connect` function, [7-2](#)

D

data

- about moving, [7-9](#)
- exploring, [8-18](#)
- filtering, [8-13](#)
- preparing, [8-1](#)
- selecting, [8-4](#)

Data lineage

- `build_source`
 - query used for build data, [1-3](#)

data parallel processing, [12-12](#)

database

- connecting to an on-premises, [7-4](#)

datastores

- about, [7-20](#)
- database views for, [12-7–12-9](#)
- deleting objects, [7-28](#)
- describing objects in, [7-27](#)
- getting information about, [7-25](#)
- granting or revoking access to, [7-30](#)
- loading objects from, [7-24](#)
- saving objects in, [7-21](#)

Date Types, [8-32](#)

DCLI

- Exadata, [5-5](#)
- python, [5-2](#)

Decision Tree algorithm, [9-27](#)

Distributed Command Line Interface, [5-2](#)

`doc2vec`, [1-2](#)

Download environment from object storage, [6-13](#)

dropping

- tables, [7-15](#)

E

EM model, [9-34](#)

Embedded Python Execution

- about, [12-12](#)
- about the SQL interface for, [12-36](#)
- SQL interface for, [12-35](#), [12-56](#)

ESA embeddings, [1-2](#)

ESA model, [9-48](#)

Exadata, [5-1](#)

- compute nodes, [5-2](#)
- DCLI, [5-5](#)

Expectation Maximization, [1-3](#)

Expectation Maximization algorithm, [9-34](#)

explainability, [9-12](#)

Explicit Semantic Analysis algorithm, [9-48](#)

Exponential Smoothing Model, [1-3](#), [9-112](#)

exporting models, [9-7](#)

F

feature extraction algorithm, [9-48](#)

feature extraction class, [9-94](#)

feature selection class, [10-8](#)

function

- `pyqGrant`, [12-50](#), [12-112](#)

functions

- `cx_Oracle.connect`, [7-2](#)
- Embedded Python Execution, [12-12](#)
- for graphics, [8-40](#)
- for managing user-defined Python functions, [12-28](#)
- `oml.boxplot`, [8-40](#)
- `oml.check_embed`, [7-2](#), [7-4](#)
- `oml.connect`, [7-2](#), [7-4](#)
- `oml.create`, [7-15](#)
- `oml.cursor`, [7-9](#), [7-15](#)
- `oml.dir`, [7-9](#), [7-13](#)
- `oml.disconnect`, [7-2](#), [7-4](#)
- `oml.do_eval`, [12-14](#)
- `oml.drop`, [7-15](#)
- `oml.ds.delete`, [7-28](#)
- `oml.ds.describe`, [7-27](#)
- `oml.ds.dir`, [7-25](#)
- `oml.ds.load`, [7-24](#)
- `oml.ds.save`, [7-21](#)
- `oml.grant`, [7-30](#)
- `oml.group_apply`, [12-18](#)
- `oml.hist`, [8-40](#)
- `oml.index_apply`, [12-25](#)
- `oml.isconnected`, [7-2](#), [7-4](#)
- `oml.row_apply`, [12-22](#)
- `oml.script.create`, [12-28](#)
- `oml.script.dir`, [12-31](#)
- `oml.script.drop`, [12-34](#)
- `oml.script.load`, [12-33](#)
- `oml.set_connection`, [7-2](#)
- `oml.sync`, [7-13](#)
- `oml.table_apply`, [12-15](#)
- `pyqEval`, [12-37](#)
- `pyqGroupEval`, [12-47](#)
- `pyqRowEval`, [12-43](#)
- `pyqTableEval`, [12-40](#)

G

GLM link functions, [1-2](#)

GLM models, [9-53](#)

granting

- access to scripts and datastores, [7-30](#)
- user privileges, [4-12](#)

graphics

- rendering, [8-40](#)

I

- importing models, [9-7](#)
- improved data preparation, [1-3](#)
- installing
 - client for Linux for Autonomous Database, [3-1](#)
 - client for Linux on-premises, [4-18](#)
 - server for Linux on-premises, [4-7](#)
- Instant Client
 - installing for Linux on-premises, [4-17](#)

K

- KM model, [9-63](#)
- KMeans, [1-3](#)

L

- libraries in OML4Py, [2-6](#)
- Linux
 - installing Python for, [4-2](#)
 - requirements, [4-1](#)
 - uninstalling on-premises client for, [4-22](#)
 - uninstalling on-premises server for, [4-15](#)
- Linux for Autonomous Database
 - installing client for, [3-1](#)
- Linux on-premises
 - installing client for, [4-18](#)
 - installing Oracle Instant Client for, [4-17](#)
 - installing server for, [4-7](#)
 - supporting packages for, [4-5](#)

M

- machine learning
 - classes, [9-2](#)
- methods
 - drop, [8-13](#)
 - drop_duplicates, [8-13](#)
 - dropna, [8-13](#)
 - for exploring data, [8-18](#)
 - for preparing data, [8-1](#)
 - pull, [7-11](#)
- Minimum Description Length algorithm, [9-18](#)
- model selection, [10-15](#)
- model tuning, [10-11](#)
- models
 - association rules, [9-21](#)
 - attribute importance, [9-18](#)
 - Clustering, [9-112](#)
 - Decision Tree, [9-11](#), [9-27](#)
 - Expectation Maximization, [9-34](#)
 - explainability, [9-12](#)
 - Explicit Semantic Analysis, [9-48](#)
 - exporting and importing, [9-7](#)
 - for anomaly detection, [9-100](#)

- models (*continued*)
 - for classification, [9-11](#), [9-27](#), [9-53](#), [9-69](#), [9-77](#), [9-86](#), [9-100](#)
 - for classification and regression, [9-119](#)
 - for clustering, [9-34](#), [9-63](#)
 - for Clustering, [9-112](#)
 - for feature extraction, [9-48](#), [9-94](#)
 - for regression, [9-53](#), [9-77](#), [9-100](#)
 - Generalized Linear Model, [9-53](#)
 - k-Means, [9-63](#)
 - Naive Bayes, [9-69](#)
 - Neural Network, [9-77](#)
 - Non-Negative Matrix Factorization, [9-106](#)
 - parametric, [9-53](#)
 - persisting, [9-2](#)
 - Random Forest, [9-86](#)
 - Singular Value Decomposition, [9-94](#)
 - Support Vector Machine, [9-100](#)
 - XGBoost, [9-119](#)
- moving data
 - about, [7-9](#)
 - to a local Python session, [7-11](#)
 - to the database, [7-9](#)

N

- Naive Bayes model, [9-69](#)
- Neural Network model, [9-77](#)
- NMF models, [9-106](#)

O

- ODMS_BOXCOX, [1-3](#)
- oml_input_type argument, [12-12](#)
- oml_na_omit argument, [12-12](#)
- oml.ai class, [9-18](#)
- oml.ar class, [9-21](#)
- oml.automl.AlgorithmSelection class, [10-6](#)
- oml.automl.FeatureSelection class, [10-8](#)
- oml.automl.ModelSelection class, [10-15](#)
- oml.automl.ModelTuning class, [10-11](#)
- oml.boxplot function, [8-40](#)
- oml.check_embed function, [7-2](#), [7-4](#)
- oml.connect function, [7-2](#), [7-4](#)
- oml.create function, [7-15](#)
- oml.cursor function, [7-9](#), [7-15](#)
- oml.Datetime, [8-32](#)
- oml.dir function, [7-9](#), [7-13](#)
- oml.disconnect function, [7-2](#), [7-4](#)
- oml.do_eval function, [12-14](#)
- oml.drop function, [7-15](#)
- oml.ds.delete function, [7-28](#)
- oml.ds.describe function, [7-27](#)
- oml.ds.dir function, [7-25](#)
- oml.ds.load function, [7-24](#)
- oml.ds.save function, [7-21](#)

oml.dt class, [9-11](#), [9-27](#)
 oml.em class, [9-34](#)
 oml.esa class, [9-48](#)
 oml.esm class, [9-112](#)
 oml.glm class, [9-53](#)
 oml.grant function, [7-30](#)
 oml.graphics class, [8-40](#)
 oml.group_apply function, [12-18](#)
 oml.hist function, [8-40](#)
 oml.index_apply function, [12-25](#)
 oml.Integer, [8-32](#)
 oml.isconnected function, [7-2](#), [7-4](#)
 oml.km class, [9-63](#)
 oml.nb class, [9-69](#)
 oml.nn class, [9-77](#)
 oml.push function, [7-9](#)
 oml.revoke function, [7-30](#)
 oml.rf class, [9-86](#)
 oml.row_apply function, [12-22](#)
 oml.script.create function, [12-28](#)
 oml.script.dir function, [12-31](#)
 oml.script.drop function, [12-34](#)
 oml.script.load function, [12-33](#)
 oml.set_connection function, [7-2](#), [7-4](#)
 oml.svd class, [9-94](#)
 oml.svm class, [9-100](#)
 oml.sync function, [7-13](#)
 oml.table_apply function, [12-15](#)
 oml.Timedelta, [8-32](#)
 oml.Timezone, [8-32](#)
 oml.xgb class, [9-119](#)
 OML4Py, [2-1](#), [5-1](#)
 Exadata, [5-5](#)
 on-premises client
 installing, [4-16](#)
 uninstalling, [4-22](#)
 on-premises server
 installing, [4-7](#)
 uninstalling, [4-15](#)
 on-premises system requirements, [4-1](#)
 Oracle Machine Learning Notebooks, [7-1](#)
 Oracle Machine Learning Python interpreter, [7-1](#)
 Oracle wallets
 about, [7-3](#)
 ore.nmf function, [9-106](#)

P

packages
 supporting for Linux on-premises, [4-5](#)
 parallel processing, [12-12](#)
 parametric models, [9-53](#)
 PL/SQL procedures
 sys.pyqScriptCreate, [12-52](#)
 sys.pyqScriptDrop, [12-55](#)
 predict method, [9-69](#)

predict.proba method, [9-69](#)
 privileges
 required, [4-12](#)
 proxy objects, [2-4](#)
 for database tables, [7-13](#), [7-15](#)
 storing, [7-20](#)
 pull method, [7-11](#)
 PYQADMIN role, [4-12](#)
 pyqEval function, [12-37](#)
 pyqGrant function, [12-50](#), [12-112](#)
 pyqGroupEval function, [12-47](#)
 pyqRowEval function, [12-43](#)
 pyqTableEval function, [12-40](#)
 pyquser.sql script, [4-13](#)
 Python, [5-1](#)
 installing for Linux, [4-2](#)
 libraries in OML4Py, [2-6](#)
 requirements, [4-1](#)
 version used, [2-6](#)
 Python interpreter, [7-1](#)
 Python objects
 storing, [7-20](#)
 python packages, [6-9](#)
 Python to SQL conversion, [2-4](#)

R

Random Forest algorithm, [9-86](#)
 ranking
 attribute importance, [9-18](#)
 read privilege
 granting or revoking, [7-30](#)
 regression models, [9-53](#), [9-77](#)
 requirements
 on-premises system, [4-1](#)
 resources
 managing, [13-1](#)
 revoking
 access to scripts and datastores, [7-30](#)
 roles
 PYQADMIN, [4-12](#)

S

scoring new data, [2-2](#), [9-2](#)
 script repository
 granting or revoking access to, [7-30](#)
 managing user-defined Python functions in,
 [12-28](#)
 registering a user-defined function, [12-28](#)
 scripts
 pyquser, [4-13](#)
 server
 installing for Linux on-premises, [4-7](#)
 settings
 about model, [9-4](#)

settings (*continued*)

- Apriori algorithm, [9-21](#)
 - association rules, [9-21](#)
 - Automatic data preparation algorithm, [9-11](#)
 - Decision Tree algorithm, [9-27](#)
 - Expectation Maximization model, [9-34](#)
 - Explicit Semantic Analysis algorithm, [9-48](#)
 - Exponential Smoothing Model, [9-112](#)
 - Generalized Linear Model algorithm, [9-53](#)
 - k-Means algorithm, [9-63](#)
 - Minimum Description Length algorithm, [9-18](#)
 - Naive Bayes algorithm, [9-69](#)
 - Neural Network algorithm, [9-77](#)
 - Random Forest algorithm, [9-86](#)
 - shared algorithm, [9-5](#)
 - Singular Value Decomposition algorithm, [9-94](#)
 - sttribute importance, [9-18](#)
 - Support Vector Machine algorithm, [9-100](#)
 - XGBoost algorithm, [9-119](#)
- special control arguments, [12-12](#)
- SQL APIs
- pyqEval function, [12-37](#)
 - pyqGrant function, [12-50](#), [12-112](#)
 - pyqGroupEval function, [12-47](#)
 - pyqRowEval function, [12-43](#)
 - pyqTableEval function, [12-40](#)
- SQL to Python conversion, [2-4](#)
- supporting packages
- for Linux on-premises, [4-5](#)
- SVD model, [9-94](#)
- SVM models, [9-100](#)
- synchronizing database tables, [7-13](#)
- sys.pyqScriptCreate procedure, [12-52](#)
- sys.pyqScriptDrop procedure, [12-55](#)

T

tables

- creating, [7-15](#)

tables (*continued*)

- dropping, [7-13](#), [7-15](#)
 - proxy objects for, [7-13](#), [7-15](#)
- task parallel processing, [12-12](#)
- transparency layer, [2-4](#)

U

uninstalling

- on-premises client, [4-22](#)
 - on-premises server, [4-15](#)
- USER_PYQ_DATASTORES view, [12-9](#)
- USER_PYQ_SCRIPTS view, [12-11](#)
- user-defined Python functions
- Embedded Python Execution of, [12-12](#)
- users
- creating new, [4-13](#)

V

views

- ALL_PYQ_DATASTORE_CONTENTS, [12-7](#)
- ALL_PYQ_DATASTORES, [12-8](#)
- ALL_PYQ_SCRIPTS, [12-10](#)
- USER_PYQ_DATASTORES, [12-9](#)
- USER_PYQ_SCRIPTS, [12-11](#)

W

wallets

- about Oracle, [7-3](#)
- WINSORIZE, [1-3](#)

X

- XGBoost algorithm, [9-119](#)
- XGBoost constraints, [1-2](#)
- XGBoost survival analysis, [1-2](#)