

Oracle® Machine Learning for Python

Use Cases



Release 2.0
G24181-02
March 2025

ORACLE®

Oracle Machine Learning for Python Use Cases, Release 2.0

G24181-02

Copyright © 2025, 2025, Oracle and/or its affiliates.

Primary Author: Dhanish Kumar

Contributors: Mark Hornick, Sherry Lamonica, Qin Wang, 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

1 Overview

1.1	Machine Learning Overview	1-1
1.1.1	What Is Machine Learning?	1-1
1.1.2	Benefits of Machine Learning	1-2
1.1.3	Define Your Business Problem	1-3
1.1.4	What Do You Want to Do?	1-3
1.1.5	Discover More Through Interfaces	1-4
1.2	Machine Learning Process	1-5
1.2.1	Workflow	1-6
1.2.2	Define Business Goals	1-7
1.2.3	Understand Data	1-8
1.2.4	Prepare Data	1-8
1.2.5	Develop Models	1-9
1.2.6	Evaluate	1-10
1.2.7	Deploy	1-10
1.3	Machine Learning Techniques and Algorithms	1-11
1.3.1	What is a Machine Learning Algorithm	1-11
1.3.2	Supervised Learning	1-11
1.3.3	Unsupervised Learning	1-12

2 Get Started

2.1	Access OML Notebooks	2-1
2.1.1	Access Oracle Machine Learning User Interface	2-1
2.1.2	Create a Notebook from the Example Templates	2-2
2.1.3	Edit Your Notebook Classic	2-3
2.2	Access Autonomous Database	2-5
2.2.1	Provision an Autonomous Database	2-6
2.2.2	Create and Update User Accounts for Oracle Machine Learning Components on Autonomous Database	2-6
2.2.3	Create User	2-6
2.2.4	Add Existing Database User Account to Oracle Machine Learning Components	2-7

3 Use Cases

3.1	Regression Use case	3-1
3.1.1	Load Data	3-2
3.1.1.1	Import Data	3-3
3.1.2	Explore Data	3-5
3.1.2.1	Data Preparation	3-9
3.1.3	Build Model	3-17
3.1.4	Evaluate	3-18
3.2	Classification Use Case	3-22
3.2.1	Load Data	3-24
3.2.2	Explore Data	3-24
3.2.3	Build Model	3-28
3.2.4	Evaluate	3-30
3.3	Clustering Use Case	3-36
3.3.1	Load Data	3-38
3.3.2	Explore Data	3-38
3.3.3	Build Model	3-43
3.3.4	Evaluate	3-45

4 Reference

4.1	About Machine Learning Classes and Algorithms	4-1
4.2	About Model Settings	4-3
4.3	Shared Settings	4-4

Index

1

Overview

- [Machine Learning Overview](#)
Machine learning is a subset of Artificial Intelligence (AI) that focuses on building systems that learn or improve performance based on the data they consume.
- [Machine Learning Process](#)
The lifecycle of a machine learning project is divided into six phases. The process begins by defining a business problem and restating the business problem in terms of a machine learning objective. The end goal of a machine learning process is to produce accurate results for solving your business problem.
- [Machine Learning Techniques and Algorithms](#)
Machine learning problems are categorized into mining techniques. Each machine learning function specifies a class of problems that can be modeled and solved. An algorithm is a mathematical procedure for solving a specific kind of problem.

1.1 Machine Learning Overview

Machine learning is a subset of Artificial Intelligence (AI) that focuses on building systems that learn or improve performance based on the data they consume.

- [What Is Machine Learning?](#)
Machine learning is a technique that discovers previously unknown relationships in data.
- [Benefits of Machine Learning](#)
Machine learning is a powerful technology that can help you find patterns and relationships within your data.
- [Define Your Business Problem](#)
Enterprises face problems such as classifying documents, predicting the financial outcomes, detecting hidden patterns and anomalies, and so on. Machine learning can help solve such problems provided that you have clear understanding of the business problem with enough data and learn to ask the right questions to obtain meaningful results.
- [What Do You Want to Do?](#)
Multiple machine learning techniques, also referred to as "mining function", are available through Oracle Database and Oracle Autonomous Database. Depending on your business problem, you can identify the appropriate mining function, or combination of mining functions, and select the algorithm or algorithms that may best support the solution.
- [Discover More Through Interfaces](#)
Oracle supports programming language interfaces for SQL, R, and Python, and no-code user interfaces such as OML AutoML UI and Oracle Data Miner, and REST model management and deployment through OML Services.

1.1.1 What Is Machine Learning?

Machine learning is a technique that discovers previously unknown relationships in data.

Machine learning and AI are often discussed together. An important distinction is that although all machine learning is AI, not all AI is machine learning. Machine learning automatically searches potentially large stores of data to discover patterns and trends that go beyond simple statistical analysis. Machine learning uses sophisticated algorithms that identify patterns in data creating models. Those models can be used to make predictions and forecasts, and categorize data.

The key features of machine learning are:

- Automatic discovery of patterns
- Prediction of likely outcomes
- Creation of actionable information
- Ability to analyze potentially large volumes of data

Machine learning can answer questions that cannot be addressed through traditional deductive query and reporting techniques.

1.1.2 Benefits of Machine Learning

Machine learning is a powerful technology that can help you find patterns and relationships within your data.

Find trends and patterns - Machine learning discovers hidden information in your data. You might already be aware of important patterns as a result of working with your data over time. Machine learning can confirm or qualify such empirical observations in addition to finding new patterns that are not immediately distinguishable through simple observation. Machine learning can discover predictive relationships that are not causal relationships. For example, machine learning might determine that males with incomes between \$50,000 and \$65,000 who subscribe to certain magazines are likely to buy a given product. You can use this information to help you develop a marketing strategy. Machine learning can handle large volume of data and can be used in financial analysis. Some of the benefits include stock price predictions (algorithmic trading) and portfolio management.

Make data driven decisions - Many companies have big data and extracting meaningful information from that data is important in making data driven business decisions. By leveraging machine learning algorithms, organizations are able to transform data into knowledge and actionable intelligence. With the changing demands, companies are able to make better decisions faster by using machine learning techniques.

Recommend products - Machine learning results can also be used to influence customer decisions by promoting or recommending relevant and useful products based on behavior patterns of customers online or their response to a marketing campaign.

Detect fraud, anomalies, and security risks - The Financial Services sector has benefited from machine learning algorithms and techniques by discovering unusual patterns or fraud and responding to new fraud behaviors much more quickly. Today companies and governments are conducting business and sharing information online. In such cases, network security is a concern. Machine learning can help in detecting anomalous behavior and automatically take corrective actions.

Retail analysis - Machine learning helps to analyze customer purchase patterns to provide promotional offers for target customers. This service ensures superior customer experience and improves customer loyalty.

Healthcare - Machine learning in medical use is becoming common, helping patients and doctors. Advanced machine learning techniques are used in radiology to make an intelligent decision by reviewing images such as radiographs, CT, MRI, PET images, and radiology

reports. It is reported that machine learning-based automatic detection and diagnosis are at par or better than the diagnosis of an actual radiologist. Some of the machine learning applications are trained to detect breast cancer. Another common use of machine learning in the medical field is that of automated billing. Some applications using machine learning can also point out patient's risk for various conditions such as stroke, diabetes, coronary artery diseases, and kidney failures and recommend medication or procedure that may be necessary.

To summarize, machine learning can:

- easily identify trends and patterns
- simplify product marketing and sales forecast
- facilitate early anomaly detection
- minimize manual intervention by "learning"
- handle multidimensional data

1.1.3 Define Your Business Problem

Enterprises face problems such as classifying documents, predicting the financial outcomes, detecting hidden patterns and anomalies, and so on. Machine learning can help solve such problems provided that you have clear understanding of the business problem with enough data and learn to ask the right questions to obtain meaningful results.

You require skills in preparing data, applying ML techniques, and evaluating results. The patterns you find through machine learning may be very different depending on how you formulate the problem. For example, rather than trying to learn how to "improve the response to a direct mail campaign," you might try to find the characteristics of people who have responded to your campaigns in the past. You can then classify if a given profile of a prospect would respond to a direct email campaign.

Many forms of machine learning are predictive. For example, a model can predict income level based on education and other demographic factors. Predictions have an associated probability (How likely is this prediction to be true?). Prediction probabilities are also known as confidence (How confident can I be of this prediction?). Some forms of predictive machine learning generate rules, which are conditions that imply a given outcome. For example, a rule can specify that a person who has a bachelor's degree and lives in a certain neighborhood is likely to have an income greater than the regional average. Rules have an associated support (What percentage of the population satisfies the rule?).

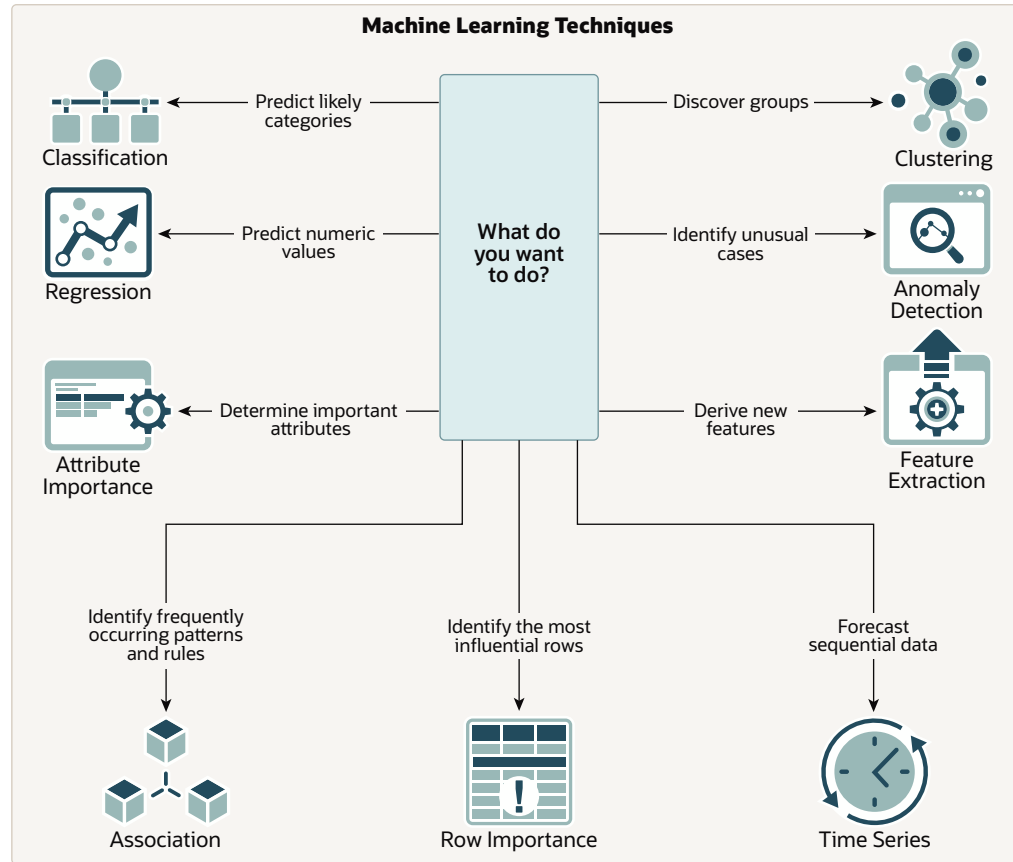
Other forms of machine learning identify groupings in the data. For example, a model might identify the segment of the population that has an income within a specified range, that has a good driving record, and that leases a new car on a yearly basis.

1.1.4 What Do You Want to Do?

Multiple machine learning techniques, also referred to as "mining function", are available through Oracle Database and Oracle Autonomous Database. Depending on your business problem, you can identify the appropriate mining function, or combination of mining functions, and select the algorithm or algorithms that may best support the solution.

For some mining functions, you can choose from among multiple algorithms. For specific problems, one technique or algorithm may be a better fit than the other or more than one algorithm can be used to solve the problem.

The following diagram provides a basic idea on how to select machine learning techniques that are available across Oracle Database and Oracle Autonomous Database.

Figure 1-1 Machine Learning Techniques

OML provides machine learning capabilities within Oracle Database by offering a broad set of in-database algorithms to perform a variety of machine learning techniques such as Classification, Regression, Clustering, Feature Extraction, Anomaly Detection, Association (Market Basket Analysis), and Time Series. Others include Attribute Importance, Row Importance, and Ranking. OML uses built-in features of Oracle Database to maximize scalability, improved memory, and performance. OML is also integrated with open source languages such as Python and R. Through the use of open source packages from R and Python, users can extend this set of techniques and algorithms in combination with embedded execution from OML4Py and OML4R.

1.1.5 Discover More Through Interfaces

Oracle supports programming language interfaces for SQL, R, and Python, and no-code user interfaces such as OML AutoML UI and Oracle Data Miner, and REST model management and deployment through OML Services.

Oracle Machine Learning Notebooks (OML Notebooks) is based on Apache Zeppelin technology enabling you to perform machine learning in Oracle Autonomous Database (Autonomous Data Warehouse (ADW), Autonomous Transactional Database (ATP), and Autonomous JSON Database (AJD)). OML Notebooks helps users explore, visualize, and prepare data, and develop and document analytical methodologies.

AutoML User Interface (AutoML UI) is an Oracle Machine Learning interface that provides you no-code automated machine learning. When you create and run an experiment in AutoML UI, it automatically performs algorithm and feature selection, as well as model tuning and selection,

thereby enhancing productivity as well as model accuracy and performance. Business users without extensive data science background can use AutoML UI to create and deploy machine learning models.

Oracle Machine Learning Services (OML Services) extends OML functionality to support model deployment and model lifecycle management for both in-database OML models and third-party Open Neural Networks Exchange (ONNX) format machine learning models through REST APIs. The REST API for Oracle Machine Learning Services provides REST API endpoints hosted on Oracle Autonomous Database. These endpoints enable you to store machine learning models along with its metadata, and create scoring endpoints for the model.

Oracle Machine Learning for Python (OML4Py) enables you to run Python commands and scripts for data transformations and for statistical, machine learning, and graphical analysis on data stored in or accessible through Oracle Autonomous Database service using a Python API. 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 users can 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. Users can use Embedded Python Execution to run user-defined Python functions in Python engines spawned by the Autonomous Database environment.

Oracle Machine Learning for R (OML4R) provides a database-centric environment for end-to-end analytical processes in R, with immediate deployment of user-defined R functions to production environments. OML4R is a set of R packages and Oracle Database features that enable an R user to operate on database-resident data without using SQL and to run user-defined R functions, also referred to as "scripts", in one or more database-controlled R engines. OML4R is included with Oracle Database and Oracle Database Cloud Service.

Oracle Machine Learning for SQL (OML4SQL) provides SQL access to powerful, in-database machine learning algorithms. You can use OML4SQL to build and deploy predictive and descriptive machine learning models that can add intelligent capabilities to applications and dashboards. OML4SQL is included with Oracle Database, Oracle Database Cloud Service, and Oracle Autonomous Database.

Oracle Data Miner (ODMr) is an extension to Oracle SQL Developer. Oracle Data Miner is a graphical user interface to discover hidden patterns, relationships, and insights in data. ODMr provides a drag-and-drop workflow editor to define and capture the steps that users take to explore and prepare data and apply machine learning technology.

1.2 Machine Learning Process

The lifecycle of a machine learning project is divided into six phases. The process begins by defining a business problem and restating the business problem in terms of a machine learning objective. The end goal of a machine learning process is to produce accurate results for solving your business problem.

- [Workflow](#)
The machine learning process workflow illustration is based on the CRISP-DM methodology. Each stage in the workflow is illustrated with points that summarize the key tasks. The CRISP-DM methodology is the most commonly used methodology for machine learning.
- [Define Business Goals](#)
The first phase of machine learning process is to define business objectives. This initial phase of a project focuses on understanding the project objectives and requirements.

- **Understand Data**
The data understanding phase involves data collection and exploration which includes loading the data and analyzing the data for your business problem.
- **Prepare Data**
The preparation phase involves finalizing the data and covers all the tasks involved in making the data in a format that you can use to build the model.
- **Develop Models**
In this phase, you select and apply various modeling techniques and tune the algorithm parameters, called *hyperparameters*, to desired values.
- **Evaluate**
At this stage of the project, it is time to evaluate how well the model satisfies the originally-stated business goal.
- **Deploy**
Deployment is the use of machine learning within a target environment. In the deployment phase, one can derive data driven insights and actionable information.

1.2.1 Workflow

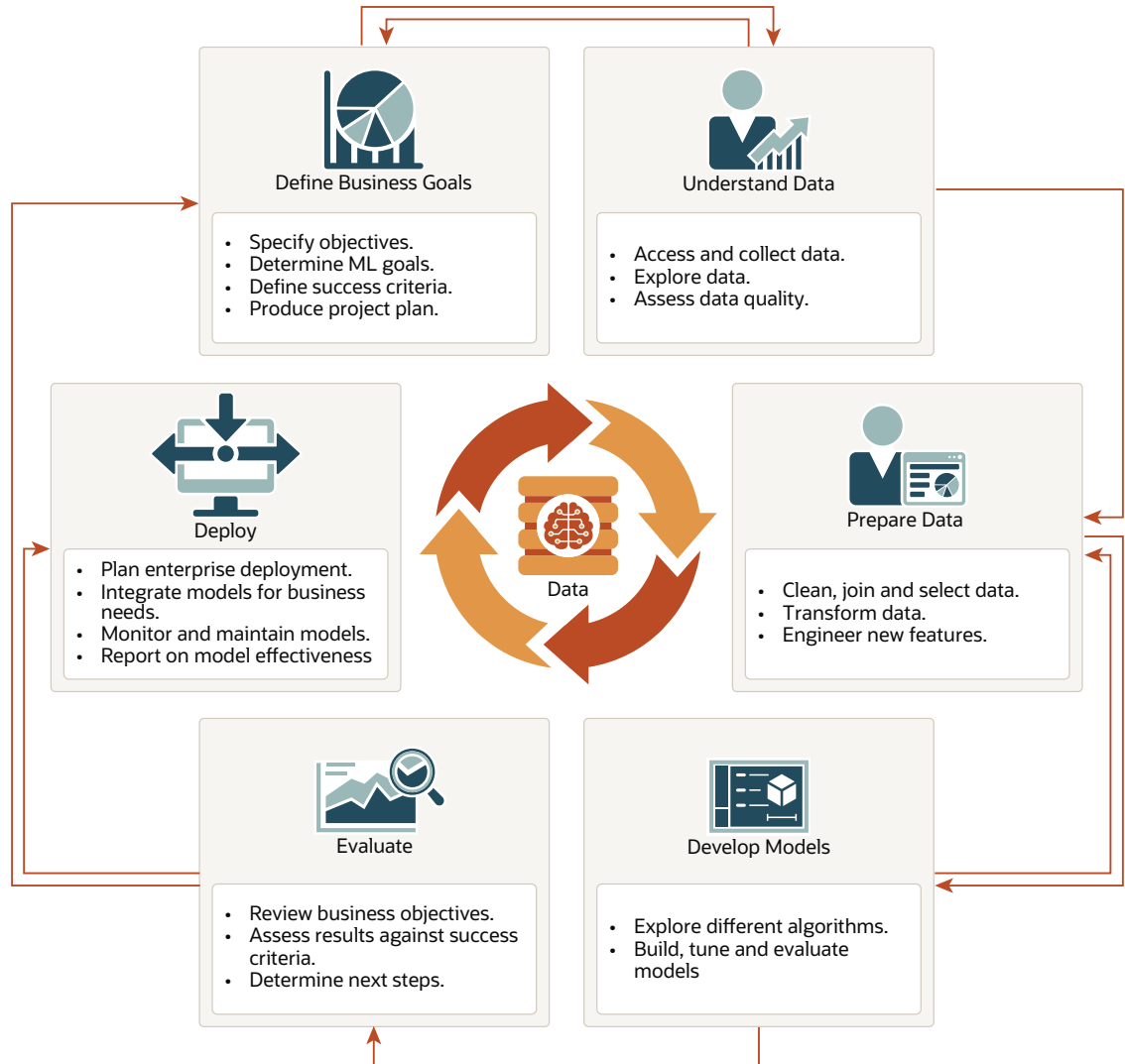
The machine learning process workflow illustration is based on the CRISP-DM methodology. Each stage in the workflow is illustrated with points that summarize the key tasks. The CRISP-DM methodology is the most commonly used methodology for machine learning.

The following are the phases of the machine learning process:

- Define business goals
- Understand data
- Prepare data
- Develop models
- Evaluate
- Deploy

Each of these phases are described separately. The following figure illustrates machine learning process workflow.

Figure 1-2 Machine Learning Process Workflow



Related Topics

- <https://www.datasciencecentral.com/profiles/blogs/crisp-dm-a-standard-methodology-to-ensure-a-good-outcome>
- <https://www.sv-europe.com/crisp-dm-methodology/>

1.2.2 Define Business Goals

The first phase of machine learning process is to define business objectives. This initial phase of a project focuses on understanding the project objectives and requirements.

Once you have specified the problem from a business perspective, you can formulate it as a machine learning problem and develop a preliminary implementation plan. Identify success criteria to determine if the machine learning results meet the business goals defined. For example, your business problem might be: "How can I sell more of my product to customers?" You might translate this into a machine learning problem such as: "Which customers are most likely to purchase the product?" A model that predicts who is most likely to purchase the

product is typically built on data that describes the customers who have purchased the product in the past.

To summarize, in this phase, you will:

- Specify objectives
- Determine machine learning goals
- Define success criteria
- Produce project plan

1.2.3 Understand Data

The data understanding phase involves data collection and exploration which includes loading the data and analyzing the data for your business problem.

Assess the various data sources and formats. Load data into appropriate data management tools, such as Oracle Database. Explore relationships in data so it can be properly integrated. Query and visualize the data to address specific data mining questions such as distribution of attributes, relationship between pairs or small number of attributes, and perform simple statistical analysis. As you take a closer look at the data, you can determine how well it can be used to address the business problem. You can then decide to remove some of the data or add additional data. This is also the time to identify data quality problems such as:

- Is the data complete?
- Are there missing values in the data?
- What types of errors exist in the data and how can they be corrected?

To summarize, in this phase, you will:

- Access and collect data
- Explore data
- Assess data quality

1.2.4 Prepare Data

The preparation phase involves finalizing the data and covers all the tasks involved in making the data in a format that you can use to build the model.

Data preparation tasks are likely to be performed multiple times, iteratively, and not in any prescribed order. Tasks can include column (attributes) selection as well as selection of rows in a table. You may create views to join data or materialize data as required, especially if data is collected from various sources. To cleanse the data, look for invalid values, foreign key values that don't exist in other tables, and missing and outlier values. To refine the data, you can apply transformations such as aggregations, normalization, generalization, and attribute constructions needed to address the machine learning problem. For example, you can transform a `DATE_OF_BIRTH` column to `AGE`; you can insert the median income in cases where the `INCOME` column is null; you can filter out rows representing outliers in the data or filter columns that have too many missing or identical values.

Additionally you can add new computed attributes in an effort to tease information closer to the surface of the data. This process is referred as *Feature Engineering*. For example, rather than using the purchase amount, you can create a new attribute: "Number of Times Purchase Amount Exceeds \$500 in a 12 month time period." Customers who frequently make large purchases can also be related to customers who respond or don't respond to an offer.

Thoughtful data preparation and feature engineering that capture domain knowledge can significantly improve the patterns discovered through machine learning. Enabling the data professional to perform data assembly, data preparation, data transformations, and feature engineering inside the Oracle Database is a significant distinction for Oracle.

**Note:**

Oracle Machine Learning supports Automatic Data Preparation (ADP), which greatly simplifies the process of data preparation.

To summarize, in this phase, you will:

- Clean, join, and select data
- Transform data
- Engineer new features

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

1.2.5 Develop Models

In this phase, you select and apply various modeling techniques and tune the algorithm parameters, called *hyperparameters*, to desired values.

If the algorithm requires specific data transformations, then you need to step back to the previous phase to apply them to the data. For example, some algorithms allow only numeric columns such that string categorical data must be "exploded" using one-hot encoding prior to modeling. In preliminary model building, it often makes sense to start with a sample of the data since the full data set might contain millions or billions of rows. Getting a feel for how a given algorithm performs on a subset of data can help identify data quality issues and algorithm setting issues sooner in the process reducing time-to-initial-results and compute costs. For supervised learning problem, data is typically split into train (build) and test data sets using an 80-20% or 60-40% distribution. After splitting the data, build the model with the desired model settings. Use default settings or customize by changing the model setting values. Settings can be specified through OML's PL/SQL, R and Python APIs. Evaluate model quality through metrics appropriate for the technique. For example, use a confusion matrix, precision, and recall for classification models; RMSE for regression models; cluster similarity metrics for clustering models and so on.

Automated Machine Learning (AutoML) features may also be employed to streamline the iterative modeling process, including algorithm selection, attribute (feature) selection, and model tuning and selection.

To summarize, in this phase, you will:

- Explore different algorithms
- Build, evaluate, and tune models

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

1.2.6 Evaluate

At this stage of the project, it is time to evaluate how well the model satisfies the originally-stated business goal.

During this stage, you will determine how well the model meets your business objectives and success criteria. If the model is supposed to predict customers who are likely to purchase a product, then does it sufficiently differentiate between the two classes? Is there sufficient lift? Are the trade-offs shown in the confusion matrix acceptable? Can the model be improved by adding text data? Should transactional data such as purchases (market-basket data) be included? Should costs associated with false positives or false negatives be incorporated into the model?

It is useful to perform a thorough review of the process and determine if important tasks and steps are not overlooked. This step acts as a quality check based on which you can determine the next steps such as deploying the project or initiate further iterations, or test the project in a pre-production environment if the constraints permit.

To summarize, in this phase, you will:

- Review business objectives
- Assess results against success criteria
- Determine next steps

1.2.7 Deploy

Deployment is the use of machine learning within a target environment. In the deployment phase, one can derive data driven insights and actionable information.

Deployment can involve scoring (applying a model to new data), extracting model details (for example the rules of a decision tree), or integrating machine learning models within applications, data warehouse infrastructure, or query and reporting tools.

Because Oracle Machine Learning builds and applies machine learning models inside Oracle Database, the results are immediately available. Reporting tools and dashboards can easily display the results of machine learning. Additionally, machine learning supports scoring single cases or records at a time with dynamic, batch, or real-time scoring. Data can be scored and the results returned within a single database transaction. For example, a sales representative can run a model that predicts the likelihood of fraud within the context of an online sales transaction.

To summarize, in this phase, you will:

- Plan enterprise deployment
- Integrate models with application for business needs
- Monitor, refresh, retire, and archive models
- Report on model effectiveness

Related Topics

- *Oracle Machine Learning for SQL User's Guide*

1.3 Machine Learning Techniques and Algorithms

Machine learning problems are categorized into mining techniques. Each machine learning function specifies a class of problems that can be modeled and solved. An algorithm is a mathematical procedure for solving a specific kind of problem.

- **What is a Machine Learning Algorithm**
An algorithm is a mathematical procedure for solving a specific kind of problem. For some machine learning techniques, you can choose among several algorithms.
- **Supervised Learning**
Supervised learning is also known as directed learning. The learning process is directed by a previously known dependent attribute or target.
- **Unsupervised Learning**
Unsupervised learning is non-directed. There is no distinction between dependent and independent attributes. There is no previously-known result to guide the algorithm in building the model.

1.3.1 What is a Machine Learning Algorithm

An algorithm is a mathematical procedure for solving a specific kind of problem. For some machine learning techniques, you can choose among several algorithms.

Each algorithm produces a specific type of model, with different characteristics. Some machine learning problems can best be solved by using more than one algorithm in combination. For example, you might first use a feature extraction model to create an optimized set of predictors, then a classification model to make a prediction on the results.

1.3.2 Supervised Learning

Supervised learning is also known as directed learning. The learning process is directed by a previously known dependent attribute or target.

Supervised machine learning attempts to explain the behavior of the target as a function of a set of independent attributes or predictors. Supervised learning generally results in predictive models.

The building of a supervised model involves training, a process whereby the software analyzes many cases where the target value is already known. In the training process, the model "learns" the patterns in the data that enable making predictions. For example, a model that seeks to identify the customers who are likely to respond to a promotion must be trained by analyzing the characteristics of many customers who are known to have responded or not responded to a promotion in the past.

Oracle Machine Learning supports the following supervised machine learning functions:

Table 1-1 Supervised Machine Learning Functions

Function	Description	Sample Problem	Supported Algorithms
Feature Selection or Attribute Importance	Identifies the attributes that are most important in predicting a target attribute	Given customer response to an affinity card program, find the most significant predictors	<ul style="list-style-type: none"> • cur Matrix Decomposition • Expectation Maximization • Minimum Description Length
Classification	Assigns items to discrete classes and predicts the class to which an item belongs	Given demographic data about a set of customers, predict customer response to an affinity card program	<ul style="list-style-type: none"> • Decision Tree • Explicit Semantic Analysis • XGBoost • Generalized Linear Model • Naive Bayes • Neural Network • Random Forest • Support Vector Machine
Regression	Approximates and forecasts continuous values	Given demographic and purchasing data about a set of customers, predict customers' age	<ul style="list-style-type: none"> • XGBoost • Generalized Linear Model • Neural Network • Support Vector Machine
Ranking	Predicts the probability of one item over other items	Recommend products to online customers based on their browsing history	XGBoost
Time Series	Forecasts target value based on known history of target values taken at equally spaced points in time	Predict the length of the ocean waves, address tactical issues such as projecting costs, inventory requirements and customer satisfaction, and so on.	Exponential Smoothing

1.3.3 Unsupervised Learning

Unsupervised learning is non-directed. There is no distinction between dependent and independent attributes. There is no previously-known result to guide the algorithm in building the model.

Unsupervised learning can be used for descriptive purposes. In unsupervised learning, the goal is pattern detection. It can also be used to make predictions.

Oracle Machine Learning supports the following unsupervised machine learning functions:

Table 1-2 Unsupervised Machine Learning Functions

Function	Description	Sample Problem	Supported Algorithms
Anomaly Detection	Identifies rows (cases, examples) that do not satisfy the characteristics of "normal" data	Given demographic data about a set of customers, identify which customer purchasing behaviors are unusual in the dataset, which may be indicative of fraud.	<ul style="list-style-type: none"> • One-Class SVM • Multivariate State Estimation Technique - Sequential Probability Ratio Test
Association	Finds items that tend to co-occur in the data and specifies the rules that govern their co-occurrence	Find the items that tend to be purchased together and specify their relationship	Apriori
Clustering	Finds natural groupings in the data	Segment demographic data into clusters and rank the probability that an individual belongs to a given cluster	<ul style="list-style-type: none"> • Expectation Maximization • k-Means • O-Cluster
Feature Extraction	Creates new attributes (features) using linear combinations of the original attributes	Given demographic data about a set of customers, transform the original attributes into fewer new attributes.	<ul style="list-style-type: none"> • Explicit Semantic Analysis • Non-Negative Matrix Factorization • PCA scoring • Singular Value Decomposition
Row Importance	Row importance technique is used in dimensionality reduction of large data sets. Row importance identifies the most influential rows of the data set.	Given a data set, select rows that meet a minimum importance value prior to model building.	cur Matrix Decomposition

2

Get Started

- [Access OML Notebooks](#)
To perform Oracle Machine Learning tasks, you can access Oracle Machine Learning Notebooks from Autonomous Database
- [Access Autonomous Database](#)
Oracle Autonomous Database is a family of self-driving, self-securing, and self-repairing cloud services. You can sign up for an Oracle Cloud Free Tier account and create a database instance.

2.1 Access OML Notebooks

To perform Oracle Machine Learning tasks, you can access Oracle Machine Learning Notebooks from Autonomous Database

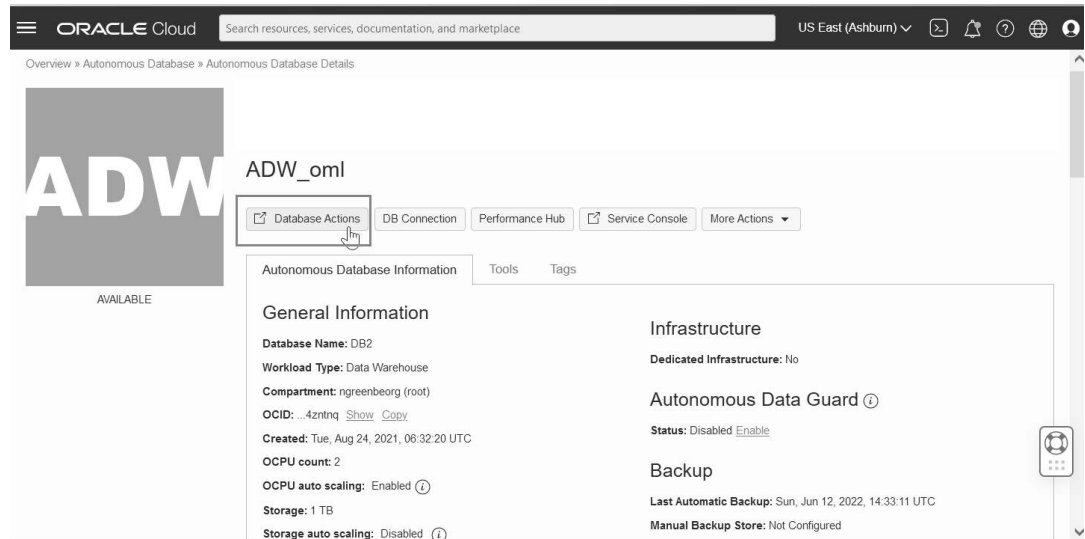
- [Access Oracle Machine Learning User Interface](#)
You can access Oracle Machine Learning **User Interface** from Autonomous Database.
- [Create a Notebook from the Example Templates](#)
Using the Oracle Machine Learning Example Templates, you can create a notebook from the available templates.
- [Edit Your Notebook Classic](#)
Upon creating an OML Notebook Classic, it opens automatically, presenting you with a single paragraph using the default `%sql` interpreter. You can change the interpreter by explicitly specifying one of `%script`, `%python`, `%sql`, `%r`, `%md` or `%conda`.

2.1.1 Access Oracle Machine Learning User Interface

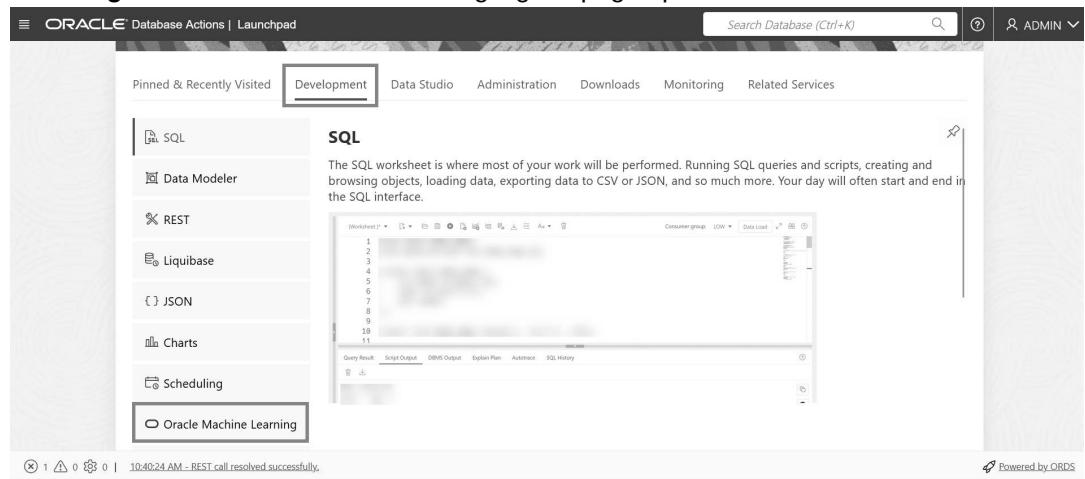
You can access Oracle Machine Learning **User Interface** from Autonomous Database.

To access Oracle Machine Learning User Interface (UI) from the Autonomous Database:

1. Select your Autonomous Database instance and on the Autonomous Database details page click **Database Actions**.



2. On the Database Actions page, go to the **Development** section and click **Oracle Machine Learning**. The Oracle Machine Learning sign in page opens.



3. On the Oracle Machine Learning sign in page, enter your username and password.
4. Click **Sign In**.

This opens the Oracle Machine Learning user application.


2.1.2 Create a Notebook from the Example Templates

Using the Oracle Machine Learning Example Templates, you can create a notebook from the available templates.

To create a notebook:

1. On the Example Templates page, select the template based on which you want to create a notebook.
2. Click **New Notebook**.
The Create Notebook dialog box opens.
3. In the **Create Notebook** dialog, the name of the selected template appears. In the **Name** field, you can change the notebook name.

The screenshot shows the Oracle Machine Learning interface. On the left, under 'Example Templates', there are two notebook templates: 'OML4Py Classification NB' and 'OML4Py Clustering EM'. The 'OML4Py Classification NB' template is selected, showing its details: 'This notebook builds and applies a Naïve Bayes C...', 'Author: Oracle', 'Date Added: 1/17/22 5:35 AM', and 'Tags: '19c' '21c' 'Python' 'Classification' 'Split' 'SH...'. A 'Create Notebook' button is visible. On the right, the 'Create Notebook' dialog box is open. It contains the following fields: 'Name' with the value 'OML4Py Classification NB (1)', 'Comment' with the value 'Copy of OML4Py classification notebook', 'Project' with the value 'OMLUSER Project [OMLUSER Workspace]' and an edit icon, and 'Connection' with the value 'Global'. At the bottom right of the dialog are 'Cancel' and 'OK' buttons.

4. In the **Comment** field, if any comment is available for the template, then it is displayed. You can edit the comment.
5. In the **Project** field, click the edit icon .
6. Select the project in which you want to save the notebook.
7. In the **Connection** field, the default connection is selected.
8. Click **OK**.

The notebook is created and is available on the Notebooks page.

2.1.3 Edit Your Notebook Classic

Upon creating an OML Notebook Classic, it opens automatically, presenting you with a single paragraph using the default `%sql` interpreter. You can change the interpreter by explicitly specifying one of `%script`, `%python`, `%sql`, `%r`, `%md` or `%conda`.

Set the context with a project with which your notebook is associated.

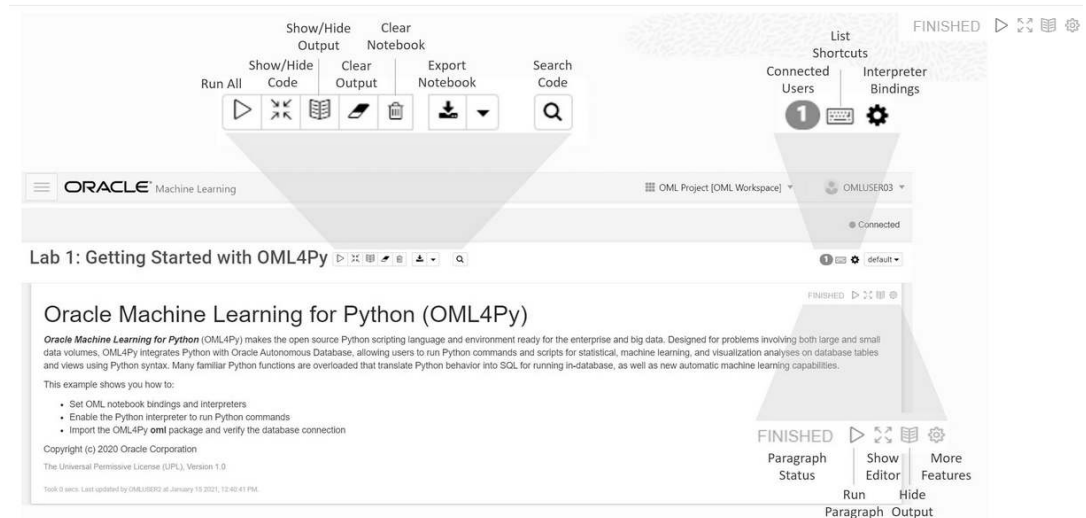
You can edit an existing Notebook Classic in your project. To edit an existing Notebook Classic:

1. On Oracle Machine Learning UI home page, select the project in which your notebook is available.
2. Go to the Oracle Machine Learning UI navigator, and select **Notebooks Classic**. All notebooks that are available in the project are listed.
3. Click the notebook that you want to open and edit.

The selected notebook opens in edit mode.


4. In the edit mode, you can use the Oracle Machine Learning Notebooks Classic toolbar options to run code in paragraphs, for configuration settings, and display options.


Figure 2-1 Notebook toolbar





You can perform the following tasks:


- Write code to fetch data


- Click  to run one or all paragraphs in the notebook.

- Click  to hide all codes from all the paragraphs in the notebook. Click it again to display the codes.


- Click  to hide all outputs from all the paragraphs in the notebook. Click it again to view the outputs.


- Click  to remove all outputs from all the paragraphs in the notebook. To view the output, click the run icon again.


- Click  to delete all the paragraphs in the notebook.



- Click  to export the notebook.

- Click  to search any information in the codes present in the notebook.

- Click  to view the list of keyboard shortcuts.

- Click  to set the order for interpreter bindings for the notebook.



- Click  to select one of the three notebook display options.
 - Click **default** to view the codes, output, and metadata in all paragraphs in the notebook.
 - Click **Simple** to view only the code and output in all paragraphs in the notebook. In this view, the notebook toolbar and all edit options are hidden. You must hover your mouse to view the edit options.
 - Click **Report** to view only the output in all paragraphs in the notebook.
- Click  to access paragraph specific edit options such as clear output, remove paragraph, adjust width, font size, run all paragraphs above or below the selected paragraph and so on.
- Add dynamic forms such as the Text Input form, Select form, Check box form for easy selection of inputs and easy filtering of data in your notebook. Oracle Machine Learning supports the following Apache Zeppelin dynamic forms:
 - Text Input form — Allows you to create a simple form for text input.
 - Select form — Allows you to create a form containing a range of values that the user can select.
 - Check Box form — Allows you to insert check boxes for multiple selection of inputs.

 **Note:**

The Apache Zeppelin dynamic forms are supported only on SQL interpreter notebooks.

5. Once you have finished editing the notebook, click **Back**.

This takes you back to the Notebooks Classic page.

2.2 Access Autonomous Database

Oracle Autonomous Database is a family of self-driving, self-securing, and self-repairing cloud services. You can sign up for an Oracle Cloud Free Tier account and create a database instance.

- [Provision an Autonomous Database](#)
A LiveLabs workshop (a set of labs) that teaches you to manage and monitor Autonomous Database (ADB) is available. A part of the workshop aims to provision an Autonomous Database instance on Oracle Cloud.
- [Create and Update User Accounts for Oracle Machine Learning Components on Autonomous Database](#)
An administrator can add an existing database user account to use with Oracle Machine Learning components or create a new user account and user credentials with the Oracle Machine Learning User Management interface.

- [Create User](#)
An administrator creates new user accounts and user credentials for Oracle Machine Learning in the User Management interface.
- [Add Existing Database User Account to Oracle Machine Learning Components](#)
As the ADMIN user you can add an existing database user account for Oracle Machine Learning components.

2.2.1 Provision an Autonomous Database

A LiveLabs workshop (a set of labs) that teaches you to manage and monitor Autonomous Database (ADB) is available. A part of the workshop aims to provision an Autonomous Database instance on Oracle Cloud.

[Manage and Monitor Autonomous Database](#)

2.2.2 Create and Update User Accounts for Oracle Machine Learning Components on Autonomous Database

An administrator can add an existing database user account to use with Oracle Machine Learning components or create a new user account and user credentials with the Oracle Machine Learning User Management interface.

2.2.3 Create User

An administrator creates new user accounts and user credentials for Oracle Machine Learning in the User Management interface.



Note:

You must have the administrator role to access the Oracle Machine Learning User Management interface.

To create a user account:

1. On the Autonomous Databases page, under the **Display Name**, select an Autonomous Database.
2. On the Autonomous Database Details page, select **Database Actions** and click **Database users**.

As an alternative, select Database Actions and click **View all database actions** to access the Database Actions Launchpad. From the Database Actions launchpad, under **Administration** click **Database Users**.
3. Click **+ Create User**.
4. In the **User Name** field, enter a username for the account. Using the username, the user will log in to an Oracle Machine Learning instance.
5. (Optional) Select the option **Password Expired (user must change)**, to required the user to change their password when they login for the first time.
6. In the **Password** field, enter a password for the user.
7. In the **Confirm Password** field, enter a password to confirm the value that you entered in the **Password** field.

8. Select **OML** to enable Oracle Machine Learning for the user.
9. Click **Create User**.

This creates a new database user and grants the required privileges to use Oracle Machine Learning.

 **Note:**

With a new database user, an administrator needs to issue grant commands on the database to grant table access to the new user for the tables associated with the user's Oracle Machine Learning notebooks.


2.2.4 Add Existing Database User Account to Oracle Machine Learning Components

As the ADMIN user you can add an existing database user account for Oracle Machine Learning components.

 **Note:**

You must have the ADMIN role to access the Oracle Machine Learning User Management interface.

To add an existing database user account:

1. On the Autonomous Databases page, under the **Display Name** column, select an Autonomous Database.
2. On the Autonomous Database Details page, select **Database Actions** and click **View all database actions**.
3. On the Database Actions Launchpad, under **Development**, click **Oracle Machine Learning**.
4. Expand the navigator by clicking the  next to Oracle Machine Learning.
5. Under Admin, select **Manage OML Users** to add Oracle Machine Learning Notebooks users.
6. Click **Show All Users** to display the existing database users.

ORACLE[®] Machine Learning User Administration

Users

[+ Create](#) [X Delete](#) ☒ Show All Users

User Name	Full Name	Role	Email	Created On	Status
ADMIN		System Administrator		8/26/18 7:30 PM	Open
ANALYST1		Developer		10/1/18 10:03 PM	Open

 **Note:**

Initially, the **Role** field shows the role **None** for existing database users. After adding a user the role **Developer** is assigned to the user.

7. Select a user. To select a user select a name in the **User Name** column. For example, select **ANALYST1**.

Selecting the user shows the Oracle Machine Learning **Edit User** page.

8. Enter a name in the **First Name** field. (Optional)
9. Enter the last name of the user in the **Last Name** field. (Optional)
10. In the **Email Address** field, enter the email ID of the user.

Making any change on this page adds the existing database user with the required privileges as an Oracle Machine Learning component user.

11. Click **Save**.

This grants the required privileges to use the Oracle Machine Learning application. In Oracle Machine Learning this user can then access any tables the user has privileges to access in the database.

3

Use Cases

- [Regression Use case](#)
The Brooklyn housing dataset contains the sale prices of homes in brooklyn borough, along with various factors that influence these prices, such as the area of the house, its location, and the type of dwelling. You are tasked with analyzing years of historical home sales data to estimate sales prices, which will help optimize real estate operations. In this case study, you will learn how to predict sales prices using the regression technique and the GLM algorithm.
- [Classification Use Case](#)
A retail store has information about its customers' behavior and the purchases they make. Now with the available data, they would like you to analyze and identify the type of customers they should target which would result in an increase in the volume of the most profitable product sold, and an increase in profit. In this use case, you will demonstrate how to identify such customers using the Random Forest algorithm.
- [Clustering Use Case](#)
A retail store has information about its customers' behavior and the purchases they make. Now with the available data, they would like you to analyze and identify if there are any similarities between the customers. Use Oracle Machine Learning to segment customers by finding clusters in the data set which can be then used to support targeted marketing campaigns to increase retail sales. In this use case, you will learn how to identify such segments using the k-Means algorithm.

3.1 Regression Use case

The Brooklyn housing dataset contains the sale prices of homes in brooklyn borough, along with various factors that influence these prices, such as the area of the house, its location, and the type of dwelling. You are tasked with analyzing years of historical home sales data to estimate sales prices, which will help optimize real estate operations. In this case study, you will learn how to predict sales prices using the regression technique and the GLM algorithm.

Related Contents

Topic	Link
OML4Py GitHub Example	OML4Py Regression GLM
About Generalized Linear Model	About Generalized Linear Model
About Machine Learning Classes and Algorithms	About Machine Learning Classes and Algorithms
Shared Settings	Shared Settings

Before you start your OML4Py use case journey, ensure that you have the following:

- Data Set
Download the data set from [Brooklyn housing dataset](#) .
- DatabaseSelect or create database out of the following options:

- Get your FREE cloud account. Go to <https://cloud.oracle.com/database> and select Oracle Database Cloud Service (DBCS), or Oracle Autonomous Database. Create an account and create an instance. See [Autonomous Database Quick Start Workshop](#).
 - Download the latest version of [Oracle Database](#) (on premises).
- Machine Learning Tools
Depending on your database selection,
 - Use OML Notebooks for Oracle Autonomous Database.
 - Install and use Oracle SQL Developer connected to an on-premises database or DBCS. See [Installing and Getting Started with SQL Developer](#).
- Other Requirements
Data Mining Privileges (this is automatically set for ADW). See [System Privileges for Oracle Machine Learning for SQL](#).
- [Load Data](#)
Load the data in your database and examine the data set and its attributes.
- [Explore Data](#)
Explore the data to understand and assess the quality of the data. At this stage assess the data to identify data types and noise in the data. Look for missing values and numeric outlier values.
- [Build Model](#)
Build your model using the training data set. Use the `oml.glm` function to build your model and specify model settings.
- [Evaluate](#)
Before you make predictions using your model on new data, you should first evaluate model accuracy. You can evaluate the model using different methods.

3.1.1 Load Data

Load the data in your database and examine the data set and its attributes.

You can download the dataset from [Rolling sales data](#) for the Brooklyn borough. If you are using the Oracle Autonomous Database, you will upload files to the Oracle Cloud Infrastructure (OCI) Object Storage, create a sample table, load data into the sample table from files on the OCI Object Storage, and explore the data.

Examine Data

There are 110 attributes in the dataset; below are descriptions of a few important ones. For a complete description of the attribute, see [Rolling sales data](#).

Attribute Name	Information
Borough	The borough in which the tax lot is located.
BoroCode	The borough in which the tax lot is located.
Year Built	The year construction of the building was completed.
Zip Code	A ZIP code that is valid for one of the addresses assigned to the tax lot.
Address	An address for the tax lot
BUILDING CLASS	A code describing the major use of structures on the tax lot.

Attribute Name	Information
HEALTH CENTER DISTRICT	The health center district in which the tax lot is located. Thirty health center districts were created by the City in 1930 to conduct neighborhood focused health interventions
LAND USE CATEGORY	A code for the tax lot's land use category
LOT AREA	Total area of the tax lot, expressed in square feet rounded to the nearest integer.
Building Area	The total gross area in square feet. Same as 'gross_sqft'
Gross Square Feet	The total area of all the floors of a building as measured from the exterior surfaces of the outside walls of the building, including the land area and space within any building or structure on the property.
Year Built	Year the structure on the property was built.
Sales Price	Price paid for the property.

- [Import Data](#)
Import data into the database by using Object Storage (for Cloud).

Related Topics

- [Glossary of Terms for Property Sales Files](#)
- [PLUTO DATA DICTIONARY](#)

3.1.1.1 Import Data

Import data into the database by using Object Storage (for Cloud).

If using a cloud account, one of the methods of importing the data is through Object Storage. Upload the data set to an Object Storage. The Object Storage URI will be used in another procedure. You can load data into your Oracle Autonomous Database (Autonomous Data Warehouse [ADW] or Autonomous Transaction Processing [ATP]) using Oracle Database tools, and Oracle and 3rd party data integration tools. You can load data:

- from local files in your client computer, or
- from files stored in a cloud-based object store

Follow the steps to upload your data file to the Object Storage bucket.

1. Login to your cloud account.
2. Click the left-side hamburger menu and select **Storage** from the menu.
3. Select **Buckets** from the Object Storage & Archive Storage option.
4. Select the compartment in which you want to upload the data.
5. Click **Create Bucket**.
6. Enter a name for your bucket. For example, Bucket1. Leave the rest of the fields as default.
7. Click **Create**.
8. Click on the bucket that you created. Scroll down and click **Upload** under Objects.

9. Leave the Object Name Prefix field blank. Click **select files** to navigate to the data file that you want to upload or drag and drop the data file. In this use case, select the modified .csv file.
10. Click **Upload**. The data file appears under Objects.
11. Click the ellipses on the right side of the data file to view the menu. Click **View Object Details**.
12. Copy the URL PATH (URI) to a text file. This URI is used in the `DBMS_CLOUD.COPY_DATA` procedure.

This procedure creates an object storage containing the data file in your cloud account.

Create Auth Token

The Auth Token is required in the `DBMS_CLOUD.CREATE_CREDENTIAL` procedure. You can generate the Auth Token in your cloud account.

1. Login into your ADW Cloud account.
2. Hover your mouse cursor over the human figure icon at the top right of the console and click **User Settings** from the drop-down menu.
3. Click **Auth Tokens** under Resources on the left of the console.
4. Click **Generate Token**. A pop-up dialog appears.
5. Enter a description (optional).
6. Click **Generate Token**.
7. Copy the generated token to a text file. The token does not appear again.
8. Click **Close**.

Create Object Storage Credential

The object storage credential is used in the `DBMS_CLOUD.COPY_DATA` procedure.

1. Login to the OML Notebooks page and create a notebook. See [Create a Notebook Classic](#)
2. Open the notebook that you just created.
3. Enter the following query to create an object storage credentials:

```
%script
begin
  DBMS_CLOUD.create_credential (
    credential_name => 'CRED',
    username => '<your cloud account username>',
    password => '<your Auth Token>'
  );
end;
/
```

```
----- PL/SQL procedure successfully completed.
-----
```

Examine the query:

- `credential_name`: The name of the credential to be stored. Provide any name. Here, CRED is the name given.

- `username`: This is your cloud account username.
 - `password`: Enter your Auth Token password that you copied after generating the Auth Token.
4. Click the play icon to run the query in your notebook. Your credentials are stored in the ADW user schema.
 5. In another para, run the following query to check the user credentials:

```
SELECT* FROM USER_CREDENTIALS;
```

3.1.2 Explore Data

Explore the data to understand and assess the quality of the data. At this stage assess the data to identify data types and noise in the data. Look for missing values and numeric outlier values.

Data Understanding and Preparation

This stage focuses on building a clear understanding of the dataset through the following steps:

- `Import necessary libraries`: Load essential Python libraries along with Oracle Machine Learning (OML).
- `Load the dataset`: Import the dataset for initial exploration.
- `Create a DataFrame proxy object`: Represent the table using a proxy object to simplify data manipulation.
- `Perform initial analysis`: Examine the dataset's structure, including its shape, data types, missing values, and categorical feature cardinality.

These steps provide a solid foundation for deeper data exploration and preprocessing.

For data preparation and understanding run the following steps:

1. Import necessary libraries

Run the following script in a %python interpreter paragraph to import the `oml` modules, the Panda's module, and set the display options:

```
import oml
import ssl
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

2. Load the dataset

```
url="https://objectstorage.us-ashburn-1.oraclecloud.com/n/adwc4pm/b/
OML_Data/o/brooklyn_sales.csv"
ssl._create_default_https_context = ssl._create_unverified_context
```

```
brooklyn_sales = pd.read_csv(url, engine='python')
z.show(brooklyn_sales.head())
```

Figure 3-1 Raw Brooklyn Data

ID ↕	borough ↕	neighborhood ↕	building_class_category ↕	tax_class ↕	block ↕	lot ↕	easement
1	3	DOWNTOWN-METROTECH	28 COMMERCIAL CONDOS	4	140	1001	nan
2	3	DOWNTOWN-FULTON FERRY	29 COMMERCIAL GARAGES	4	54	1	nan
3	3	BROOKLYN HEIGHTS	21 OFFICE BUILDINGS	4	204	1	nan
4	3	MILL BASIN	22 STORE BUILDINGS	4	8470	55	nan
5	3	BROOKLYN HEIGHTS	26 OTHER HOTELS	4	230	1	nan

3. Replace missing values (NaN with None)

```
df = brooklyn_sales.apply(lambda x: x.replace(np.nan, None) if x.dtypes ==
'object' else x)
```

4. Impute missing values based on data type

First, the code removes any columns that are entirely empty or contain only missing values. Then, it goes through each remaining column, checks the data type of the non-missing values, and fills in any missing data with the most appropriate replacement based on the column's type.

```
# Drop columns where all values are missing
brooklyn_sales1 = brooklyn_sales.dropna(axis=1, how="all")

d = {}

# Iterate through each column and fill missing values based on its data
type
for col in brooklyn_sales1:
    x = brooklyn_sales1[col].dropna().tolist() # Get non-null values to
check the column type

    if len(x) > 0:
        # For text columns, replace missing values with an empty string
        if isinstance(x[0], str):
            y = brooklyn_sales1[col].fillna("")
        # For integer columns, missing values are left unchanged
        elif isinstance(x[0], int):
            y = brooklyn_sales1[col]
        # For other numeric columns (e.g., floats), replace missing values
with 0.0
        else:
            y = brooklyn_sales1[col].fillna(float(0))

    d[col] = y # Store the modified column

# Convert the dictionary back into a DataFrame
brooklyn_sales2 = pd.DataFrame.from_dict(d)
```

```
# Print the shape of the updated DataFrame
print(brooklyn_sales2.shape)
```

```
(390883, 110)
```

5. Create dataframe proxy object

```
try:
    oml.drop(table = 'BROOKLYN')
except:
    pass
```

```
# Create a persistent table named BROOKLYN in the Oracle database
BROOKLYN = oml.create(brooklyn_sales2, table="BROOKLYN")
```

6. Analyze the dataframe

Examine and interpret the shape, data types, missing values and find columns having low cardinality.

- Shape of DataFrame:

```
BROOKLYN.shape
```

```
(390883, 110)
```

- Data Types of Columns:

```
BROOKLYN.dtypes
```

```
ID                <class 'oml.core.integer.Integer'>
borough           <class 'oml.core.integer.Integer'>
neighborhood      <class 'oml.core.string.String'>
building_class_category <class 'oml.core.string.String'>
tax_class         <class 'oml.core.string.String'>
...
PFIRM15_FL        <class 'oml.core.float.Float'>
Version           <class 'oml.core.string.String'>
MAPPLUTO_F        <class 'oml.core.float.Float'>
SHAPE_Leng        <class 'oml.core.float.Float'>
SHAPE_Area        <class 'oml.core.float.Float'>
Length: 110, dtype: object
```

- Identify columns with missing values (>75%):

```
def percent_missing(dat):
    per_miss={}
    large_miss_columns=[]
    for i in dat.columns:
        l=len(dat)
```

```

a=100-(dat[i].count()/l)*100
if a>=75:
    per_miss[i]=round(a)
return per_miss

z.show(pd.DataFrame(list(percent_missing(BROOKLYN).items()),
columns=["Columns", "% Missing"]))

```

Figure 3-2 View columns and their missing percentage.

Columns ↕	% Missing ↕
apartment_number	78
ZoneDist2	96
ZoneDist3	100
ZoneDist4	100
Overlay1	89

- Identify columns with low cardinality:

```

def unique_values_less_10(data):
    cols=[]
    for x in data.columns:
        unique_values=data[x].nunique()
        if unique_values< 10:
            cols.append(x)
    return cols

print(unique_values_less_10(BROOKLYN))

['borough', 'tax_class_at_sale', 'Borough', 'SanitBoro', 'ZoneDist4',
'Overlay1', 'Overlay2',
 'SPDist2', 'SPDist3', 'LtdHeight', 'SplitZone', 'Easements',
'OwnerType', 'AreaSource', 'Ext',
 'ProxCode', 'IrrLotCode', 'BsmtCode', 'BoroCode', 'CondoNo',
'ZMCode', 'PLUTOMapID',
 'FIRM07_FLA', 'PFIRM15_FL', 'Version', 'MAPPLUTO_F']

```

- Data Preparation**

Data preparation is the process of cleaning (handling missing values, outliers, and inconsistencies), transforming (scaling, encoding, and creating new features) and organizing raw data to make it more compatible with machine learning algorithms.

3.1.2.1 Data Preparation

Data preparation is the process of cleaning (handling missing values, outliers, and inconsistencies), transforming (scaling, encoding, and creating new features) and organizing raw data to make it more compatible with machine learning algorithms.

Data Preparation

This stage focuses on building a clear understanding of the dataset through the following steps:

- **Redundant Columns:**
 - **Clean the Data:** Identify and remove duplicate records, redundant columns, and highly correlated columns.
- **Data Subset Creation:** Filter/select relevant columns.
- **Visualizations:** Generate visual representations to understand data patterns and relationships.
- **Feature Engineering:** Create new features like "Decade Built", "Sale Age", and "Street Address".
- **Clean the DataFrame:** Handle missing values, outliers, and inconsistencies.
- **Dataframe Dimensions:** Ensure correct dimensions after cleaning.

These steps provide a solid foundation for data preparation.

Redundant Columns: Cleaning Up the Data

To identify redundant columns, compare column names and descriptions for clues. Then analyze the data within the columns to see if the values are identical or highly correlated. Once you identify the redundant columns, you need to decide which one to remove. The column with more missing values, inconsistent formatting, or mostly a unique value can be removed.

Analyze Columns with Similar Naming Conventions

Analyse and compare pairs or groups of columns that appear similar due to their content or variations in naming conventions or formatting. The goal is to ensure data consistency, detect redundancy, and verify if columns represent the same information.

The following columns are compared:

- borough, Borough, and BoroCode
- YearBuilt and year_built
- ZipCode and zip_code

Similarly, this can be applied to other columns, such as building_class, BldgClass and building_class_at_sale; Address and address; etc.

1. Compare columns: borough, Borough, and BoroCode

The column names "borough," "Borough," and "BoroCode" are quite similar, suggesting they may contain the same information. To verify this, print five random samples from each column for comparison. Additionally, based on the column descriptions, all three columns represent the Brooklyn borough, with values such as "BK" or "3.0." Since the data is

specific to Brooklyn, these columns are redundant and can be safely removed from the dataset.

```
z.show(BROOKLYN[['borough', 'Borough', 'BoroCode']].sample(n=5))
```

borough ↕	Borough ↕	BoroCode ↕
3	BK	3
3	None	0
3	BK	3
3	BK	3
3	BK	3

2. Identify Matching Data Percentage

```
def matching_percentage(data, cols, threshold=0.9):
    # Filter rows where both columns have non-zero values
    filtered_df = data[(data[cols[0]] != 0) & (data[cols[1]] != 0)]

    # Calculate matching percentage
    total_rows = len(filtered_df)
    matching_rows = len(filtered_df[filtered_df[cols[0]] ==
    filtered_df[cols[1]]])
    matching_percentage = matching_rows / total_rows if total_rows else 0

    # Output result
    if matching_percentage >= threshold:
        print(f"The columns have a high percentage ({matching_percentage *
    100:.2f}%) of matching values, suggesting similarity.")
    else:
        print("The columns do not have a high percentage of matching
    values.")
```

3. Compare columns: YearBuilt and year_built

The column names "YearBuilt" and "year_built" are quite similar, suggesting they may contain the same information. To verify this, we should print out 5 random samples from each column.

```
matching_percentage(BROOKLYN, ['YearBuilt', 'year_built'])
```

The columns have a high percentage (99.25%) of matching values, suggesting similarity.

The columns "YearBuilt", "year_built" contain similar information, so remove one. Remove "year_built" from the dataset.

4. Compare columns: ZipCode and zip_code

```
matching_percentage(BROOKLYN, ['ZipCode', 'zip_code'])
```

The columns have a high percentage (98.72%) of matching values, suggesting similarity.

The column names "ZipCode" and "zip_code" are quite similar, suggesting they may contain the same information. To verify this, we should print out 5 random samples from each column.

```
z.show(BROOKLYN[['ZipCode', 'zip_code']].sample(n=5))
```

ZipCode ↕	zip_code ↕
11207	11207
11221	11221
11216	11216
11236	11236
11229	11229

Filter data by selecting the desired columns

To focus on more reliable data, only houses built after the 1800s are included in the dataset. This is because houses built before the 1800s frequently have a single YearBuilt value of 0, indicating potentially missing or inaccurate information and more is explained in the next step.

```
BROOKLYN2 = BROOKLYN[(BROOKLYN['YearBuilt']>= 1800)]  
[['building_class_at_sale', 'HealthCent', 'YearBuilt', 'ResidFAR',  
 'sale_date', 'building_class_category', 'GarageArea', 'CD', 'YearAlter1',  
 'ID', 'SchoolDist', 'SanitDistr', 'PolicePrct', 'address',  
 'CT2010', 'commercial_units', 'BldgArea', 'NumFloors',  
 'sale_price', 'AssessTot', 'ResArea', 'land_sqft', 'LotFront',  
 'LotArea', 'AssessLand', 'SHAPE_Area', 'year_of_sale',  
 'gross_sqft', 'XCoord', 'YCoord', 'SHAPE_Leng']]
```

```
# Dataframe dimension  
BROOKLYN2.shape
```

```
(295356, 31)
```

Feature Engineering and Visualization

Create new columns and modify existing features based on insights gathered from visualizations. The newly engineered features are then merged back into the dataset to enhance its quality and readiness for modeling.

1. Built periods and their counts:

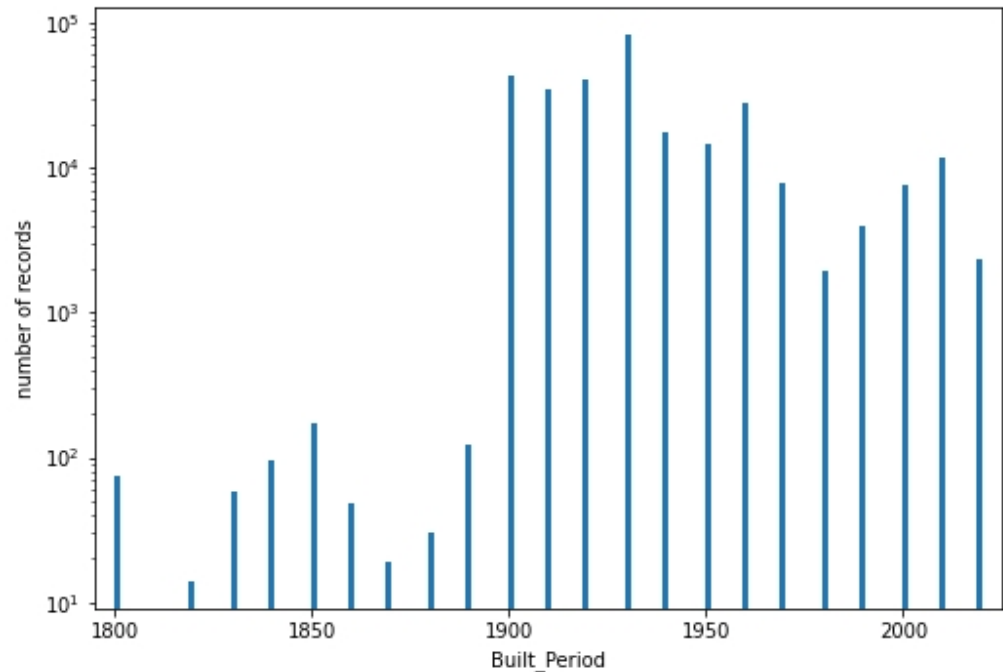
- Analyze the distribution of the periods in which the houses were built and identify the least and most common periods within our dataset. The column, YearBuilt, is first rounded to the nearest integer value. These rounded values will then be categorized into predefined intervals, or bins. Count the number of YearBuilt within each bin to determine the frequency distribution of built periods.

```
built_period = (BROOKLYN2['YearBuilt'] // 10) * 10 +  
oml.Integer(BROOKLYN2['YearBuilt'] % 10 >= 5) * 10  
bins_str =  
built_period.cut(bins=[1700,1800,1880,1900,1920,1940,1960,1980,2000,2020  
,2040])  
bins = sorted(bins_str.drop_duplicates().pull())  
  
z.show(pd.DataFrame({'Built Period':bins, 'Count':[oml.Integer(bins_str  
== b).sum() for b in bins]}))
```

Built Period ↕	Count ↕
(1700, 1800]	74
(1800, 1880]	438
(1880, 1900]	42855
(1900, 1920]	75342
(1920, 1940]	99553

- Visualisation of built periods and their counts

```
# Data might be incomplete when DECADE_BUILT < 1900  
  
Nbins = 141  
n, bins, patches = plt.hist(built_period.pull(), Nbins)  
plt.xlabel('Built_Period')  
plt.ylabel('number of records')  
plt.yscale('log')  
p = plt.xlim(1795, 2025)
```



2. Preview sale price and count by binning

- Analyze the distribution of the sale price of the houses and identify the least and most common sale price within our dataset. The column, `sale_price`, is rounded and then are categorized into predefined intervals, or bins. Count the number of `sale_price` within each bin to determine the frequency distribution of `sale_price`.

```
Sale_Price=(BROOKLYN2['sale_price'].cut(bins=[-100000000,0,20000,40000,60000,80000,100000,1000000,10000000,500000000]))
```

```
# bins_str =
decade_built.cut(bins=[1700,1800,1880,1900,1920,1940,1960,1980,2000,2020,2040])
bins = sorted(Sale_Price.drop_duplicates().pull())
```

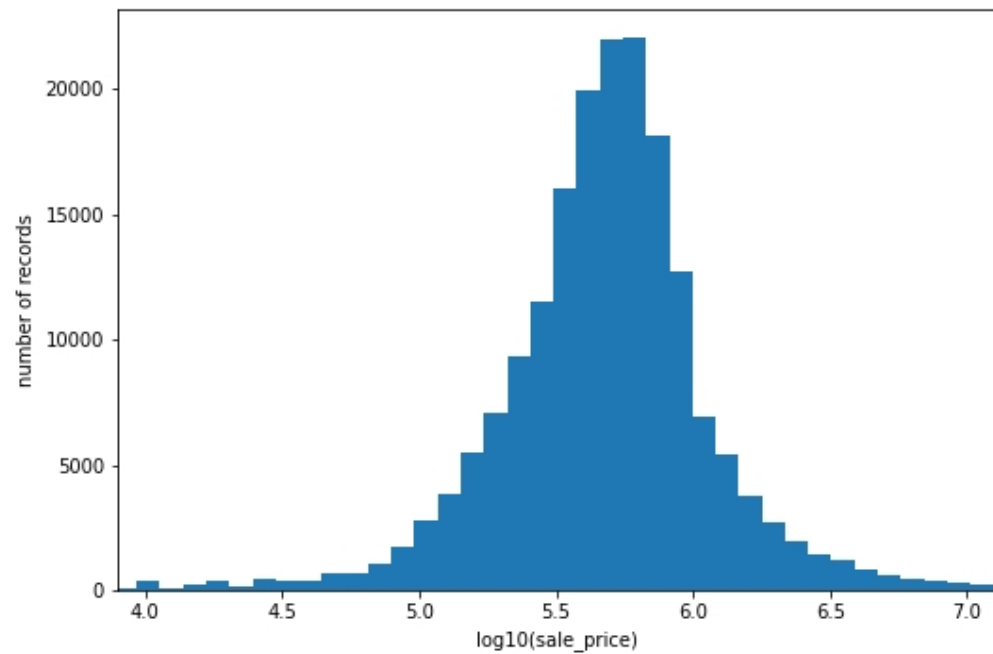
```
z.show(pd.DataFrame({'Sale Price':bins, 'Count':[oml.Integer(Sale_Price == b).sum() for b in bins]}))
```

Sale Price ↕	Count ↕
(-100000000,0]	106063
(0,20000]	6295
(100000,1000000]	151515
(1000000,10000000]	24115
(10000000,500000000]	938

- Examine logarithmic sales price distribution

```
# Most properties have  $10^5 < \text{sale\_price} < 10^{6.5}=3.2\text{M}$ 
Nbins = 101

n, bins, patches = plt.hist((BROOKLYN2[BROOKLYN2['sale_price']>0]
['sale_price']).log(10).pull(), Nbins)
plt.xlabel('log10(sale_price)')
plt.ylabel('number of records')
p = plt.xlim(3.9, 7.1)
```



3. Preview building class category, count and count percentage

Analyze the categorical column, `building_class_category`, by computing a cross-tabulation. Sort this table in descending order. Finally, determine the frequency of each building class category.

```
build_category=
BROOKLYN2.crosstab('building_class_category').sort_values('count',
ascending=False)
count_percentage= ((build_category['count'] / len(BROOKLYN2)) *
100).round(decimals=2)
z.show(build_category.concat({'count_percentage':count_percentage}))
```

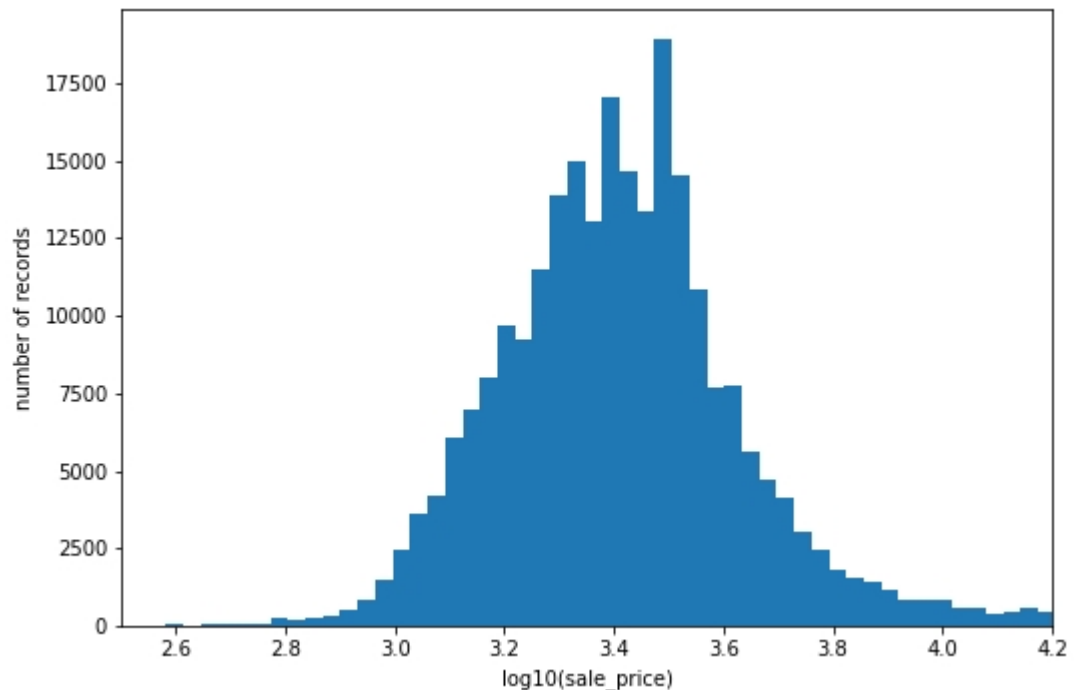
building_class_category ↕	count ↕	count_percentage
02 TWO FAMILY HOMES	104105	35.25
01 ONE FAMILY HOMES	52897	17.91
03 THREE FAMILY HOMES	37489	12.69
10 COOPS - ELEVATOR APARTMENTS	31320	10.6
07 RENTALS - WALKUP APARTMENTS	27634	9.36

4. Examine logarithmic gross qft distribution

Values in the gross qft column, when log-transformed, approximate a normal distribution.

```
# Most properties have  $10^{2.9}=800 < \text{sale\_price} < 10^{3.7}=5000$ 
Nbins = 201

n, bins, patches = plt.hist((BROOKLYN2[BROOKLYN2['gross_sqft']>0]
['gross_sqft']).log(10).pull(), Nbins)
plt.xlabel('log10(sale_price)')
plt.ylabel('number of records')
p = plt.xlim(2.5, 4.2)
```



Feature Engineering

Feature engineering is the process of creating new input features from existing data which explains the underlying patterns of a data. These new features help to improve the model's predictability.

The following features have been engineered:

- **Built Period:** The Period in which the house was built.

```
built_period=(BROOKLYN2['YearBuilt'] // 10) * 10 +
oml.Integer(BROOKLYN2['YearBuilt'] % 10 >= 5) * 10
BROOKLYN2=BROOKLYN2.concat({'Built_Period':built_period})
```

- **Age_At_Sale:** Age of the house at sale is the number of years from its construction to the sale date.

```
Age_At_Sale2 = abs(BROOKLYN2['year_of_sale'] - BROOKLYN2['YearBuilt'])
BROOKLYN2= BROOKLYN2.concat({'Age_At_Sale2': Age_At_Sale2})
```

- **Quarter:** Refers to the quarter in which the house was built.

```
time_period = oml.Datetime.strptime(BROOKLYN2['sale_date'], format="MM/DD/
YYYY")
Quarter= (oml.Integer((time_period.month - 1) // 3 + 1))
```

Clean the Dataframe

After feature engineering, remove columns that no longer contribute to the analysis and drop any rows that have a missing value.

```
BROOKLYN2=BROOKLYN2.drop(['sale_date'])
BROOKLYN2=BROOKLYN2.dropna()
```

Filter the data to include properties whose sale price, gross square footage, and the decade year of construction fall within a specific, relevant range.

```
BROOKLYN3 = BROOKLYN2[(BROOKLYN2['sale_price']>=1.0e5) &
(BROOKLYN2['sale_price']<=5.0e6) &
(BROOKLYN2['gross_sqft']>=800) & (BROOKLYN2['gross_sqft']<=5000)
&
(BROOKLYN2['Built_Period']>=1900) &
(BROOKLYN2['Built_Period']<=2010) ]
```

Apply a log transformation to normalize the column. The original, untransformed column is then removed.

```
BROOKLYN3 = BROOKLYN3.concat({'log_gross_sqft':
BROOKLYN3['gross_sqft'].log(10)})
BROOKLYN3=BROOKLYN3.drop(['gross_sqft'])
```

To improve the model's performance, filter the data to focus on properties with higher probabilities of belonging to specific building classes and categories.

```
BROOKLYN4 = BROOKLYN3[(BROOKLYN3['building_class_at_sale']=='A1') |
(BROOKLYN3['building_class_at_sale']=='A2')
| (BROOKLYN3['building_class_at_sale']=='A4') |
(BROOKLYN3['building_class_at_sale']=='A5')
| (BROOKLYN3['building_class_at_sale']=='B1') |
(BROOKLYN3['building_class_at_sale']=='B2')]
```

```
| (BROOKLYN3['building_class_at_sale']=='B3') ]
BROOKLYN5 = BROOKLYN4[((BROOKLYN4['building_class_category']=='02 TWO FAMILY
HOMES') | (BROOKLYN4['building_class_category']=='01 ONE FAMILY HOMES'))]
```

Dataframe Dimensions

Run the following script to verify the dataframe dimensions:

```
BROOKLYN5.shape
```

```
(67083, 32)
```

3.1.3 Build Model

Build your model using the training data set. Use the `oml.glm` function to build your model and specify model settings.

For a supervised learning, like Regression, before creating the model, split the data in to training and test data. Although you can use the entire data set to build a model, it is difficult to validate the model unless there are new data sets available. Therefore, to evaluate the model and to accurately assess the performance of the model on the same data, you generally split or separate the data into training and test data. You use the training data set to train the model and then use the test data set to test the accuracy of the model by running prediction queries. The testing data set already contains known values for the attribute that you want to predict. It is thus easy to determine whether the model's predictions are correct.

Algorithm Selection

Before you build a model, choose the suitable algorithm. You can choose one of the following algorithms to solve a regression problem:

- Extreme Gradient Boosting
- Generalized Linear Model
- Neural Network
- Support Vector Machine

When you want to understand the data set, you always start from a simple and easy baseline model. The Generalized Linear Model algorithm is the right choice because it is simple and easy to interpret since it fits a linear relationship between the feature and the target. You can get an initial understanding of a new data set from the result of the linear model.

The following steps guide you to split your data and build your model with the selected algorithm.

1. Split Data: Train/Test:

Split the data into training and test data, with a 80/20 ratio respectively. The seed parameter is used for random splitting. The split method splits the data referenced by the DataFrame proxy object `BROOKLYN5` into two new DataFrame proxy objects `train`, and `test`.

```
TRAIN, TEST = BROOKLYN5.split(ratio = (0.8,0.2), seed=15)
TRAIN_X = TRAIN.drop('sale_price')
TRAIN_Y = TRAIN['sale_price']
```

```
TEST_X = TEST  
TEST_Y = TEST['sale_price']
```

2. Model Building:

Specify the model settings and build a Generalized Linear Model (GLM) model object for predicting the `sale_price` attribute, run the following script. The settings are given as key-value or dictionary pairs where it refers to parameters name and value setting respectively.

```
try:  
    oml.drop(model = 'BROOKLYN_GLM_REGRESSION_MODEL')  
except:  
    print('No such model')  
  
setting = {'PREP_AUTO':'ON',  
           'GLMS_ROW_DIAGNOSTICS':'GLMS_ROW_DIAG_ENABLE',  
           'GLMS_FTR_SELECTION':'GLMS_FTR_SELECTION_ENABLE',  
           'GLMS_FTR_GENERATION':'GLMS_FTR_GENERATION_ENABLE'}  
  
glm_mod = oml.glm("regression", **setting)  
glm_mod = glm_mod.fit(TRAIN_X, TRAIN_Y, model_name =  
    'BROOKLYN_GLM_REGRESSION_MODEL', case_id = 'ID')
```

Model setting parameters:

- **PREP_AUTO:** Used to specify fully automated or user-directed general data preparation. By default, it is enabled with a constant value as 'PREP_AUTO': PREP_AUTO_ON.
- **GLMS_ROW_DIAGNOSTICS:** Enables or disables the row diagnostics. By default, row diagnostics are disabled.
- **GLMS_FTR_SELECTION:** Enables or disables feature selection for GLM. By default, feature selection is not enabled.
- **GLMS_FTR_GENERATION:** Specifies whether or not feature generation is enabled for GLM. By default, feature generation is not enabled.

Note:

Feature generation can only be enabled when feature selection is also enabled.

3.1.4 Evaluate

Before you make predictions using your model on new data, you should first evaluate model accuracy. You can evaluate the model using different methods.

Information about Model settings

Evaluate the model by examining the various statistics generated after building the model. The statistics indicate the model's quality.

- **Model details:** Run the following script for model details available through the GLM model object, like the model settings, attribute coefficients, fit details, etc.

```
glm_mod
```

They can also be displayed and viewed individually as shown below.

- **Attribute Coefficient:** Run the following script to display the model's attribute coefficient.

```
z.show(glm_mod.coef.round(2).head())
```

attribute name ↕	attribute value ↕	coefficient ↕	std error ↕	t value ↕	p value
(Intercept)	None	7943491860.15	313153278.31	25.37	0
CT2010	None	33851.43	2027.35	16.7	0
BldgArea	None	-13147.49	852.39	-15.42	0
year_of_sale	None	-110725697.2	3974468.36	-27.86	0
XCoord	None	2703.18	171.95	15.72	0

- **Fit Details:** Run the following script to display the fit details of the model.

```
z.show(glm_mod.fit_details.round(2).head())
```

name ↕	value ↕
ADJUSTED_R_SQUARE	0.65
AIC	1340512.07
COEFF_VAR	41.9
CORRECTED_TOTAL_DF	53743
CORRECTED_TOT_SS	1042052796

Score

Scoring is the process of applying the model on the test data to access its performance.

1. **Predict sale price:** Use the model to make predictions on test data.

```
BROOKLYN_RES = glm_mod.predict(TEST.drop('sale_price'), supplemental_cols
= TEST[:, ['ID', 'sale_price']])
z.show(BROOKLYN_RES.round(2).head())
```

ID ↕	sale_price ↕	PREDICTION
2492	5000000	3305640.36
2509	5000000	1185078
27111	1100000	1041649.91
26746	1100000	842356.97
27302	1100000	827528.32

2. Evaluate Model Performance: Evaluate the model's performance by using the score function.

```
model_coef = len(glm_mod.coef)
no_rows_test = TEST_X.shape[0]
R_squared = glm_mod.score(TEST_X, TEST_Y).round(3)

print(
    f"RMSE : {(((BROOKLYN_RES['PREDICTION'] - BROOKLYN_RES['sale_price'])
    ** 2).mean() ** .5).round(2)}\n"
    f"MAE: {(abs(BROOKLYN_RES['PREDICTION'] -
    BROOKLYN_RES['sale_price'])).mean().round(2)}\n"
    f"R_squared: {R_squared}\n"
    f"Adjusted R^2: {(1 - (1 - R_squared)*(no_rows_test-1)/(no_rows_test
    - model_coef - 1)).round(4)}"
)
```

The interpretation of the model's performance based on the metrics are as follows:

- **RMSE** (286,137.56): The model's predictions, on average, deviate by approximately 286,137.56 units from the actual values.
- **MAE** (170,992.05): The average absolute error in predictions is 170,992.05 units, which provides a sense of the model's accuracy without penalizing large errors.
- **R²** (0.581): The model explains 58.1% of the variance in the target variable, suggesting a moderate fit to the data.
- **Adjusted R²** (0.5604): After adjusting for the number of predictors, the model explains about 56.04% of the variance

Overall, the model demonstrates moderate predictive accuracy, with potential for further improvement.

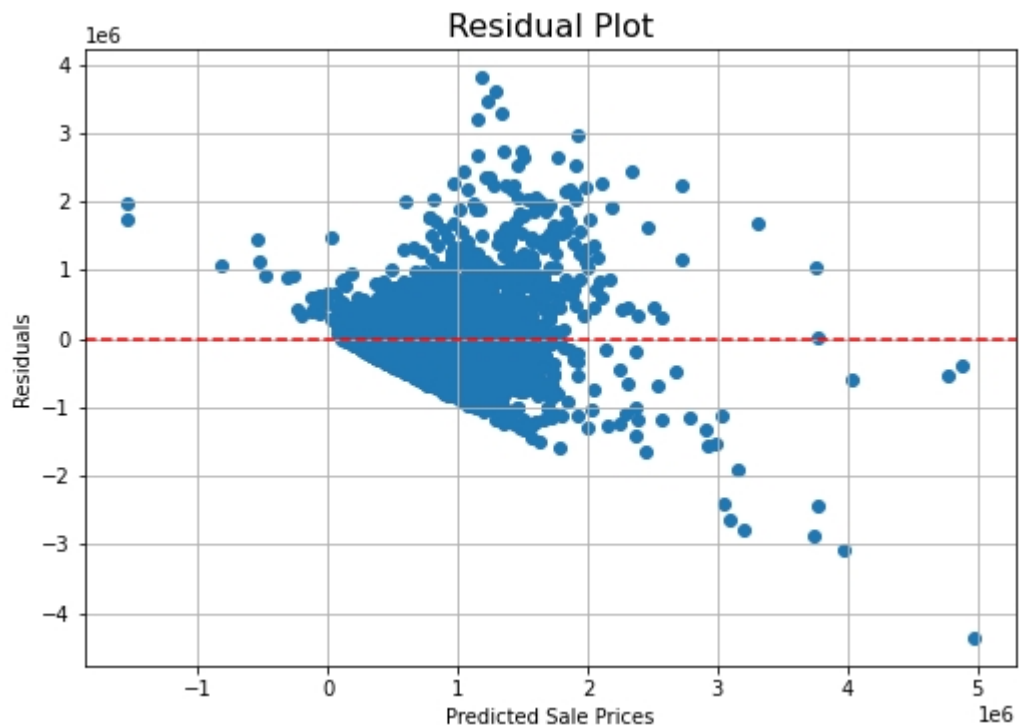
3. Residual Graph: The Residual plot is showing heteroscedastic pattern. This indicates that the errors in the model are inconsistent. The non-linear relationship between the fitted values (predicted values) and the residuals, suggests that the model can be improved further.

```
import matplotlib.pyplot as plt

y_pred = BROOKLYN_RES['PREDICTION'].pull()
residuals = (BROOKLYN_RES['sale_price'] -
BROOKLYN_RES['PREDICTION']).pull()
```

```
plt.scatter(y_pred, residuals)

plt.xlabel('Predicted Sale Prices')
plt.ylabel('Residuals')
plt.title('Residual Plot', fontsize=16)
plt.grid(True)
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



SQL Interface to Score Data and Display Prediction Details

You can score data and make predictions using the SQL interface. The test data is materialized into `BROOKLYN_GLM_TEST_DATA` so that you can query it using SQL. The materialized method writes the contents of an Oracle Machine Learning `oml.DataFrame` proxy table to an Oracle Database table.

1. Materialize `BROOKLYN_GLM_TEST_DATA` for use in query below:

```
try:
    oml.drop(table = 'BROOKLYN_GLM_TEST_DATA')
except:
    pass
_ = TEST.materialize(table = 'BROOKLYN_GLM_TEST_DATA')
```

2. Display prediction with explanatory prediction details in SQL: The SQL command to score and display the prediction details. The prediction functions apply a glm regressional model named `BROOKLYN_GLM_REGRESSION_MODEL` to the data from the materialized table

BROOKLYN_GLM_TEST_DATA. The query includes information about the predictors that have the greatest influence on the prediction.

```
SELECT ID,
       round(PREDICTION_YRS_RES,3) PRED_YRS_RES,
       round(PRED_LOWER_LIMIT,1) LOWER_BOUND,
       round(PRED_UPPER_LIMIT,1) UPPER_BOUND,
       RTRIM(TRIM(SUBSTR(OUTPRED."Attribute1",17,100)), 'rank="1"/>')
FIRST_ATTRIBUTE,
       RTRIM(TRIM(SUBSTR(OUTPRED."Attribute2",17,100)), 'rank="2"/>')
SECOND_ATTRIBUTE,
       RTRIM(TRIM(SUBSTR(OUTPRED."Attribute3",17,100)), 'rank="3"/>')
THIRD_ATTRIBUTE
FROM (SELECT ID,
            PREDICTION(BROOKLYN_GLM_REGRESSION_MODEL USING *)
PREDICTION_YRS_RES,
            PREDICTION_BOUNDS(BROOKLYN_GLM_REGRESSION_MODEL USING
*) .LOWER PRED_LOWER_LIMIT,
            PREDICTION_BOUNDS(BROOKLYN_GLM_REGRESSION_MODEL USING
*) .UPPER PRED_UPPER_LIMIT,
            PREDICTION_DETAILS(BROOKLYN_GLM_REGRESSION_MODEL USING *) PD
FROM BROOKLYN_GLM_TEST_DATA
WHERE ID < 100015
ORDER BY ID) OUT,
XMLTABLE('/Details'
        PASSING OUT.PD
        COLUMNS
        "Attribute1" XMLType PATH 'Attribute[1]',
        "Attribute2" XMLType PATH 'Attribute[2]',
        "Attribute3" XMLType PATH 'Attribute[3]') OUTPRED
```

ID ↕	PRED_YRS_RES ↕	LOWER_BOUND ↕	UPPER_BOUND ↕	FIRST_ATTRIBUTE ↕
2492	3305640.357	3247333.7	3363947	"year_of_sale" actualValue="2016" weight="1"
2509	1185077.996	1165498.2	1204657.8	"year_of_sale" actualValue="2017" weight="1"
2536	2731191.702	2654003	2808380.4	"year_of_sale" actualValue="2015" weight="1"
2571	1290879.075	1244202.5	1337555.7	"year_of_sale" actualValue="2014" weight="1"
2596	1917010.67	1883723.3	1950298.1	"year_of_sale" actualValue="2014" weight="1"

To conclude, you have successfully predicted the median house prices in Boston using Generalized Linear Model algorithm.

3.2 Classification Use Case

A retail store has information about its customers' behavior and the purchases they make. Now with the available data, they would like you to analyze and identify the type of customers they should target which would result in an increase in the volume of the most profitable product

sold, and an increase in profit. In this use case, you will demonstrate how to identify such customers using the Random Forest algorithm.

Related Contents

Topic	Link
OML4Py GitHub Example	Classification Random Forest
About Random Forest	About Random Forest
Random Forest	Random Forest Algorithm
Shared Settings	Shared Settings

Before you start your OML4Py use case journey, ensure that you have the following:

- **Data Set**
The data set used for this use case is from the SH schema. The SH schema can be readily accessed in Oracle Autonomous Database. For on-premises databases, the schema is installed during the installation or can be manually installed by downloading the scripts. See [Installing the Sample Schemas](#).
- **Database**
Select or create database out of the following options:
 - Get your FREE cloud account. Go to <https://cloud.oracle.com/database> and select Oracle Database Cloud Service (DBCS), or Oracle Autonomous Database. Create an account and create an instance. See [Autonomous Database Quick Start Workshop](#).
 - Download the latest version of [Oracle Database](#) (on premises).
- **Machine Learning Tools**
Depending on your database selection,
 - Use OML Notebooks for Oracle Autonomous Database.
 - Install and use Oracle SQL Developer connected to an on-premises database or DBCS. See [Installing and Getting Started with SQL Developer](#).
- **Other Requirements**
Data Mining Privileges (this is automatically set for ADW). See [System Privileges for Oracle Machine Learning for SQL](#).
- **[Load Data](#)**
Access the data set from the SH Schema and explore the data to understand the attributes.
- **[Explore Data](#)**
Explore the data to understand and assess the quality of the data. At this stage assess the data to identify data types and noise in the data. Look for missing values and numeric outlier values.
- **[Build Model](#)**
Build your model using the training data set. Use the `oml.rf` function to build your model and specify the model settings.
- **[Evaluate](#)**
Before you make predictions using your model on new data, you should first evaluate model accuracy. You can evaluate the model using different methods.

Related Topics

- [Create a Notebook](#)
- [Edit your Notebook](#)

- Installing Sample Schemas

3.2.1 Load Data

Access the data set from the SH Schema and explore the data to understand the attributes.

Examine Data

Attribute Name	Information
CUST_ID	The ID of the customer
EDUCATION	Educational information of the customer
OCCUPATION	Occupation of the customer
HOUSEHOLD_SIZE	People per house
YRS_RESIDENCE	Number of years of residence
AFFINITY_CARD	Whether the customer holds an affinity card
BULK_PACK_DISKETTES	Product. Indicates whether the customer already owns the product. 1 means Yes. 0 means No
FLAT_PANEL_MONITOR	Product. Indicates whether the customer already owns the product. 1 means Yes. 0 means No
HOME_THEATER_PACKAGE	Product. Indicates whether the customer already owns the product. 1 means Yes. 0 means No
BOOKKEEPING_APPLICATION	Product. Indicates whether the customer already owns the product. 1 means Yes. 0 means No
PRINTER_SUPPLIES	Product. Indicates whether the customer already owns the product. 1 means Yes. 0 means No
Y_BOX_GAMES	Product. Indicates whether the customer already owns the product. 1 means Yes. 0 means No
OS_DOC_SET_KANJI	Product. Indicates whether the customer already owns the product. 1 means Yes. 0 means No
COMMENTS	Comments from customers

To learn more about `CUSTOMERS` table in `SH` Schema, see [SH Sample Schema](#) .

3.2.2 Explore Data

Explore the data to understand and assess the quality of the data. At this stage assess the data to identify data types and noise in the data. Look for missing values and numeric outlier values.

Data Understanding and Preparation

To access database data from Python using OML4Py, you must first create a `oml.DataFrame` proxy object in Python which represents a database table, view, or query. Create a `oml.DataFrame` proxy object for `SUPPLEMENTARY_DEMOGRAPHICS` and `CUSTOMERS`

and then merge them by inner join on an identical and unique column. Assess the data to identify data types and noise in the data. Look for missing values (systematic or random), outlier numeric values, or inconsistently labeled categorical values.

For data preparation and understanding run the following steps:

1. Run the following script in a %python interpreter paragraph to import the `oml` modules, the Panda's module, and set the display options:

```
import pandas as pd
import oml

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

2. Use the `oml.sync` function to create the Python object `DEMOGRAPHICS` as a proxy for a database table `SUPPLEMENTARY_DEMOGRAPHICS`. 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.

```
DEMOGRAPHICS = oml.sync(table = "SUPPLEMENTARY_DEMOGRAPHICS", schema =
"SH")
z.show(DEMOGRAPHICS.head())
```

CUST_ID ◊	EDUCATION ◊	OCCUPATION ◊	HOUSEHOLD_SIZE ◊	YRS_RESIDENCE ◊	AFFINITY_CARD
102547	10th	Other	1	0	0
101050	10th	Other	1	0	0
100040	11th	Sales	1	0	0
102117	HS-grad	Farming	1	0	0
101074	10th	Handler	1	1	0

3. Run the shape function to view the rows and columns of an `oml.DataFrame` object `DEMO`.

```
print("Shape:", DEMOGRAPHICS.shape)

(4500, 14)
```

4. Use the `oml.sync` function to create the Python object `CUSTOMERS` as a proxy for a database table `SH.CUSTOMERS`. Like the second step here no schema is used. With the schema argument, you can specify the schema in which to create a Python environment and proxy objects. Only one environment for a given database schema can exist at a time. If `schema=None`, then objects are created in the current user's schema.

```
CUSTOMERS = oml.sync(query = 'SELECT CUST_ID, CUST_GENDER,
CUST_MARITAL_STATUS, CUST_YEAR_OF_BIRTH, CUST_INCOME_LEVEL,
```

```
CUST_CREDIT_LIMIT FROM SH.CUSTOMERS')
z.show(CUSTOMERS.head())
```

CUST_ID ↕	CUST_GENDER ↕	CUST_MARITAL_STATUS ↕	CUST_YEAR_OF_BIRTH ↕	CUST_INCOME_LEVEL
49671	M	married	1976	G: 130,000 - 149,999
3228	M	None	1964	G: 130,000 - 149,999
6783	M	single	1942	G: 130,000 - 149,999
10338	M	married	1977	G: 130,000 - 149,999
13894	M	None	1949	G: 130,000 - 149,999

- Run the shape function to view the rows and columns of an oml.DataFrame object CUSTOMERS.

```
print("Shape:", CUSTOMERS.shape)
```

```
(55500, 6)
```

- Create a new oml.dataframe CUSTOMER_DATA by merging the table CUSTOMERS and DEMOGRAPHICS with an inner join on the common column CUST_ID . The merge function joins one oml.dataframe to another oml.dataframe . The suffixes parameter is used when the two oml.dataframe have conflicting column names.

```
CUSTOMER_DATA = CUSTOMERS[["CUST_ID", "CUST_INCOME_LEVEL",
"CUST_CREDIT_LIMIT"]].merge(DEMOGRAPHICS[["CUST_ID", "HOUSEHOLD_SIZE",
"OCCUPATION", "HOME_THEATER_PACKAGE"]], how = "inner", on =
'CUST_ID', suffixes = ["", ""])
z.show(CUSTOMER_DATA.head())
```

CUST_ID ↕	CUST_INCOME_LEVEL ↕	CUST_CREDIT_LIMIT ↕	HOUSEHOLD_SIZE ↕	OCCUPATION
100134	L: 300,000 and above	9000	2	Cleric.
102828	E: 90,000 - 109,999	10000	1	Machine
101232	J: 190,000 - 249,999	9000	1	Other
100696	F: 110,000 - 129,999	7000	3	Prof.
103948	J: 190,000 - 249,999	9000	1	Cleric.

- Run the shape function to view the rows and columns of an oml.DataFrame object CUSTOMER_DATA.

```
print("Shape:", CUSTOMER_DATA.shape)
```

```
(4500, 6)
```

8. Run the following script to view the data types of all the columns.

```
print("The datatypes of the column: ", "\n")
print(CUSTOMER_DATA.dtypes)
```

The datatypes of the column:

```
CUST_ID                <class 'oml.core.float.Float'>
CUST_INCOME_LEVEL      <class 'oml.core.string.String'>
CUST_CREDIT_LIMIT      <class 'oml.core.float.Float'>
HOUSEHOLD_SIZE         <class 'oml.core.string.String'>
OCCUPATION              <class 'oml.core.string.String'>
HOME_THEATER_PACKAGE   <class 'oml.core.integer.Integer'>
dtype: object
```

9. To check if there are any missing values run the following script. The count function returns the number of elements that are not NULL for each column and the len function returns the number of rows in the dataset.

```
print("Number of missing values in each column is : \n")
print(len(CUSTOMER_DATA)-CUSTOMER_DATA.count())
```

Number of missing values in each column is :

```
CUST_ID                0
CUST_INCOME_LEVEL      0
CUST_CREDIT_LIMIT      0
HOUSEHOLD_SIZE         0
OCCUPATION              0
HOME_THEATER_PACKAGE   0
dtype: int64
```

10. Use the crosstab method to perform a cross-column analysis of an oml.DataFrame object in the database. 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. In this example, the crosstab function displays the distribution of HOME_THEATER_PACKAGE responders.

```
z.show(CUSTOMER_DATA.crosstab('HOME_THEATER_PACKAGE'))
```

HOME_THEATER_PACKAGE ↕	count
0	1961
1	2539

11. To know how customers respond to HOME_THEATER_PACKAGE according to their income level run the following code:

```
z.show(CUSTOMER_DATA.crosstab('CUST_INCOME_LEVEL', 'HOME_THEATER_PACKAGE').sort_values('count', ascending=False).rename(columns = {'count': 'NUM_CUSTOMERS'}))
```

CUST_INCOME_LEVEL ↕	HOME_THEATER_PACKAGE ↕	NUM_CUSTOMERS
J: 190,000 - 249,999	1	519
J: 190,000 - 249,999	0	446
L: 300,000 and above	0	328
I: 170,000 - 189,999	1	319
L: 300,000 and above	1	315

This completes the data understanding and data preparation stage.

3.2.3 Build Model

Build your model using the training data set. Use the `oml.rf` function to build your model and specify the model settings.

For a supervised learning, like Classification, before creating the model, split the data into training and test data. Although you can use the entire data set to build a model, it is difficult to validate the model unless there are new data sets available. Therefore, to evaluate the model and to accurately assess the performance of the model on the same data, you generally split or separate the data into training and test data. You use the training data set to train the model and then use the test data set to test the accuracy of the model by running prediction queries. The testing data set already contains known values for the attribute that you want to predict. It is thus easy to determine whether the predictions of the model are correct.

Algorithm Selection

Before you build a model, choose the suitable algorithm. You can choose one of the following algorithms to solve a classification problem:

- Decision Tree
- Generalized Linear Model
- Naive Bayes
- Neural Network
- Random Forest
- Support Vector Machine

Here you will be using Random forest algorithms as interpretability is not a major concern. The Random Forest algorithm is a type of ensemble method used for classification. Random forest builds a number of independent decision trees and combines the output of the multiple decision trees to make predictions. Each of these decision trees is built using a random sample from the input and each tree uses a random subset of the features. This avoids the problem of overfitting while increasing accuracy. To build a model using a supervised learning algorithm

(Random Forest Model), you need to first split the data into train and test data. After splitting the data, build the model using the train data and once the model is built, score the test data using the model.

1. You will split the CUSTOMER_DATA data with 60 percent of the records for the train data set and 40 percent for the test data set. The seed parameter is used for random splitting. The split method splits the data referenced by the DataFrame proxy object CUSTOMER_DATA into two new DataFrame proxy objects train, and test. Run the following script.

```
TRAIN, TEST = CUSTOMER_DATA.split(ratio = (0.6,0.4),seed=1)
TRAIN_X = TRAIN.drop('HOME_THEATER_PACKAGE')
TRAIN_Y = TRAIN['HOME_THEATER_PACKAGE']
TEST_X = TEST
TEST_Y = TEST['HOME_THEATER_PACKAGE']
```

2. Run the following statement to view a few rows of the test dataset.

```
z.show(TRAIN)
```

3. To specify model settings and build a Random Forest model object for predicting the HOME_THEATER_PACKAGE attribute, run the following script. The settings are given as key-value or dictionary pairs where it refers to parameters name and value setting respectively. Here some of the settings specified are PREP_AUTO and RFOR_NUM_TREES . The Random Forest makes use of the Decision Tree settings to configure the construction of individual trees. The fit function builds the rf model according to the training data and parameter settings.

```
try:
    oml.drop(model = 'MODEL_RF')
except:
    pass

settings = {'PREP_AUTO': 'ON',
            'ALGO_NAME': 'ALGO_RANDOM_FOREST',
            'RFOR_NUM_TREES': '25'}

rf_mod = oml.rf(**settings)
rf_mod.fit(TRAIN_X, TRAIN_Y, case_id = 'CUST_ID', model_name = 'MODEL_RF')
```

Model setting parameters:

- RFOR_NUM_TREES: Denotes the number of trees the random forest can have.
- PREP_AUTO: Used to specify fully automated or user-directed general data preparation. By default, it is enabled with a constant value as 'PREP_AUTO': PREP_AUTO_ON. Alternatively, it can also be given as 'PREP_AUTO': 'ON'.

Note:

Any parameters or settings not specified are either system-determined or default values are used.

3.2.4 Evaluate

Before you make predictions using your model on new data, you should first evaluate model accuracy. You can evaluate the model using different methods.

Information about Model settings

Evaluate the model by examining the various statistics generated after building the model. The statistics indicate the model's quality.

- Run the following script for model details available through the Random Forest model object, like the model settings, coefficients, fit details, etc.

```
rf_mod
```

```
Model Name: MODEL_RF
```

```
Model Owner: OMLUSERDK
```

```
Algorithm Name: Random Forest
```

```
Mining Function: CLASSIFICATION
```

```
Target: HOME_THEATER_PACKAGE
```

```
Settings:
```

	setting name	setting value
0	ALGO_NAME	ALGO_RANDOM_FOREST
1	CLAS_MAX_SUP_BINS	32
2	CLAS_WEIGHTS_BALANCED	OFF
3	ODMS_DETAILS	ODMS_ENABLE
4	ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO
5	ODMS_RANDOM_SEED	0
6	ODMS_SAMPLING	ODMS_SAMPLING_DISABLE
7	PREP_AUTO	ON
8	RFOR_NUM_TREES	25
9	RFOR_SAMPLING_RATIO	.5
10	TREE_IMPURITY_METRIC	TREE_IMPURITY_GINI
11	TREE_TERM_MAX_DEPTH	16
12	TREE_TERM_MINPCT_NODE	.05
13	TREE_TERM_MINPCT_SPLIT	.1
14	TREE_TERM_MINREC_NODE	10

They can also be displayed and viewed individually as shown below.

- Run the following script to display the model's global statistics.

```
z.show(rf_mod.global_stats)
```

attribute name ↕	attribute value ↕
AVG_DEPTH	8.72AVG_NODECOUNT
MAX_DEPTH	10
MAX_NODECOUNT	89
MIN_DEPTH	10
MIN_NODECOUNT	35

- Run the following script to display the attribute importance of the `rf_mod` model.

```
z.show(rf_mod.importance)
```

ATTRIBUTE_NAME ↕	ATTRIBUTE_SUBNAME ↕	ATTRIBUTE_IMPORTANCE
CUST_CREDIT_LIMIT	None	1.570499020749605
CUST_INCOME_LEVEL	None	0.8044975606753664
HOUSEHOLD_SIZE	None	1.794760876267874
OCCUPATION	None	0.840247963640243

Score

Here you will make predictions on the test case using the model and then evaluate the model by using methods like Confusion Matrix, Lift Chart, Gains Chart, and ROC curve chart.

- Make predictions on the test data and add the `CASE_ID` as a supplemental column so you can uniquely associate scores with the original data. To do so run the below script:

```
# Set the case ID attribute
case_id = 'CUST_ID'
# Gather the Predictions
RES_DF = rf_mod.predict(TEST_X, supplemental_cols = TEST_X)
# Additionally collect the PROBABILITY_OF_0 and PROBABILITY_OF_1
RES_PROB = rf_mod.predict_proba(TEST_X, supplemental_cols =
TEST_X[case_id])
# Join the entire result into RES_DF
RES_DF = RES_DF.merge(RES_PROB, how = "inner", on = case_id, suffixes =
["", "_1"])
```

- To evaluate the model, pass a proxy object `oml.DataFrame` containing predictions and the target columns in a user-defined function named `evaluate_model`. Evaluate your model using standard metrics. For a classification example, you can evaluate your model using the following:

- **Confusion Matrix:** It displays the number and type of correct and incorrect predictions made with respect to the actual classification in the test data. It is an n-by-n matrix where n is the number of classes.
- **Lift Chart:** Applies only to binary classification requiring the designation of the positive class. It measures the degree to which the predictions of a classification model are better than randomly generated predictions.
- **ROC curve chart:** Applies to binary classification and requires the designation of the positive class. These are metrics for comparing predicted and actual target values in a classification model.

Run the below script to generate the metrics and charts:

```
def evaluate_model(pred_data='', settings_name='', name='', target=''):
    """Evaluate the models by passing an proxy oml.Dataframe containing
    Predictions
    and the target column,
    The Settings name (for the charts),
    The name of the model used (for the charts),
    Supply the target column name for evaluation
    for computing the confusion matrix with the test dataset"""
    import oml
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.metrics import auc
    from sklearn.metrics import roc_curve

    conf_matrix = pred_data.crosstab(target, 'PREDICTION', pivot=True)

    # Extract Statistics from the Confusion Matrix
    cf_local = conf_matrix.pull()
    TN = int(cf_local[cf_local[target]==0]['count_(0)'])
    FN = int(cf_local[cf_local[target]==0]['count_(1)'])
    TP = int(cf_local[cf_local[target]==1]['count_(1)'])
    FP = int(cf_local[cf_local[target]==1]['count_(0)'])
    TPR = TP/(TP+FN)
    FPR = FP/(FP+TN)
    TNR = TN/(TN+FP)
    FNR = FN/(FN+TP)
    Precision = TP/(TP+FP)
    Accuracy = (TP+TN)/(TP+TN+FP+FN)
    NPV = TN/(FN+TN)
    DetectionRate = TN/(FN+TN)
    BalancedAccuracy = (TPR+TNR)/2

    # Estimated AUC via Triangle (not very precise) could be
    # AUC = (1/2)*FPR*TPR + (1/2)*(1-FPR)*(1-TPR) + (1-FPR)*TPR
    # Compute real AUC using roc_curve by loading the
    # data locally and using the roc_curve() function
    pred_local = pred_data.pull()
    fpr, tpr, _ =
roc_curve(pred_local[[target]], pred_local[['PROBABILITY_OF_1']])
    AUC = auc(fpr, tpr)
    opt_index = np.argmax(tpr - fpr)
    FPR_OPT = fpr[opt_index]
    TPR_OPT = tpr[opt_index]
```

```

F1Score = 2*Precision*TPR/(Precision+TPR)
MathewsCorrCoef = ((TP*TN)-(FP*FN))/
((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))**0.5

# Store all statistics to export
statistics = {'Algorithm' : name,
             'Algorithm_setting' : settings_name,
             'TN' : TN,
             'TP' : TP,
             'FP' : FP,
             'FN' : FN,
             'TPR' : TPR,
             'FPR' : FPR,
             'TNR' : TNR,
             'FNR' : FNR,
             'Precision' : Precision,
             'Accuracy' : Accuracy,
             'NPV' : NPV,
             'DetectionRate' : DetectionRate,
             'BalancedAccuracy' : BalancedAccuracy,
             'AUC' : AUC,
             'F1Score' : F1Score,
             'MathewsCorrCoef' : MathewsCorrCoef
            }

# Nice round stats for printing to screen
TOTAL = TP+TN+FP+FN
TN_P = round((TN/TOTAL*100),2)
FP_P = round((FP/TOTAL*100),2)
FN_P = round((FN/TOTAL*100),2)
TP_P = round((TP/TOTAL*100),2)
# Print the output nicely on Zeppelin native Table
print("%table CONFUSION MATRIX\tPREDICTED 0\tPREDICTED 1\nACTUAL 0\t"+
      "True Negative: "+str(TN)+" (" +str(TN_P)+"%)\t"+
      "False Positive: "+str(FP)+" (" +str(FP_P)+"%)\nACTUAL 1\t"+
      "False Negative: "+str(FN)+" (" +str(FN_P)+"%)\t"+
      "True Positive: "+str(TP)+" (" +str(TP_P)+"%)\n"+
      "Accuracy: "+str(round(Accuracy*100,4))+"%\t"+
      "AUC: "+str(round(AUC,4))+"%\t"+
      "F1Score: "+str(round(F1Score,4))
      )

# Multiple Charts for Evaluation
fig, axes = plt.subplots(nrows=1, ncols=4,figsize=[22,5])
ax1, ax2, ax3, ax4 = axes.flatten()
fig.suptitle('Evaluation of the '+str(name)+' Model, with settings:
'+str(settings_name), size=16)

# Statistics
ax1.axis('off')

# Function to return rounded numbers if the string is float, return
# integers otherwise and return characters if not a number
def round_if_float(content):
    try:
        val = float(content)
    except ValueError:

```

```

        return(content)
    else:
        if val.is_integer():
            return(int(content))
        else:
            return(round(float(content),4))

for num, name in enumerate(statistics):
    ax1.text(0.01,
             (-num*0.06+0.94),
             "{0}: {1}".format(name,round_if_float(statistics[name])),
             ha='left',
             va='bottom',
             fontsize=12)

# Produce Lift Chart
ax2.set_title('Lift Chart')
data = pred_local.sort_values(by='PROBABILITY_OF_1', ascending=False)
data['row_id'] = range(0,0+len(data))
data['decile'] = ( data['row_id'] / (len(data)/10) ).astype(int)
lift = data.groupby('decile')[target].agg(['count','sum'])
lift.columns = ['count', target]
lift['decile'] = range(1,11)

data_ideal = pred_local.sort_values(by=target, ascending=False)
data_ideal['row_id'] = range(0,0+len(data))
data_ideal['decile'] = ( data_ideal['row_id'] /
(len(data_ideal)/10) ).astype(int)
lift_ideal = data_ideal.groupby('decile')[target].agg(['count','sum'])
lift_ideal.columns = ['count', 'IDEAL']
lift['IDEAL']=lift_ideal['IDEAL']

ax2.bar(lift['decile'],lift['IDEAL']/lift['count'],
color='darkorange', label='Ideal')
ax2.bar(lift['decile'],lift[target]/lift['count'],
color='blue', alpha=0.6, label='Model')
ax2.axhline((lift[target]/lift['count']).mean(),
color='grey', linestyle='--', label='Avg TARGET')
ax2.set_ylim(0,1.15)
ax2.set_xlabel('Decile', size=13)
ax2.set_ylabel('Percent of Actual Targets', size=13)
# Print labels.
for dec in lift['decile']:
    ax2.text(dec, lift[lift.decile==dec][target]/lift[lift.decile==dec]
['count'] + 0.05,
             ("%.0f" % int(round((lift[(lift.decile==dec)][target]/
lift[lift.decile==dec]['count'])*100,0))+"%",
             ha='center', va='bottom')
    ax2.legend(loc="upper right")

# Produce Gains Chart
ax3.set_title('Distributions of Predictions')
pred_local[pred_local[target]==1]['PROBABILITY_OF_1'].rename("Target =
1").plot(kind='density', bw_method=0.1, grid=True, ax=ax3)
pred_local[pred_local[target]==0]['PROBABILITY_OF_1'].rename("Target =
0").plot(kind='density', bw_method=0.1, grid=True, ax=ax3)

```

```

ax3.axvline(.5, color='grey', linestyle='--', label='Cutoff at 0.5')
ax3.set_xlim([0,1])
ax3.set_xlabel('Probability of 1', size=13)
ax3.set_ylabel('Density', size=13)
ax3.legend(loc="upper right")

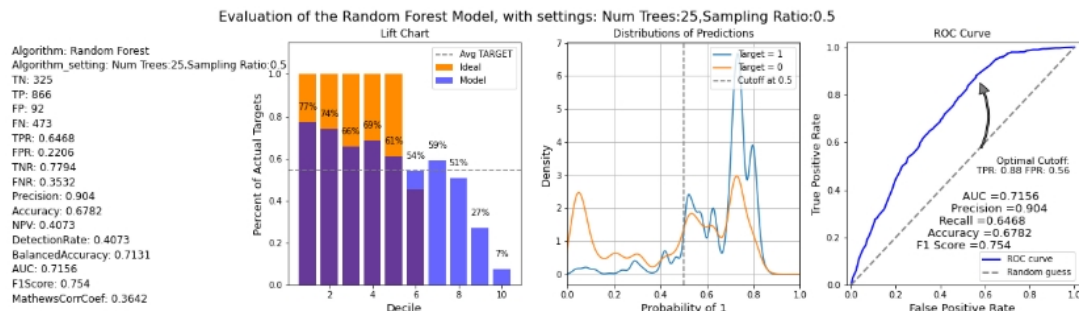
# ROC curve Chart
ax4.set_title('ROC Curve')
ax4.plot(fpr, tpr, color='blue', lw=2, label='ROC curve')
ax4.plot(FPR_OPT, TPR_OPT, color='orange', markersize=6)
ax4.plot([0, 1], [0, 1], lw=2, linestyle='--', color='grey',
label='Random guess')
ax4.annotate('Optimal Cutoff:\nTPR: '+str(round(TPR_OPT,2))+ ' FPR: '
'+str(round(FPR_OPT,2)),
            fontsize=11, xy=(FPR_OPT, TPR_OPT), xycoords='data',
xytext=(0.98, 0.54),
            textcoords='data',
            arrowprops=dict(facecolor='gray', shrink=0.1, width=2,
                            connectionstyle='arc3, rad=0.3'),
            horizontalalignment='right', verticalalignment='top')
ax4.annotate('AUC =' +str(round(AUC,4)), xy=(0.5, 0.35),
            xycoords='axes fraction', size=13)
ax4.annotate('Precision =' +str(round(Precision,4)), xy=(0.45, 0.3),
            xycoords='axes fraction', size=13)
ax4.annotate('Recall =' +str(round(TPR,4)), xy=(0.4, 0.25),
            xycoords='axes fraction', size=13)
ax4.annotate('Accuracy =' +str(round(Accuracy,4)), xy=(0.35, 0.2),
            xycoords='axes fraction', size=13)
ax4.annotate('F1 Score =' +str(round(F1Score,4)), xy=(0.3, 0.15),
            xycoords='axes fraction', size=13)
ax4.set_xlim([-0.02, 1.02])
ax4.set_ylim([0.0, 1.02])
ax4.set_xlabel('False Positive Rate', size=13)
ax4.set_ylabel('True Positive Rate', size=13)
ax4.legend(loc="lower right")

return(statistics, pred_local)

_ = evaluate_model(pred_data=RES_DF, settings_name='Num Trees:25,Sampling
Ratio:0.5', name='Random Forest', target='HOME_THEATER_PACKAGE')

```

CONFUSION MATRIX ↕	PREDICTED 0 ↕	PREDICTED 1 ↕
ACTUAL 0	True Negative: 325 (18.51%)	False Positive: 92 (5.24%)
ACTUAL 1	False Negative: 473 (26.94%)	True Positive: 866 (49.32%)
Accuracy: 67.8246%	AUC: 0.7156	F1Score: 0.754



- Display the results of customers responding to HOME_THEATER_PACKAGE with a probability greater than 0.5. Select the columns from the RES_DF dataset to display. To do so, run the following script:

```
z.show(RES_DF[RES_DF['PROBABILITY_OF_1'] > 0.5])
```

CUST_ID	CUST_INCOME_LEVEL	CUST_CREDIT_LIMIT	HOUSEHOLD_SIZE	OCCUPATION
100718	L: 300,000 and above	11000	3	Transp.
101020	K: 250,000 - 299,999	11000	3	Transp.
101900	C: 50,000 - 69,999	5000	3	Transp.
100229	D: 70,000 - 89,999	10000	3	Transp.
103311	E: 90,000 - 109,999	10000	3	Transp.

- Run the following script to get the model accuracy of the rf_mod. The score function makes prediction on the Test data and the target test data and gives the mean accuracy.

```
print("RF accuracy score = {:.2f}".format(rf_mod.score(TEST_X, TEST_Y)))
```

```
RF accuracy score = 0.68
```

You obtain an accuracy of 0.68 or approximately 68% of the result are correctly predicted.

To conclude, you have successfully identified customers who are likely to purchase HOME_THEATER_PACKAGE. This prediction helps to promote and offer home theater package to the target customers.

3.3 Clustering Use Case

A retail store has information about its customers' behavior and the purchases they make. Now with the available data, they would like you to analyze and identify if there are any similarities between the customers. Use Oracle Machine Learning to segment customers by finding clusters in the data set which can be then used to support targeted marketing campaigns to

increase retail sales. In this use case, you will learn how to identify such segments using the k-Means algorithm.

Related Contents

Table 3-1 Related Contents

Topic	Link
OML4Py GitHub Example	Clustering k-Means
About Clustering	About Clustering
Model Settings	About Model Settings
Shared Settings	Shared Settings
k-Means - Model Detail Views	Model Detail Views for k-Means

(Optional) Enter contextual information here, including the purpose of the task.

Before you start your OML4Py use case journey, ensure that you have the following:

- **Data Set**
The data set used for this use case is from the SH schema. The SH schema can be readily accessed in Oracle Autonomous Database. For on-premises databases, the schema is installed during the installation or can be manually installed by downloading the scripts. See [Installing the Sample Schemas](#).
- **Database**
Select or create database out of the following options:
 - Get your FREE cloud account. Go to <https://cloud.oracle.com/database> and select Oracle Database Cloud Service (DBCS), or Oracle Autonomous Database. Create an account and create an instance. See [Autonomous Database Quick Start Workshop](#).
 - Download the latest version of [Oracle Database](#) (on premises).
- **Machine Learning Tools**
Depending on your database selection,
 - Use OML Notebooks for Oracle Autonomous Database.
 - Install and use Oracle SQL Developer connected to an on-premises database or DBCS. See [Installing and Getting Started with SQL Developer](#).
- **Other Requirements**
Data Mining Privileges (this is automatically set for ADW). See [System Privileges for Oracle Machine Learning for SQL](#).
- **[Load Data](#)**
Access the data set from the SH Schema and explore the data to understand the attributes.
- **[Explore Data](#)**
Once the data is accessible, explore the data to understand and assess the quality of the data. At this stage assess the data to identify data types and noise in the data. Look for missing values and numeric outlier values.
- **[Build Model](#)**
To evaluate a model's performance, it is common practice to split the data into training and test sets. This allows you to assess how well the model generalizes to unseen data. However, in unsupervised learning, such as clustering, there are no labels or predictors available to calculate accuracy or evaluate performance. As a result, you can use the entire dataset to build the model without the need to split it. Since there is no ground truth

to compare the results against, the training-test split is neither applicable nor useful in unsupervised learning.

- **Evaluate**
Evaluate a model by assessing its performance using various metrics and techniques to determine how effectively it generalizes to new, unseen data. This process involves comparing predictions to actual outcomes using metrics like accuracy, precision, recall, F1 score, or mean squared error, depending on the model type. The evaluation helps identify the model's strengths and weaknesses, guiding further improvement or tuning.

Related Topics

- Create a Notebook
- Edit your Notebook
- Installing Sample Schemas

3.3.1 Load Data

Access the data set from the SH Schema and explore the data to understand the attributes.



Remember:

The data set used for this use case is from the SH schema. The SH schema can be readily accessed in Oracle Autonomous Database. For on-premises databases, the schema is installed during the installation or can be manually installed by downloading the scripts. See Installing the Sample Schemas.

To understand the data, you will perform the following:

- Access the data.
- Examine the various attributes or columns of the data set.
- Assess data quality (by exploring the data).

Access Data

You will use `CUSTOMERS` and `SUPPLEMENTARY_DEMOGRAPHICS` table data from the SH schema.

3.3.2 Explore Data

Once the data is accessible, explore the data to understand and assess the quality of the data. At this stage assess the data to identify data types and noise in the data. Look for missing values and numeric outlier values.

Assess Data Quality

To access database data from Python using OML4Py, you must first create a `oml.DataFrame` proxy object in Python which represents a database table, view, or query. Create a `oml.DataFrame` proxy object for `SUPPLEMENTARY_DEMOGRAPHICS` and `CUSTOMERS` and then merge them by inner join on a key column, e.g., `CUST_ID`. Assess the data to identify data types and noise in the data. Look for missing values, outlier numeric values, or inconsistently labeled categorical values.

The following steps help you with the exploratory analysis of the data:

1. Run the following script in a `%python` interpreter paragraph to import the `oml` modules, the Panda's module, and set the display options:

```
import pandas as pd
import matplotlib.pyplot as plt
import oml

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

2. Use the `oml.sync` function to create the Python object `DEMOGRAPHICS` as a proxy for a database table `SUPPLEMENTARY_DEMOGRAPHICS`. The `oml.sync` function returns an `oml.DataFrame` object. The `oml.DataFrame` object returned by `oml.sync` is a proxy for the database object.

 **Note:**

Only one environment for a given database schema can exist at a time. If "schema=None", then objects are created searched in the current user's schema.

```
DEMOGRAPHICS = oml.sync(table = "SUPPLEMENTARY_DEMOGRAPHICS", schema =
"SH")
z.show(DEMOGRAPHICS.head())
```

CUST_ID ↕	EDUCATION ↕	OCCUPATION ↕	HOUSEHOLD_SIZE ↕	YRS_RESIDENCE ↕	AFFINITY_CARD
102547	10th	Other	1	0	0
101050	10th	Other	1	0	0
100040	11th	Sales	1	0	0
102117	HS-grad	Farming	1	0	0
101074	10th	Handler	1	1	0

3. To determine the number of rows and columns in the `oml.DataFrame` object `DEMOGRAPHICS`, use `DataFrame.shape`.

```
print("Shape:", DEMOGRAPHICS.shape)

(4500, 14)
```

4. Use the `oml.sync` function to create the Python object `CUSTOMERS` as a proxy for a database table `SH.CUSTOMERS`. The query argument uses the SQL `SELECT` statement for selecting columns to include for use through the proxy object.c

```
CUSTOMERS = oml.sync(query = 'SELECT CUST_ID, CUST_GENDER,
CUST_MARITAL_STATUS, CUST_YEAR_OF_BIRTH, CUST_INCOME_LEVEL,
CUST_CREDIT_LIMIT FROM SH.CUSTOMERS')
z.show(CUSTOMERS.head())
```

CUST_ID ↕	CUST_GENDER ↕	CUST_MARITAL_STATUS ↕	CUST_YEAR_OF_BIRTH ↕	CUST_INCOME_LEVEL
100134	F	Divorc.	1965	L: 300,000 and above
102828	F	NeverM	1967	E: 90,000 - 109,999
101232	M	NeverM	1979	J: 190,000 - 249,999
100696	M	Married	1971	F: 110,000 - 129,999

5. To determine the number of rows and columns in the `oml.DataFrame` object `CUSTOMERS`, use `DataFrame.shape`.

```
print("Shape:",CUSTOMERS.shape)

(55500, 6)
```

6. Create a new `oml.DataFrame` `CUSTOMER_DATA` by merging the table `CUSTOMERS` and `DEMOGRAPHICS` with an inner join on the common column `CUST_ID`. The merge function joins one `oml.DataFrame` to another `oml.DataFrame`. The suffixes parameter is used when the two `oml.DataFrame` have conflicting column names.

```
CUSTOMER_DATA = CUSTOMERS[["CUST_ID", "CUST_GENDER",
"CUST_MARITAL_STATUS", "CUST_YEAR_OF_BIRTH", "CUST_INCOME_LEVEL",
"CUST_CREDIT_LIMIT"]].merge(DEMOGRAPHICS[["CUST_ID",
"HOUSEHOLD_SIZE","YRS_RESIDENCE", "Y_BOX_GAMES"]], how = "inner", on =
'CUST_ID',suffixes = ["", ""])
```

7. To determine the number of rows and columns in the `oml.DataFrame` object `CUSTOMER_DATA`, use `DataFrame.shape`.

```
print("Shape:",CUSTOMER_DATA.shape)

Shape: (4500, 9)
```

8. Use the `concat` function to concatenate the new column `CUST_AGE` in an `oml.DataFrame` object `CUSTOMER_DATA`. The column `CUST_AGE` contains the age based on the column `CUST_YEAR_OF_BIRTH` where the year of birth is converted to age in the year 2005. The information in the `CUST_YEAR_OF_BIRTH` column has been modified and maintained in `CUST_AGE`, so drop the `CUST_YEAR_OF_BIRTH` column.

```
CUSTOMER_DATA=CUSTOMER_DATA.concat({'CUST_AGE':abs(CUSTOMER_DATA['CUST_YEAR
_OF_BIRTH'] -2005)})
```

```
CUSTOMER_DATA=CUSTOMER_DATA.drop('CUST_YEAR_OF_BIRTH')
CUSTOMER_DATA.head()
```

	CUST_ID	CUST_GENDER	CUST_MARITAL_STATUS	CUST_INCOME_LEVEL	CUST_CREDIT_LIMIT	HOUSEHOLD_SIZE	YRS_RESIDENCE	Y_BOX_GAMES	CUST_AGE
0	100134	F	Divorc.	L: 300,000 and above	9000	2	2	0	40
1	102828	F	NeverM	E: 90,000 - 109,999	10000	1	4	0	38
2	101232	M	NeverM	J: 190,000 - 249,999	9000	1	2	1	26
3	100696	M	Married	F: 110,000 - 129,999	7000	3	3	0	34
4	103948	M	NeverM	J: 190,000 - 249,999	9000	1	4	0	39

- Run the following script to view the data type of each column.

```
print("The datatypes of the column: ", "\n")
print(CUSTOMER_DATA.dtypes)
```

The datatypes of the column:

```
CUST_ID                <class 'oml.core.float.Float'>
CUST_GENDER            <class 'oml.core.string.String'>
CUST_MARITAL_STATUS    <class 'oml.core.string.String'>
CUST_INCOME_LEVEL      <class 'oml.core.string.String'>
CUST_CREDIT_LIMIT      <class 'oml.core.float.Float'>
HOUSEHOLD_SIZE         <class 'oml.core.string.String'>
YRS_RESIDENCE          <class 'oml.core.float.Float'>
Y_BOX_GAMES            <class 'oml.core.integer.Integer'>
CUST_AGE               <class 'oml.core.integer.Integer'>
dtype: object
```

- To check if there are any missing values run the following script. The count function returns the number of elements that are not NULL for each column and the len() function returns the number of rows in the dataset.

```
print("Number of missing values in each column is : \n")
print(len(CUSTOMER_DATA)-CUSTOMER_DATA.count())
```

Number of missing values in each column is

```
CUST_ID                0
CUST_GENDER            0
CUST_MARITAL_STATUS    0
CUST_YEAR_OF_BIRTH     0
CUST_INCOME_LEVEL      0
CUST_CREDIT_LIMIT      0
HOUSEHOLD_SIZE         0
YRS_RESIDENCE          0
Y_BOX_GAMES            0
dtype: int64
```

- Use the crosstab method to perform a cross-column analysis of an oml.DataFrame object in the database. 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. In this example, the crosstab function

displays the distribution of unique values of CUST_CREDIT_LIMIT along the x-axis and its occurrence frequency along the y-axis.

```
z.show(CUSTOMER_DATA.crosstab('CUST_CREDIT_LIMIT'))
```

CUST_CREDIT_LIMIT ↕	count
1500	580
3000	415
5000	405
7000	556
9000	946

12. Use the transparency layer method `describe` to calculate descriptive statistics that summarize the central tendency, dispersion, and shape of the CUSTOMER_DATA table in each numeric column.

 **Note:**

All computations are computed in the database and only the result statistics are returned to the Python client, in this case, the notebook. Eliminating the need to move data and using the database as a high-performance compute engine greatly increases scalability.

```
CUSTOMER_DATA.describe()
```

	CUST_ID	CUST_CREDIT_LIMIT	YRS_RESIDENCE	Y_BOX_GAMES	CUST_AGE
count	4500.000000	4500.000000	4500.000000	4500.000000	4500.000000
mean	102250.500000	7924.222222	4.022000	0.312444	40.375556
std	1299.182435	3989.265430	1.901955	0.463541	13.679428
min	100001.000000	1500.000000	0.000000	0.000000	19.000000
25%	101125.750000	5000.000000	3.000000	0.000000	29.000000
50%	102250.500000	9000.000000	4.000000	0.000000	39.000000
75%	103375.250000	11000.000000	5.000000	1.000000	49.000000
max	104500.000000	15000.000000	14.000000	1.000000	92.000000

13. Before building the model, it's important to ensure that the data is clean. Data often contains outliers, which can form separate clusters that negatively impact model quality. The following script defines a function, `IQR`, to calculate the interquartile range for a dataframe. It takes two arguments: `SUMMARY_DF` (which contains summary statistics of

the dataframe, generated using the `describe` method) and a list of features. The IQR function uses a for loop to compute the interquartile range for each feature in the list.

Run the script to calculate the interquartile range for the specified columns:

```
def IQR(SUMMARY_DF, features):
    result = [0]*len(features)
    for i, feature in enumerate(features):
        result[i] = abs(SUMMARY_DF[feature]['75%'] - SUMMARY_DF[feature]
['25%'])
    return result

print(IQR(CUSTOMER_DATA.describe(), ['CUST_AGE', 'CUST_CREDIT_LIMIT',
'YRS_RESIDENCE', 'Y_BOX_GAMES']))

[20.0, 6000.0, 2.0, 1.0]
```

The user-defined function `remove_outlier` uses the interquartile range to find outliers in the data and remove them. In boxplot, outliers are points that lie outside of the upper and lower quartiles by 1.5 times the interquartile range ($Q1 - 1.5 * IQR$ or $Q3 + 1.5 * IQR$). Another form of outlier treatment is clipping or capping, where more extreme values are replaced with a max or min value, e.g., the 1.5 IQR values.

The following function removes rows with outliers of a given feature based on quantiles:

```
def remove_outlier(DF, SUMMARY_DF, features):
    iqrs = IQR(SUMMARY_DF, features)
    for i, iqr in enumerate(iqrs):
        H = 1.5*iqr
        DF = DF[ ( DF[features[i]] > SUMMARY_DF[features[i]]['25%'] - H )
& ( DF[features[i]] < SUMMARY_DF[features[i]]['75%'] + H )]
    print(DF.shape)
    return DF

CUSTOMER_DATA_CLEAN= remove_outlier(CUSTOMER_DATA,
CUSTOMER_DATA.describe(), ['CUST_AGE', 'CUST_CREDIT_LIMIT',
'YRS_RESIDENCE', 'Y_BOX_GAMES'])

print("Shape:",CUSTOMER_DATA_CLEAN.shape)

Shape: (4233, 9)
```

This completes the data understanding and data preparation stage.

3.3.3 Build Model

To evaluate a model's performance, it is common practice to split the data into training and test sets. This allows you to assess how well the model generalizes to unseen data. However, in unsupervised learning, such as clustering, there are no labels or predictors available to calculate accuracy or evaluate performance. As a result, you can use the entire dataset to

build the model without the need to split it. Since there is no ground truth to compare the results against, the training-test split is neither applicable nor useful in unsupervised learning.

Algorithm Selection

Using OML4Py, you can choose one of the following algorithms to solve a clustering problem:

1. Expectation-Maximization (EM)
2. K-Means (KM)

The Expectation-Maximization (EM) algorithm uses a probabilistic clustering based on a density estimation algorithm. The EM algorithm is used when data contains hidden components or when some data points are absent. In contrast, the *k*-Means (KM) algorithm is a distance-based clustering algorithm that partitions data into a specified number of clusters. Distance-based algorithms are based on the principle that nearby data points are more closely related to one another than to those that are farther away. This algorithm works iteratively to minimize the within-cluster variance in relation to the nearest cluster centroid.

The *k*-Means algorithm is chosen, as it is simpler than the Expectation-Maximization (EM) algorithm. Since, the optimal number of clusters is unknown, start with one cluster and gradually increase the number of clusters. Use the Elbow method to determine the optimal number of clusters.

To specify model settings and build a *k*-Means model object that will partition and segment the data, run the following script. The settings are provided as key-value pairs, or dictionary pairs, where each key represents a parameter name and its corresponding value represents the setting. Some of the specified settings include KMNS_ITERATIONS and KMNS_DISTANCE. The *k*-Means algorithm utilizes the number of clusters (*k*) along with these settings to configure the algorithm.

The following steps guide you to build your model with the selected algorithm.

- Use the `oml.km` algorithm to build your model and specify the model settings. Run the following script:

```
try:
    oml.drop(model="CUST_CLUSTER_MODEL")
except:
    pass

setting = {'KMNS_ITERATIONS': 10,
          'KMNS_DISTANCE': 'KMNS_EUCLIDEAN',
          'KMNS_NUM_BINS': 10,
          'KMNS_DETAILS': 'KMNS_DETAILS_ALL',
          'PREP_AUTO': 'ON'}

km_mod1 = oml.km(n_clusters = 1, **setting).fit(CUSTOMER_DATA_CLEAN,
model_name = "CUST_CLUSTER_MODEL", case_id = 'CUST_ID')
```

Examine the script:

- **KMNS_ITERATIONS:** Specifies the maximum number of allowed iterations, with a default of 20.
- **KMNS_DISTANCE:** Specify the type of distance functions used, by default distance function is the Euclidean distance.
- **KMNS_NUM_BINS:** Specifies the number of bins in the attribute histogram produced by *k*-Means.

- **KMNS_DETAILS:** Determines the level of cluster details that are computed during the build. The value `KMNS_DETAILS_ALL` indicates that the cluster hierarchy, record counts, and descriptive statistics such as variances, modes, histograms, and rules are computed.
- **PREP_AUTO:** Used for Automatic Data Preparation. By default, it is enabled as `'PREP_AUTO': PREP_AUTO_ON`, which requires the `DBMS_DATA_MINING` package. Alternatively, it can be set as `'PREP_AUTO': 'ON'`. This allows the compiler to validate that the PL/SQL constant name is correct.

3.3.4 Evaluate

Evaluate a model by assessing its performance using various metrics and techniques to determine how effectively it generalizes to new, unseen data. This process involves comparing predictions to actual outcomes using metrics like accuracy, precision, recall, F1 score, or mean squared error, depending on the model type. The evaluation helps identify the model's strengths and weaknesses, guiding further improvement or tuning.

Information and Model Settings



Note:

To get a complete list of information on the settings available in the k-Means module, run the below script:

```
help(oml.algo.km)
```

The following steps help you to view different model detail views.

- Use `km_mod1` to access the model details available through the k-Means model object, like the model settings, coefficients, fit details, and more.

```
km_mod1
```

```

Model Owner:

Algorithm Name: K-Means

Mining Function: CLUSTERING

Settings:
      setting name      setting value
0      ALGO_NAME        ALGO_KMEANS
1      CLUS_NUM_CLUSTERS          1
2      KMNS_CONV_TOLERANCE        .001
3      KMNS_DETAILS      KMNS_DETAILS_ALL
4      KMNS_DISTANCE      KMNS_EUCLIDEAN
5      KMNS_ITERATIONS          10
6      KMNS_MIN_PCT_ATTR_SUPPORT  .1
7      KMNS_NUM_BINS           10
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

```

- Use `km_mod1.clusters` to list the clusters information.

```
z.show(km_mod1.clusters)
```

CLUSTER_ID	ROW_CNT	PARENT_CLUSTER_ID	TREE_LEVEL	DISPERSION
1	4233	None	1	6.739342467

- Run the following script to display the model's centroid concerning each column. It gives mean, variance for a numeric attribute and mode for a categorical attribute.

```
z.show(km_mod1.centroids)
```

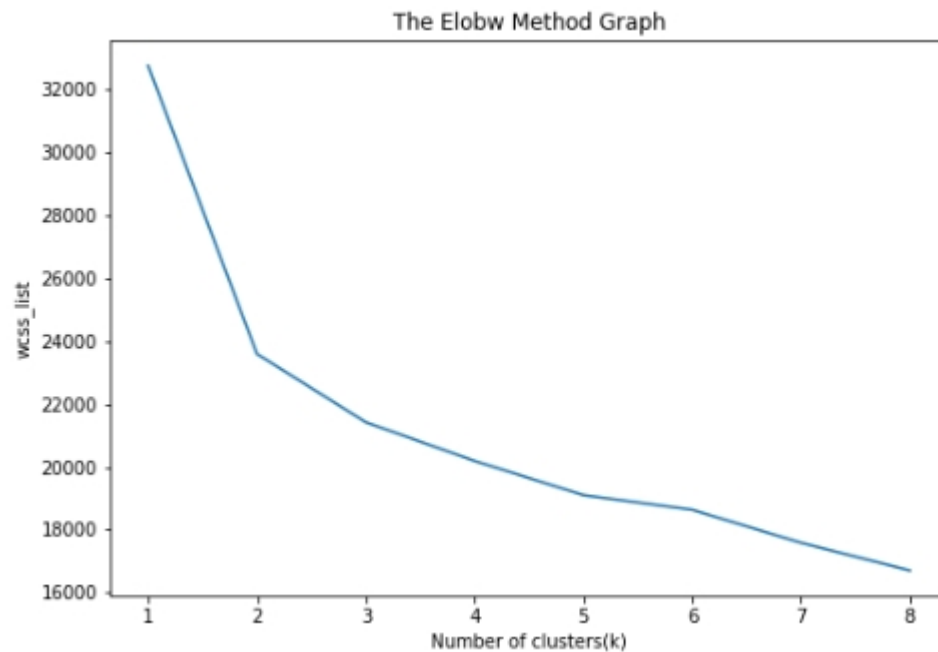
CLUSTER_ID	ATTRIBUTE_NAME	MEAN	MODE_VALUE	VARIANCE
1	CUST_AGE	39.35506732813605	None	160.730938
1	CUST_CREDIT_LIMIT	7956.886369005431	None	15893757.50
1	CUST_GENDER		M	
1	CUST_INCOME_LEVEL		J: 190,000 - 249,999	
1	CUST_MARITAL_STATUS		Married	

- To determine the optimal value of k you will use the Elbow method. Assuming that k lies in a given range, search for the best k by running k-Means over each k in the given range. For each k find the within-cluster variance. As the value of k increases within-cluster variance should decrease. This is because more centers mean that the data points are on

average at less distance from each other for a given centroid. Imagine that each data point is a center then within-cluster variance would be zero. Before the optimal value of k , the decrease in within-cluster-variance will be relatively large as new clusters will have increasingly closer data points within them. After the optimal value of k , the decrease will be slow as the new clusters formed will be similar to each other. The range of k should be chosen such that we can see the sharp decrease at optimal value and the slow decrease afterward resulting in an almost linear pattern being formed in the plot. The entire curve usually looks like an L shape and the best K lies in the turning point or the elbow of the L shape.

```
incluster_sum = []
for cluster in range(1, 9):
    setting = {'kmns_iterations': 15, 'KMNS_RANDOM_SEED': 1}
    km_mod2 = oml.km(n_clusters = cluster,
**setting).fit(CUSTOMER_DATA_CLEAN)
    incluster_sum.append(abs(km_mod2.score(CUSTOMER_DATA_CLEAN)))

plt.plot(range(1, 9),incluster_sum)
plt.title('The Elbow Method Graph')
plt.xlabel('Number of clusters(k)')
plt.ylabel('wcss_list')
plt.show()
```



The elbow joint or the number of optimal clusters can be observed at $k=3$.

- Build the final model according to the optimal value of clusters.

```
try:
    oml.drop(model="CUST_CLUSTER_MODEL")
except:
    pass

setting = {'KMNS_ITERATIONS': 20,
```

```

        'KMNS_DISTANCE': 'KMNS_EUCLIDEAN',
        'KMNS_NUM_BINS': 10,
        'KMNS_DETAILS': 'KMNS_DETAILS_ALL',
        'PREP_AUTO': 'ON'}
km_mod3 = oml.km(n_clusters = 3, **setting).fit(CUSTOMER_DATA_CLEAN,
model_name = "CUST_CLUSTER_MODEL", case_id = 'CUST_ID')

```

- Run the following script to display the model's clusters and the parent-child relationship in the cluster hierarchy.

```
z.show(km_mod3.clusters)
```

CLUSTER_ID	ROW_CNT	PARENT_CLUSTER_ID	TREE_LEVEL	DISPERSION
1	4233		1	4.027751954761431
2	2864	1	2	4.102983341996107
3	1369	1	2	3.870365035082749
4	1442	2	3	3.1837323525598094
5	1422	2	3	5.035163318625603

The clusters with cluster_id 3, 4, and 5 are the leaf clusters having 1369, 1442, and 1422 rows of data points in them. The cluster with cluster_id equal to 1 is the root of the binary tree as its parent_cluster_id is equal to nan.

- Run the following script to get only the parent/child relationship.

```
z.show(km_mod3.taxonomy)
```

PARENT_CLUSTER_ID	CHILD_CLUSTER_ID
1	2
1	3
2	4
2	5
3	

- To view the per cluster-attribute center (centroid) information of leaf clusters use the model attribute centroids which displays model statistics like mean, variance, and mode value for each cluster and attribute.

```
km_mod3.centroids[km_mod3.centroids["CLUSTER_ID"]>=3]
```

	CLUSTER_ID	ATTRIBUTE_NAME	MEAN	MODE_VALUE	VARIANCE
0	3	CUST_AGE	25.736304	None	1.157734e+01
1	3	CUST_CREDIT_LIMIT	8322.498174	None	1.575575e+07
2	3	CUST_GENDER	NaN	M	NaN
3	3	CUST_INCOME_LEVEL	NaN	J: 190,000 - 249,999	NaN
4	3	CUST_MARITAL_STATUS	NaN	NeverM	NaN
5	3	HOUSEHOLD_SIZE	NaN	1	NaN
6	3	YRS_RESIDENCE	2.333820	None	7.810264e-01
7	3	Y_BOX_GAMES	1.000000	None	0.000000e+00
8	4	CUST_AGE	46.255895	None	9.852503e+01
9	4	CUST_CREDIT_LIMIT	7729.195562	None	1.628244e+07
10	4	CUST_GENDER	NaN	M	NaN
11	4	CUST_INCOME_LEVEL	NaN	J: 190,000 - 249,999	NaN
12	4	CUST_MARITAL_STATUS	NaN	Married	NaN
13	4	HOUSEHOLD_SIZE	NaN	3	NaN

Cluster ID 3 has users with the highest mean for Y_BOX_GAMES and CUST_INCOME_LEVEL.

- For the model km_mod3, use the model attribute cluster_hists to view the cluster histogram details. To see the CUST_INCOME_LEVEL attribute's histogram details for Cluster ID 5.

```
mod_histogram=km_mod3.cluster_hists

z.show(mod_histogram[(mod_histogram['cluster.id']==3) &
    ((mod_histogram['variable']=='CUST_INCOME_LEVEL:A: Below 30,000') |
    (mod_histogram['variable']=='CUST_INCOME_LEVEL:B: 30,000 -
49,999') |
    (mod_histogram['variable']=='CUST_INCOME_LEVEL:C: 50,000 -
69,999') |
    (mod_histogram['variable']=='CUST_INCOME_LEVEL:D: 70,000 -
89,999') |
    (mod_histogram['variable']=='CUST_INCOME_LEVEL:E: 90,000 -
109,999') |
    (mod_histogram['variable']=='CUST_INCOME_LEVEL:F: 110,000 -
129,999') |
    (mod_histogram['variable']=='CUST_INCOME_LEVEL:G: 130,000 -
149,999') |
    (mod_histogram['variable']=='CUST_INCOME_LEVEL:H: 150,000 -
169,999') |
    (mod_histogram['variable']=='CUST_INCOME_LEVEL:I: 170,000 -
189,999') |
    (mod_histogram['variable']=='CUST_INCOME_LEVEL:J: 190,000 -
249,999') |
    (mod_histogram['variable']=='CUST_INCOME_LEVEL:K: 250,000 -
299,999') |
    (mod_histogram['variable']=='CUST_INCOME_LEVEL:L: 300,000 and
above'))])
```

cluster.id ↕	variable ↕	bin.id ↕	lower.bound ↕	upper.bound
3	CUST_INCOME_LEVEL:A: Below 30,000	1	None	None
3	CUST_INCOME_LEVEL:B: 30,000 - 49,999	2	None	None
3	CUST_INCOME_LEVEL:C: 50,000 - 69,999	3	None	None
3	CUST_INCOME_LEVEL:D: 70,000 - 89,999	4	None	None
3	CUST_INCOME_LEVEL:E: 90,000 - 109,999	5	None	None

This histogram groups cluster-id 3 into bins based on the CUST_INCOME_LEVEL. The bin with the highest count of customers earns salaries ranging from 190,000 to 249,999 annually.

- Check the support and confidence level of leaf clusters(3,4 and 5) using the model attribute rules which give the conditions for a case to be assigned with some probability to a cluster. Support and confidence are metrics that describe the relationships between clustering rules and cases. Support is the percentage of cases for which the rule holds. Confidence is the probability that a case described by this rule is assigned to the cluster.

```
km_mod3.rules
```

cluster.id ↕	rhs.support ↕	rhs.conf ↕	lhs.support ↕	lhs.conf ↕	lhs.var ↕	lhs.var.support
1	4233	1	3861	0.9121190644932672	CUST_AGE	3861
1	4233	1	3861	0.9121190644932672	CUST_AGE	3861
1	4233	1	4233	0.9121190644932672	CUST_CREDIT_LIMIT	3861
1	4233	1	4233	0.9121190644932672	CUST_CREDIT_LIMIT	3861
1	4233	1	4233	0.9121190644932672	CUST_GENDER	3861

The columns headers in the above data frame specify the following:

- cluster.id: The ID of a cluster in the model
- rhs.support: The record count
- rhs.conf: The record confidence
- lhs.support: The rule support
- lhs.conf: The rule confidence
- lhs.var: The attribute predicate name
- lhs.var.support: The attribute predicate support
- lhs.var.conf: The attribute predicate confidence
- predicate: The attribute predicate
- Run the following script to get the cluster id and the total number of data points present in each leaf node. The total number of data points at each tree level should always be conserved. The total number of data points present in the root node should equal the sum of all the data points present in the leaf nodes (1369+1442+1422=4233).

```
z.show(km_mod3.leaf_cluster_counts)
```

Score

The clusters discovered by k-Means are used to score a new record by estimating the probabilities that the new record belongs to each of the k clusters. The cluster with the highest probability is assigned to the record.

1. In this step, you will make predictions on the CUSTOMER_DATA_CLEAN and add the CUST_ID as a supplemental column so that you can uniquely associate scores with the data. To do so run the below script:

```
pred = km_mod3.predict(CUSTOMER_DATA_CLEAN, supplemental_cols =
CUSTOMER_DATA_CLEAN[["CUST_ID"]])
z.show(pred)
```

CUST_ID ↕	CLUSTER_ID
100134	5
102828	5
101232	3
100696	4
103948	5

2. To make predictions that return probability for each cluster on the data use predict_proba function.

```
pred = km_mod3.predict_proba(CUSTOMER_DATA_CLEAN, supplemental_cols =
CUSTOMER_DATA_CLEAN[["CUST_ID"]])
z.show(pred)
```

CUST_ID ↕	PROBABILITY_OF_3 ↕	PROBABILITY_OF_4 ↕	PROBABILITY_OF_5 ↕
100100	0.15813238770968768	0.37173314588610795	0.47013446640420437
100200	0.717076208547178	0.11908279462078367	0.16384099683203837
100300	0.14818781232209655	0.5018260144227255	0.3499861732551779
100400	0.0715221435473418	0.44410008587469424	0.4843777705779639
100500	0.722855164600747	0.11569166769553961	0.16145316770371337

3. With Embedded Python Execution, all the above tasks can be achieved. You can invoke user-defined Python functions in Python engines spawned and managed by the database

environment. Use the `oml.do_eval` function to run a user-defined input function that builds a k-Means model, scores records, and displays the results.

```
def build_km_1():

    setting = {'KMNS_ITERATIONS': 20,
              'KMNS_DISTANCE': 'KMNS_EUCLIDEAN',
              'KMNS_NUM_BINS': 10,
              'KMNS_DETAILS': 'KMNS_DETAILS_ALL',
              'PREP_AUTO': 'ON'}

    # Create a KM model object and fit it.
    try:
        oml.drop(model="CUST_CLUSTER_MODEL_EPE")
    except:
        pass
    km_mod_epe = oml.km(n_clusters = 3,
**setting).fit(CUSTOMER_DATA_CLEAN, model_name = "CUST_CLUSTER_MODEL_EPE",
case_id = 'CUST_ID')

    # Show model details.
    #km_mod_epe
    pred=(km_mod_epe.predict(CUSTOMER_DATA_CLEAN, supplemental_cols
=CUSTOMER_DATA_CLEAN[:, ['CUST_ID']]))
    return pred

z.show(oml.do_eval(func = build_km_1))
```

CUST_ID ↕	CLUSTER_ID
100134	5
102828	5
101232	3
100696	4
103948	5

- Run the following script to display the probability score of customer c1 with CUST_ID (102308) belonging to each cluster.

```
c1=CUSTOMER_DATA_CLEAN[CUSTOMER_DATA_CLEAN['CUST_ID']==102308]
km_mod3.predict_proba(c1, supplemental_cols =c1['CUST_ID'])
```

	CUST_ID	PROBABILITY_OF_3	PROBABILITY_OF_4	PROBABILITY_OF_5
0	102308	0.681144	0.125552	0.193304

To sell a new gaming product, you must target customers who have already purchased Y_BOX_GAMES and have a high credit limit. You have successfully segmented the population into different clusters and the cluster with cluster-id 3 has the target population with the greatest percentage of customers who have already purchased Y_BOX_GAMES, with a mean CUST_CREDIT_LIMIT of 8322. So, you can confidently target customers in cluster-id 3 to sell a new game product.

4

Reference

- [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.

4.1 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.
<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.

Class	Algorithm	Function of Algorithm	Description
<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	Feature extraction	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.
<code>oml.onnx</code>	Open Neural Network Exchange	Regression Classification Clustering Embedding	An ONNX-format model can be loaded into the database and used to score data using the prediction operators of Oracle Machine Learning. Although not an "algorithm", this supports models from multiple algorithms and machine learning techniques.

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.

4.2 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 4-1 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)
```

4.3 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 4-1 Shared Model Settings

Setting Name	Setting Value	Description
ODMS_DETAILS	ODMS_ENABLE	<p>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.</p> <p>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.</p> <p>If the value is ODMS_DISABLE, then model detail tables are not created and tables relevant to model details are also not created.</p> <p>The reduction in the space depends on the algorithm. Model size reduction can be on the order of 10x .</p>
	ODMS_DISABLE	
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	<p>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.</p> <p>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.</p> <p>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.</p> <p>The value ODMS_MISSING_VALUE_DELETE_ROW is applicable to all algorithms.</p>
	ODMS_MISSING_VALUE_MEAN_MODE	
	ODMS_MISSING_VALUE_DELETE_ROW	
ODMS_PARTITION_BUILD_TYPE	ODMS_PARTITION_BUILD_INTRA	<p>Controls the parallel building of partitioned models.</p> <p>ODMS_PARTITION_BUILD_INTRA builds each partition in parallel using all slaves.</p> <p>ODMS_PARTITION_BUILD_INTER builds each partition entirely in a single slave, but multiple partitions may be built at the same time because multiple slaves are active.</p> <p>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.</p>
	ODMS_PARTITION_BUILD_INTER	
	ODMS_PARTITION_BUILD_HYBRID	

Table 4-1 (Cont.) Shared Model Settings

Setting Name	Setting Value	Description
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	$0 < \text{value}$	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	$1 \leq \text{value}$	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.
PREP_SCALE_2DNUM	pprep_scale_stddev pprep_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].

Table 4-1 (Cont.) Shared Model Settings

Setting Name	Setting Value	Description
PREP_SCALE_NNUM	PREP_SCALE_MAXABS	This data preparation 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 data preparation 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.

Glossary

Index

A

algorithms

- machine learning, [4-1](#)

- settings common to all, [4-4](#)

C

classes

- machine learning, [4-1](#)

clustering

- use case, [3-36](#)

K

k-means algorithm, [3-36](#)

M

machine learning

- classes, [4-1](#)

models

- persisting, [4-1](#)

S

scoring new data, [4-1](#)

settings

- about model, [4-3](#)

- shared algorithm, [4-4](#)