2024-05-02

Oracle Machine Learning for Python API Reference

Release 2.0

CONTENTS:

Oracle Machine Learning for Python

A component of Oracle Machine Learning, Oracle Machine Learning for Python (OML4Py) makes the open source Python programming language and environment ready for enterprise in-database data. Designed for problems involving both large and small volumes of data, Oracle Machine Learning for Python integrates Python with Oracle Database. Python users can run Python commands and scripts for statistical, machine learning, and graphical analyses on data stored in Oracle Database. Python users can develop, refine, and deploy Python scripts that leverage the parallelism and scalability of Oracle Database to automate data analysis. Data analysts and data scientists can run Python modules and develop and operationalize Python scripts for machine learning applications in one step without having to learn SQL. Oracle Machine Learning for Python performs function pushdown to leverage the database as a high-performance compute engine for core Python and popular Python module functions.

CHAPTER

ONE

METHODS AND CLASSES

Oracle Machine Learning for Python Core Methods and Classes

Provides connection, data transfer, manipulation, processing, and access control.

1.1 Connection

Oracle Machine Learning for Python client components connect a Python session to an Oracle Database instance running the OML4Py server. The connection makes the data in a database schema available to the Python user. It also makes the processing power, memory, and storage capacities of the database server available to the Python session through the OML4Py client interface.

`oml.connect`([user, password, host, port, ...])	Establishes an Oracle Database connection.
`oml.disconnect`([cleanup])	Terminates the Oracle Database connection.
`oml.isconnected`([check_automl])	Indicates whether an active Oracle Database connection exists.
`oml.check_embed`()	Indicates whether embedded Python is set up in the connected Oracle Database.

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

Establishes an Oracle Database connection.

Just as with `cx_Oracle.connect()`, the user, password, and data source name can be provided separately or with host, port, sid or service_name.

There can be only one active connection. Calling this method when an active connection already exists replaces the active connection with a new one. This results in the previous connection being implicitly disconnected with the corresponding release of resources.

Parameters

user
[str or None (default)]

password
[str or None (default)]

host
[str or None (default)] Host name of the Oracle Database.

port
[int, str or None (default)] The Oracle Database port number.

sid
[str or None (default)] The Oracle Database SID.

service_name
[str or None (default)] The service name to be used in the connection identifier for the Oracle Database.

dsn
[str or None (default)] Data source name. The TNS entry of the database, or an TNS alias in the Oracle Wallet.

encoding
[str, 'UTF-8' (default)] Encoding to use for regular database strings.

nenconding
[str, 'UTF-8' (default)] Encoding to use for national character set database strings.

automl
[str, or bool or None (default)]

To enable automl, specify:

- True: if `host`, `port`, `sid` or `service_name` are specified and a connection pool is running for this (`host`, `port`, `sid` or `service_name`).
- Data source name: for a running connection pool if `dsn` is specified with a data source name.
- TNS alias in an Oracle Wallet: for a running connection pool if `dsn` is also specified with Wallet TNS alias.

Otherwise, automl is disabled.

Notes

- Parameters `sid` and `service_name` are exclusive.
- Parameters (`host`, `port`, `sid` or `service_name`), and `dsn` can only be specified exclusively.
- Parameter `user` and `password` must be provided when (`host`, `port`, `sid` or `service_name`) is specified, or `dsn` (and optionally `automl`) is specified with a data source name.
- Parameter `user` and `password` should be set to empty str "", when `dsn` (and optionally `automl`) is specified with Wallet TNS alias, to establish connection with Oracle Wallet.
- Automl requires Database Resident Connection Pooling (DRCP) running on the Database server.

`oml.`**`disconnect`**`(`*cleanup=True*`)`
Terminates the Oracle Database connection. By default, the OML objects created through this connection will be deleted.

Parameters

cleanup
[bool, True (default)] Cleans up OML objects defined in Python's main module before disconnecting from the database.

`oml.isconnected(check_automl=False)`
Indicates whether an active Oracle Database connection exists.

Parameters

check_automl: bool, False (default)
Indicates whether to check the connection is automl-enabled.

Returns

connected
[bool]

`oml.check_embed()`
Indicates whether embedded Python is set up in the connected Oracle Database.

Returns

embed_status
[bool or None] None when not connected.

Examples

Connect to database using dsn

```
>>> dsn = "(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST="+host+")(PORT="+port+
↪"))(CONNECT_DATA=(SERVICE_NAME="+service_name+")))"
>>> oml.connect(user="pyquser", password="pyquser", dsn=dsn)
>>> oml.isconnected()
True
>>> oml.isconnected(check_automl=True)
False
```

Connect to database using host, port, service_name

```
>>> oml.connect(user="pyquser", password="pyquser", host=host, port=port, service_
↪name=service_name)
>>> oml.isconnected()
True
```

```
>>> oml.isconnected(check_automl=True)
False
```

Disconnect from database

```
>>> oml.disconnect()
>>> oml.isconnected()
False
```

Connect to database to use automl

```
>>> dsn_pool = "(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST="+host+")(PORT="+port+
↪"))(CONNECT_DATA=(SERVICE_NAME="+service_name+")(SERVER=POOLED)))"
>>> oml.connect(user="pyquser", password="pyquser", dsn=dsn, automl=dsn_pool)
>>> oml.isconnected(check_automl=True)
True
```

```
>>> oml.disconnect()
```

Connect to database using host, port, service_name, with automl enabled

```
>>> oml.connect(user="pyquser", password="pyquser", host=host, port=port, service_
↪name=service_name, automl = True)
>>> oml.isconnected(check_automl=True)
True
```

Use connection alias ‘mycon1’ in Oracle Wallet

```
>>> oml.connect(user="", password="", dsn="mycon1")
>>> oml.isconnected()
True
```

Also use connection pool alias ‘mycon1_pool’ in Oracle Wallet to enable AutoML

```
>>> oml.connect(user="", password="", dsn="mycon1", automl="mycon1_pool")
>>> oml.isconnected(check_automl=True)
True
```

1.2 Data Transfer

With transparency layer functions you can connect to an Oracle Database instance and interact with data structures in a database schema. You can move data to and from the database and create database tables.

oml.create(x, table[, oranumber, dbtypes, . . .])	Creates a table in Oracle Database from a Python data set.
oml.push(x[, oranumber, dbtypes])	Pushes data into Oracle Database.
oml.sync([schema, regex_match])	Creates a DataFrame proxy object in Python that represents an Oracle Database data set.
oml.drop([table, view, model])	Drops a database table, view, or model.
oml.dir([schema])	Returns the names of OML objects in the workspace or the names of tables and views from specified schema
oml.comment(table[, column, comments, append])	Returns if the add/update of COMMENT was sucesfully if a COMMENT was specified if the COMMENT is ‘’ it will delete the COMMENT
oml.cursor()	Returns a cx_Oracle cursor object of the current OML database connection.

oml.**create**(*x*, *table*, *oranumber=True*, *dbtypes=None*, *append=False*, *comments=None*)
Creates a table in Oracle Database from a Python data set.

Parameters

x [pandas.DataFrame or a list of tuples of equal size] If `x` is a list of tuples of equal size, each tuple represents a row in the table. The column names are set to COL1, COL2, ... and so on.

table
[str] A name for the table.

oranumber
[bool, True (default)] If True, use SQL NUMBER for numeric columns. Otherwise, use BINARY_DOUBLE. Ignored if `append` is True.

dbtypes
[dict mapping str to str or list of str] A list of SQL types to use on the new table. If a list, its length should be equal to the number of columns. If a dict, the keys are the names of the columns. Ignored if `append` is True.

append
[bool, False (default)] Indicates whether to append the data to the existing table.

comments
[str - comment for table] list of str : Multiline comments for table

Returns

new_table
[oml.DataFrame] A proxy object that represents the newly-created table.

Notes

- When creating a new table, for columns whose SQL types are not specified in `dbtypes`, NUMBER is used for numeric columns when `oranumber` is True and BINARY_DOUBLE is used when `oranumber` is False. Users should set `oranumber` to False when the data contains NaN values. For string columns, the default type is VARCHAR2(4000), and for bytes columns, the default type is BLOB.
- When `x` is specified with an empty pandas.DataFrame, OML creates an empty table. NUMBER is used for numeric columns when `oranumber` is True and BINARY_DOUBLE is used when `oranumber` is False. VARCHAR2(4000) is used for columns of object dtype in the pandas.DataFrame.
- OML does not support columns containing values of multiple data types, data conversion is needed or a TypeError may be raised.
- OML determines default column types by looking at 20 random rows sampled from the table. For tables with less than 20 rows, all rows are used in column type determination. NaN values are considered as float type. If a column has all Nones, or has inconsistent data types that are not None in the sampled rows, a default column type cannot be determined, and a ValueError is raised unless a SQL type for the column is specified in `dbtypes`.

Examples

See *Data Transfer Pipeline*.

`oml.`**`push`**(*x*, *oranumber=True*, *dbtypes=None*)
Pushes data into Oracle Database.

Creates an internal table in Oracle Database and inserts the data into the table. The table exists as long as an OML object (either in the Python client or saved in the datastore) references the table.

Parameters

x [pandas.DataFrame or a list of tuples of equal size] If `x` is a list of tuples of equal size, each tuple represents a row in the table. The column names are set to COL1, COL2, . . . and so on.

oranumber
[bool] If True (default), use SQL NUMBER for numeric columns. Otherwise use BINARY_DOUBLE. Ignored if `append` is True.

dbtypes
[dict or list of str] The SQL data types to use in the table.

Returns

temp_table
[oml.DataFrame]

Notes

- When creating a new table, for columns whose SQL types are not specified in `dbtypes`, NUMBER is used for numeric columns when `oranumber` is True and BINARY_DOUBLE is used when `oranumber` is False. Users should set `oranumber` to False when the data contains NaN values. For string columns, the default type is VARCHAR2(4000), and for bytes columns, the default type is BLOB.
- When `x` is specified with an empty pandas.DataFrame, OML creates an empty table. NUMBER is used for numeric columns when `oranumber` is True and BINARY_DOUBLE is used when `oranumber` is False. VARCHAR2(4000) is used for columns of object dtype in the pandas.DataFrame.
- OML does not support columns containing values of multiple data types, data conversion is needed or a TypeError may be raised.
- OML determines default column types by looking at 20 random rows sampled from the table. For tables with less than 20 rows, all rows are used in column type determination. NaN values are considered as float type. If a column has all Nones, or has inconsistent data types that are not None in the sampled rows, a default column type cannot be determined, and a ValueError is raised unless a SQL type for the column is specified in `dbtypes`.

Examples

See *Data Transfer Pipeline*.

`oml.`**`sync`**(*schema=None*, *regex_match=False*, *table=None*, *view=None*, *query=None*)
Creates a DataFrame proxy object in Python that represents an Oracle Database data set.

The data set can be one of the following: a database table, view, or query.

Parameters

schema
[str or None (default)] The name of the schema where the database object exists; if None, then the current schema is used.

regex_match
[bool, False (default)] Synchronizes tables or views that match a regular expression. Ignored if `query` is used.

table, view, query
[str or None (default)] The name of a table, of a view, or of an Oracle SQL query to select from the database. When `regex_match` is True, this specifies the name pattern. Exactly one of these parameters must be a str and the other two must be None.

Returns

data_set
[oml.DataFrame, or if `regex_match` is used, returns] a dict of oml.DataFrame

Notes

When `regex_match` is True, synchronizes the matched tables or views to a dict with the table or view name as the key.

Examples

See *Data Transfer Pipeline*.

`oml.`**`drop`**`(`*table=None*, *view=None*, *model=None*`)`
Drops a database table, view, or model.

Parameters

table
[str or None (default)] The name of the table to drop.

view
[str or None (default)] The name of the view to drop.

model
[str or None (default)] The name of the model to drop.

Examples

See *Data Transfer Pipeline*.

`oml.`**`dir`**`(`*schema="*`)`
Returns the names of OML objects in the workspace or the names of tables and views from specified schema

Parameters

schema
[str, list of str, or None] str : The name of one schema example: 'SYS' list of str : Multiples schemas example: ['SYS','TEST'] None : Current schema is used

Returns

obj_names
[list of str] Names of oml objects in current Python workspace when schema is set to an empty string (default)

table_names
[list of tuples] (owner,name,type) of database tables and views that current user can access when schema is set to None or specified

Examples

See *Data Transfer Pipeline*.

`oml.comment(table, column=None, comments=None, append=False)`

Returns if the add/update of COMMENT was sucesfully if a COMMENT was specified if the COMMENT is ‘’ it will delete the COMMENT

or the COMMENT if not COMMENT was specified and only table/view and column were specified

Parameters

table
[str] The name of table to set comments on or get comments from.

column: str, None (default)
The name of column to set comments on or get comments from. By default set comments on or get comments from table.

comments: str, list of str, None (default)
Comments string or a list of strings to set multiline comments. By default get comments from the specified table or column.

append: boolean, False
Indicates whether to append the comments

Returns

result_comment
[str] The comments on the specified table or column when `comments` is None

Examples

See *Data Transfer Pipeline*.

`oml.cursor()`

Returns a cx_Oracle cursor object of the current OML database connection. It can be used to execute queries against Oracle Database.

Returns

cursor_obj
[a cx_Oracle cx:cursorobj.]

Examples

See *Data Transfer Pipeline*.

An example showing the complete OML Data Transfer pipeline

```
>>> import oml
>>> import pandas as pd
>>> x = pd.DataFrame({
...         'GENDER': ['M', 'M', 'F', 'M', 'F', 'M', 'F', 'F', None, 'F', 'M', 'F'],
...         'HAND': ['L', 'R', 'R', 'L', 'R', None, 'L', 'R', 'R', 'R', 'R', 'R'],
...         'SPEED': [40.5, 30.4, 60.8, 51.2, 54, 29.3, 34.1, 39.6, 46.4, 12, 25.3, 37.
↪5],
...         'ACCURACY': [.92, .94, .87, .9, .85, .97, .96, .93, .89, .84, .91, .95]
...         })
```

Create a table *DRIVER* from pandas DataFrame *x* specifying *oranumber* and *dbtypes* and return an oml.DataFrame

```
>>> oml_dr = oml.create(x, table='DRIVER', oranumber=False, dbtypes = {'HAND':
↪'CHAR(1)'})
>>> oml_dr.shape
(12, 4)
>>> oml_dr.dtypes
GENDER        <class 'oml.core.string.String'>
HAND          <class 'oml.core.string.String'>
SPEED           <class 'oml.core.float.Float'>
ACCURACY        <class 'oml.core.float.Float'>
dtype: object
>>> oml.dir()
['oml_dr']
```

Create and get comment on table *DRIVER* from Oracle Database

```
>>> oml.comment(table='DRIVER', column='GENDER', comments='some comments')
>>> oml.comment(table='DRIVER', column='GENDER')
'some comments'
```

Get column type information of table *DRIVER* from Oracle Database

```
>>> cr = oml.cursor()
>>> _ = cr.execute("select column_name, data_type from all_tab_columns where table_
↪name = 'DRIVER'")
>>> cr.fetchall()
[('GENDER', 'VARCHAR2'), ('HAND', 'CHAR'), ('SPEED', 'BINARY_DOUBLE'), ('ACCURACY',
↪'BINARY_DOUBLE')]
>>> cr.close()
```

Sync table *DRIVER* from Oracle Database into an oml.DataFrame

```
>>> oml_dr2 = oml.sync(table = 'DRIVER')
>>> oml_dr2.shape
(12, 4)
```

Push the pandas DataFrame *x* to an Oracle Database internal table

```
>>> oml_dr3 = oml.push(x)
>>> oml_dr3.shape
(12, 4)
>>> sorted(oml.dir())
['oml_dr', 'oml_dr2', 'oml_dr3']
```

Drop table *DRIVER* from Oracle Database

```
>>> oml.drop(table = 'DRIVER')
>>> del oml_dr, oml_dr2, oml_dr3
>>> oml.dir()
[]
```

1.3 Data Types

OML4Py supports both series objects of Boolean, Bytes, Float, and String and tabular objects of DataFrame.

oml.Boolean()	Boolean series data class.
oml.Bytes(other, dbtype)	Binary series data class.
oml.Float(other[, dbtype])	Numeric series data class.
oml.Integer(other)	Numeric series data class.
oml.String(other, dbtype)	Character series data class.
oml.DataFrame(other)	Tabular dataframe class.
oml.Datetime()	Datetime series data class.
oml.Timedelta()	Timedelta series data class.
oml.Timezone()	Timezone series data class.

1.3.1 oml.Boolean

class oml.**Boolean**

Boolean series data class.

Represents a single column of 0, 1, and NULL values in Oracle Database.

Attributes

`shape`	The dimensions of the data set.

Special Methods

__init__()	Initialize self.
__getitem__(key)	Index self.
__len__()	Returns number of rows.
__eq__(other)	Equivalent to `self == other`.
__ne__(other)	Equivalent to `self != other`.
__gt__(other)	Equivalent to `self > other`.
__ge__(other)	Equivalent to `self >= other`.
__lt__(other)	Equivalent to `self < other`.
__le__(other)	Equivalent to `self <= other`.

Special Method Examples

Methods

KFold([n_splits, seed, use_hash, nvl])	Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.
all()	Checks whether all elements in the Boolean series data object are True.
any()	Checks whether any elements in the Boolean series data object are True.
append(other[, all])	Appends the `other` OML data object of the same class to this data object.
concat(other[, auto_name])	Combines current OML data object with the `other` data objects column-wise.
count()	Returns the number of elements that are not NULL.
create_view([view, use_colname])	Creates an Oracle Database view for the data represented by the OML data object.
describe()	Generates descriptive statistics that summarize the central tendency, dispersion, and shape of an OML series data distribution.
drop_duplicates()	Removes duplicated elements.
dropna()	Removes missing values.
head([n])	Returns the first n elements.
isnull()	Detects the missing value None.
materialize([table])	Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.
max([skipna])	Returns the maximum value.
min([skipna])	Returns the minimum value.
nunique([dropna])	Returns the number of unique values.
pull()	Pulls data represented by this object from Oracle Database into an in-memory Python object.
sort_values([ascending, na_position])	Sorts the values in the series data object.
split([ratio, seed, use_hash, nvl])	Splits the series data object randomly into multiple sets.
tail([n])	Returns the last n elements.

__init__()

Initialize self. See help(type(self)) for accurate signature.

__getitem__(*key*)

Index self. Equivalent to `self[key]`.

Parameters

key

[oml.Boolean] Must be from the same data source as self.

Returns

subset

[same type as self] Contains only the rows satisfying the condition in `key`.

__len__()

Returns number of rows. Equivalent to `len(self)`.

Returns

rownum
[int]

`__eq__`(*other*)
Equivalent to `self == other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

equal
[oml.Boolean]

`__ne__`(*other*)
Equivalent to `self != other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

notequal
[oml.Boolean]

`__gt__`(*other*)
Equivalent to `self > other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterthan
[oml.Boolean]

`__ge__`(*other*)
Equivalent to `self >= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterequal
[oml.Boolean]

`__lt__`(*other*)
Equivalent to `self < other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessthan
[oml.Boolean]

`__le__`(*other*)
Equivalent to `self <= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series

can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessequal
[oml.Boolean]

KFold(*n_splits=3*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.

Parameters

n_splits
[int, 3 (default)] The number of folds. Must be greater than or equal to 2.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default):] If not None, the specified values are used to hash in place of Null.

Returns

kfold_data
[a list of pairs of series objects of the same type as caller] Each pair within the list is a fold. The first element of the pair is the train set, and the second element is the test set, which consists of all elements not in the train set.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

Examples
See *Boolean.split()*

all()
Checks whether all elements in the Boolean series data object are True.

Returns

all: bool

any()
Checks whether any elements in the Boolean series data object are True.

Returns

any: bool

append(*other*, *all=True*)

Appends the `other` OML data object of the same class to this data object.

Parameters

other
[An OML data object of the same class]

all
[boolean, True (default)] Keeps the duplicated elements from the two data objects.

Returns

appended
[type of caller] A new data object containing the elements from both objects.

Examples
See `Boolean.concat()`

concat(*other*, *auto_name=False*)

Combines current OML data object with the `other` data objects column-wise.

Current object and the `other` data objects must be combinable, that is, they both represent data from the same underlying database table, view, or query.

Parameters

other
[an OML data object, a list of OML data objects, or a dict mapping str to OML data objects.]

- OML data object: an OML series data object or an oml.DataFrame
- list: a sequence of OML series and DataFrame objects to concat.
- dict: a dict mapping str to OML series and DataFrame objects, the column name of concatenated OML series object is replaced with str, column names of concatenated OML DataFrame object is prefixed with str. Need to specify a `collections.OrderedDict` if the concatenation order is expected to follow the key insertion order.

auto_name
[boolean, False (default)] Indicates whether to automatically resolve conflict column names. If True, append duplicated column names with suffix `[column_index]`.

Returns

concat_table
[oml.DataFrame] An oml.DataFrame data object with its original columns followed by the columns of `other`.

Raises

ValueError

- If `other` is not a single nor a list/dict of OML objects, or if `other` is empty.

- If objects in `other` are not from same data source.
- If `auto_name` is False and duplicated column names are detected.

Notes

After concatenation is done, if there is any empty column names in the resulting oml.DataFrame, they will be renamed with `COL[column_index]`.

Examples

```
>>> import oml
>>> import pandas as pd
>>> from collections import OrderedDict
>>> oml_frame = oml.push(pd.DataFrame({"x":[1, 2, 3, 4, 5],
...                                    "y":[1, 3, 5, 2, 4]}))
>>> x = oml_frame["x"] > 3
>>> y = oml_frame["y"] > 3
>>> k = oml_frame["y"] < 2
>>> x
[False, False, False, True, True]
>>> y
[False, False, True, False, True]
>>> k
[True, False, False, False, False]
```

Concat oml.Boolean.

```
>>> z = x.concat(y)
>>> z
    COL1   COL2
0  False  False
1  False  False
2  False   True
3   True  False
4   True   True
```

Concat oml.Boolean and rename the concatenated column.

```
>>> x.concat({'y > 3':y})
    COL1  y > 3
0  False  False
1  False  False
2  False   True
3   True  False
4   True   True
```

Concat multiple OML data objects.

```
>>> x.concat([y, k])
    COL1   COL2   COL3
0  False  False   True
1  False  False  False
2  False   True  False
3   True  False  False
4   True   True  False
```

Concat multiple OML data objects and perform customized renaming.

```
>>> x.concat(OrderedDict([('y < 2', k), ('y > 3', y), ('New_', oml_
↪frame)]))
    COL1  y < 2  y > 3  New_x  New_y
0  False   True  False      1      1
1  False  False  False      2      3
2  False  False   True      3      5
3   True  False  False      4      2
4   True  False   True      5      4
```

Append oml.Boolean.

```
>>> x.append(y)
[False, False, False, True, True, False, False, True, False, True]
```

count()

Returns the number of elements that are not NULL.

Returns

nobs
[int]

Examples

See *Boolean.describe()*

create_view(*view=None*, *use_colname=False*)

Creates an Oracle Database view for the data represented by the OML data object.

Parameters

view
[str or None (default)] The name of a database view. If `view` is None, the created view is managed by OML and the view is automatically dropped when no longer needed. If a `view` is specified, then it is up to the user to drop the view.

use_colname
[bool, False (default)] Indicates whether to create the view with the same column names as the DataFrame object. Ignored if `view` is specified.

Returns

new_view
[oml.DataFrame]

Raises

TypeError

- If the object represents data from a temporary table or view, and the view to create is meant to persist past the current session.

Examples

See *Float.pull()*

describe()

Generates descriptive statistics that summarize the central tendency, dispersion, and shape of an OML series data distribution.

Returns

summary

[pandas.Series] Includes `count` (number of non-null entries), `unique` (number of unique entries), `top` (most common value), `freq`, (frequency of the most common value).

Examples

Set up a Boolean series data object.

```
>>> import oml
>>> import numpy as np
>>> oml_bool = oml.push(np.array([0, 0, 1, None, 1, 1, 1])) > 0
```

Get a general description of the data.

```
>>> oml_bool.describe()
count         6
unique        2
top        True
freq          4
dtype: object
```

Find the count of all not-Null values in the data.

```
>>> oml_bool.count()
6
```

Get the maximum value in the data.

```
>>> oml_bool.max()
True
```

Find the minimum value in the data.

```
>>> oml_bool.min()
False
```

drop_duplicates()

Removes duplicated elements.

Returns

deduplicated

[type of caller]

Examples

See *Boolean.dropna()*

dropna()

Removes missing values.

Missing values include None and/or nan if applicable.

Returns

dropped
[type of caller]

Examples
Setup.

```
>>> import oml
>>> oml_bool = oml.push([1, 2, 3, None, 5]) > 3
>>> oml_bool
[False, False, False, None, True]
```

Drop NA values

```
>>> oml_bool.dropna()
[False, False, False, True]
```

Drop duplicate values

```
>>> oml_bool.drop_duplicates().sort_values()
[False, True, None]
```

Number of unique values

```
>>> oml_bool.nunique()
2
>>> oml_bool.nunique(dropna=False)
3
```

Null values

```
>>> oml_bool.isnull()
[False, False, False, True, False]
```

head(*n=5*)
Returns the first n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_head
[type of caller]

Examples
See *Float.pull()*

isnull()
Detects the missing value None.

Returns

isnull
[oml.Boolean] Indicates missing value None for each element.

Examples

See `Boolean.dropna()`

materialize(*table=None*)

Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.

Parameters

table

[str or None (default)] The name of a table. If `table` is None, an OML-managed table is created, that is, OML drops the table when it is no longer used by any OML object or when you invoke `oml.disconnect(cleanup=True)` to terminate the database connection. If a table is specified, then it's up to the user to drop the named table.

Returns

new_table

[same type as self]

Examples

See `Float.pull()`

max(*skipna=True*)

Returns the maximum value.

Parameters

skipna

[boolean, True (default)] Excludes NaN values when computing the result.

Returns

max

[Python type corresponding to the column or numpy.nan]

Examples

See `Boolean.describe()`

min(*skipna=True*)

Returns the minimum value.

Parameters

skipna

[boolean, True (default)] Excludes NaN values when computing the result.

Returns

min

[Python type corresponding to the column or numpy.nan]

Examples

See `Boolean.describe()`

nunique(*dropna=True*)

Returns the number of unique values.

Parameters

dropna
[bool, True (default)] If True, NULL values are not included in the count.

Returns

nunique
[int]

Examples
See *Boolean.dropna()*

pull()
Pulls data represented by this object from Oracle Database into an in-memory Python object.

Returns

pulled_obj
[list of bool and None]

sort_values(*ascending=True*, *na_position='last'*)
Sorts the values in the series data object.

Parameters

ascending
[bool, True (default)] If True, sorts in ascending order. Otherwise, sorts in descending order.

na_position
[{'first', 'last'}, 'last' (default)] `first` places NANs and Nones at the beginning; `last` places them at the end.

Returns

sorted_obj
[type of caller]

Examples
See *Float.pull()*

split(*ratio=(0.7, 0.3)*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into multiple sets.

Parameters

ratio
[a list of float values or (0.7, 0.3) (default)] All the numbers must be positive and the sum of them are no more than 1. Each number represents the ratio of split data in one set.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default)] If not None, the specified values are used to hash in place of Null.

Returns

split_data
[a list of series objects of the same type as caller] Each of which contains the portion of data by the specified ratio.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

Examples

```
>>> import oml
>>> import numpy as np
>>> oml_bool = oml.push(np.random.randint(0, 2, 100)) > 0
```

Split into four equal-sized sets.

```
>>> splits = oml_bool.split(ratio = (.25, .25, .25, .25),
...                         seed = 5678, use_hash = False)
>>> [len(split) for split in splits]
[26, 21, 28, 25]
```

Split randomly into 4 consecutive folds

```
>>> folds = oml_bool.KFold(n_splits=4, use_hash=False)
>>> [(len(x[0]), len(x[1])) for x in folds]
[(69, 31), (79, 21), (76, 24), (76, 24)]
```

tail(*n=5*)
Returns the last n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_tail
[type of caller]

Examples
See *Float.pull()*

Special Method Examples

```
>>> import numpy as np
>>> oml_bool = oml.push(np.array([4.5, -100, 38, -1.32, 443])) > 0
>>> oml_bool
[True, False, True, False, True]
```

Get number of False entries in `oml_bool`.

```
>>> oml_bool == False
[False, True, False, True, False]
>>> oml_bool[oml_bool == False]
[False, False]
>>> len(oml_bool[oml_bool == False])
2
```

1.3.2 oml.Bytes

class oml.**Bytes**(*other*, *dbtype*)

Binary series data class.

Represents a single column of RAW or BLOB data in Oracle Database.

Attributes

`shape`	The dimensions of the data set.

Special Methods

`__init__`(other, dbtype)	Convert underlying Oracle Database type.
`__getitem__`(key)	Index self.
`__len__`()	Returns number of rows.
`__eq__`(other)	Equivalent to `self == other`.
`__ne__`(other)	Equivalent to `self != other`.
`__gt__`(other)	Equivalent to `self > other`.
`__ge__`(other)	Equivalent to `self >= other`.
`__lt__`(other)	Equivalent to `self < other`.
`__le__`(other)	Equivalent to `self <= other`.

Special Method Examples

Methods

`KFold`([n_splits, seed, use_hash, nvl])	Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.
`append`(other[, all])	Appends the `other` OML data object of the same class to this data object.

Continued on next page

Table 9 – continued from previous page

`concat`(other[, auto_name])	Combines current OML data object with the `other` data objects column-wise.
`count`()	Returns the number of elements that are not NULL.
`create_view`([view, use_colname])	Creates an Oracle Database view for the data represented by the OML data object.
`describe`()	Generates descriptive statistics that summarize the central tendency, dispersion, and shape of an OML series data distribution.
`drop_duplicates`()	Removes duplicated elements.
`dropna`()	Removes missing values.
`head`([n])	Returns the first n elements.
`isnull`()	Detects the missing value None.
`len`()	Computes the length of each byte string.
`materialize`([table])	Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.
`max`([skipna])	Returns the maximum value.
`min`([skipna])	Returns the minimum value.
`nunique`([dropna])	Returns the number of unique values.
`pull`()	Pulls data represented by this object from Oracle Database into an in-memory Python object.
`sort_values`([ascending, na_position])	Sorts the values in the series data object.
`split`([ratio, seed, use_hash, nvl])	Splits the series data object randomly into multiple sets.
`tail`([n])	Returns the last n elements.

`__init__`(*other*, *dbtype*)

Convert underlying Oracle Database type.

Parameters

other
[oml.Bytes]

dbtype
['raw' or 'blob']

`__getitem__`(*key*)

Index self. Equivalent to `self[key]`.

Parameters

key
[oml.Boolean] Must be from the same data source as self.

Returns

subset
[same type as self] Contains only the rows satisfying the condition in `key`.

`__len__`()

Returns number of rows. Equivalent to `len(self)`.

Returns

rownum
[int]

`__eq__`(*other*)
Equivalent to `self == other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

equal
[oml.Boolean]

`__ne__`(*other*)
Equivalent to `self != other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

notequal
[oml.Boolean]

`__gt__`(*other*)
Equivalent to `self > other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series

can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterthan
[oml.Boolean]

`__ge__`(*other*)
Equivalent to `self >= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterequal
[oml.Boolean]

`__lt__`(*other*)
Equivalent to `self < other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessthan
[oml.Boolean]

`__le__`(*other*)
Equivalent to `self <= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.

- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessequal
[oml.Boolean]

KFold(*n_splits=3*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.

Parameters

n_splits
[int, 3 (default)] The number of folds. Must be greater than or equal to 2.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default):] If not None, the specified values are used to hash in place of Null.

Returns

kfold_data
[a list of pairs of series objects of the same type as caller] Each pair within the list is a fold. The first element of the pair is the train set, and the second element is the test set, which consists of all elements not in the train set.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

Examples
See *Bytes.split()*

append(*other*, *all=True*)
Appends the `other` OML data object of the same class to this data object.

Parameters

other
[An OML data object of the same class]

all
[boolean, True (default)] Keeps the duplicated elements from the two data objects.

Returns

appended
[type of caller] A new data object containing the elements from both objects.

Examples
See `Bytes.concat()`

concat(*other*, *auto_name=False*)
Combines current OML data object with the `other` data objects column-wise.

Current object and the `other` data objects must be combinable, that is, they both represent data from the same underlying database table, view, or query.

Parameters

other
[an OML data object, a list of OML data objects, or a dict mapping str to OML data objects.]

- OML data object: an OML series data object or an oml.DataFrame
- list: a sequence of OML series and DataFrame objects to concat.
- dict: a dict mapping str to OML series and DataFrame objects, the column name of concatenated OML series object is replaced with str, column names of concatenated OML DataFrame object is prefixed with str. Need to specify a `collections.OrderedDict` if the concatenation order is expected to follow the key insertion order.

auto_name
[boolean, False (default)] Indicates whether to automatically resolve conflict column names. If True, append duplicated column names with suffix `[column_index]`.

Returns

concat_table
[oml.DataFrame] An oml.DataFrame data object with its original columns followed by the columns of `other`.

Raises

ValueError

- If `other` is not a single nor a list/dict of OML objects, or if `other` is empty.
- If objects in `other` are not from same data source.
- If `auto_name` is False and duplicated column names are detected.

Notes

After concatenation is done, if there is any empty column names in the resulting oml.DataFrame, they will be renamed with `COL[column_index]`.

Examples

```
>>> import oml
>>> import pandas as pd
>>> from collections import OrderedDict
>>> import numpy as np
>>> import pickle
>>> np.random.seed(1234)
>>> oml_frame = oml.push(pd.DataFrame({"x":[pickle.dumps(x) for x in
...                                     np.random.randint(0, 1000,␣
↪100)],
...                                     "y":[pickle.dumps(x) for x in
...                                     np.random.randint(0, 1000,␣
↪100)]}))
>>> oml_x = oml_frame["x"]
>>> oml_y = oml_frame["y"]
```

Concat oml.Bytes.

```
>>> z = oml_x.concat(oml_y)
>>> z.head(n=2)
                                                x  \
0  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
1  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...

                                                y
0  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
1  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
>>> z.shape
(100, 2)
```

Concat oml.Bytes and rename the concatenated column.

```
>>> oml_x.concat({'Bytes_y':oml_y}).head()
                                                x  \
0  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
1  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
2  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
3  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
4  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...

                                          Bytes_y
0  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
1  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
2  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
3  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
4  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
```

Concat multiple OML data objects and turn on automatic name conflict resolution.

```
>>> oml_x.concat([oml_frame, oml_y], auto_name=True).head()
                                                x  \
0  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
1  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
2  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
3  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
4  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...

                                               x2  \
```

(continues on next page)

(continued from previous page)

```
0  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
1  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
2  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
3  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
4  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...

                                                   y   \
0  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
1  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
2  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
3  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
4  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...

                                                  y4
0  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
1  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
2  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
3  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
4  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
```

Concat multiple OML data objects and perform customized renaming.

```
>>> oml_x.concat(OrderedDict([('Bytes_y',oml_y), ('New_',oml_
↪frame)])).head()
                                                   x   \
0  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
1  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
2  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
3  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
4  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...

                                             Bytes_y   \
0  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
1  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
2  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
3  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
4  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...

                                               New_x   \
0  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
1  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
2  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
3  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
4  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...

                                               New_y
0  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
1  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
2  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
3  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
4  b'\x80\x04\x95i\x00\x00\x00\x00\x00\x00\x00\x8...
```

Append oml.Bytes.

```
>>> z = oml_x.append(oml_y)
>>> z.shape
(200, 1)
```

count()

Returns the number of elements that are not NULL.

Returns

nobs
[int]

Examples
See `Bytes.describe()`

create_view(*view=None*, *use_colname=False*)

Creates an Oracle Database view for the data represented by the OML data object.

Parameters

view
[str or None (default)] The name of a database view. If `view` is None, the created view is managed by OML and the view is automatically dropped when no longer needed. If a `view` is specified, then it is up to the user to drop the view.

use_colname
[bool, False (default)] Indicates whether to create the view with the same column names as the DataFrame object. Ignored if `view` is specified.

Returns

new_view
[oml.DataFrame]

Raises

TypeError

- If the object represents data from a temporary table or view, and the view to create is meant to persist past the current session.

Examples
See `String.pull()`

describe()

Generates descriptive statistics that summarize the central tendency, dispersion, and shape of an OML series data distribution.

Returns

summary
[`pandas.Series`] Includes `count` (number of non-null entries), `unique` (number of unique entries), `top` (most common value), `freq`, (frequency of the most common value).

Examples
Set up a Bytes series data object.

```
>>> import oml
>>> import pandas as pd
>>> df = pd.DataFrame({'bytes' : [b'a', b'b', b'c', b'c', b'd', b'e
↪']})
```

(continues on next page)

(continued from previous page)

```
>>> oml_df = oml.push(df, oranumber = False, dbtypes = 'RAW(5)')
>>> oml_byte = oml_df['bytes']
```

Get a general description of the data.

```
>>> oml_byte.describe()
count        6
unique       5
top       b'c'
freq         2
Name: bytes, dtype: object
```

Find the count of not-Null values in the data.

```
>>> oml_byte.count()
6
```

Get the maximum value in the data.

```
>>> oml_byte.max()
b'e'
```

Find the minimum value in the data.

```
>>> oml_byte.min()
b'a'
```

drop_duplicates()

Removes duplicated elements.

Returns

deduplicated
[type of caller]

dropna()

Removes missing values.

Missing values include None and/or nan if applicable.

Returns

dropped
[type of caller]

Examples

Setup.

```
>>> import oml
>>> oml_bytes = oml.push([b'a', None, b'c', None, b'd', b'a'])
>>> oml_bytes
[b'a', None, b'c', None, b'd', b'a']
```

Drop NA values

```
>>> oml_bytes.dropna()
[b'a', b'c', b'd', b'a']
```

Null values

```
>>> oml_bytes.isnull()
[False, True, False, True, False, False]
```

head(*n=5*)

Returns the first n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_head
[type of caller]

Examples
See `String.pull()`

isnull()

Detects the missing value None.

Returns

isnull
[oml.Boolean] Indicates missing value None for each element.

Examples
See `Bytes.dropna()`

len()

Computes the length of each byte string.

Returns

length
[oml.Float]

materialize(*table=None*)

Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.

Parameters

table
[str or None (default)] The name of a table. If `table` is None, an OML-managed table is created, that is, OML drops the table when it is no longer used by any OML object or when you invoke `oml.disconnect(cleanup=True)` to terminate the database connection. If a table is specified, then it's up to the user to drop the named table.

Returns

new_table
[same type as self]

Examples

See `String.pull()`

`max(skipna=True)`

Returns the maximum value.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

max
[Python type corresponding to the column or numpy.nan]

Examples

See `Bytes.describe()`

`min(skipna=True)`

Returns the minimum value.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

min
[Python type corresponding to the column or numpy.nan]

Examples

See `Bytes.describe()`

`nunique(dropna=True)`

Returns the number of unique values.

Parameters

dropna
[bool, True (default)] If True, NULL values are not included in the count.

Returns

nunique
[int]

`pull()`

Pulls data represented by this object from Oracle Database into an in-memory Python object.

Returns

pulled_obj
[list of bytes and None]

sort_values (*ascending=True*, *na_position='last'*)
Sorts the values in the series data object.

Parameters

ascending
[bool, True (default)] If True, sorts in ascending order. Otherwise, sorts in descending order.

na_position
[{'first', 'last'}, 'last' (default)] `first` places NANs and Nones at the beginning; `last` places them at the end.

Returns

sorted_obj
[type of caller]

Examples
See `String.pull()`

split (*ratio=(0.7, 0.3)*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into multiple sets.

Parameters

ratio
[a list of float values or (0.7, 0.3) (default)] All the numbers must be positive and the sum of them are no more than 1. Each number represents the ratio of split data in one set.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default)] If not None, the specified values are used to hash in place of Null.

Returns

split_data
[a list of series objects of the same type as caller] Each of which contains the portion of data by the specified ratio.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

Examples

```
>>> import oml
>>> import numpy as np
>>> import pickle
>>> np.random.seed(1234)
>>> oml_bytes = oml.push([pickle.dumps(x) for x in np.random.
↪randint(0, 1000, 100)])
```

Split into four equal-sized sets.

```
>>> splits = oml_bytes.split(ratio = (.25, .25, .25, .25), seed =␣
↪1234,
...                               use_hash = False)
>>> [len(split) for split in splits]
[26, 31, 17, 26]
```

Split randomly into 4 consecutive folds

```
>>> folds = oml_bytes.KFold(n_splits=4, use_hash = False)
>>> [(len(x[0]), len(x[1])) for x in folds]
[(69, 31), (79, 21), (76, 24), (76, 24)]
```

tail(*n=5*)

Returns the last n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_tail
[type of caller]

Examples

See *String.pull()*

Special Method Examples

```
>>> import pandas as pd
>>> col1 = [b'#w9', b'r4)', b'D+i', b'NV%']
>>> col2 = [b'2,>', b'E3C', b'-0"', b'i19']
>>> df = oml.push(pd.DataFrame({'col1' : col1, 'col2': col2}))
```

Find the length of the series.

```
>>> len(df['col1'])
4
```

Convert underlying Oracle Database type to raw. Then return the elements in `rawcol1` that are lexicographically before their counterparts in `rawcol2`.

```
>>> rawcol1 = oml.Bytes(df['col1'], 'raw')
>>> rawcol2 = oml.Bytes(df['col2'], 'raw')
>>> rawcol1 < rawcol2
```

(continues on next page)

(continued from previous page)

```
[True, False, False, True]
>>> rawcol1[rawcol1 < rawcol2]
[b'#w9', b'NV%']
```

1.3.3 oml.Float

class oml.**Float**(*other*, *dbtype=None*)

Numeric series data class.

Represents a single column of NUMBER, BINARY_DOUBLE or BINARY_FLOAT data in Oracle Database.

Attributes

`shape`	The dimensions of the data set.

Special Methods

`__init__`(other[, dbtype])	Convert to oml.Float, or convert underlying Oracle Database type.
`__contains__`(item)	Check whether all elements in `item` exists in the Float series
`__getitem__`(key)	Index self.
`__len__`()	Returns number of rows.
`__eq__`(other)	Equivalent to `self == other`.
`__ne__`(other)	Equivalent to `self != other`.
`__gt__`(other)	Equivalent to `self > other`.
`__ge__`(other)	Equivalent to `self >= other`.
`__lt__`(other)	Equivalent to `self < other`.
`__le__`(other)	Equivalent to `self <= other`.
`__add__`(other)	Equivalent to `self + other`.
`__sub__`(other)	Equivalent to `self - other`.
`__mul__`(other)	Equivalent to `self * other`.
`__truediv__`(other)	Equivalent to `self / other`.
`__floordiv__`(other)	Equivalent to `self // other`.
`__pow__`(other)	Equivalent to `self ** other`.
`__mod__`(other)	Equivalent to `self % other`.
`__divmod__`(other)	Equivalent to `divmod(self, other)`.
`__abs__`()	Return the absolute value of every element in `self`.
`__neg__`()	Return the negation of every element in `self`.
`__matmul__`(other)	Equivalent to `self @ other` and `self.dot(other)`.

Special Method Examples

Methods

KFold([n_splits, seed, use_hash, nvl])	Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.
append(other[, all])	Appends the other OML data object of the same class to this data object.
ceil()	Returns the ceiling of each element in the Float series data object.
concat(other[, auto_name])	Combines current OML data object with the other data objects column-wise.
count()	Returns the number of elements that are not NULL.
create_view([view, use_colname])	Creates an Oracle Database view for the data represented by the OML data object.
cumsum([ascending, na_position, skipna])	Gets the cumulative sum after the OML series data object is sorted.
cut(bins[, right, labels, retbins, ...])	Returns the indices of half-open bins to which each value belongs.
describe([percentiles])	Generates descriptive statistics that summarize the central tendency, dispersion, and shape of the OML series data distribution.
dot([other, skipna])	Returns the inner product with an oml.Float.
drop_duplicates()	Removes duplicated elements.
dropna()	Removes missing values.
exp()	Returns element-wise e to the power of values in the Float series data object.
floor()	Returns the floor of each element in the Float series data object.
head([n])	Returns the first n elements.
isinf()	Detects infinite values element-wise in the Float series data object.
isnan()	Detects a NaN (not a number) element from Float object.
isnull()	Detects the missing value None.
kurtosis([skipna])	Returns the sample kurtosis of the values.
log([base])	Returns element-wise logarithm, to the given base, of values in the Float series data object.
materialize([table])	Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.
max([skipna])	Returns the maximum value.
mean([skipna])	Returns the mean of the values.
median([skipna])	Returns the median of the values.
min([skipna])	Returns the minimum value.
nunique([dropna])	Returns the number of unique values.
pull()	Pulls data represented by the series data object from Oracle Database into an in-memory Python object.
replace(old, new[, default])	Replace values given in *old* with *new*.
round([decimals])	Rounds oml.Float values to the specified decimal place.
skew([skipna])	Returns the sample skewness of the values.
sort_values([ascending, na_position])	Sorts the values in the series data object.
split([ratio, seed, use_hash, nvl])	Splits the series data object randomly into multiple sets.

Continued on next page

Table 12 – continued from previous page

sqrt()	Returns the square root of each element in the Float series data object.
std([skipna])	Returns the sample standard deviation of the values.
sum([skipna])	Returns the sum of the values.
tail([n])	Returns the last n elements.

__init__(*other*, *dbtype=None*)

Convert to oml.Float, or convert underlying Oracle Database type.

Parameters

other
[oml.Boolean, oml.Integer, or oml.Float]

- oml.Boolean : initialize a oml.Float object that has value 1 (resp. 0) wherever `other` has value True (resp. False).
- oml.Integer : initialize a oml.Float object with the same value of oml.Integer object as `other`.
- oml.Float : initialize a oml.Float object with the same data as `other`, except the underlying Oracle Database type has been converted to the one specified by `dbtype`.

dbtype
['number' or 'binary_double'] Ignored if `other` is type `oml.Boolean`. Must be specified if `other` is type `oml.Float`.

__contains__(*item*)

Check whether all elements in `item` exists in the Float series

Equivalent to `item in self`.

Parameters

item
[int/float, list of int/float, oml.Float] Values to check in series

Returns

contains
[bool] Returns *True* if all elements exists, otherwise *False*

__getitem__(*key*)

Index self. Equivalent to `self[key]`.

Parameters

key
[oml.Boolean] Must be from the same data source as self.

Returns

subset
[same type as self] Contains only the rows satisfying the condition in `key`.

__len__()

Returns number of rows. Equivalent to `len(self)`.

Returns

rownum
[int]

`__eq__`(*other*)
Equivalent to `self == other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

equal
[oml.Boolean]

`__ne__`(*other*)
Equivalent to `self != other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

notequal
[oml.Boolean]

`__gt__`(*other*)
Equivalent to `self > other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series

can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterthan
[oml.Boolean]

`__ge__`(*other*)
Equivalent to `self >= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterequal
[oml.Boolean]

`__lt__`(*other*)
Equivalent to `self < other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessthan
[oml.Boolean]

`__le__`(*other*)
Equivalent to `self <= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.

- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessequal
[oml.Boolean]

`__add__`(*other*)
Equivalent to `self + other`.

Parameters

other
[int, oml.Integer, float, or oml.Float]

- scalar : add the scalar to each element in `self`.
- oml.Float : must come from the same data source. Add corresponding elements in `self` and `other`.
- oml.Integer : must come from the same data source. Add corresponding elements in `self` and `other`.

Returns

sum
[oml.Float]

`__sub__`(*other*)
Equivalent to `self - other`.

Parameters

other
[int, oml.Integer, float, or oml.Float]

- scalar : subtract the scalar from each element in `self`.
- oml.Float : must come from the same data source. From each element in `self`, subtract the corresponding element in `other`.
- oml.Integer : must come from the same data source. From each element in `self`, subtract the corresponding element in `other`.

Returns

difference
[oml.Float]

`__mul__`(*other*)
Equivalent to `self * other`.

Parameters

other
[int, oml.Integer, float, or oml.Float]

- scalar : multiply the scalar with each element in `self`.
- oml.Float : must come from the same data source. Multiply corresponding elements in `self` and `other`.
- oml.Integer : must come from the same data source. Multiply corresponding elements in `self` and `other`.

Returns

product
[oml.Float]

`__truediv__`(*other*)
Equivalent to `self / other`.

Parameters

other
[int, oml.Integer, float, or oml.Float]

- scalar : divide each element in `self` by the scalar.
- oml.Float : must come from the same data source. Divide each element in `self` by the corresponding element in `other`.
- oml.Integer : must come from the same data source. Divide each element in `self` by the corresponding element in `other`.

Returns

quotient
[oml.Float]

`__floordiv__`(*other*)
Equivalent to `self // other`.

Parameters

other
[int, oml.Integer, float, or oml.Float]

- scalar : divide each element in `self` by the scalar.
- oml.Float : must come from the same data source. Divide each element in `self` by the corresponding element in `other`.
- oml.Integer : must come from the same data source. Divide each element in `self` by the corresponding element in `other`.

Returns

quotient
[oml.Float]

`__pow__`(*other*)
Equivalent to `self ** other`.

Parameters

other
[int, oml.Integer, float, or oml.Float]

- scalar : Raise each element in `self` to the power of the scalar.
- oml.Float : must come from the same data source. Raise each element in `self` to the power of the corresponding element in `other`.
- oml.Integer : must come from the same data source. Raise each element in `self` to the power of the corresponding element in `other`.

Returns

power
[oml.Float]

`__mod__`(*other*)
Equivalent to `self % other`.

Parameters

other
[int, oml.Integer, float, or oml.Float]

- scalar : Find the remainder when each element in `self` is divided by the scalar.
- oml.Float : must come from the same data source. Find the remainder when each element in `self` is divided by the corresponding element in `other`.
- oml.Integer : must come from the same data source. Find the remainder when each element in `self` is divided by the corresponding element in `other`.

Returns

remainder
[oml.Float]

`__divmod__`(*other*)
Equivalent to `divmod(self, other).`

Parameters

other
[int, oml.Integer, float, or oml.Float]

- scalar : Find the quotient and remainder when each element in `self` is divided by the scalar.
- oml.Float : must come from the same data source. Find the quotient and remainder when each element in `self` is divided by the corresponding element in `other`.
- oml.Integer : must come from the same data source. Find the quotient and remainder when each element in `self` is divided by the corresponding element in `other`.

Returns

divrem
[oml.DataFrame] The first column contains the floor of the quotient, and the second column contains the remainder.

`__abs__``()`
Return the absolute value of every element in `self`.

Equivalent to `abs(self)`.

Returns

absval
[oml.Float]

`__neg__``()`
Return the negation of every element in `self`. Equivalent to `-self`.

Returns

negation
[oml.Float]

`__matmul__``(`*other*`)`
Equivalent to `self @ other` and `self.dot(other)`.

Returns the inner product with an oml.Float. Matrix multiplication with a oml.DataFrame.

Parameters

other
[oml.Float or oml.DataFrame]

Returns

matprod
[oml.Float]

See also:

`Float.dot`

`KFold``(`*n_splits=3*, *seed=12345*, *use_hash=True*, *nvl=None*`)`
Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.

Parameters

n_splits
[int, 3 (default)] The number of folds. Must be greater than or equal to 2.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default):] If not None, the specified values are used to hash in place of Null.

Returns

kfold_data
[a list of pairs of series objects of the same type as caller] Each pair within the list is a fold. The first element of the pair is the train set, and the second element is the test set, which consists of all elements not in the train set.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

Examples
See *`Float.split()`*

`append`(*other*, *all=True*)
Appends the `other` OML data object of the same class to this data object.

Parameters

other
[An OML data object of the same class]

all
[boolean, True (default)] Keeps the duplicated elements from the two data objects.

Returns

appended
[type of caller] A new data object containing the elements from both objects.

Examples
See *`Float.concat()`*

`ceil`()
Returns the ceiling of each element in the Float series data object.

Returns

ceil
[oml.Float]

Examples
See *`Float.cut()`*

`concat`(*other*, *auto_name=False*)
Combines current OML data object with the `other` data objects column-wise.

Current object and the `other` data objects must be combinable, that is, they both represent data from the same underlying database table, view, or query.

Parameters

other
[an OML data object, a list of OML data objects, or a dict mapping str to OML data objects.]

- OML data object: an OML series data object or an oml.DataFrame
- list: a sequence of OML series and DataFrame objects to concat.
- dict: a dict mapping str to OML series and DataFrame objects, the column name of concatenated OML series object is replaced with str, column names of concatenated OML DataFrame object is prefixed with str. Need to specify a `collections.OrderedDict` if the concatenation order is expected to follow the key insertion order.

auto_name
[boolean, False (default)] Indicates whether to automatically resolve conflict column names. If True, append duplicated column names with suffix `[column_index]`.

Returns

concat_table
[oml.DataFrame] An oml.DataFrame data object with its original columns followed by the columns of `other`.

Raises

ValueError

- If `other` is not a single nor a list/dict of OML objects, or if `other` is empty.
- If objects in `other` are not from same data source.
- If `auto_name` is False and duplicated column names are detected.

Notes

After concatenation is done, if there is any empty column names in the resulting oml.DataFrame, they will be renamed with `COL[column_index]`.

Examples

```
>>> import oml
>>> import pandas as pd
>>> from collections import OrderedDict
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_
↪Width',
...                                         'Petal_Length', 'Petal_
↪Width'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor
↪',
...          2:'virginica'}[x], iris.target)), columns = ['Species'])
>>> iris_df = pd.concat([x, y], axis=1)
>>> oml_iris = oml.create(iris_df, table = 'IRIS')
>>> x = oml_iris['Sepal_Length']
>>> y = oml_iris['Petal_Length']
```

(continues on next page)

(continued from previous page)

```
>>> oml_frame = oml_iris[['Sepal_Width', 'Petal_Width']]
>>> x.shape
(150, 1)
>>> y.shape
(150, 1)
>>> oml_frame.shape
(150, 2)
```

Concat oml.Float.

```
>>> z = x.concat(y)
>>> z.head()
   Sepal_Length  Petal_Length
0           5.1           1.4
1           4.9           1.4
2           4.7           1.3
3           4.6           1.5
4           5.0           1.4
>>> z.shape
(150, 2)
>>> z = x.concat(oml_frame)
>>> z.head()
   Sepal_Length  Sepal_Width  Petal_Width
0           5.1          3.5          0.2
1           4.9          3.0          0.2
2           4.7          3.2          0.2
3           4.6          3.1          0.2
4           5.0          3.6          0.2
>>> z.shape
(150, 3)
```

Concat oml.Float and rename the concatenated column.

```
>>> x.concat({'petal length':y}).head()
   Sepal_Length  petal length
0           5.1           1.4
1           4.9           1.4
2           4.7           1.3
3           4.6           1.5
4           5.0           1.4
```

Concat multiple OML data objects and turn on automatic name conflict resolution.

```
>>> x.concat([y, oml_iris], auto_name=True).head()
   Sepal_Length  Petal_Length  Sepal_Length3  Sepal_Width  Petal_
↪Length5  \
0           5.1           1.4            5.1          3.5         ␣
↪ 1.4
1           4.9           1.4            4.9          3.0         ␣
↪ 1.4
2           4.7           1.3            4.7          3.2         ␣
↪ 1.3
3           4.6           1.5            4.6          3.1         ␣
↪ 1.5
4           5.0           1.4            5.0          3.6         ␣
↪ 1.4
```

(continues on next page)

(continued from previous page)

```
   Petal_Width Species
0          0.2  setosa
1          0.2  setosa
2          0.2  setosa
3          0.2  setosa
4          0.2  setosa
```

Concat multiple OML data objects and perform customized renaming.

```
>>> x.concat(OrderedDict([('petal length',y), ('New_',oml_frame)])).
↪head()
   Sepal_Length  petal length  New_Sepal_Width  New_Petal_Width
0           5.1           1.4              3.5              0.2
1           4.9           1.4              3.0              0.2
2           4.7           1.3              3.2              0.2
3           4.6           1.5              3.1              0.2
4           5.0           1.4              3.6              0.2
```

Append oml.Floats.

```
>>> z = x.head().append(y.head())
>>> z
[5.1, 4.9, 4.7, 4.6, 5, 1.4, 1.4, 1.3, 1.5, 1.4]
>>> oml.drop("IRIS")
```

count()

Returns the number of elements that are not NULL.

Returns

nobs
[int]

Examples

See *Float.describe()*

create_view(*view=None*, *use_colname=False*)

Creates an Oracle Database view for the data represented by the OML data object.

Parameters

view
[str or None (default)] The name of a database view. If `view` is None, the created view is managed by OML and the view is automatically dropped when no longer needed. If a `view` is specified, then it is up to the user to drop the view.

use_colname
[bool, False (default)] Indicates whether to create the view with the same column names as the DataFrame object. Ignored if `view` is specified.

Returns

new_view
[oml.DataFrame]

Raises

TypeError

- If the object represents data from a temporary table or view, and the view to create is meant to persist past the current session.

Examples
See `Float.pull()`

`cumsum`(*ascending=True*, *na_position='last'*, *skipna=True*)
Gets the cumulative sum after the OML series data object is sorted.

Parameters

ascending
[bool, True (default)] Sorts ascending, otherwise descending.

na_position
[{'first', 'last'}, 'last' (default)] `first` places NaN and None at the beginning, `last` places them at the end.

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

cumsum
[oml.Float]

Examples
See `Float.describe()`

`cut`(*bins*, *right=True*, *labels=None*, *retbins=False*, *precision=3*, *include_lowest=False*)
Returns the indices of half-open bins to which each value belongs.

Parameters

bins
[int or strictly monotonically increasing sequence of float/int] If int, defines number of equal-width bins in the range of this column. In this case, to include the min and max value, the range is extended by .1% on each side where the bin does not include the end-point. If a sequence, defines bin edges allowing for non-uniform bin-widths. In this case, the range of x is not extended.

right
[bool, True (default)] Indicates whether the bins include the rightmost edge or the leftmost edge.

labels
[sequence of unique str, int, or float values, False, or None (default)] If a sequence, must be the same length as the resulting number of bins and must have values of same type. If False, bins are sequentially labeled with integers. If None, bins are labeled with the intervals they correspond to.

retbins
[bool, False (default)] Indicates whether to return the bin edges or not.

precision
[int, 3 (default)] When `labels` is None, determines the precision of the bin labels.

include_lowest
[bool, False (default)] Indicates whether the first interval should be left-inclusive.

Returns

out
[oml.Float or oml.String] If labels are ints or floats, return oml.Float. If labels are str, return oml.String.

bins
[numpy.ndarray of floats] Returned only if retbins is True.

Examples

```
>>> import oml
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(1234)
>>> oml_float = oml.push(np.random.random_sample(10)*10)
```

oml.Float floor, ceil, round, exp, log, sqrt, isinf, isnan, dot, cut

```
>>> oml_float
[1.9151945037889229, 6.221087710398319, 4.377277390071145, 7.
↪853585837137692, 7.799758081188035, 2.7259260528264164, 2.
↪764642551430967, 8.018721775350192, 9.581393536837052, 8.
↪759326347420947]
```

Returns the floor of each element in the Float series data object.

```
>>> oml_float.floor()
[1, 6, 4, 7, 7, 2, 2, 8, 9, 8]
```

Returns the ceiling of each element in the Float series data object.

```
>>> oml_float.ceil()
[2, 7, 5, 8, 8, 3, 3, 9, 10, 9]
```

Round all the elements to the specified decimal place.

```
>>> oml_float.round()
[2, 6, 4, 8, 8, 3, 3, 8, 10, 9]
```

```
>>> oml_float.round(3)
[1.915, 6.221, 4.377, 7.854, 7.8, 2.726, 2.765, 8.019, 9.581, 8.759]
```

Returns exponential of all the elements.

```
>>> oml_float.exp()
[6.788259010699638, 503.2503250384189, 79.62096121817044, 2574.
↪951137366237, 2440.0116215055973, 15.270548715867625, 15.
↪87336506479903, 3037.292508480158, 14492.601210512114, 6369.
↪819087273758]
```

Returns the natural logarithm of all the elements.

```
>>> oml_float.log()
[0.6498191860330569, 1.82794476451697, 1.4764269306743263, 2.
↪0609702220352855, 2.0540927179822606, 1.0028082062836694, 1.
↪01691135046225, 2.0817790295439944, 2.259823044539425, 2.
↪170119001011924]
```

```
>>> oml_float.log(base = 2)
[0.9374909171643938, 2.6371668467875766, 2.130033811118764, 2.
↪9733515187501323, 2.9634293777591396, 1.4467464261682066, 1.
↪4670929623355868, 3.0033722821497593, 3.260235499643627, 3.
↪1308199208988143]
```

```
>>> oml_float.log(10)
[0.28221288672901923, 0.7938663244536591, 0.6412040689252149, 0.
↪8950679947968442, 0.8920811327373482, 0.4355140703962955, 0.
↪44163898809053903, 0.9041051450728633, 0.9814286783212787, 0.
↪9424707072128761]
```

Returns the square root of each element in the Float series data object.

```
>>> oml_float.sqrt()
[1.3839055256009793, 2.4942108392031175, 2.092194395860754, 2.
↪802424992241129, 2.792804698003073, 1.6510378714088954, 1.
↪6627214292932437, 2.8317347643008857, 3.0953826155803505, 2.
↪959615912144842]
```

Returns the indices of half-open bins to which each value belongs.

```
>>> oml_float.cut(4, precision = 5)
['(1.90753, 3.83174]', '(5.74829, 7.66484]', '(3.83174, 5.74829]',
↪'(7.66484, 9.58139]', '(7.66484, 9.58139]', '(1.90753, 3.83174]',
↪'(1.90753, 3.83174]', '(7.66484, 9.58139]', '(7.66484, 9.58139]',
↪'(7.66484, 9.58139]']
```

```
>>> oml_float.cut(4, right = False)
[None, '[5.748, 7.665)', '[3.832, 5.748)', '[7.665, 9.589)', '[7.
↪665, 9.589)', '[1.915, 3.832)', '[1.915, 3.832)', '[7.665, 9.589)
↪', '[7.665, 9.589)', '[7.665, 9.589)']
```

```
>>> oml_float.cut(4, include_lowest = True)
[None, '(5.748, 7.665]', '(3.832, 5.748]', '(7.665, 9.581]', '(7.
↪665, 9.581]', '[1.915, 3.832]', '[1.915, 3.832]', '(7.665, 9.581]
↪', '(7.665, 9.581]', '(7.665, 9.581]']
```

```
>>> oml_float.cut(4, labels = False)
[0, 2, 1, 3, 3, 0, 0, 3, 3, 3]
```

```
>>> oml_float.cut(4, labels = ['a', 'b', 'c', 'd'])
['a', 'c', 'b', 'd', 'd', 'a', 'a', 'd', 'd', 'd']
```

```
>>> oml_float.cut([1, 5, 10, float('inf')], retbins = True, labels=[
↪'a', 'b', 'c'])
(['a', 'b', 'a', 'b', 'b', 'a', 'a', 'b', 'b', 'b'], array([ 1.,  5.
↪, 10., inf]))
```

Create oml.Float object with NaN and Inf values

```
>>> float_types = oml.push(pd.DataFrame({'FLOAT_NAN': [None, -12.1,␣
↪1234, 40.00, 95.6], 'FLOAT_INF': [98.3, 99.9, -0.003, float('Inf
↪'), 11.1]}), oranumber = False)
>>> oml_float_nan = float_types['FLOAT_NAN']
>>> oml_float_inf = float_types['FLOAT_INF']
>>> type(oml_float_nan)
<class 'oml.core.float.Float'>
>>> type(oml_float_inf)
<class 'oml.core.float.Float'>
```

Detects infinite values element-wise in the Float series data object.

```
>>> oml_float_inf.isinf()
[False, False, False, True, False]
```

Detects a NaN (not a number) element from Float object.

```
>>> oml_float_nan.isnan()
[True, False, False, False, False]
```

Create oml.Float object with NaN values for dot() test

```
>>> float_dot = oml.push(pd.DataFrame({'FLOAT1': np.random.random_
↪sample(10)*10, 'FLOAT2': np.random.random_sample(10)*10, 'FLOAT_
↪NAN': [None, -12.1, 1234, 40.00, 95.6, 98.3, 99.9, -0.003, float(
↪'Inf'), 11.1]}), oranumber = False)
>>> oml_float_dot1 = float_dot['FLOAT1']
>>> oml_float_dot2 = float_dot['FLOAT2']
>>> oml_float_dot3 = float_dot['FLOAT_NAN']
>>> type(oml_float_dot1)
<class 'oml.core.float.Float'>
>>> type(oml_float_dot2)
<class 'oml.core.float.Float'>
```

Returns the inner product with an oml.Float. Matrix multiplication with a oml.DataFrame.

```
>>> oml_float_dot1.dot()
343.5720558076067
```

```
>>> oml_float_dot1.dot(oml_float_dot2)
242.12738327014995
```

```
>>> oml_float_dot2.dot(oml_float_dot1, skipna = False)
242.12738327014995
```

```
>>> oml_float_dot2.dot(oml_float_dot3, skipna = True)
inf
```

```
>>> oml_float_dot2.dot(oml_float_dot3, skipna = False)
nan
```

```
>>> oml_float_dot2.dot(float_dot, skipna = True)
FLOAT1        242.127383
FLOAT2        315.159799
```

(continues on next page)

(continued from previous page)

```
FLOAT_NAN           inf
dtype: float64
```

```
>>> oml_float_dot2.dot(float_dot, skipna = False)
FLOAT1       242.127383
FLOAT2       315.159799
FLOAT_NAN           NaN
dtype: float64
```

```
>>> oml_float_dot1.dot(float_dot[['FLOAT1', 'FLOAT2']])
FLOAT1    343.572056
FLOAT2    242.127383
dtype: float64
```

@ is the same as dot(other, skipna = True)

```
>>> oml_float_dot1@oml_float_dot2
242.12738327014995
```

```
>>> oml_float_dot1@float_dot[['FLOAT1', 'FLOAT2']]
FLOAT1    343.572056
FLOAT2    242.127383
dtype: float64
```

describe(*percentiles=None*)

Generates descriptive statistics that summarize the central tendency, dispersion, and shape of the OML series data distribution.

Parameters

percentiles
[list-like of numbers, optional] The percentiles to include in the output. All must be between 0 and 1. The default is [.25, .5, .75], which corresponds to the inclusion of the 25th, 50th, and 75th percentiles.

Returns

summary
[`pandas.Series`] Includes `count` (number of non-null entries), `mean`, `std`, `min`, `max`, and the specified `percentiles`. The 50th percentile is always included.

Examples

Set up a Float series data object.

```
>>> import oml
>>> import pandas as pd
>>> df = pd.DataFrame({'numeric': [1, 1.4, -4, 3.145, 5, 10.5, 3.5,␣
↪None]})
>>> oml_df = oml.push(df, oranumber = False)
>>> oml_float = oml_df['numeric']
```

Describe the Float series data.

```
>>> oml_float.describe()
count      7.000000
mean       2.935000
std        4.397944
min       -4.000000
25%        1.200000
50%        3.145000
75%        4.250000
max       10.500000
Name: numeric, dtype: float64
```

Describe the data with given percentiles.

```
>>> oml_float.describe(percentiles=[.2, .4, .6, .8])
count      7.000000
mean       2.935000
std        4.397944
min       -4.000000
20%        1.080000
40%        2.098000
50%        3.145000
60%        3.358000
80%        4.700000
max       10.500000
Name: numeric, dtype: float64
```

Get the count of all not-Null values in the data.

```
>>> oml_float.count()
7
```

Find the maximum value in the data.

```
>>> oml_float.max()
10.5
```

Get the minimum value in the data.

```
>>> oml_float.min()
-4.0
```

Find the average value in the data.

```
>>> oml_float.mean()
2.935
```

Get the median in the data.

```
>>> oml_float.median()
3.145
```

Find the standard deviation of the data.

```
>>> oml_float.std()
4.3979436482671455
```

Get the skewness of the data.

```
>>> oml_float.skew()
0.20641985983222347
```

Find the kurtosis of the data.

```
>>> oml_float.kurtosis()
-0.07581984600909619
```

Get the sum of the data.

```
>>> oml_float.sum()
20.545
```

Find the cumulative sum after the data is sorted in descending order.

```
>>> oml_float.cumsum(ascending = False, na_position = 'first')
[None, 10.5, 15.5, 19.0, 22.145, 23.544999999999998, 24.
↪544999999999998, 20.544999999999998]
```

dot(*other=None*, *skipna=True*)

Returns the inner product with an oml.Float. Matrix multiplication with a oml.DataFrame.

Can be called using self @ other.

Parameters

other
[oml.Float or oml.DataFrame, optional] If not specified, self is used.

skipna
[bool, True (default)] Treats NaN entries as 0.

Returns

dot_product
[`pandas.Series` or float]

Examples
See `Float.cut()`

drop_duplicates()

Removes duplicated elements.

Returns

deduplicated
[type of caller]

Examples
See `Float.dropna()`

dropna()

Removes missing values.

Missing values include None and/or nan if applicable.

Returns

dropped
[type of caller]

Examples

Setup.

```
>>> import oml
>>> oml_float = oml.push([1, None, 3, 3, 6, None, None])
>>> oml_float
[1, None, 3, 3, 6, None, None]
```

Drop NA values

```
>>> oml_float.dropna()
[1, 3, 3, 6]
```

Drop duplicate values

```
>>> oml_float.drop_duplicates().sort_values()
[1, 3, 6, None]
```

Number of unique values

```
>>> oml_float.nunique()
3
>>> oml_float.nunique(dropna=False)
4
```

Null values

```
>>> oml_float.isnull()
[False, True, False, False, False, True, True]
```

Replace values with non-matched values perserved

```
>>> oml_float.replace(old=[1.0,3.0,None], new=[100.0, 300.0,float(
↪'nan')])
[100.0, nan, 300.0, 300.0, 6.0, nan, nan]
```

Replace values and use default value for non-matched values

```
>>> oml_float.replace(old=[1.0,3.0], new=[100.0, 300.0], default=0.
↪0)
[100, 0, 300, 300, 0, 0, 0]
```

Replace float values with str values

```
>>> oml_float.replace(old=[1.0,3.0], new=['T','F'], default='F')
['T', 'F', 'F', 'F', 'F', 'F', 'F']
```

exp()

Returns element-wise e to the power of values in the Float series data object.

Returns

exp
[oml.Float]

Examples
See *Float.cut()*

floor()
Returns the floor of each element in the Float series data object.

Returns

floor
[oml.Float]

Examples
See *Float.cut()*

head(*n=5*)
Returns the first n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_head
[type of caller]

Examples
See *Float.pull()*

isinf()
Detects infinite values element-wise in the Float series data object.

Returns

isinf
[oml.Boolean]

Examples
See *Float.cut()*

isnan()
Detects a NaN (not a number) element from Float object.

Returns

isnull
[oml.Boolean] Indicates NaN for each element.

Examples
See *Float.cut()*

isnull()
Detects the missing value None.

Returns

isnull
[oml.Boolean] Indicates missing value None for each element.

Examples
See `Float.dropna()`

`kurtosis`(*skipna=True*)
Returns the sample kurtosis of the values.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

kurt
[float or nan]

Examples
See `Float.describe()`

`log`(*base=None*)
Returns element-wise logarithm, to the given `base`, of values in the Float series data object.

Parameters

base
[int, float, optional] The base of the logarithm, by default natural logarithm

Returns

log
[oml.Float]

Examples
See `Float.cut()`

`materialize`(*table=None*)
Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.

Parameters

table
[str or None (default)] The name of a table. If `table` is None, an OML-managed table is created, that is, OML drops the table when it is no longer used by any OML object or when you invoke `oml.disconnect(cleanup=True)` to terminate the database connection. If a table is specified, then it's up to the user to drop the named table.

Returns

new_table
[same type as self]

Examples
See `Float.pull()`

max(*skipna=True*)
Returns the maximum value.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

max
[Python type corresponding to the column or numpy.nan]

Examples
See `Float.describe()`

mean(*skipna=True*)
Returns the mean of the values.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

mean
[float or numpy.nan]

Examples
See `Float.describe()`

median(*skipna=True*)
Returns the median of the values.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

median
[float or numpy.nan]

Examples
See `Float.describe()`

min(*skipna=True*)
Returns the minimum value.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

min
[Python type corresponding to the column or numpy.nan]

Examples
See *Float.describe()*

nunique (*dropna=True*)
Returns the number of unique values.

Parameters

dropna
[bool, True (default)] If True, NULL values are not included in the count.

Returns

nunique
[int]

Examples
See *Float.dropna()*

pull ()
Pulls data represented by the series data object from Oracle Database into an in-memory Python object.

Returns

pulled_obj
[list]

Examples
Create table from pandas DataFrame and return oml series object

```
>>> import oml
>>> oml_float = oml.create([float(i) for i in list(range(100))],␣
↪table = "FLOAT")
>>> oml_bool = oml_float < 50
>>> type(oml_float)
<class 'oml.core.float.Float'>
>>> type(oml_bool)
<class 'oml.core.boolean.Boolean'>
>>> oml_float.shape
(100, 1)
>>> len(oml_float)
100
```

Pull the data from oml series object into a python list

```
>>> type(oml_float.pull())
<class 'list'>
>>> oml_float.pull()[:10]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> oml_bool.pull()[:10]
[True, True, True, True, True, True, True, True, True, True]
```

Show first and last 10 elements

```
>>> oml_float.head(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> oml_float.tail(10)
[90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
>>> oml_bool.head()
[True, True, True, True, True]
>>> oml_bool.tail()
[False, False, False, False, False]
```

Sort oml series object by value

```
>>> oml_float.sort_values(ascending = False).head(10)
[99, 98, 97, 96, 95, 94, 93, 92, 91, 90]
```

Create a database view from oml.Float object

```
>>> oml_float_v = oml_float.create_view(view = "FLOAT_V")
>>> oml.sync(view = "FLOAT_V").head()
   COL1
0     0
1     1
2     2
3     3
4     4
```

Materialize oml.Float object into another table

```
>>> oml_float_v.materialize(table = "FLOAT_T").shape
(100, 1)
>>> oml.sync(table = "FLOAT_T").head()
   COL1
0     0
1     1
2     2
3     3
4     4
```

```
>>> oml.drop(table = 'FLOAT_T')
>>> oml.drop(view = 'FLOAT_V')
>>> oml.drop(table = 'FLOAT')
```

replace(*old*, *new*, *default=None*)

Replace values given in *old* with *new*.

Parameters

old
[list of float, or list of str] Specifying the old values. When specified with a list of float, it can contain float('nan') and None. When specified with a list of str, it can contain None.

new
[list of float, or list of str] A list of the same length as argument *old* specifying the new values. When specified with a list of float, it can contain float('nan') and None. When specified with a list of str, it can contain None.

default
[float, str, or None (default)] A single value to use for the non-matched elements in argu-

ment *old*. If None, non-matched elements will preserve their original values. If not None, data type should be consistent with values in *new*. Must be set when *old* and *new* contain values of different data types.

Returns

replaced
[oml.Float]

Raises

ValueError

- if values in *old* have data types inconsistent with original values
- if *default* is specifed with a non-None value which has data type inconsistent with values in *new*
- if *default* is None when *old* and *new* contain values of different data types

Examples
See `Float.dropna()`

round(*decimals=0*)
Rounds oml.Float values to the specified decimal place.

Parameters

decimals
[non-negative int]

Returns

rounded
[oml.Float]

Examples
See `Float.cut()`

skew(*skipna=True*)
Returns the sample skewness of the values.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

skew
[float or nan]

Examples
See `Float.describe()`

sort_values(*ascending=True*, *na_position='last'*)
Sorts the values in the series data object.

Parameters

ascending
[bool, True (default)] If True, sorts in ascending order. Otherwise, sorts in descending order.

na_position
[{'first', 'last'}, 'last' (default)] `first` places NANs and Nones at the beginning; `last` places them at the end.

Returns

sorted_obj
[type of caller]

Examples
See `Float.pull()`

split (*ratio=(0.7, 0.3)*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into multiple sets.

Parameters

ratio
[a list of float values or (0.7, 0.3) (default)] All the numbers must be positive and the sum of them are no more than 1. Each number represents the ratio of split data in one set.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default)] If not None, the specified values are used to hash in place of Null.

Returns

split_data
[a list of series objects of the same type as caller] Each of which contains the portion of data by the specified ratio.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

Examples

```
>>> import oml
>>> import numpy as np
```

(continues on next page)

(continued from previous page)

```
>>> np.random.seed(1234)
>>> oml_float = oml.push(np.random.randint(0, 1000, 100))
```

Split into four equal-sized sets.

```
>>> splits = oml_float.split(ratio = (.25, .25, .25, .25), seed =␣
↪1234)
>>> [len(split) for split in splits]
[29, 21, 16, 34]
```

Split randomly into 4 consecutive folds

```
>>> folds = oml_float.KFold(n_splits=4)
>>> [(len(x[0]), len(x[1])) for x in folds]
[(72, 28), (80, 20), (78, 22), (70, 30)]
```

sqrt()

Returns the square root of each element in the Float series data object.

Returns

sqrt
[oml.Float]

Examples
See `Float.cut()`

std(*skipna=True*)

Returns the sample standard deviation of the values.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

std
[float or numpy.nan]

Examples
See `Float.describe()`

sum(*skipna=True*)

Returns the sum of the values.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

sum
[float or numpy.nan]

Examples
See *Float.describe()*

tail(*n=5*)
Returns the last n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_tail
[type of caller]

Examples
See *Float.pull()*

Special Method Examples

```
>>> import pandas as pd
>>> flts = oml.push(
...         pd.DataFrame({'a': [-1, 3.1, None, 0, -5.2],
...         'b': [4.3, 10, -2.2, 0, -1]}), oranumber = False)
```

Convert underlying column type from binary_double to number.

```
>>> numflt = oml.Float(flts['a'], 'number')
>>> flts['a']
[-1.0, 3.1, nan, 0.0, -5.2]
>>> numflt
[-1, 3.1, None, 0, -5.2]
```

Some arithmetic.

```
>>> -flts['b'] - abs(flts['a'])
[-5.3, -13.1, nan, 0.0, -4.2]
>>> numflt ** 2
[1, 9.610000000000001, None, 0, 27.040000000000003]
>>> flts['b'] @ flts['a']
31.9
```

Select the elements with absolute value less than or equal to 1 in `flts['b']`, see how many there are, and see if they are all in `flts['a']`.

```
>>> abs(flts['b']) <= 1
[False, False, False, True, True]
>>> flts['b'][abs(flts['b']) <= 1]
[0.0, -1.0]
>>> len(flts['b'][abs(flts['b']) <= 1])
2
>>> flts['b'][abs(flts['b']) <= 1] in numflt
True
```

1.3.4 oml.Integer

class `oml.Integer(other)`

Numeric series data class.

Represents a single column of INTEGER data (NUMBER with no precision) in Oracle Database.

Attributes

`shape`	The dimensions of the data set.

Special Methods

`__init__`(other)	Convert to oml.Integer
`__getitem__`(key)	Index self.
`__len__`()	Returns number of rows.
`__eq__`(other)	Equivalent to `self == other`.
`__ne__`(other)	Equivalent to `self != other`.
`__gt__`(other)	Equivalent to `self > other`.
`__ge__`(other)	Equivalent to `self >= other`.
`__lt__`(other)	Equivalent to `self < other`.
`__le__`(other)	Equivalent to `self <= other`.

Special Method Examples

Methods

`KFold`([n_splits, seed, use_hash, nvl])	Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.
`append`(other[, all])	Appends the `other` OML data object of the same class to this data object.
`concat`(other[, auto_name])	Combines current OML data object with the `other` data objects column-wise.
`count`()	Returns the number of elements that are not NULL.
`create_view`([view, use_colname])	Creates an Oracle Database view for the data represented by the OML data object.
`cumsum`([ascending, na_position, skipna])	Gets the cumulative sum after the OML series data object is sorted.
`cut`(bins[, right, labels, retbins, . . .])	Returns the indices of half-open bins to which each value belongs.
`describe`([percentiles])	Generates descriptive statistics that summarize the central tendency, dispersion, and shape of the OML series data distribution.
`dot`([other, skipna])	Returns the inner product with an oml.Integer.
`drop_duplicates`()	Removes duplicated elements.
`dropna`()	Removes missing values.
`exp`()	Returns element-wise e to the power of values in the Integer series data object.

Continued on next page

Table 15 – continued from previous page

head([n])	Returns the first n elements.
isnan()	Detects a NaN (not a number) element from Integer object.
isnull()	Detects the missing value None.
kurtosis([skipna])	Returns the sample kurtosis of the values.
log([base])	Returns element-wise logarithm, to the given `base`, of values in the Integer series data object.
materialize([table])	Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.
max([skipna])	Returns the maximum value.
mean([skipna])	Returns the mean of the values.
median([skipna])	Returns the median of the values.
min([skipna])	Returns the minimum value.
nunique([dropna])	Returns the number of unique values.
pull()	Pulls data represented by the series data object from Oracle Database into an in-memory Python object.
replace(old, new[, default])	Replace values given in *old* with *new*.
skew([skipna])	Returns the sample skewness of the values.
sort_values([ascending, na_position])	Sorts the values in the series data object.
split([ratio, seed, use_hash, nvl])	Splits the series data object randomly into multiple sets.
sqrt()	Returns the square root of each element in the Integer series data object.
std([skipna])	Returns the sample standard deviation of the values.
sum([skipna])	Returns the sum of the values.
tail([n])	Returns the last n elements.

__init__(*other*)

Convert to oml.Integer

Parameters

other

[oml.Boolean or oml.Integer]

- oml.Boolean : initialize a oml.Integer object that has value 1 (resp. 0) wherever `other` has value True (resp. False).
- oml.Float : initialize a oml.Integer object with the rounded data from `other`.

__getitem__(*key*)

Index self. Equivalent to `self[key]`.

Parameters

key

[oml.Boolean] Must be from the same data source as self.

Returns

subset

[same type as self] Contains only the rows satisfying the condition in `key`.

`__len__``()`

Returns number of rows. Equivalent to `len(self)`.

Returns

rownum
[int]

`__eq__``(`*other*`)`

Equivalent to `self == other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

equal
[oml.Boolean]

`__ne__``(`*other*`)`

Equivalent to `self != other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

notequal
[oml.Boolean]

`__gt__``(`*other*`)`

Equivalent to `self > other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.

- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterthan
[oml.Boolean]

`__ge__`(*other*)
Equivalent to `self >= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterequal
[oml.Boolean]

`__lt__`(*other*)
Equivalent to `self < other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessthan
[oml.Boolean]

`__le__`(*other*)
Equivalent to `self <= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessequal
[oml.Boolean]

KFold(*n_splits=3*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.

Parameters

n_splits
[int, 3 (default)] The number of folds. Must be greater than or equal to 2.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default):] If not None, the specified values are used to hash in place of Null.

Returns

kfold_data
[a list of pairs of series objects of the same type as caller] Each pair within the list is a fold. The first element of the pair is the train set, and the second element is the test set, which consists of all elements not in the train set.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

append(*other*, *all=True*)
Appends the `other` OML data object of the same class to this data object.

Parameters

other
[An OML data object of the same class]

all
[boolean, True (default)] Keeps the duplicated elements from the two data objects.

Returns

appended
[type of caller] A new data object containing the elements from both objects.

concat (*other*, *auto_name=False*)
Combines current OML data object with the `other` data objects column-wise.

Current object and the `other` data objects must be combinable, that is, they both represent data from the same underlying database table, view, or query.

Parameters

other
[an OML data object, a list of OML data objects, or a dict mapping str to OML data objects.]

- OML data object: an OML series data object or an oml.DataFrame
- list: a sequence of OML series and DataFrame objects to concat.
- dict: a dict mapping str to OML series and DataFrame objects, the column name of concatenated OML series object is replaced with str, column names of concatenated OML DataFrame object is prefixed with str. Need to specify a `collections.OrderedDict` if the concatenation order is expected to follow the key insertion order.

auto_name
[boolean, False (default)] Indicates whether to automatically resolve conflict column names. If True, append duplicated column names with suffix `[column_index]`.

Returns

concat_table
[oml.DataFrame] An oml.DataFrame data object with its original columns followed by the columns of `other`.

Raises

ValueError

- If `other` is not a single nor a list/dict of OML objects, or if `other` is empty.
- If objects in `other` are not from same data source.
- If `auto_name` is False and duplicated column names are detected.

Notes

After concatenation is done, if there is any empty column names in the resulting oml.DataFrame, they will be renamed with `COL[column_index]`.

count ()
Returns the number of elements that are not NULL.

Returns

nobs
[int]

create_view(*view=None*, *use_colname=False*)
Creates an Oracle Database view for the data represented by the OML data object.

Parameters

view
[str or None (default)] The name of a database view. If `view` is None, the created view is managed by OML and the view is automatically dropped when no longer needed. If a `view` is specified, then it is up to the user to drop the view.

use_colname
[bool, False (default)] Indicates whether to create the view with the same column names as the DataFrame object. Ignored if `view` is specified.

Returns

new_view
[oml.DataFrame]

Raises

TypeError

- If the object represents data from a temporary table or view, and the view to create is meant to persist past the current session.

cumsum(*ascending=True*, *na_position='last'*, *skipna=True*)
Gets the cumulative sum after the OML series data object is sorted.

Parameters

ascending
[bool, True (default)] Sorts ascending, otherwise descending.

na_position
[{'first', 'last'}, 'last' (default)] `first` places NaN and None at the beginning, `last` places them at the end.

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

cumsum
[oml.Float]

cut(*bins*, *right=True*, *labels=None*, *retbins=False*, *precision=3*, *include_lowest=False*)
Returns the indices of half-open bins to which each value belongs.

Parameters

bins
[int or strictly monotonically increasing sequence of float/int] If int, defines number of equal-width bins in the range of this column. In this case, to include the min and max

value, the range is extended by .1% on each side where the bin does not include the endpoint. If a sequence, defines bin edges allowing for non-uniform bin-widths. In this case, the range of x is not extended.

right
[bool, True (default)] Indicates whether the bins include the rightmost edge or the leftmost edge.

labels
[sequence of unique str, int, or float values, False, or None (default)] If a sequence, must be the same length as the resulting number of bins and must have values of same type. If False, bins are sequentially labeled with integers. If None, bins are labeled with the intervals they correspond to.

retbins
[bool, False (default)] Indicates whether to return the bin edges or not.

precision
[int, 3 (default)] When `labels` is None, determines the precision of the bin labels.

include_lowest
[bool, False (default)] Indicates whether the first interval should be left-inclusive.

Returns

out
[oml.Integer or oml.Float or oml.String] If labels are ints or floats, return oml.Integer or oml.Float. If labels are str, return oml.String.

bins
[`numpy.ndarray` of ints] Returned only if `retbins` is True.

describe(*percentiles=None*)
Generates descriptive statistics that summarize the central tendency, dispersion, and shape of the OML series data distribution.

Parameters

percentiles
[list-like of numbers, optional] The percentiles to include in the output. All must be between 0 and 1. The default is [.25, .5, .75], which corresponds to the inclusion of the 25th, 50th, and 75th percentiles.

Returns

summary
[`pandas.Series`] Includes `count` (number of non-null entries), `mean`, `std`, `min`, `max`, and the specified `percentiles`. The 50th percentile is always included.

dot(*other=None*, *skipna=True*)
Returns the inner product with an oml.Integer. Matrix multiplication with a oml.DataFrame.

Can be called using self @ other.

Parameters

other
[oml.Integer or oml.Float or oml.DataFrame, optional] If not specified, self is used.

skipna
[bool, True (default)] Treats NaN entries as 0.

Returns

dot_product
[`pandas.Series` or float]

`drop_duplicates``()`
Removes duplicated elements.

Returns

deduplicated
[type of caller]

`dropna``()`
Removes missing values.

Missing values include None and/or nan if applicable.

Returns

dropped
[type of caller]

`exp``()`
Returns element-wise e to the power of values in the Integer series data object.

Returns

exp
[oml.Integer]

`head``(`*n=5*`)`
Returns the first n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_head
[type of caller]

`isnan``()`
Detects a NaN (not a number) element from Integer object.

Returns

isnull
[oml.Boolean] Indicates NaN for each element.

`isnull``()`
Detects the missing value None.

Returns

isnull
[oml.Boolean] Indicates missing value None for each element.

kurtosis(*skipna=True*)
Returns the sample kurtosis of the values.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

kurt
[float or nan]

log(*base=None*)
Returns element-wise logarithm, to the given `base`, of values in the Integer series data object.

Parameters

base
[int, float, optional] The base of the logarithm, by default natural logarithm

Returns

log
[oml.Float]

materialize(*table=None*)
Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.

Parameters

table
[str or None (default)] The name of a table. If `table` is None, an OML-managed table is created, that is, OML drops the table when it is no longer used by any OML object or when you invoke `oml.disconnect(cleanup=True)` to terminate the database connection. If a table is specified, then it's up to the user to drop the named table.

Returns

new_table
[same type as self]

max(*skipna=True*)
Returns the maximum value.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

max
[Python type corresponding to the column or numpy.nan]

mean(*skipna=True*)
Returns the mean of the values.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

mean
[float or numpy.nan]

median(*skipna=True*)
Returns the median of the values.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

median
[float or numpy.nan]

min(*skipna=True*)
Returns the minimum value.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

min
[Python type corresponding to the column or numpy.nan]

nunique(*dropna=True*)
Returns the number of unique values.

Parameters

dropna
[bool, True (default)] If True, NULL values are not included in the count.

Returns

nunique
[int]

pull()
Pulls data represented by the series data object from Oracle Database into an in-memory Python object.

Returns

pulled_obj
[list]

replace(*old*, *new*, *default=None*)
Replace values given in *old* with *new*.

Parameters

old
[list of int, or list of str] Specifying the old values.

new
[list of int, or list of str] A list of the same length as argument *old* specifying the new values.

default
[int, str, or None (default)] A single value to use for the non-matched elements in argument *old*. If None, non-matched elements will preserve their original values. If not None, data type should be consistent with values in *new*. Must be set when *old* and *new* contain values of different data types.

Returns

replaced
[oml.Integer]

Raises

ValueError

- if values in *old* have data types inconsistent with original values
- if *default* is specifed with a non-None value which has data type inconsistent with values in *new*
- if *default* is None when *old* and *new* contain values of different data types

skew(*skipna=True*)
Returns the sample skewness of the values.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

skew
[float or nan]

`sort_values`(*ascending=True*, *na_position='last'*)
Sorts the values in the series data object.

Parameters

ascending
[bool, True (default)] If True, sorts in ascending order. Otherwise, sorts in descending order.

na_position
[{'first', 'last'}, 'last' (default)] `first` places NANs and Nones at the beginning; `last` places them at the end.

Returns

sorted_obj
[type of caller]

`split`(*ratio=(0.7, 0.3)*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into multiple sets.

Parameters

ratio
[a list of float values or (0.7, 0.3) (default)] All the numbers must be positive and the sum of them are no more than 1. Each number represents the ratio of split data in one set.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default)] If not None, the specified values are used to hash in place of Null.

Returns

split_data
[a list of series objects of the same type as caller] Each of which contains the portion of data by the specified ratio.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

`sqrt`()
Returns the square root of each element in the Integer series data object.

Returns

sqrt
[oml.Float]

std(*skipna=True*)
Returns the sample standard deviation of the values.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

std
[float or numpy.nan]

sum(*skipna=True*)
Returns the sum of the values.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

sum
[float or numpy.nan]

tail(*n=5*)
Returns the last n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_tail
[type of caller]

Special Method Examples

/ / $Header: Integer.special.txt 07-oct-2022.01:59:59 ffeli Exp $ / / Integer.special.txt / / Copyright (c) 2022, Oracle and/or its affiliates. / / NAME / Integer.special.txt - <one-line expansion of the name> / / DESCRIPTION / <short description of component this file declares/defines> / / NOTES / <other useful comments, qualifications, etc.> / / MODIFIED (MM/DD/YY) / ffeli 10/07/22 - Creation /

```
>>> import oml
>>> import pandas as pd
```

```
>>> integer1 = oml.push(pd.DataFrame({'INTEGER': [0, -12, 1234, 40, 95]}),␣
↪dbtypes = "NUMBER(*, 0)")
>>> integer1
   INTEGER
0        0
1      -12
2     1234
3       40
4       95
```

```
>>> integer2 = oml.push(pd.DataFrame({'INTEGER': [0.1, -12.2, 1234.5, 40.0, 95.9]}
↪), dbtypes = "NUMBER(*, 0)")
>>> integer2
   INTEGER
0        0
1      -12
2     1235
3       40
4       96
```

```
>>> integer3 = oml.push([0, -12, 1234, 40, 95], dbtypes = "NUMBER(*, 0)")
>>> integer3
[0, -12, 1234, 40, 95]
```

```
>>> integer4 = oml.push([0.1, -12.2, 1234.5, 40.0, 95.9], dbtypes = "NUMBER(*, 0)
↪")
>>> integer4
[0, -12, 1235, 40, 96]
```

```
>>> integer5 = oml.push(pd.DataFrame({'INTEGER': [0, -12, 1234, 40, 95]}))
>>> integer5
   INTEGER
0        0
1      -12
2     1234
3       40
4       95
```

```
>>> integer6 = oml.push([0, -12, 1234, 40, 95])
>>> integer6
[0, -12, 1234, 40, 95]
```

```
>>> integer7 = oml.create(pd.DataFrame({'INTEGER': [0, -12, 1234, 40, 95]}),␣
↪table = "INTEGER1", dbtypes = "NUMBER(*, 0)")
>>> integer7
   INTEGER
0        0
1      -12
2     1234
3       40
4       95
```

1.3.5 oml.String

class oml.**String**(*other*, *dbtype*)

Character series data class.

Represents a single column of VARCHAR2, CHAR, or CLOB data in Oracle Database.

Attributes

shape	The dimensions of the data set.

Special Methods

__init__(other, dbtype)	Convert underlying Oracle Database type.
__contains__(item)	Check whether all elements in item exists in the String series
__getitem__(key)	Index self.
__len__()	Returns number of rows.
__eq__(other)	Equivalent to self == other.
__ne__(other)	Equivalent to self != other.
__gt__(other)	Equivalent to self > other.
__ge__(other)	Equivalent to self >= other.
__lt__(other)	Equivalent to self < other.
__le__(other)	Equivalent to self <= other.

Special Method Examples

Methods

KFold([n_splits, seed, use_hash, nvl])	Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.
append(other[, all])	Appends the other OML data object of the same class to this data object.
concat(other[, auto_name])	Combines current OML data object with the other data objects column-wise.
count()	Returns the number of elements that are not NULL.
count_pattern(pat[, flags])	Counts the number of occurrences of the pattern in each string.
create_view([view, use_colname])	Creates an Oracle Database view for the data represented by the OML data object.
describe()	Generates descriptive statistics that summarize the central tendency, dispersion, and shape of an OML series data distribution.
drop_duplicates()	Removes duplicated elements.
dropna()	Removes missing values.
find(sub[, start])	Returns the lowest index in each string where substring is found that is greater than or equal to start.
head([n])	Returns the first n elements.

Continued on next page

Table 18 – continued from previous page

`isnull`()	Detects the missing value None.
`len`()	Computes the length of each string.
`materialize`([table])	Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.
`max`([skipna])	Returns the maximum value.
`min`([skipna])	Returns the minimum value.
`nunique`([dropna])	Returns the number of unique values.
`pull`()	Pulls data represented by this object from Oracle Database into an in-memory Python object.
`replace`(old, new[, default])	Replace values given in *old* with *new*.
`sort_values`([ascending, na_position])	Sorts the values in the series data object.
`split`([ratio, seed, use_hash, nvl])	Splits the series data object randomly into multiple sets.
`tail`([n])	Returns the last n elements.

`__init__`(*other*, *dbtype*)

Convert underlying Oracle Database type.

Parameters

other
[oml.String]

dbtype
['varchar2' or 'clob']

`__contains__`(*item*)

Check whether all elements in `item` exists in the String series

Equivalent to `item in self`.

Parameters

item
[str, list of str, oml.String] Values to check in series

Returns

contains
[bool] Returns `True` if all elements exist, otherwise `False`.

`__getitem__`(*key*)

Index self. Equivalent to `self[key]`.

Parameters

key
[oml.Boolean] Must be from the same data source as self.

Returns

subset
[same type as self] Contains only the rows satisfying the condition in `key`.

__len__()

Returns number of rows. Equivalent to `len(self)`.

Returns

rownum
[int]

__eq__(*other*)

Equivalent to `self == other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

equal
[oml.Boolean]

__ne__(*other*)

Equivalent to `self != other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

notequal
[oml.Boolean]

__gt__(*other*)

Equivalent to `self > other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.

- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterthan
[oml.Boolean]

`__ge__`(*other*)
Equivalent to `self >= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterequal
[oml.Boolean]

`__lt__`(*other*)
Equivalent to `self < other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessthan
[oml.Boolean]

`__le__`(*other*)
Equivalent to `self <= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessequal
[oml.Boolean]

KFold(*n_splits=3*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.

Parameters

n_splits
[int, 3 (default)] The number of folds. Must be greater than or equal to 2.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default):] If not None, the specified values are used to hash in place of Null.

Returns

kfold_data
[a list of pairs of series objects of the same type as caller] Each pair within the list is a fold. The first element of the pair is the train set, and the second element is the test set, which consists of all elements not in the train set.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

Examples
See *String.split()*

append(*other*, *all=True*)
Appends the `other` OML data object of the same class to this data object.

Parameters

other
[An OML data object of the same class]

all
[boolean, True (default)] Keeps the duplicated elements from the two data objects.

Returns

appended
[type of caller] A new data object containing the elements from both objects.

Examples
See *String.concat()*

concat(*other*, *auto_name=False*)
Combines current OML data object with the `other` data objects column-wise.

Current object and the `other` data objects must be combinable, that is, they both represent data from the same underlying database table, view, or query.

Parameters

other
[an OML data object, a list of OML data objects, or a dict mapping str to OML data objects.]

- OML data object: an OML series data object or an oml.DataFrame
- list: a sequence of OML series and DataFrame objects to concat.
- dict: a dict mapping str to OML series and DataFrame objects, the column name of concatenated OML series object is replaced with str, column names of concatenated OML DataFrame object is prefixed with str. Need to specify a `collections.OrderedDict` if the concatenation order is expected to follow the key insertion order.

auto_name
[boolean, False (default)] Indicates whether to automatically resolve conflict column names. If True, append duplicated column names with suffix `[column_index]`.

Returns

concat_table
[oml.DataFrame] An oml.DataFrame data object with its original columns followed by the columns of `other`.

Raises

ValueError

- If `other` is not a single nor a list/dict of OML objects, or if `other` is empty.
- If objects in `other` are not from same data source.
- If `auto_name` is False and duplicated column names are detected.

Notes

After concatenation is done, if there is any empty column names in the resulting oml.DataFrame, they will be renamed with `COL[column_index]`.

Examples

```
>>> import oml
>>> import pandas as pd
>>> from collections import OrderedDict
>>> oml_frame = oml.push(pd.DataFrame({"COL1":['a', 'b', 'c', 'd',
↪'e'],
...                                    "COL2":['p', 'w', 'x', 'y',
↪'z'],
...                                    "COL3":['f', 'g', 'h', 'i',
↪'j']}))
>>> oml_x = oml_frame["COL1"]
>>> oml_y = oml_frame["COL2"]
>>> oml_z = oml_frame["COL3"]
```

Concat oml.String.

```
>>> oml_x.concat(oml_y)
  COL1 COL2
0    a    p
1    b    w
2    c    x
3    d    y
4    e    z
```

Concat oml.String and rename the concatenated column.

```
>>> oml_x.concat({'Alphabet':oml_y})
  COL1 Alphabet
0    a        p
1    b        w
2    c        x
3    d        y
4    e        z
```

Concat multiple OML data objects and turn on automatic name conflict resolution.

```
>>> oml_x.concat([oml_z, oml_y, oml_frame], auto_name=True)
  COL1 COL3 COL2 COL14 COL25 COL36
0    a    f    p     a     p     f
1    b    g    w     b     w     g
2    c    h    x     c     x     h
3    d    i    y     d     y     i
4    e    j    z     e     z     j
```

Concat multiple OML data objects and perform customized renaming.

```
>>> oml_x.concat(OrderedDict([('Alphabet',oml_z), ('New_',oml_
↪frame)]))
  COL1 Alphabet New_COL1 New_COL2 New_COL3
0    a        f        a        p        f
1    b        g        b        w        g
2    c        h        c        x        h
3    d        i        d        y        i
4    e        j        e        z        j
```

Append oml.String.

```
>>> oml_x.append(oml_y)
['a', 'b', 'c', 'd', 'e', 'p', 'w', 'x', 'y', 'z']
```

count()

Returns the number of elements that are not NULL.

Returns

nobs
[int]

Examples
See `String.describe()`

count_pattern(*pat*, *flags=0*)

Counts the number of occurrences of the pattern in each string.

Parameters

pat
[str that is a valid regular expression conforming to the POSIX standard]

flags
[int, 0 (default, no flags)] The following `re` module flags are supported:

- `re.I`/`re.IGNORECASE` : Performs case-insensitive matching.
- `re.M`/`re.MULTILINE` : Treats the source string as multiple lines. Interprets the caret (^) and dollar sign ($) as the start and end, respectively, of any line anywhere in source string. Without this flag, the caret and dollar sign match only the start and end, respectively, of the source string.
- `re.S`/`re.DOTALL` : Allows the period (.) to match all characters, including the newline character. Without this flag, the period matches all characters except the newline character.

Multiple flags can be specifed by bitwise OR-ing them.

Returns

counts
[oml.Float]

Examples
See `String.find()`

create_view(*view=None*, *use_colname=False*)

Creates an Oracle Database view for the data represented by the OML data object.

Parameters

view
[str or None (default)] The name of a database view. If `view` is None, the created view is managed by OML and the view is automatically dropped when no longer needed. If a `view` is specified, then it is up to the user to drop the view.

use_colname
[bool, False (default)] Indicates whether to create the view with the same column names as the DataFrame object. Ignored if `view` is specified.

Returns

new_view
[oml.DataFrame]

Raises

TypeError

- If the object represents data from a temporary table or view, and the view to create is meant to persist past the current session.

Examples
See *String.pull()*

describe()
Generates descriptive statistics that summarize the central tendency, dispersion, and shape of an OML series data distribution.

Returns

summary
[pandas.Series] Includes count (number of non-null entries), unique (number of unique entries), top (most common value), freq, (frequency of the most common value).

Examples
Set up a String series data object.

```
>>> import oml
>>> import pandas as pd
>>> df = pd.DataFrame({'string' : [None, None, 'd', 'a', 'a', 'b']})
>>> oml_df = oml.push(df, oranumber = False)
>>> oml_str = oml_df['string']
```

Describe the String series data.

```
>>> oml_str.describe()
count      4
unique     3
top        a
freq       2
Name: string, dtype: object
```

Get the count of all not-Null values in the data.

```
>>> oml_str.count()
4
```

Find the maximum value in the String series data.

```
>>> oml_str.max()
'd'
```

Get the minimum value in the String series data.

```
>>> oml_str.min()
'a'
```

drop_duplicates()
Removes duplicated elements.

Returns

deduplicated
[type of caller]

Examples
See *String.dropna()*

dropna()
Removes missing values.

Missing values include None and/or nan if applicable.

Returns

dropped
[type of caller]

Examples
Setup.

```
>>> import oml
>>> oml_string = oml.push(['a', None, 'b', 'b', 'e'])
>>> oml_string
['a', None, 'b', 'b', 'e']
```

Drop NA values

```
>>> oml_string.dropna()
['a', 'b', 'b', 'e']
```

Drop duplicate values

```
>>> oml_string.drop_duplicates().sort_values()
['a', 'b', 'e', None]
```

Number of unique values

```
>>> oml_string.nunique()
3
>>> oml_string.nunique(dropna=False)
4
```

Null values

```
>>> oml_string.isnull()
[False, True, False, False, False]
```

Replace values with non-matched values perserved

```
>>> oml_string.replace(old=['a','b',None], new=['amy','baker','zoey
↪'])
['amy', 'zoey', 'baker', 'baker', 'e']
```

Replace values and use default value for non-matched values

```
>>> oml_string.replace(old=['a','b'], new=['amy','baker'], default=
↪'eve')
['amy', 'eve', 'baker', 'baker', 'eve']
```

Replace str values with float values

```
>>> oml_string.replace(old=['a','b',None], new=[1.0,2.0,float('nan
↪')], default=0.0)
[1.0, nan, 2.0, 2.0, 0.0]
```

find (*sub*, *start=0*)

Returns the lowest index in each string where substring is found that is greater than or equal to `start`. Returns -1 on failure.

Parameters

sub
[str] The text expression to search.

start
[int] A nonnegative integer indicating when the function begins the search.

Returns

found
[oml.Float]

Examples

Create table from pandas DataFrame and return oml series object

```
>>> import oml
>>> import re
>>> oml_str = oml.create(['s1234', 's123', 's12', 's1', 's123', 's12
↪', 's', 's1234', r'\s12', '\\'], table = "STRING")
>>> type(oml_str)
<class 'oml.core.string.String'>
```

oml.String find

```
>>> oml_str
['s1234', 's123', 's12', 's1', 's123', 's12', 's', 's1234', '\\s12',
↪ '\\']
```

Find string 's12' in each string of oml_str and return the lowest index

```
>>> oml_str.find('s12')
[0, 0, 0, -1, 0, 0, -1, 0, 1, -1]
```

Find string '4' in each string of oml_str beginning from 2nd character and return the lowest index

```
>>> oml_str.find('4', 2)
[4, -1, -1, -1, -1, -1, -1, 4, -1, -1]
```

oml.count_pattern

```
>>> oml_str = oml.create(['fresh bananas Fran\xe7ais \'␣
↪Vertical\x0bTab \\\n\\ bunnies, puppies, and kittens', 'can-can␣
↪dance fa\xe7ade \tHorizontal-tab! [a-e].*$ cotton 1candy.',
↪'Panama canal \xc9galit\xe9 Carriage Return\r [[:lower:]]+␣
↪Samarkand, 12Uzbekistan.'], table = "STRING2")
```

Counts the number of occurrences of the pattern in each string

```
>>> oml_str.count_pattern('.*dy\\.$')
[0, 1, 0]
```

```
>>> oml_str.count_pattern('[CN]A', re.I)
[3, 3, 4]
```

```
>>> oml.drop(table = 'STRING')
>>> oml.drop(table = 'STRING2')
```

head(*n=5*)
Returns the first n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_head
[type of caller]

Examples
See *String.pull()*

isnull()
Detects the missing value None.

Returns

isnull
[oml.Boolean] Indicates missing value None for each element.

Examples
See *String.dropna()*

len()
Computes the length of each string.

Returns

length
[oml.Float]

materialize(*table=None*)
Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.

Parameters

table
[str or None (default)] The name of a table. If `table` is None, an OML-managed table is created, that is, OML drops the table when it is no longer used by any OML object or when you invoke `oml.disconnect(cleanup=True)` to terminate the database connection. If a table is specified, then it's up to the user to drop the named table.

Returns

new_table
[same type as self]

Examples
See `String.pull()`

max(*skipna=True*)
Returns the maximum value.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

max
[Python type corresponding to the column or numpy.nan]

Examples
See `String.describe()`

min(*skipna=True*)
Returns the minimum value.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

min
[Python type corresponding to the column or numpy.nan]

Examples
See `String.describe()`

nunique(*dropna=True*)
Returns the number of unique values.

Parameters

dropna
[bool, True (default)] If True, NULL values are not included in the count.

Returns

nunique
[int]

Examples
See *String.dropna()*

pull()

Pulls data represented by this object from Oracle Database into an in-memory Python object.

Returns

pulled_obj
[list of str and None]

Examples

Create table from pandas DataFrame and return oml series object

```
>>> import oml
>>> oml_str = oml.create([('s' + str(x)) for x in range(100)],␣
↪table = "STRING")
>>> oml_bytes = oml.create([bytes([x]) for x in range(100)],
...                        dbtypes = ['RAW(10)'], table = "BYTES")
>>> type(oml_str)
<class 'oml.core.string.String'>
>>> type(oml_bytes)
<class 'oml.core.bytes.Bytes'>
>>> oml_str.shape
(100, 1)
>>> len(oml_bytes)
100
```

Pull the data from oml series object into a python list

```
>>> oml_str.pull()[:10]
['s0', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9']
>>> oml_bytes.pull()[:5]
[b'\x00', b'\x01', b'\x02', b'\x03', b'\x04']
```

Show first and last 10 elements and sort values

```
>>> oml_str.head(10)
['s0', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9']
>>> oml_str.tail(10)
['s90', 's91', 's92', 's93', 's94', 's95', 's96', 's97', 's98', 's99
↪']
>>> oml_bytes.head()
[b'\x00', b'\x01', b'\x02', b'\x03', b'\x04']
>>> oml_bytes.tail(10)
[b'Z', b'[', b'\\', b']', b'^', b'_', b'`', b'a', b'b', b'c']
>>> oml_str.sort_values(ascending = False).head(10)
['s99', 's98', 's97', 's96', 's95', 's94', 's93', 's92', 's91', 's90
↪']
```

Create a database view from oml series object

```
>>> oml_str_v = oml_str.create_view(view = "STRING_V")
>>> oml.sync(view = "STRING_V").head()
  COL1
0   s0
1   s1
2   s2
3   s3
4   s4
>>> oml_bytes_v = oml_bytes.create_view(view = "BYTES_V")
>>> oml.sync(view = "BYTES_V").head()
      COL1
0  b'\x00'
1  b'\x01'
2  b'\x02'
3  b'\x03'
4  b'\x04'
```

Materialize oml series object into another table

```
>>> oml_str_v.materialize(table = "STRING_T").shape
(100, 1)
>>> oml.sync(table = "STRING_T").head()
  COL1
0   s0
1   s1
2   s2
3   s3
4   s4
>>> oml_bytes_v.materialize(table = "BYTES_T").shape
(100, 1)
>>> oml.sync(table = "BYTES_T").head()
      COL1
0  b'\x00'
1  b'\x01'
2  b'\x02'
3  b'\x03'
4  b'\x04'
```

```
>>> oml.drop(table = 'STRING_T')
>>> oml.drop(view = 'STRING_V')
>>> oml.drop(table = 'STRING')
>>> oml.drop(table = 'BYTES_T')
>>> oml.drop(view = 'BYTES_V')
>>> oml.drop(table = 'BYTES')
```

replace(*old*, *new*, *default=None*)

Replace values given in *old* with *new*.

Parameters

old

[list of float, or list of str] Specifying the old values. When specified with a list of float, it can contain float('nan') and None. When specified with a list of str, it can contain None.

new

[list of float, or list of str] A list of the same length as argument *old* specifying the new values. When specified with a list of float, it can contain float('nan') and None. When specified with a list of str, it can contain None.

default
[float, str, or None (default)] A single value to use for the non-matched elements in argument *old*. If None, non-matched elements will preserve their original values. If not None, data type should be consistent with values in *new*. Must be set when *old* and *new* contain values of different data types.

Returns

replaced
[oml.String]

Raises

ValueError

- if values in *old* have data types inconsistent with original values
- if *default* is specifed with a non-None value which has data type inconsistent with values in *new*
- if *default* is None when *old* and *new* contain values of different data types

Examples
See `String.dropna()`

`sort_values`(*ascending=True*, *na_position='last'*)
Sorts the values in the series data object.

Parameters

ascending
[bool, True (default)] If True, sorts in ascending order. Otherwise, sorts in descending order.

na_position
[{'first', 'last'}, 'last' (default)] `first` places NANs and Nones at the beginning; `last` places them at the end.

Returns

sorted_obj
[type of caller]

Examples
See `String.pull()`

`split`(*ratio=(0.7, 0.3)*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into multiple sets.

Parameters

ratio
[a list of float values or (0.7, 0.3) (default)] All the numbers must be positive and the sum of them are no more than 1. Each number represents the ratio of split data in one set.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default)] If not None, the specified values are used to hash in place of Null.

Returns

split_data
[a list of series objects of the same type as caller] Each of which contains the portion of data by the specified ratio.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

Examples

```
>>> import oml
>>> import numpy as np
>>> np.random.seed(1234)
>>> oml_str = oml.push([str(x) for x in np.random.randint(0, 1000,␣
↪100)])
```

Split into four equal-sized sets.

```
>>> splits = oml_str.split(ratio = (.25, .25, .25, .25), seed =␣
↪1234)
>>> [len(split) for split in splits]
[29, 19, 22, 30]
```

Split randomly into 4 consecutive folds

```
>>> folds = oml_str.KFold(n_splits=4)
>>> [(len(x[0]), len(x[1])) for x in folds]
[(78, 22), (73, 27), (73, 27), (79, 21)]
```

tail(*n=5*)
Returns the last n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_tail
[type of caller]

Examples

See *String.pull()*

Special Method Examples

```
>>> import pandas as pd
>>> fruit_list = ['strawberry', 'cantaloupe', 'plum',
...   'pear', 'apricot', 'guava', 'apple', 'banana',
...   'papaya', 'grape', 'cherry', 'tomato']
>>> oml_fruit = oml.push(
...                 pd.DataFrame({
...                   'fruit' : fruit_list,
...                   'int' : [len(x) for x in fruit_list]}),
...                 dbtypes={'fruit':'clob'})
```

Convert series to have underlying varchar2 type.

```
>>> varfruit = oml.String(oml_fruit['fruit'], 'varchar2')
```

Get all the fruits whose names come after ‘m’ in the alphabet and count them.

```
>>> varfruit > 'm'
[True, False, True, True, False, False, False, False, True, False, False, True]
>>> varfruit[varfruit > 'm']
['strawberry', 'plum', 'pear', 'papaya', 'tomato']
>>> len(varfruit[varfruit > 'm'])
5
```

Check that `varfruit` contains all the fruits in another set.

```
>>> fruit_basket = ['apple', 'cranberry', 'pear']
>>> fruit_basket in varfruit
False
```

1.3.6 oml.DataFrame

class `oml.`**`DataFrame`**`(`*other*`)`

Tabular dataframe class.

Represents multiple columns of Boolean, Bytes, Float, and/or String data.

Attributes

`columns`	The column names of the data set.
`dtypes`	The types of the columns of the data set.
`shape`	The dimensions of the data set.

Special Methods

`__init__`(other)	Convert OML series data object(s) to oml.DataFrame.
`__getitem__`(key)	Index self.
`__len__`()	Returns number of rows.

Special Method Examples

Methods

`KFold`([n_splits, seed, strata_cols, ...])	Splits the oml.DataFrame object randomly into k consecutive folds.
`append`(other[, all])	Appends the `other` OML data object of the same class to this data object.
`concat`(other[, auto_name])	Combines current OML data object with the `other` data objects column-wise.
`corr`([method, min_periods, skipna])	Computes pairwise correlation between all columns where possible, given the type of coefficient.
`count`([numeric_only])	Returns the number of elements that are not NULL for each column.
`create_view`([view, use_colname])	Creates an Oracle Database view for the data represented by the OML data object.
`crosstab`(index[, columns, values, rownames, ...])	Computes a simple cross-tabulation of two or more columns.
`cumsum`(by[, ascending, na_position, skipna])	Gets the cumulative sum of each `Float` or `Integer` or `Boolean` column after the `DataFrame` object is sorted.
`describe`([percentiles, include, exclude])	Generates descriptive statistics that summarize the central tendency, dispersion, and shape of the data in each column
`drop`(columns)	Drops specified columns.
`drop_duplicates`([subset])	Removes duplicated rows from oml.DataFrame object.
`dropna`([how, thresh, subset])	Removes rows containing missing values.
`fillna`(new[, columns])	Fill None values with new value
`head`([n])	Returns the first n elements.
`info`([verbose, max_cols, memory_usage, ...])	Print a concise summary of a DataFrame.
`isnull`()	Detects the missing value None.
`kurtosis`([skipna])	Returns the sample kurtosis of the values for each `Float` column.
`materialize`([table])	Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.
`max`([skipna, numeric_only])	Returns the maximum value in each column.
`mean`([skipna])	Returns the mean of the values for each `Float` or `Integer` or `Boolean` column.
`median`([skipna])	Returns the median of the values for each `Float` or `Integer` column.
`merge`(other[, on, left_on, right_on, how, ...])	Joins data sets.
`min`([skipna, numeric_only])	Returns the minimum value in each column.

Continued on next page

Table 21 – continued from previous page

nunique([dropna])	Returns number of unique values for each column of DataFrame.
pivot_table(index[, columns, values, . . .])	Converts data set to a spreadsheet-style pivot table.
pull([aslist])	Pulls data represented by the DataFrame from Oracle Database into an in-memory Python object.
rename(columns)	Renames columns.
replace(old, new[, default, columns])	Replace values given in *old* with *new* in specified columns.
round([decimals])	Rounds oml.Float values in the oml.DataFrame object to the specified decimal place.
sample([frac, n, random_state])	Return a random sample data sets from an oml.DataFrame object.
select_types([include, exclude])	Returns the subset of columns include/excluding columns based on their OML type.
skew([skipna])	Returns the sample skewness of the values for each `Float` column.
sort_values(by[, ascending, na_position])	Specifies the order in which rows appear in the result set.
split([ratio, seed, strata_cols, use_hash, . . .])	Splits the oml.DataFrame object randomly into multiple data sets.
std([skipna])	Returns the sample standard deviation of the values of each `Float` or `Integer` or `Boolean` column.
sum([skipna])	Returns the sum of the values of each `Float` or `Integer` or `Boolean` column.
t_dot([other, skipna, pull_from_db])	Calculates the matrix cross-product of self with other.
tail([n])	Returns the last n elements.

__init__(*other*)

Convert OML series data object(s) to oml.DataFrame.

Parameters

other

[OML series data object or dict mapping str to OML series data objects]

- OML series data object : initializes a single-column oml.DataFrame containing the same data.
- dict : initializes a oml.DataFrame that comprises all the OML series data objects in the dict in an arbitrary order. Each column in the resulting oml.DataFrame has as its column name its corresponding key in the dict.

__getitem__(*key*)

Index self. Equivalent to `self[key]`.

Parameters

key

[oml.Boolean, str, list of str, 2-tuple]

- oml.Boolean : select only the rows satisfying the condition. Must be from the same data source as self.
- str : select the column of the same name

- list of str : select the columns whose names matches the elements in the list.
- 2-tuple : The first element in the tuple denotes which rows to select. It can be either a oml.Boolean or `slice(None)` (this selects all rows). The second element in the tuple denotes which columns to select. It can be either `slice(None)` (this selects all columns), str, list of str, int, or list of int. If int or list of int, selects the column(s) in the corresponding position(s).

Returns

subset
[OML data object] Is a oml.DataFrame if has more than one column, otherwise is a OML series data object.

`__len__``()`
Returns number of rows. Equivalent to `len(self)`.

Returns

rownum
[int]

`KFold``(`*n_splits=3*, *seed=12345*, *strata_cols=None*, *use_hash=True*, *hash_cols=None*, *nvl=None*`)`
Splits the oml.DataFrame object randomly into k consecutive folds.

Parameters

n_splits
[int, 3 (default)] The number of folds. Must be greater than or equal to 2.

seed
[int or 12345 (default)] The seed to use for random splitting.

strata_cols
[a list of string values or None (default)] Names of the columns used for stratification. If None, stratification is not performed. Must be None when `use_hash` is False.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

hash_cols
[a list of string values or None (default)] If a list of string values, use the values from these named columns to hash to split the data. If None, use the values from the 1st 10 columns to hash.

nvl
[numeric value, str, or None (default)] If not None, the specified values are used to hash in place of Null.

Returns

kfold_data
[a list of k 2-tuples of oml.DataFrame objects]

Raises

ValueError

- If `hash_cols` refers to a single LOB column.

Examples

See `DataFrame.split()`

append(*other*, *all=True*)

Appends the `other` OML data object of the same class to this data object.

Parameters

other
[An OML data object of the same class]

all
[boolean, True (default)] Keeps the duplicated elements from the two data objects.

Returns

appended
[type of caller] A new data object containing the elements from both objects.

Examples

See `DataFrame.merge()`

concat(*other*, *auto_name=False*)

Combines current OML data object with the `other` data objects column-wise.

Current object and the `other` data objects must be combinable, that is, they both represent data from the same underlying database table, view, or query.

Parameters

other
[an OML data object, a list of OML data objects, or a dict mapping str to OML data objects.]

- OML data object: an OML series data object or an oml.DataFrame
- list: a sequence of OML series and DataFrame objects to concat.
- dict: a dict mapping str to OML series and DataFrame objects, the column name of concatenated OML series object is replaced with str, column names of concatenated OML DataFrame object is prefixed with str. Need to specify a `collections.OrderedDict` if the concatenation order is expected to follow the key insertion order.

auto_name
[boolean, False (default)] Indicates whether to automatically resolve conflict column names. If True, append duplicated column names with suffix `[column_index]`.

Returns

concat_table
[oml.DataFrame] An oml.DataFrame data object with its original columns followed by the columns of `other`.

Raises

ValueError

- If `other` is not a single nor a list/dict of OML objects, or if `other` is empty.
- If objects in `other` are not from same data source.
- If `auto_name` is False and duplicated column names are detected.

Notes

After concatenation is done, if there is any empty column names in the resulting oml.DataFrame, they will be renamed with `COL[column_index]`.

Examples

See `DataFrame.merge()`

corr(*method='pearson'*, *min_periods=1*, *skipna=True*)

Computes pairwise correlation between all columns where possible, given the type of coefficient.

Parameters

method

['pearson' (default), 'kendall', or 'spearman']

- pearson : Uses Pearson's correlation coefficient. Can only calculate correlations between Float or Integer or Boolean columns.
- kendall : Uses Kendall's tau-b coefficient.
- spearman : Uses Spearman's rho coefficient.

min_periods

[int, optional, 1 (default)] The minimum number of observations required per pair of columns to have a valid result.

skipna

[bool, True (default)] If True, NaN and (+/-)Inf values are mapped to NULL.

Returns

y [`pandas.DataFrame`]

Examples

See `DataFrame.describe()`

count(*numeric_only=False*)

Returns the number of elements that are not NULL for each column.

Parameters

numeric_only

[boolean, False (default)] Includes only Float, Integer and Boolean columns.

Returns

count

[`pandas.Series`]

Examples

See `DataFrame.describe()`

create_view(*view=None*, *use_colname=False*)

Creates an Oracle Database view for the data represented by the OML data object.

Parameters

view
[str or None (default)] The name of a database view. If `view` is None, the created view is managed by OML and the view is automatically dropped when no longer needed. If a `view` is specified, then it is up to the user to drop the view.

use_colname
[bool, False (default)] Indicates whether to create the view with the same column names as the DataFrame object. Ignored if `view` is specified.

Returns

new_view
[oml.DataFrame]

Raises

TypeError

- If the object represents data from a temporary table or view, and the view to create is meant to persist past the current session.

Examples

See `DataFrame.pull()`

crosstab(*index*, *columns=None*, *values=None*, *rownames=None*, *colnames=None*, *aggfunc=None*, *margins=False*, *margins_name='All'*, *dropna=True*, *normalize=False*, *pivot=False*)

Computes a simple cross-tabulation of two or more columns. By default, computes a frequency table for the columns unless a column and an aggregation function have been passed.

Parameters

index
[str or list of str] Names of the column(s) of the DataFrame to group by. If `pivot` is True, these columns are displayed in the rows of the result table.

columns
[str or list of str, optional] Names of the other column(s) of the Dataframe to group by. If `pivot` is True, these columns are displayed in the columns of the result table.

values
[str, optional] The name of the column to aggregate according to the grouped columns. Requires `aggfunc` to be specified.

aggfunc
[OML DataFrame aggregation function object, optional] The supported oml.DataFrame aggregation functions include: count, max, mean, median, min, nunique, std and sum. To use `aggfunc`, specify the function object using its full name, for example, `oml.DataFrame.sum`, `oml.DataFrame.nunique`, and so on. If specified, requires `values` to also be specified.

rownames
[str or list of str, None (default)] If specified, must match number of names in `index`. If None, names in `index` are used.

colnames
[str or list of str, None (default)] If specified, must match number of strings in `columns`. If None, names in `columns` are used. Ignored if `pivot` is True.

margins
[bool, False (default)] Includes row and column margins (subtotals)

margins_name
[str, 'All' (default)] Names of the row and column that contain the totals when `margins` is True. Should be a value not contained in any of the columns specified by `index` and `columns`.

dropna
[bool, True (default)] In addition, if `pivot` is True, drops columns from the result table if all the entries of the column are NaN.

normalize
[boolean, {'all', 'index', 'columns'} or {0, 1}, False (default)] Normalizes by dividing the values by their sum.

- If 'all' or True, normalizes over all values.
- If 'index' or 0, normalizes over each row.
- If 'columns' or 1, normalizes over each column.
- If `margins` is True, also normalizes margin values.

pivot
[bool, False (default)] If True, returns results in pivot table format. Else, returns results in relational table format.

Returns

crosstab
[oml.DataFrame]

See also:

[`DataFrame.pivot_table`]

Examples
See `DataFrame.pivot_table()`

cumsum(*by*, *ascending=True*, *na_position='last'*, *skipna=True*)
Gets the cumulative sum of each `Float` or `Integer` or `Boolean` column after the `DataFrame` object is sorted.

Parameters

by
[str or list of str] A single column name or list of column names by which to sort the DataFrame object. Columns in `by` do not have to be `Float` or `Integer` or `Boolean`.

ascending
[bool or list of bool, True (default)] If True, sort is in ascending order, otherwise descending. Specify list for multiple sort orders. If this is a list of bools, must match the length of `by`.

na_position
[{'first', 'last'}, 'last' (default)] `first` places NaN and None at the beginning, `last` places them at the end.

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

cumsum
[oml.DataFrame]

Examples
See `DataFrame.describe()`

describe(*percentiles=None*, *include=None*, *exclude=None*)
Generates descriptive statistics that summarize the central tendency, dispersion, and shape of the data in each column

Parameters

percentiles
[bool, list-like of numbers, or None (default), optional] The percentiles to include in the output for *Float* columns. All must be between 0 and 1. If `percentiles` is None or True, `percentiles` is set to `[.25, .5, .75]`, which corresponds to the 25th, 50th, and 75th percentiles. If *percentiles* is False, only `min` and `max` stats and no other percentiles is included.

include
['all', list-like of OML column types or None (default), optional] Types of columns to include in the result. Available options:

- 'all': Includes all columns.
- List of OML column types : Only includes specified types in the results.
- None (default) : If `Float` columns exist and `exclude` is None, only includes `Float` columns. Otherwise, includes all columns.

exclude
[list of OML column types or None (default), optional] Types of columns to exclude from the result. Available options:

- List of OML column types : Excludes specified types from the results.
- None (default) : Result excludes nothing.

Returns

summary
[`pandas.DataFrame`] The concatenation of the summary statistics for each column.

See also:

DataFrame.count

DataFrame.max

DataFrame.min

DataFrame.mean

DataFrame.std

DataFrame.select_types

Examples

Set up an oml.DataFrame object.

```
>>> import oml
>>> import pandas as pd
>>> df = pd.DataFrame({'numeric': [1, 1.4, -4, 3.145, 5, None],
...                    'string' : [None, None, 'a', 'a', 'a', 'b'],
...                    'bytes' : [b'a', b'b', b'c', b'c', b'd', b'e
↪']})
>>> oml_df = oml.push(df, dbtypes = {'numeric': 'BINARY_DOUBLE',
...                                  'string':'CHAR(1)', 'bytes':
↪'RAW(1)'})
```

Specify percentiles in the output.

```
>>> oml_df.describe(percentiles=[.33, .66])
        numeric
count  5.000000
mean   1.309000
std    3.364655
min   -4.000000
33%    1.128000
50%    1.400000
66%    2.516800
max    5.000000
```

Describe all columns.

```
>>> oml_df.describe(include='all')
         numeric string bytes
count   5.000000      4     6
unique       NaN      2     5
top          NaN      a  b'c'
freq         NaN      3     2
mean    1.309000    NaN   NaN
std     3.364655    NaN   NaN
min    -4.000000    NaN   NaN
25%     1.000000    NaN   NaN
50%     1.400000    NaN   NaN
75%     3.145000    NaN   NaN
max     5.000000    NaN   NaN
```

Exclude Float columns.

```
>>> oml_df.describe(exclude=[oml.Float])
       string bytes
count       4     6
unique      2     5
top         a  b'c'
freq        3     2
```

Let's explore the iris dataset.

```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_
↪Width',
...                                        'Petal_Length', 'Petal_
↪Width'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor
↪',
...          2:'virginica'}[x], iris.target)), columns = ['Species'])
```

Modify the dataset by replacing a few entries with NA so that the columns have different counts of not-Null values.

```
>>> x['Sepal_Width'].replace({2.7: None}, inplace=True)
>>> x['Petal_Length'].replace({6.0: None}, inplace=True)
>>> x['Petal_Width'].replace({2.5: None}, inplace=True)
>>> iris_df = pd.concat([x, y], axis=1)
>>> oml_iris = oml.create(iris_df, table = 'IRIS')
```

Get a general description of the data with specified percentiles.

```
>>> oml_iris.describe(percentiles=[.3, .7])
       Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
count    150.000000   141.000000    148.000000   147.000000
mean       5.843333     3.080142      3.727703     1.172789
std        0.828066     0.439841      1.757658     0.746642
min        4.300000     2.000000      1.000000     0.100000
30%        5.270000     2.900000      1.700000     0.400000
50%        5.800000     3.000000      4.300000     1.300000
70%        6.300000     3.300000      4.900000     1.700000
max        7.900000     4.400000      6.900000     2.400000
```

Exclude Float columns.

```
>>> oml_iris.describe(exclude=[oml.Float])
       Species
count      150
unique       3
top     setosa
freq        50
```

Find pairwise correlation between all columns where possible.

```
>>> oml_iris.corr()
              Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
Sepal_Length      1.000000    -0.120002      0.871226     0.816090
Sepal_Width      -0.120002     1.000000     -0.428795    -0.396155
```

(continues on next page)

(continued from previous page)

```
Petal_Length      0.871226    -0.428795      1.000000      0.964162
Petal_Width       0.816090    -0.396155      0.964162      1.000000
```

Get the count of all not-Null values in each column.

```
>>> oml_iris.count()
Sepal_Length    150
Sepal_Width     141
Petal_Length    148
Petal_Width     147
Species         150
dtype: int64
```

Find the maximum value in each `Float` or `Boolean` column.

```
>>> oml_iris.max(numeric_only = True)
Sepal_Length    7.9
Sepal_Width     4.4
Petal_Length    6.9
Petal_Width     2.4
dtype: float64
```

Get the minimum value in each `Float` or `Boolean` column.

```
>>> oml_iris.min(numeric_only = True)
Sepal_Length    4.3
Sepal_Width     2.0
Petal_Length    1.0
Petal_Width     0.1
dtype: float64
```

Find the mean value in each `Float` or `Boolean` column.

```
>>> oml_iris.mean()
Sepal_Length    5.843333
Sepal_Width     3.080142
Petal_Length    3.727703
Petal_Width     1.172789
dtype: float64
```

Get the median value in each `Float` or `Boolean` column.

```
>>> oml_iris.median()
Sepal_Length    5.8
Sepal_Width     3.0
Petal_Length    4.3
Petal_Width     1.3
dtype: float64
```

Find the standard deviation of values in each `Float` or `Boolean` column.

```
>>> oml_iris.std()
Sepal_Length    0.828066
Sepal_Width     0.439841
Petal_Length    1.757658
Petal_Width     0.746642
dtype: float64
```

Get the skewness of values in each `Float` or `Boolean` column.

```
>>> oml_iris.skew()
Sepal_Length    0.311753
Sepal_Width     0.205437
Petal_Length   -0.255895
Petal_Width    -0.112624
dtype: float64
```

Find the kurtosis of values in each `Float` or `Boolean` column.

```
>>> oml_iris.kurtosis()
Sepal_Length   -0.573568
Sepal_Width     0.178920
Petal_Length   -1.400858
Petal_Width    -1.369464
dtype: float64
```

Get the sum of values in each `Float` or `Boolean` column.

```
>>> oml_iris.sum()
Sepal_Length    876.5
Sepal_Width     434.3
Petal_Length    551.7
Petal_Width     172.4
dtype: float64
```

Find the cumulative sum of values in each `Float` or `Boolean` column after sorted by 'Sepal_Length'.

```
>>> oml_iris.cumsum(by='Sepal_Length')
      Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
 0             4.3          3.0           1.1          0.1
 1             8.7          6.0           2.4          0.3
 2            13.1          9.2           3.7          0.5
 3            17.5         12.1           5.1          0.7
...            ...          ...           ...          ...
 146         853.2        424.1         531.7        165.9
 147         860.9        427.9         538.4        168.1
 148         868.6        430.5         545.3        170.4
 149         876.5        434.3         551.7        172.4

[150 rows x 4 columns]
```

```
>>> oml.drop(table = 'IRIS')
```

drop(*columns*)

Drops specified columns.

Parameters

columns
[str or list of str] Columns to drop from the object.

Returns

dropped
[oml.DataFrame]

Examples
See *DataFrame.pull()*

drop_duplicates(*subset=None*)
Removes duplicated rows from oml.DataFrame object.

Use subset to consider a set of rows duplicates if they have identical values for only a subset of the columns. In this case, after deduplication, each of the other columns contains the minimum value found across the set.

Parameters

subset
[str or list of str, optional] Columns to consider for identifying duplicates. If None, use all columns.

Returns

deduplicated
[oml.DataFrame]

dropna(*how='any'*, *thresh=None*, *subset=None*)
Removes rows containing missing values.

Parameters

how
[{'any', 'all'}, 'any' (default)] Determines if row is removed from DataFrame when at least one or all values are missing.

thresh
[int, optional] Requires that many of missing values to drop a row from DataFrame.

subset
[list, optional] The names of the columns to check for missing values.

Returns

dropped
[oml.DataFrame] DataFrame without missing values.

Examples
Setup.

```
>>> import oml
>>> import pandas as pd
>>> df = pd.DataFrame({'numeric': [1, 1.4, -4, -4, 5.432, None,␣
↪None],
...                    'string' : [None, None, 'a', 'a', 'a', 'b',␣
↪None],
...                    'string2': ['x', None, 'a', 'z', None, 'x',␣
↪None]})
>>> oml_df = oml.push(df, dbtypes = {'numeric': 'BINARY_DOUBLE',
...                               'string':'CHAR(1)', 'string1':
↪'CHAR(1)'})
```

(continues on next page)

(continued from previous page)

```
>>> oml_df
   numeric string string2
0    1.000   None       x
1    1.400   None    None
2   -4.000      a       a
3   -4.000      a       z
4    5.432      a    None
5      NaN      b       x
6      NaN   None    None
```

Drop NA values

```
>>> oml_df.dropna(how='any')
   numeric string string2
0     -4.0      a       a
1     -4.0      a       z
```

```
>>> oml_df.dropna(how='all')
   numeric string string2
0    1.000   None       x
1    1.400   None    None
2   -4.000      a       a
3   -4.000      a       z
4    5.432      a    None
5      NaN      b       x
```

```
>>> oml_df.dropna(how='any', subset=['numeric'])
   numeric string string2
0    1.000   None       x
1    1.400   None    None
2   -4.000      a       a
3   -4.000      a       z
4    5.432      a    None
```

Drop Duplicates

```
>>> oml_df.drop_duplicates().sort_values(['numeric', 'string',
↪'string2'])
   numeric string string2
0   -4.000      a       a
1   -4.000      a       z
2    1.000   None       x
3    1.400   None    None
4    5.432      a    None
5      NaN      b       x
6      NaN   None    None
```

```
>>> oml_df.drop_duplicates(subset=['string', 'string2'])
   numeric string string2
0    5.432      a    None
1   -4.000      a       z
2    1.400   None    None
3    1.000   None       x
4   -4.000      a       a
5      NaN      b       x
```

Number of unique values

```
>>> oml_df.nunique()
numeric     4
string      2
string2     3
Name: 0, dtype: int64
```

Replace values in column 'numeric' and perserve non-matched values

```
>>> oml_df.replace(old=[-4.0,float('nan')], new=[-400.0,0.0],␣
↪columns=['numeric'])
   numeric string string2
0    1.000   None       x
1    1.400   None    None
2 -400.000      a       a
3 -400.000      a       z
4    5.432      a    None
5    0.000      b       x
6    0.000   None    None
```

Replace values in column 'numeric' with default value for non-matched values

```
>>> oml_df.replace(old=[1.0, -4.0], new=[100.0, -400.0], default=0.
↪0, columns=['numeric'])
   numeric string string2
0      100   None       x
1        0   None    None
2     -400      a       a
3     -400      a       z
4        0      a    None
5        0      b       x
6        0   None    None
```

Replace str values in two columns

```
>>> oml_df.replace(old=['a','b',None], new=['amy','baker','zoey'],␣
↪columns=['string','string2'])
   numeric string string2
0    1.000   zoey       x
1    1.400   zoey    zoey
2   -4.000    amy     amy
3   -4.000    amy       z
4    5.432    amy    zoey
5      NaN  baker       x
6      NaN   zoey    zoey
```

Replace str values with numeric values

```
>>> oml_df.replace(old=['a','b',None], new=[0.0,1.0,float('nan')],␣
↪default=0.0, columns=['string'])
   numeric  string string2
0    1.000     NaN       x
1    1.400     NaN    None
2   -4.000     0.0       a
3   -4.000     0.0       z
4    5.432     0.0    None
```

(continues on next page)

(continued from previous page)

```
5       NaN      1.0       x
6       NaN      NaN    None
```

Check None or NaN values

```
>>> oml_df.isnull()
   numeric  string  string2
0    False    True    False
1    False    True     True
2    False   False    False
3    False   False    False
4    False   False     True
5     True   False    False
6     True    True     True
```

Fill None or NaN values

```
>>> oml_df.fillna(new = {'numeric':0,'string':'NA','string2':'NA'})
   numeric string string2
0    1.000     NA       x
1    1.400     NA      NA
2   -4.000      a       a
3   -4.000      a       z
4    5.432      a      NA
5    0.000      b       x
6    0.000     NA      NA
```

fillna (*new*, *columns=None*)

Fill None values with new value

Parameters

new

[dict, str, int, float or bool] `dict` specifies column names as key with the corresponding new values to fill. `str` specifies new str value to replace None in the oml.String columns `int` specifies new int value to replace None in the oml.Integer columns `float` specifies new float value to replace None or NaN in the oml.Float `bool` specifies new Boolean values to replace None in the oml.Boolean columns

columns

[list of str] names of columns to replace None or NaN

Returns

filledna

[oml.DataFrame]

Examples

See *DataFrame.dropna()*

head (*n=5*)

Returns the first n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_head
[type of caller]

Examples
See `DataFrame.pull()`

info(*verbose=None*, *max_cols=None*, *memory_usage=None*, *show_counts=None*)
Print a concise summary of a DataFrame.

Parameters

verbose
[bool] `bool` specifies if all info is going to be display

max_cols
[bools] `bool` specifies if max_cols is going to be display

memory_usage:
`bool` specifies if memory_usage is going to be display

show_counts:
`bool` specifies if show_counts is going to be display

Returns

This method prints a summary of a DataFrame

including the index dtype and columns, non-null values

isnull()
Detects the missing value None.

Returns

isnull
[oml.DataFrame]

Indicates missing value None for each element.

Examples
See `DataFrame.dropna()`

kurtosis(*skipna=True*)
Returns the sample kurtosis of the values for each `Float` column.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result

Returns

kurt
[`pandas.Series`]

Examples
See `DataFrame.describe()`

materialize(*table=None*)
Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.

Parameters

table
[str or None (default)] The name of a table. If `table` is None, an OML-managed table is created, that is, OML drops the table when it is no longer used by any OML object or when you invoke `oml.disconnect(cleanup=True)` to terminate the database connection. If a table is specified, then it's up to the user to drop the named table.

Returns

new_table
[same type as self]

Examples
See `DataFrame.pull()`

max(*skipna=True*, *numeric_only=False*)
Returns the maximum value in each column.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result

numeric_only
[boolean, False (default)] Includes only `Float`, `Integer` and `Boolean` columns.

Returns

max
[`pandas.Series`]

Examples
See `DataFrame.describe()`

mean(*skipna=True*)
Returns the mean of the values for each `Float` or `Integer` or `Boolean` column.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result

Returns

mean
[pandas.Series]

Examples
See *DataFrame.describe()*

median(*skipna=True*)
Returns the median of the values for each Float or Integer column.

Parameters

skipna
[boolean, True (default)] Exclude NaN values when computing the result

Returns

median
[pandas.Series]

Examples
See *DataFrame.describe()*

merge(*other*, *on=None*, *left_on=None*, *right_on=None*, *how='left'*, *suffixes=('_l', '_r')*, *nvl=True*)
Joins data sets.

Parameters

other
[an OML data set object]

on
[str or list of str, optional] Column names to join on. Must be found in both self and other.

left_on
[str or list of str, optional] Column names of self to join on.

right_on
[str or list of str, optional] Column names of other to join on. If specified, must specify the same number of columns as left_on.

how
['left' (default), 'right', 'inner', 'full']

- left : left outer join
- right : right outer join
- full : full outer join
- inner : inner join

If on and left_on are both None, then how is ignored, and a cross join is performed.

suffixes
[sequence of length 2] Suffix to apply to column names on the left and right side, respectively.

nvl
[True (default), False, dict]

- True : join condition includes NULL value

- False : join condition excludes NULL value
- dict : specifies the values that join columns use in replacement of NULL value with column names as keys

Returns

merged
[oml.DataFrame]

Examples

```
>>> import oml
>>> import pandas as pd
>>> from collections import OrderedDict
>>> df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
...                     "id2" : [1.0, 2.0, 3.0, 4.0, 5.0],
...                     "val" : ["a", "b", "c", "d", "e"],
...                     "ch" : ["p", "q", "r", "a", "b"],
...                     "num" : [4, 3, 6.7, 7.2, 5]})
>>> oml_frame = oml.push(df)
```

oml.DataFrame shape

```
>>> w = oml_frame['id']
>>> x = oml_frame[['id', 'val']]
>>> x2 = oml_frame[['id2', 'val']]
>>> y = oml_frame[['num', 'ch']]
```

Concat oml.DataFrame

```
>>> x.concat(y)
   id val  num ch
0   1   a  4.0  p
1   2   b  3.0  q
2   3   c  6.7  r
3   4   d  7.2  a
4   5   e  5.0  b
>>> y.concat(w)
   num ch  id
0  4.0  p   1
1  3.0  q   2
2  6.7  r   3
3  7.2  a   4
4  5.0  b   5
```

Concat oml.DataFrame and rename the concatenated column.

```
>>> x.concat({'ID':w})
   id val  ID
0   1   a   1
1   2   b   2
2   3   c   3
3   4   d   4
4   5   e   5
```

Concat multiple OML data objects and turn on automatic name conflict resolution.

```
>>> x.concat([w, y], auto_name=True)
   id val  id3  num ch
0   1   a    1  4.0  p
1   2   b    2  3.0  q
2   3   c    3  6.7  r
3   4   d    4  7.2  a
4   5   e    5  5.0  b
```

Concat multiple OML data objects and perform customized renaming.

```
>>> x.concat(OrderedDict([('ID',w), ('New_',oml_frame)]))
   id val  ID  New_id  New_id2 New_val New_ch  New_num
0   1   a   1       1        1       a      p      4.0
1   2   b   2       2        2       b      q      3.0
2   3   c   3       3        3       c      r      6.7
3   4   d   4       4        4       d      a      7.2
4   5   e   5       5        5       e      b      5.0
```

Append oml.DataFrames

```
>>> x2.append(y)
   id2 val
0  1.0   a
1  2.0   b
2  3.0   c
3  4.0   d
4  5.0   e
5  4.0   p
6  3.0   q
7  6.7   r
8  7.2   a
9  5.0   b
```

Cross join to another OML data set object

```
>>> z = x.merge(y)
>>> z.sort_values(by = ['id_l', 'val_l', 'num_r']).head()
   id_l val_l  num_r ch_r
0     1     a    3.0    q
1     1     a    4.0    p
2     1     a    5.0    b
3     1     a    6.7    r
4     1     a    7.2    a
>>> z.shape
(25, 4)
>>> y.merge(w)
    num_l ch_l  id_r
0     4.0    p     1
1     4.0    p     2
2     4.0    p     3
3     4.0    p     4
4     4.0    p     5
5     3.0    q     1
...   ...   ...  ...
19    7.2    a     5
20    5.0    b     1
21    5.0    b     2
```

(continues on next page)

(continued from previous page)

```
22      5.0     b      3
23      5.0     b      4
24      5.0     b      5
```

Perform a left join

```
>>> x.head(4).merge(other=oml_frame[['id', 'num']], on="id",␣
↪suffixes=['.l','.r'])
   id val.l  num.r
0   1     a    4.0
1   2     b    3.0
2   3     c    6.7
3   4     d    7.2
```

Perform a right join

```
>>> x.merge(other=y, left_on="id", right_on="num", how="right").
↪sort_values(['num_r'])
   id_l val_l  num_r ch_r
0   3.0     c    3.0    q
1   4.0     d    4.0    p
2   5.0     e    5.0    b
3   NaN  None    6.7    r
4   NaN  None    7.2    a
```

min(*skipna=True*, *numeric_only=False*)

Returns the minimum value in each column.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result

numeric_only
[boolean, False (default)] Includes only `Float`, `Integer` and `Boolean` columns

Returns

min
[`pandas.Series`]

Examples
See `DataFrame.describe()`

nunique(*dropna=True*)

Returns number of unique values for each column of DataFrame.

Parameters

dropna
[bool, True (default)] If True, NULL values are not included in the count.

Returns

nunique
[pandas.Series]

Examples

See `DataFrame.dropna()`

pivot_table(*index*, *columns=None*, *values=None*, *aggfunc=<function DataFrame.mean>*, *margins=False*, *dropna=True*, *margins_name='All'*)

Converts data set to a spreadsheet-style pivot table. Due to the Oracle 1000 column limit, pivot tables with more than 1000 columns are automatically truncated to display the categories with the most entries for each value column.

Parameters

index
[str or list of str] Names of columns containing the keys to group by on the pivot table index.

columns
[str or list of str, optional] Names of columns containing the keys to group by on the pivot table columns.

values
[str or list of str, optional] Names of columns to aggregate on. If None, values are inferred as all columns not in `index` or `columns`.

aggfunc
[OML DataFrame aggregation function or a list of them, oml.DataFrame.mean (default)] The supported oml.DataFrame aggregation functions include: count, max, mean, median, min, nunique, std and sum. When using aggregation functions, specify the function object using its full name, for example, `oml.DataFrame.sum`, `oml.DataFrame.nunique`, and so on. If `aggfunc` contains more than one function, each function is applied to each column in `values`. If the function does not apply to the type of a column in `values`, the result table skips applying the function to the particular column.

margins
[bool, False (default)] Include row and column margins (subtotals)

dropna
[bool, True (default)] Unless `columns` is None, drop column labels from the result table if all the entries corresponding to the column label are NaN for all aggregations.

margins_name
[string, 'All' (default)] Names of the row and column that contain the totals when `margins` is True. Should be a value not contained in any of the columns specified by `index` and `columns`.

Returns

pivoted
[oml.DataFrame]

See also:

DataFrame.crosstab

Examples

```
>>> import pandas as pd
>>> x = pd.DataFrame({
...         'GENDER': ['M', 'M', 'F', 'M', 'F', 'M', 'F', 'F', None,
↪'F', 'M', 'F'],
...         'HAND': ['L', 'R', 'R', 'L', 'R', None, 'L', 'R', 'R', 'R
↪', 'R', 'R'],
...         'SPEED': [40.5, 30.4, 60.8, 51.2, 54, 29.3, 34.1, 39.6,␣
↪46.4, 12, 25.3, 37.5],
...         'ACCURACY': [.92, .94, .87, .9, .85, .97, .96, .93, .89, .
↪84, .91, .95]
...         })
>>> x = oml.push(x)
```

Find the categories that the most entries belonged to.

```
>>> x.crosstab('GENDER', 'HAND').sort_values(['GENDER', 'HAND'],␣
↪ascending=False)
  GENDER  HAND  count
0      M     R      2
1      M     L      2
2      M  None      1
3      F     R      5
4      F     L      1
5   None     R      1
```

For each gender value and across all entries, find the ratio of entries with different hand values.

```
>>> x.crosstab('GENDER', 'HAND', pivot = True, margins = True,␣
↪normalize = 0)
  GENDER  count_(L)  count_(R)  count_(None)
0   None   0.000000   1.000000      0.000000
1      F   0.166667   0.833333      0.000000
2      M   0.400000   0.400000      0.200000
3    All   0.250000   0.666667      0.083333
```

Find mean speed across all gender and hand combinations.

```
>>> x.pivot_table('GENDER', 'HAND', 'SPEED')
  GENDER  mean(SPEED)_(L)  mean(SPEED)_(R)  mean(SPEED)_(None)
0   None              NaN            46.40                 NaN
1      F            34.10            40.78                 NaN
2      M            45.85            27.85                29.3
```

Find median accuracy and speed for every gender and hand combination.

```
>>> x.pivot_table('GENDER', 'HAND', aggfunc = oml.DataFrame.median)
  GENDER  median(ACCURACY)_(L)  median(ACCURACY)_(R) ␣
↪median(ACCURACY)_(None)   \
0   None                   NaN                 0.890                    ␣
↪       NaN
1      F                  0.96                 0.870                    ␣
↪       NaN
2      M                  0.91                 0.925                    ␣
↪      0.97

   median(SPEED)_(L)   median(SPEED)_(R)   median(SPEED)_(None)
```

(continues on next page)

(continued from previous page)

```
0                 NaN                46.40                    NaN
1               34.10                39.60                    NaN
2               45.85                27.85                   29.3
```

Find max and min speed for every gender and hand combination and across all combinations.

```
>>> x.pivot_table('GENDER', 'HAND', 'SPEED',
...               aggfunc = [oml.DataFrame.max, oml.DataFrame.min],
...               margins = True)
  GENDER  max(SPEED)_(L)  max(SPEED)_(R)  max(SPEED)_(None) ␣
↪max(SPEED)_(All)   \
0   None             NaN            46.4                NaN              ␣
↪     46.4
1      F            34.1            60.8                NaN              ␣
↪     60.8
2      M            51.2            30.4               29.3              ␣
↪     51.2
3    All            51.2            60.8               29.3              ␣
↪     60.8

   min(SPEED)_(L)  min(SPEED)_(R)  min(SPEED)_(None)  min(SPEED)_
↪(All)
0             NaN            46.4                NaN              ␣
↪46.4
1            34.1            12.0                NaN              ␣
↪12.0
2            40.5            25.3               29.3              ␣
↪25.3
3            34.1            12.0               29.3              ␣
↪12.0
```

pull (*aslist=False*)

Pulls data represented by the DataFrame from Oracle Database into an in-memory Python object.

Parameters

aslist
[bool] If False, returns a pandas.DataFrame. Otherwise, returns the data as a list of tuples.

Returns

pulled_obj
[`pandas.DataFrame` or list of tuples]

Examples

```
>>> import oml
>>> import pandas as pd
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_
↪Width',
...                                         'Petal_Length', 'Petal_
↪Width'])
```

(continues on next page)

(continued from previous page)

```
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor
↪',
...         2:'virginica'}[x], iris.target)), columns = ['Species'])
```

Create table from pandas DataFrame and return oml.DataFrame

```
>>> iris_df = pd.concat([x, y], axis=1)
>>> oml_iris = oml.create(iris_df, table = 'IRIS')
>>> type(oml_iris)
<class 'oml.core.frame.DataFrame'>
```

oml.DataFrame columns, shape, len

```
>>> oml_iris.columns
['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width',
↪'Species']
>>> oml_iris.shape
(150, 5)
>>> len(oml_iris)
150
```

Pull the data from oml.DataFrame object to a pandas DataFrame

```
>>> pd_df = oml_iris.pull()
>>> type(pd_df)
<class 'pandas.core.frame.DataFrame'>
>>> pd_df.columns
Index(['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width',
       'Species'],
      dtype='object')
>>> pd_df.shape
(150, 5)
>>> pd_df.head()
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width Species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

Show first and last 5 rows

```
>>> oml_iris.head()
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width Species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
>>> oml_iris.tail()
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width    Species
0           6.7          3.0           5.2          2.3  virginica
1           6.3          2.5           5.0          1.9  virginica
2           6.5          3.0           5.2          2.0  virginica
3           6.2          3.4           5.4          2.3  virginica
4           5.9          3.0           5.1          1.8  virginica
```

Sort oml.DataFrame object by columns

```
>>> oml_iris.sort_values(by = ['Sepal_Length', 'Sepal_Width']).
↪head()
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width Species
0           4.3          3.0           1.1          0.1  setosa
1           4.4          2.9           1.4          0.2  setosa
2           4.4          3.0           1.3          0.2  setosa
3           4.4          3.2           1.3          0.2  setosa
4           4.5          2.3           1.3          0.3  setosa
```

Create a database view from oml.DataFrame object

```
>>> oml_iris_v = oml_iris.create_view(view = "IRIS_V")
>>> oml.sync(view = "IRIS_V").head()
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width Species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

Materialize oml.DataFrame object into another table

```
>>> oml_iris_v.materialize(table = "IRIS_T").shape
(150, 5)
>>> oml.sync(table = "IRIS_T").head()
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width Species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

Rename oml.DataFrame columns

```
>>> oml_iris.rename({'Sepal_Length':'Sepal_Length2',
...                  'Species':'Species2'}).head()
   Sepal_Length2  Sepal_Width  Petal_Length  Petal_Width Species2
0            5.1          3.5           1.4          0.2   setosa
1            4.9          3.0           1.4          0.2   setosa
2            4.7          3.2           1.3          0.2   setosa
3            4.6          3.1           1.5          0.2   setosa
4            5.0          3.6           1.4          0.2   setosa
```

Drop columns from oml.DataFrame object

```
>>> oml_iris.drop(['Petal_Width', 'Species2']).head()
   Sepal_Length2  Sepal_Width  Petal_Length
0            5.1          3.5           1.4
1            4.9          3.0           1.4
2            4.7          3.2           1.3
3            4.6          3.1           1.5
4            5.0          3.6           1.4
```

```
>>> oml.drop(table = 'IRIS_T')
>>> oml.drop(view = 'IRIS_V')
>>> oml.drop(table = 'IRIS')
```

rename(*columns*)

Renames columns.

Parameters

columns
[dict or list] `dict` contains old and new column names. `list` contains the new names for all the columns in order.

Returns

renamed
[DataFrame]

Notes

The method changes the column names of the caller DataFrame object too.

Examples
See `DataFrame.pull()`

replace(*old*, *new*, *default=None*, *columns=None*)

Replace values given in *old* with *new* in specified columns.

Parameters

columns
[list of str or None (default)] Columns to look for values in *old*. If None, then all columns of DataFrame will be replaced.

old
[list of float, or list of str] Specifying the old values. When specified with a list of float, it can contain float('nan') and None. When specified with a list of str, it can contain None.

new
[list of float, or list of str] A list of the same length as argument *old* specifying the new values. When specified with a list of float, it can contain float('nan') and None. When specified with a list of str, it can contain None.

default
[float, str, or None (default)] A single value to use for the non-matched elements in argument *old*. If None, non-matched elements will preserve their original values. If not None, data type should be consistent with values in *new*. Must be set when *old* and *new* contain values of different data types.

Returns

replaced
[oml.DataFrame]

Raises

ValueError

- if values in *old* have data types inconsistent with original values in the target columns

- if *default* is specifed with a non-None value which has data type inconsistent with values in *new*
- if *default* is None when *old* and *new* contain values of different data types

Examples

See `DataFrame.dropna()`

round (*decimals=0*)

Rounds oml.Float values in the oml.DataFrame object to the specified decimal place.

Parameters

decimals
[non-negative int]

Returns

rounded: oml.DataFrame

Examples

```
>>> import oml
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(1234)
>>> oml_dataframe = oml.push(pd.DataFrame({'FLOAT1': [None, -12.1,␣
↪1234, 40.00, 95.612], 'FLOAT2': [98.3, 99.9, -0.003, 12.4, 11.1],
↪'FLOAT3': [1.88, -2.9, 7.123, -0.3, None], 'FLOAT4': [1.001, 8, -
↪2.1, 15.67, -0.1234]}), oranumber = False)
>>> type(oml_dataframe)
<class 'oml.core.frame.DataFrame'>
```

```
>>> oml_dataframe
     FLOAT1  FLOAT2  FLOAT3   FLOAT4
0       NaN  98.300   1.880   1.0010
1   -12.100  99.900  -2.900   8.0000
2  1234.000  -0.003   7.123  -2.1000
3    40.000  12.400  -0.300  15.6700
4    95.612  11.100     NaN  -0.1234
```

oml.DataFrame round examples: Rounds oml.DataFrame values to the specified decimal place.

```
>>> oml_dataframe.round(1)
   FLOAT1  FLOAT2  FLOAT3  FLOAT4
0     NaN    98.3     1.9     1.0
1   -12.1    99.9    -2.9     8.0
2  1234.0     0.0     7.1    -2.1
3    40.0    12.4    -0.3    15.7
4    95.6    11.1     NaN    -0.1
```

oml.DataFrame t_dot examples: Calculates the matrix cross-product of self with other.

```
>>> oml_dataframe.t_dot()
              FLOAT1          FLOAT2         FLOAT3         FLOAT4
```

(continues on next page)

(continued from previous page)

```
FLOAT1  1.533644e+06    344.801200  8812.872000 -2073.198521
FLOAT2  3.448012e+02  19919.870009  -108.647369  1090.542860
FLOAT3  8.812872e+03   -108.647369    62.771529   -40.977420
FLOAT4 -2.073199e+03   1090.542860   -40.977420   314.976129
```

```
>>> oml_dataframe[['FLOAT1', 'FLOAT2']].t_dot(oml_dataframe[['FLOAT3
↪', 'FLOAT4']])
             FLOAT3       FLOAT4
FLOAT1  8812.872000 -2073.198521
FLOAT2  -108.647369  1090.542860
```

```
>>> oml_dataframe[['FLOAT1', 'FLOAT2']].t_dot(oml_dataframe[['FLOAT3
↪', 'FLOAT4']], skipna=False)
        FLOAT3      FLOAT4
FLOAT1     NaN         NaN
FLOAT2     NaN  1090.54286
```

```
>>> oml_dataframe[['FLOAT1', 'FLOAT2']].t_dot(pull_from_db=False).
↪sort_values(['ROWID', 'COLID'])
   ROWID  COLID         VALUE
0      1      1  1.533644e+06
1      1      2  3.448012e+02
2      2      1  3.448012e+02
3      2      2  1.991987e+04
```

```
>>> oml_dataframe[['FLOAT1', 'FLOAT2']].t_dot(pull_from_db=False,␣
↪skipna=False).sort_values(['ROWID', 'COLID'])
   ROWID  COLID         VALUE
0      1      1           NaN
1      1      2           NaN
2      2      1           NaN
3      2      2  19919.870009
```

sample (*frac=None*, *n=None*, *random_state=None*)

Return a random sample data sets from an oml.DataFrame object.

Parameters

frac
[a float value] Fraction of data sets to return. The value should be between 0 and 1. Cannot be used with n.

n
[an integer value] Number of rows to return. Default = 1 if frac = None. Cannot be used with frac.

random_state
[int or 12345 (default)] The seed to use for random sampling.

Returns

sample_data
[an oml.DataFrame objects] It contains the random sample rows from an oml.DataFrame object. The fraction of returned data sets is specified by the frac parameter.

select_types (*include=None*, *exclude=None*)

Returns the subset of columns include/excluding columns based on their OML type.

Parameters

include, exclude
[list of OML column types] A selection of OML column types to be included/excluded. At least one of these parameters must be supplied.

Returns

subset
[oml.DataFrame]

Raises

ValueError

- If both of `include` and `exclude` are None.
- If `include` and `exclude` have overlapping elements.

skew (*skipna=True*)

Returns the sample skewness of the values for each `Float` column.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result

Returns

skew
[`pandas.Series`]

Examples

See *`DataFrame.describe()`*

sort_values (*by*, *ascending=True*, *na_position='last'*)

Specifies the order in which rows appear in the result set.

Parameters

by
[str or list of str] Column names or list of column names.

ascending
[bool or list of bool, True (default)] If True, sort is in ascending order. Sort is in descending order otherwise. Specify list for multiple sort orders. If this is a list of bools, must match the length of `by`.

na_position
[{'first', 'last'}, 'last' (default)] `first` places NANs and Nones at the beginning; `last` places them at the end.

Returns

sorted_obj
[oml.DataFrame]

Examples
See `DataFrame.pull()`

split (*ratio=(0.7, 0.3)*, *seed=12345*, *strata_cols=None*, *use_hash=True*, *hash_cols=None*, *nvl=None*)
Splits the oml.DataFrame object randomly into multiple data sets.

Parameters

ratio
[a list of float values or (0.7, 0.3) default] All the numbers must be positive and the sum of them are no more than 1. Each number represents the ratio of split data in one set.

seed
[int or 12345 (default)] The seed to use for random splitting.

strata_cols: a list of string values or None (default):
Names of the columns used for stratification. If None, stratification is not performed. Must be None when use_hash is False.

use_hash: boolean, True (default)
If True, use hashing to randomly split the data. If False, use random number to split the data.

hash_cols: a list of string values or None (default):
If a list of string values, use the values from these named columns to hash to split the data. If None, use the values from the 1st 10 columns to hash.

nvl: numeric value, str, or None (default):
If not None, the specified values are used to hash in place of Null.

Returns

split_data
[a list of oml.DataFrame objects] Each of which contains the portion of data by the specified ratio.

Raises

ValueError

- If `hash_cols` refers to a single LOB column.

Examples

```
>>> import oml
>>> import pandas as pd
>>> from sklearn.datasets import load_digits
>>> digits = load_digits()
>>> pd_digits = pd.DataFrame(digits.data,
...                   columns=['IMG'+str(i) for i in
...                            range(digits['data'].shape[1])])
>>> pd_digits = pd.concat([pd_digits,
...                        pd.Series(digits.target, name = 'target
↪')],
```

(continues on next page)

(continued from previous page)

```
...                           axis = 1)
>>> oml_digits = oml.push(pd_digits)
```

Sample using fraction or number of rows.

```
>>> samples = oml_digits.sample(frac = 0.5, random_state = 3271)
>>> len(samples)
897
>>> samples = oml_digits.sample(n = 100, random_state = 3271)
>>> len(samples)
100
```

Split into four equal-sized sets.

```
>>> splits = oml_digits.split(ratio = (.25, .25, .25, .25),
...                               use_hash = False)
>>> [len(split) for split in splits]
[432, 460, 451, 454]
```

Stratify on the target column.

```
>>> splits = oml_digits.split(strata_cols=['target'])
>>> [split.shape for split in splits]
[(1285, 65), (512, 65)]
>>> [split['target'].drop_duplicates().sort_values().pull()
...  for split in splits]
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]]
```

Hash on the target column.

```
>>> splits = oml_digits.split(hash_cols=['target'])
>>> [split.shape for split in splits]
[(899, 65), (898, 65)]
>>> [split['target'].drop_duplicates().sort_values().pull()
...  for split in splits]
[[0, 1, 3, 5, 8], [2, 4, 6, 7, 9]]
```

std(*skipna=True*)

Returns the sample standard deviation of the values of each `Float` or `Integer` or `Boolean` column.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result

Returns

std
[`pandas.Series`]

Examples

See *DataFrame.describe()*

sum(*skipna=True*)

Returns the sum of the values of each `Float` or `Integer` or `Boolean` column.

Parameters

skipna
[boolean, True (default)] Exclude NaN values when computing the result

Returns

sum
[pandas.Series]

Examples
See *DataFrame.describe()*

t_dot(*other=None*, *skipna=True*, *pull_from_db=True*)
Calculates the matrix cross-product of self with other.

Equivalent to transposing self first, then multiplying it with other.

Parameters

other
[oml.DataFrame, optional] If not specified, self is used.

skipna
[bool, True (default)] Treats NaN entries as 0.

pull_from_db
[bool, True (default)] If True, returns a pandas.DataFrame. If False, returns a oml.DataFrame consisting of three columns:

- ROWID: the row number of the resulting matrix
- COLID: the column number of the resulting matrix
- VALUE: the value at the corresponding position of the matrix

Returns

prod
[int or float, pandas.Series, or pandas.DataFrame]

See also:

oml.Float.dot

Examples
See *DataFrame.round()*

tail(*n=5*)
Returns the last n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_tail
[type of caller]

Examples
See *DataFrame.pull()*

Special Method Examples

```
>>> import pandas as pd
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> data_cols = pd.DataFrame(iris.data, columns = ['Sepal_Length',
...              'Sepal_Width', 'Petal_Length', 'Petal_Width'])
>>> target_col = pd.DataFrame(list(map(lambda x: {0: 'setosa',
...               1: 'versicolor', 2:'virginica'}[x], iris.target)),
...               columns = ['Species'])
>>> oml_iris = oml.push(pd.concat([data_cols, target_col], axis=1),
...                    'IRIS')
```

Get number of rows.

```
>>> len(oml_iris)
150
```

Select rows by oml.Boolean.

```
>>> small_sepal = oml_iris[oml_iris['Sepal_Length'] < 5]
```

Select columns by column number.

```
>>> small_sepal[:, [0, 3]]
    Sepal_Length  Petal_Width
0            4.9          0.2
1            4.7          0.2
2            4.6          0.2
...          ...          ...
19           4.6          0.2
20           4.9          1.0
21           4.9          1.7
```

Select columns by slice while selecting a subset of rows.

```
>>> small_sepal[small_sepal['Sepal_Width'] < 3, -2:]
   Petal_Width     Species
0          0.2      setosa
1          0.3      setosa
2          1.0  versicolor
3          1.7   virginica
```

Select column by name.

```
>>> small_sepal['Species']
['setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
↪'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
↪'setosa', 'setosa', 'setosa', 'setosa', 'versicolor', 'virginica']
```

(continues on next page)

(continued from previous page)

Convert column to oml.DataFrame.

```
>>> oml.DataFrame(small_sepal['Species'])
      Species
0       setosa
1       setosa
2       setosa
...        ...
19      setosa
20  versicolor
21   virginica
```

1.3.7 oml.Datetime

class oml.**Datetime**

Datetime series data class.

Represents a single column of Date or difftime data in Oracle Database.

Attributes

`day`	Creates an oml.Float object containing days only from a given oml.Datetime object
`hour`	Creates an oml.Float object containing hours only from a given oml.Datetime object
`minute`	Creates an oml.Float object containing minutes only from a given oml.Datetime object
`month`	Creates an oml.Float object containing months only from a given oml.Datetime object
`second`	Creates an oml.Float object containing seconds only from a given oml.Datetime object
`shape`	The dimensions of the data set.
`tzinfo`	Get time zone information from a given oml.Datetime object when available
`year`	Creates an oml.Float object containing years only from a given oml.Datetime object

Special Methods

`__init__()`	Initialize self.
`__getitem__(key)`	Index self.
`__len__()`	Returns number of rows.
`__eq__(other)`	Equivalent to `self == other`.
`__ne__(other)`	Equivalent to `self != other`.
`__gt__(other)`	Equivalent to `self > other`.
`__ge__(other)`	Equivalent to `self >= other`.
`__lt__(other)`	Equivalent to `self < other`.

Continued on next page

Table 23 – continued from previous page

`__le__`(other)	Equivalent to `self <= other`.

Special Method Examples

Methods

`KFold`([n_splits, seed, use_hash, nvl])	Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.
`append`(other[, all])	Appends the `other` OML data object of the same class to this data object.
`concat`(other[, auto_name])	Combines current OML data object with the `other` data objects column-wise.
`count`()	Returns the number of elements that are not NULL.
`create_view`([view, use_colname])	Creates an Oracle Database view for the data represented by the OML data object.
`describe`()	Generates descriptive statistics that summarize the central tendency, dispersion, and shape of an OML series data distribution.
`drop_duplicates`()	Removes duplicated elements.
`dropna`()	Removes missing values.
`head`([n])	Returns the first n elements.
`isnull`()	Detects the missing value None.
`materialize`([table])	Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.
`max`([skipna])	Returns the maximum value.
`min`([skipna])	Returns the minimum value.
`nunique`([dropna])	Returns the number of unique values.
`pull`()	Pulls data represented by this object from Oracle Database into an in-memory Python object.
`replace`([year, month, day, hour, minute, . . .])	Replace current oml.Datetime object with the specified year, month, day, hour, minute, second, microsecond.
`sort_values`([ascending, na_position])	Sorts the values in the series data object.
`split`([ratio, seed, use_hash, nvl])	Splits the series data object randomly into multiple sets.
`strftime`([format])	Converts an oml.Datetime object to an oml.String object controlled by an explicit format string
`strptime`([format])	Creates an oml.Datetime object from a given oml.String controlled by an explicit format string.
`tail`([n])	Returns the last n elements.

`__init__``()`

Initialize self. See help(type(self)) for accurate signature.

`__getitem__``(`*key*`)`

Index self. Equivalent to `self[key]`.

Parameters

key
[oml.Boolean] Must be from the same data source as self.

Returns

subset
[same type as self] Contains only the rows satisfying the condition in `key`.

`__len__``()`
Returns number of rows. Equivalent to `len(self)`.

Returns

rownum
[int]

`__eq__``(`*other*`)`
Equivalent to `self == other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

equal
[oml.Boolean]

`__ne__``(`*other*`)`
Equivalent to `self != other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

notequal
[oml.Boolean]

`__gt__``(`*other*`)`
Equivalent to `self > other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterthan
[oml.Boolean]

`__ge__`(*other*)
Equivalent to `self >= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterequal
[oml.Boolean]

`__lt__`(*other*)
Equivalent to `self < other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessthan
[oml.Boolean]

__le__(*other*)
Equivalent to `self <= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessequal
[oml.Boolean]

KFold(*n_splits=3*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.

Parameters

n_splits
[int, 3 (default)] The number of folds. Must be greater than or equal to 2.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default):] If not None, the specified values are used to hash in place of Null.

Returns

kfold_data
[a list of pairs of series objects of the same type as caller] Each pair within the list is a fold. The first element of the pair is the train set, and the second element is the test set, which consists of all elements not in the train set.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

append(*other*, *all=True*)
Appends the `other` OML data object of the same class to this data object.

Parameters

other
[An OML data object of the same class]

all
[boolean, True (default)] Keeps the duplicated elements from the two data objects.

Returns

appended
[type of caller] A new data object containing the elements from both objects.

concat (*other*, *auto_name=False*)

Combines current OML data object with the `other` data objects column-wise.

Current object and the `other` data objects must be combinable, that is, they both represent data from the same underlying database table, view, or query.

Parameters

other
[an OML data object, a list of OML data objects, or a dict mapping str to OML data objects.]

- OML data object: an OML series data object or an oml.DataFrame
- list: a sequence of OML series and DataFrame objects to concat.
- dict: a dict mapping str to OML series and DataFrame objects, the column name of concatenated OML series object is replaced with str, column names of concatenated OML DataFrame object is prefixed with str. Need to specify a `collections.OrderedDict` if the concatenation order is expected to follow the key insertion order.

auto_name
[boolean, False (default)] Indicates whether to automatically resolve conflict column names. If True, append duplicated column names with suffix `[column_index]`.

Returns

concat_table
[oml.DataFrame] An oml.DataFrame data object with its original columns followed by the columns of `other`.

Raises

ValueError

- If `other` is not a single nor a list/dict of OML objects, or if `other` is empty.
- If objects in `other` are not from same data source.
- If `auto_name` is False and duplicated column names are detected.

Notes

After concatenation is done, if there is any empty column names in the resulting oml.DataFrame, they will be renamed with `COL[column_index]`.

count()
Returns the number of elements that are not NULL.

Returns

nobs
[int]

create_view(*view=None*, *use_colname=False*)
Creates an Oracle Database view for the data represented by the OML data object.

Parameters

view
[str or None (default)] The name of a database view. If `view` is None, the created view is managed by OML and the view is automatically dropped when no longer needed. If a `view` is specified, then it is up to the user to drop the view.

use_colname
[bool, False (default)] Indicates whether to create the view with the same column names as the DataFrame object. Ignored if `view` is specified.

Returns

new_view
[oml.DataFrame]

Raises

TypeError

- If the object represents data from a temporary table or view, and the view to create is meant to persist past the current session.

describe()
Generates descriptive statistics that summarize the central tendency, dispersion, and shape of an OML series data distribution.

Returns

summary
[`pandas.Series`] Includes `count` (number of non-null entries), `unique` (number of unique entries), `top` (most common value), `freq`, (frequency of the most common value).

drop_duplicates()
Removes duplicated elements.

Returns

deduplicated
[type of caller]

dropna()
Removes missing values.

Missing values include None and/or nan if applicable.

Returns

dropped
[type of caller]

head(*n=5*)
Returns the first n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_head
[type of caller]

isnull()
Detects the missing value None.

Returns

isnull
[oml.Boolean] Indicates missing value None for each element.

materialize(*table=None*)
Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.

Parameters

table
[str or None (default)] The name of a table. If `table` is None, an OML-managed table is created, that is, OML drops the table when it is no longer used by any OML object or when you invoke `oml.disconnect(cleanup=True)` to terminate the database connection. If a table is specified, then it's up to the user to drop the named table.

Returns

new_table
[same type as self]

max(*skipna=True*)
Returns the maximum value.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

max
[Python type corresponding to the column or numpy.nan]

min(*skipna=True*)
Returns the minimum value.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

min
[Python type corresponding to the column or numpy.nan]

nunique(*dropna=True*)
Returns the number of unique values.

Parameters

dropna
[bool, True (default)] If True, NULL values are not included in the count.

Returns

nunique
[int]

pull()
Pulls data represented by this object from Oracle Database into an in-memory Python object.

Returns

pulled_obj
[list of datetime and None]

replace(*year=None*, *month=None*, *day=None*, *hour=None*, *minute=None*, *second=None*, *microsecond=None*, *tzinfo=None*)
Replace current oml.Datetime object with the specified year, month, day, hour, minute, second, microsecond.

Parameters

year: int or oml.Integer
the new year to be used.

month: int or oml.Integer
the new month

day: int or oml.Integer
the new day to be used.

hour: int or oml.Integer
the new hour to be used.

minute: int or oml.Integer
the new minute to be used

second: int or oml.Integer
the new second to be used.

microsecond: int or oml.Integer
the new microsecond to be used.

tzinfo: python datetime.tzinfo object or oml.Timezone
the new tzinfo to be used.

Returns

res: oml.Datetime
Returns an oml.Datetime object with the same value, except for those parameters given new values by whichever keyword arguments are specified.

Examples

```
>>> import oml
>>> from oml import core
>>> import pandas as pd
>>> import datetime
>>> from datetime import datetime
>>> from datetime import timezone
>>> from datetime import timedelta
```

oml.Datetime replace

```
>>> d1=datetime(2005, 7, 14, 1, 1)
>>> d2=[d1]
>>> d3=core.push(d2)
>>> d4 = datetime.fromisoformat('2011-11-04 00:05:23 +05:00')
>>> tzinfo = d4.tzinfo
>>> d3.replace(day = 15)
[datetime.datetime(2005, 7, 15, 1, 1)]
```

```
>>> d3.replace(month = 10)
[datetime.datetime(2005, 10, 14, 1, 1)]
```

```
>>> d3.replace(year = 2001)
[datetime.datetime(2001, 7, 14, 1, 1)]
```

```
>>> d3.replace(hour = 12)
[datetime.datetime(2005, 7, 14, 12, 1)]
```

```
>>> d3.replace(minute = 30)
[datetime.datetime(2005, 7, 14, 1, 30)]
```

```
>>> d3.replace(second = 50)
[datetime.datetime(2005, 7, 14, 1, 1, 50)]
```

```
>>> d3.replace(tzinfo = tzinfo)
[datetime.datetime(2005, 7, 14, 1, 1, tzinfo=datetime.
↪timezone(datetime.timedelta(seconds=18000)))]
```

sort_values (*ascending=True*, *na_position='last'*)
Sorts the values in the series data object.

Parameters

ascending
[bool, True (default)] If True, sorts in ascending order. Otherwise, sorts in descending order.

na_position
[{'first', 'last'}, 'last' (default)] `first` places NANs and Nones at the beginning; `last` places them at the end.

Returns

sorted_obj
[type of caller]

split (*ratio=(0.7, 0.3)*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into multiple sets.

Parameters

ratio
[a list of float values or (0.7, 0.3) (default)] All the numbers must be positive and the sum of them are no more than 1. Each number represents the ratio of split data in one set.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default)] If not None, the specified values are used to hash in place of Null.

Returns

split_data
[a list of series objects of the same type as caller] Each of which contains the portion of data by the specified ratio.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

strftime (*format=None*)
Converts an oml.Datetime object to an oml.String object controlled by an explicit format string

Parameters:
format: a string to control the format

Returns

res: oml.String
Returns an oml.String representing oml.Datetime, controlled by an explicit format string.

strptime (*format=None*)

Creates an oml.Datetime object from a given oml.String controlled by an explicit format string.

Parameters

date_string: an oml.String object that be converted to oml.Datetime object

format: a string to control the format

Returns

res: oml.Datetime
A oml.Datetime object corresponding to date_string, parsed according to format.

Examples

```
>>> import oml
>>> from oml import core
>>> import pandas as pd
>>> import datetime
>>> from datetime import datetime
>>> from datetime import timezone
>>> from datetime import timedelta
```

oml.Datetime.strftime()

```
>>> d1 = datetime(2005, 7, 14, 20, 1, 1)
>>> d2=[d1, d1]
>>> d3=core.push(d2)
>>> d3.strftime()
['14-JUL-05 08.01.01.000000 PM', '14-JUL-05 08.01.01.000000 PM']
```

```
>>> d4=d3.strftime("DD-Mon-RR HH24:MI:SS")
>>> d4
['14-Jul-05 20:01:01', '14-Jul-05 20:01:01']
```

oml.Datetime.strptime()

```
>>> d1 = "14-Jul-05 20:01:01"
>>> d2=[d1, d1]
>>> d3=core.push(d2)
>>> d4=core.Datetime.strptime(d3, "DD-Mon-RR HH24:MI:SS")
>>> d4
[datetime.datetime(2005, 7, 14, 20, 1, 1), datetime.datetime(2005,␣
↪7, 14, 20, 1, 1)]
```

tail (*n=5*)

Returns the last n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_tail
[type of caller]

Special Method Examples

/ / $Header: Datetime.special.txt 07-oct-2022.02:00:55 ffeli Exp $ / / Datetime.special.txt / / Copyright (c) 2022, Oracle and/or its affiliates. / / NAME / Datetime.special.txt - <one-line expansion of the name> / / DESCRIPTION / <short description of component this file declares/defines> / / NOTES / <other useful comments, qualifications, etc.> / / MODIFIED (MM/DD/YY) / ffeli 10/07/22 - Creation /

```
>>> import oml
>>> import pandas as pd
>>> import datetime
>>> from datetime import datetime
```

```
>>> d1=datetime(2005, 7, 14, 12, 30)
>>> d2=pd.DataFrame({'newdate': [d1, d1]})
>>> d3=oml.push(d2)
>>> d3
              newdate
0 2005-07-14 12:30:00
1 2005-07-14 12:30:00
```

```
>>> d3=oml.push(d2, dbtypes = 'TIMESTAMP')
>>> d3
              newdate
0 2005-07-14 12:30:00
1 2005-07-14 12:30:00
```

```
>>> d2=pd.DataFrame([[d1, d1], [d1, d1]])
>>> d3=oml.push(d2)
>>> d3
                    0                    1
0 2005-07-14 12:30:00 2005-07-14 12:30:00
1 2005-07-14 12:30:00 2005-07-14 12:30:00
```

```
>>> d3=oml.push(d2, dbtypes = ['TIMESTAMP', 'TIMESTAMP'])
>>> d3
                    0                    1
0 2005-07-14 12:30:00 2005-07-14 12:30:00
1 2005-07-14 12:30:00 2005-07-14 12:30:00
```

```
>>> d1=datetime(2005, 7, 14, 12, 30)
>>> d2=pd.DataFrame({'newdatetime': [d1, d1]})
>>> d3=oml.create(d2, dbtypes = 'TIMESTAMP', table = 'newdatetime')
>>> d3
          newdatetime
0 2005-07-14 12:30:00
1 2005-07-14 12:30:00
```

1.3.8 oml.Timedelta

class oml.**Timedelta**

Timedelta series data class.

Represents a single column of difference between two dates or times in Oracle Database.

Attributes

`day`	Creates an oml.Float object containing days only from a given oml.Timedelta object
`second`	Creates an oml.Float object containing seconds only from a given oml.Timedelta object
`shape`	The dimensions of the data set.

Special Methods

`__init__`()	Initialize self.
`__getitem__`(key)	Index self.
`__len__`()	Returns number of rows.
`__eq__`(other)	Equivalent to `self == other`.
`__ne__`(other)	Equivalent to `self != other`.
`__gt__`(other)	Equivalent to `self > other`.
`__ge__`(other)	Equivalent to `self >= other`.
`__lt__`(other)	Equivalent to `self < other`.
`__le__`(other)	Equivalent to `self <= other`.

Special Method Examples

Methods

`KFold`([n_splits, seed, use_hash, nvl])	Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.
`append`(other[, all])	Appends the `other` OML data object of the same class to this data object.
`concat`(other[, auto_name])	Combines current OML data object with the `other` data objects column-wise.
`count`()	Returns the number of elements that are not NULL.
`create_view`([view, use_colname])	Creates an Oracle Database view for the data represented by the OML data object.
`describe`()	Generates descriptive statistics that summarize the central tendency, dispersion, and shape of an OML series data distribution.
`drop_duplicates`()	Removes duplicated elements.
`dropna`()	Removes missing values.
`head`([n])	Returns the first n elements.
`isnull`()	Detects the missing value None.

Continued on next page

Table 27 – continued from previous page

`materialize`([table])	Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.
`max`([skipna])	Returns the maximum value.
`min`([skipna])	Returns the minimum value.
`nunique`([dropna])	Returns the number of unique values.
`pull`()	Pulls data represented by this object from Oracle Database into an in-memory Python object.
`sort_values`([ascending, na_position])	Sorts the values in the series data object.
`split`([ratio, seed, use_hash, nvl])	Splits the series data object randomly into multiple sets.
`tail`([n])	Returns the last n elements.
`total_seconds`()	Return total duration of each element expressed in seconds.

`__init__`()

Initialize self. See help(type(self)) for accurate signature.

`__getitem__`(*key*)

Index self. Equivalent to `self[key]`.

Parameters

key
[oml.Boolean] Must be from the same data source as self.

Returns

subset
[same type as self] Contains only the rows satisfying the condition in `key`.

`__len__`()

Returns number of rows. Equivalent to `len(self)`.

Returns

rownum
[int]

`__eq__`(*other*)

Equivalent to `self == other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

equal
[oml.Boolean]

`__ne__`(*other*)
Equivalent to `self != other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

notequal
[oml.Boolean]

`__gt__`(*other*)
Equivalent to `self > other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterthan
[oml.Boolean]

`__ge__`(*other*)
Equivalent to `self >= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterequal
[oml.Boolean]

`__lt__`(*other*)
Equivalent to `self < other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessthan
[oml.Boolean]

`__le__`(*other*)
Equivalent to `self <= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessequal
[oml.Boolean]

`KFold`(*n_splits=3*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.

Parameters

n_splits
[int, 3 (default)] The number of folds. Must be greater than or equal to 2.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default):] If not None, the specified values are used to hash in place of Null.

Returns

kfold_data
[a list of pairs of series objects of the same type as caller] Each pair within the list is a fold. The first element of the pair is the train set, and the second element is the test set, which consists of all elements not in the train set.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

append(*other*, *all=True*)
Appends the `other` OML data object of the same class to this data object.

Parameters

other
[An OML data object of the same class]

all
[boolean, True (default)] Keeps the duplicated elements from the two data objects.

Returns

appended
[type of caller] A new data object containing the elements from both objects.

concat(*other*, *auto_name=False*)
Combines current OML data object with the `other` data objects column-wise.

Current object and the `other` data objects must be combinable, that is, they both represent data from the same underlying database table, view, or query.

Parameters

other
[an OML data object, a list of OML data objects, or a dict mapping str to OML data objects.]

- OML data object: an OML series data object or an oml.DataFrame
- list: a sequence of OML series and DataFrame objects to concat.
- dict: a dict mapping str to OML series and DataFrame objects, the column name of concatenated OML series object is replaced with str, column names of concatenated OML DataFrame object is prefixed with str. Need to specify a `collections.OrderedDict` if the concatenation order is expected to follow the key insertion order.

auto_name
[boolean, False (default)] Indicates whether to automatically resolve conflict column names. If True, append duplicated column names with suffix `[column_index]`.

Returns

concat_table
[oml.DataFrame] An oml.DataFrame data object with its original columns followed by the columns of `other`.

Raises

ValueError

- If `other` is not a single nor a list/dict of OML objects, or if `other` is empty.
- If objects in `other` are not from same data source.
- If `auto_name` is False and duplicated column names are detected.

Notes

After concatenation is done, if there is any empty column names in the resulting oml.DataFrame, they will be renamed with `COL[column_index]`.

count()
Returns the number of elements that are not NULL.

Returns

nobs
[int]

create_view(*view=None*, *use_colname=False*)
Creates an Oracle Database view for the data represented by the OML data object.

Parameters

view
[str or None (default)] The name of a database view. If `view` is None, the created view is managed by OML and the view is automatically dropped when no longer needed. If a `view` is specified, then it is up to the user to drop the view.

use_colname
[bool, False (default)] Indicates whether to create the view with the same column names as the DataFrame object. Ignored if `view` is specified.

Returns

new_view
[oml.DataFrame]

Raises

TypeError

- If the object represents data from a temporary table or view, and the view to create is meant to persist past the current session.

describe()

Generates descriptive statistics that summarize the central tendency, dispersion, and shape of an OML series data distribution.

Returns

summary
[`pandas.Series`] Includes `count` (number of non-null entries), `unique` (number of unique entries), `top` (most common value), `freq`, (frequency of the most common value).

drop_duplicates()

Removes duplicated elements.

Returns

deduplicated
[type of caller]

dropna()

Removes missing values.

Missing values include None and/or nan if applicable.

Returns

dropped
[type of caller]

head(*n=5*)

Returns the first n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_head
[type of caller]

isnull()

Detects the missing value None.

Returns

isnull
[oml.Boolean] Indicates missing value None for each element.

materialize(*table=None*)

Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.

Parameters

table
[str or None (default)] The name of a table. If `table` is None, an OML-managed table is created, that is, OML drops the table when it is no longer used by any OML object or when you invoke `oml.disconnect(cleanup=True)` to terminate the database connection. If a table is specified, then it's up to the user to drop the named table.

Returns

new_table
[same type as self]

max(*skipna=True*)
Returns the maximum value.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

max
[Python type corresponding to the column or numpy.nan]

min(*skipna=True*)
Returns the minimum value.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

min
[Python type corresponding to the column or numpy.nan]

nunique(*dropna=True*)
Returns the number of unique values.

Parameters

dropna
[bool, True (default)] If True, NULL values are not included in the count.

Returns

nunique
[int]

pull()
Pulls data represented by this object from Oracle Database into an in-memory Python object.

Returns

pulled_obj
[list of timedelta and None]

sort_values(*ascending=True*, *na_position='last'*)
Sorts the values in the series data object.

Parameters

ascending
[bool, True (default)] If True, sorts in ascending order. Otherwise, sorts in descending order.

na_position
[{'first', 'last'}, 'last' (default)] `first` places NANs and Nones at the beginning; `last` places them at the end.

Returns

sorted_obj
[type of caller]

split(*ratio=(0.7, 0.3)*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into multiple sets.

Parameters

ratio
[a list of float values or (0.7, 0.3) (default)] All the numbers must be positive and the sum of them are no more than 1. Each number represents the ratio of split data in one set.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default)] If not None, the specified values are used to hash in place of Null.

Returns

split_data
[a list of series objects of the same type as caller] Each of which contains the portion of data by the specified ratio.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

tail(*n=5*)
Returns the last n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_tail
[type of caller]

total_seconds()
Return total duration of each element expressed in seconds.

Returns

res
[oml.Float] A list of total seconds in the oml.timedelta series.

Special Method Examples

/ / $Header: Timedelta.special.txt 07-oct-2022.02:00:35 ffeli Exp $ / / Timedelta.special.txt / / Copyright (c) 2022, Oracle and/or its affiliates. / / NAME / Timedelta.special.txt - <one-line expansion of the name> / / DESCRIPTION / <short description of component this file declares/defines> / / NOTES / <other useful comments, qualifications, etc.> / / MODIFIED (MM/DD/YY) / ffeli 10/07/22 - Creation /

```
>>> import oml
>>> import pandas as pd
>>> import datetime
>>> from datetime import datetime
>>> from datetime import timedelta
```

```
>>> d1=timedelta(seconds=1)
>>> d2=pd.DataFrame([d1, d1])
>>> d3=oml.push(d2)
>>> d3
                0
0 0 days 00:00:01
1 0 days 00:00:01
```

```
>>> d3=oml.push(d2, dbtypes = 'INTERVAL DAY TO SECOND')
>>> d3
                0
0 0 days 00:00:01
1 0 days 00:00:01
```

```
>>> d2=pd.DataFrame([[d1, d1], [d1, d1]])
>>> d3=oml.push(d2)
>>> d3
                0                1
0 0 days 00:00:01 0 days 00:00:01
1 0 days 00:00:01 0 days 00:00:01
```

```
>>> d3=oml.push(d2, dbtypes = ['INTERVAL DAY TO SECOND', 'INTERVAL DAY TO SECOND
↪'])
>>> d3
                 0                 1
0 0 days 00:00:01 0 days 00:00:01
1 0 days 00:00:01 0 days 00:00:01
```

```
>>> d1=timedelta(seconds=1)
>>> d2=pd.DataFrame({'newtimedelta': [d1, d1]})
>>> d3=oml.create(d2, dbtypes = 'INTERVAL DAY TO SECOND', table = 'newtimedelta')
>>> d3
     newtimedelta
0 0 days 00:00:01
1 0 days 00:00:01
```

1.3.9 oml.Timezone

class oml.**Timezone**

Timezone series data class.

Represents a single column of time zone data in Oracle Database.

Attributes

shape	The dimensions of the data set.

Special Methods

__init__()	Initialize self.
__getitem__(key)	Index self.
__len__()	Returns number of rows.
__eq__(other)	Equivalent to `self == other`.
__ne__(other)	Equivalent to `self != other`.
__gt__(other)	Equivalent to `self > other`.
__ge__(other)	Equivalent to `self >= other`.
__lt__(other)	Equivalent to `self < other`.
__le__(other)	Equivalent to `self <= other`.

Special Method Examples

Methods

KFold([n_splits, seed, use_hash, nvl])	Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.
append(other[, all])	Appends the `other` OML data object of the same class to this data object.
concat(other[, auto_name])	Combines current OML data object with the `other` data objects column-wise.

Continued on next page

Table 30 – continued from previous page

count()	Returns the number of elements that are not NULL.
create_view([view, use_colname])	Creates an Oracle Database view for the data represented by the OML data object.
describe()	Generates descriptive statistics that summarize the central tendency, dispersion, and shape of an OML series data distribution.
drop_duplicates()	Removes duplicated elements.
dropna()	Removes missing values.
head([n])	Returns the first n elements.
isnull()	Detects the missing value None.
materialize([table])	Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.
max([skipna])	Returns the maximum value.
min([skipna])	Returns the minimum value.
nunique([dropna])	Returns the number of unique values.
pull()	Pulls data represented by this object from Oracle Database into an in-memory Python object.
sort_values([ascending, na_position])	Sorts the values in the series data object.
split([ratio, seed, use_hash, nvl])	Splits the series data object randomly into multiple sets.
tail([n])	Returns the last n elements.

__init__()

Initialize self. See help(type(self)) for accurate signature.

__getitem__(*key*)

Index self. Equivalent to `self[key]`.

Parameters

key
[oml.Boolean] Must be from the same data source as self.

Returns

subset
[same type as self] Contains only the rows satisfying the condition in `key`.

__len__()

Returns number of rows. Equivalent to `len(self)`.

Returns

rownum
[int]

__eq__(*other*)

Equivalent to `self == other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

equal
[oml.Boolean]

`__ne__`(*other*)
Equivalent to `self != other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

notequal
[oml.Boolean]

`__gt__`(*other*)
Equivalent to `self > other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterthan
[oml.Boolean]

`__ge__`(*other*)
Equivalent to `self >= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

greaterequal
[oml.Boolean]

`__lt__`(*other*)
Equivalent to `self < other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessthan
[oml.Boolean]

`__le__`(*other*)
Equivalent to `self <= other`.

Parameters

other
[OML series data object of compatible type or corresponding built-in python scalar]

- scalar : every element in `self` will be compared to the scalar.
- a OML series : must come from the same data source. Every element in `self` will be compared to the corresponding element in `other`. oml.Float and oml.Boolean series can be compared to each other. oml.String and oml.Bytes series can be compared to each other.

Returns

lessequal
[oml.Boolean]

`KFold`(*n_splits=3*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into k consecutive folds for use with k-fold cross validation.

Parameters

n_splits
[int, 3 (default)] The number of folds. Must be greater than or equal to 2.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default):] If not None, the specified values are used to hash in place of Null.

Returns

kfold_data
[a list of pairs of series objects of the same type as caller] Each pair within the list is a fold. The first element of the pair is the train set, and the second element is the test set, which consists of all elements not in the train set.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

append(*other*, *all=True*)
Appends the `other` OML data object of the same class to this data object.

Parameters

other
[An OML data object of the same class]

all
[boolean, True (default)] Keeps the duplicated elements from the two data objects.

Returns

appended
[type of caller] A new data object containing the elements from both objects.

concat(*other*, *auto_name=False*)
Combines current OML data object with the `other` data objects column-wise.

Current object and the `other` data objects must be combinable, that is, they both represent data from the same underlying database table, view, or query.

Parameters

other
[an OML data object, a list of OML data objects, or a dict mapping str to OML data objects.]

- OML data object: an OML series data object or an oml.DataFrame
- list: a sequence of OML series and DataFrame objects to concat.
- dict: a dict mapping str to OML series and DataFrame objects, the column name of concatenated OML series object is replaced with str, column names of concatenated OML DataFrame object is prefixed with str. Need to specify a `collections.OrderedDict` if the concatenation order is expected to follow the key insertion order.

auto_name
[boolean, False (default)] Indicates whether to automatically resolve conflict column names. If True, append duplicated column names with suffix `[column_index]`.

Returns

concat_table
[oml.DataFrame] An oml.DataFrame data object with its original columns followed by the columns of `other`.

Raises

ValueError

- If `other` is not a single nor a list/dict of OML objects, or if `other` is empty.
- If objects in `other` are not from same data source.
- If `auto_name` is False and duplicated column names are detected.

Notes

After concatenation is done, if there is any empty column names in the resulting oml.DataFrame, they will be renamed with `COL[column_index]`.

count()
Returns the number of elements that are not NULL.

Returns

nobs
[int]

create_view(*view=None*, *use_colname=False*)
Creates an Oracle Database view for the data represented by the OML data object.

Parameters

view
[str or None (default)] The name of a database view. If `view` is None, the created view is managed by OML and the view is automatically dropped when no longer needed. If a `view` is specified, then it is up to the user to drop the view.

use_colname
[bool, False (default)] Indicates whether to create the view with the same column names as the DataFrame object. Ignored if `view` is specified.

Returns

new_view
[oml.DataFrame]

Raises

TypeError

- If the object represents data from a temporary table or view, and the view to create is meant to persist past the current session.

describe()
Generates descriptive statistics that summarize the central tendency, dispersion, and shape of an OML series data distribution.

Returns

summary
[`pandas.Series`] Includes `count` (number of non-null entries), `unique` (number of unique entries), `top` (most common value), `freq`, (frequency of the most common value).

drop_duplicates()
Removes duplicated elements.

Returns

deduplicated
[type of caller]

dropna()
Removes missing values.

Missing values include None and/or nan if applicable.

Returns

dropped
[type of caller]

head(*n=5*)
Returns the first n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_head
[type of caller]

isnull()
Detects the missing value None.

Returns

isnull
[oml.Boolean] Indicates missing value None for each element.

materialize(*table=None*)
Pushes the contents represented by an OML proxy object (a view, a table and so on) into a table in Oracle Database.

Parameters

table
[str or None (default)] The name of a table. If `table` is None, an OML-managed table is created, that is, OML drops the table when it is no longer used by any OML object or when you invoke `oml.disconnect(cleanup=True)` to terminate the database connection. If a table is specified, then it's up to the user to drop the named table.

Returns

new_table
[same type as self]

max(*skipna=True*)
Returns the maximum value.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

max
[Python type corresponding to the column or numpy.nan]

min(*skipna=True*)
Returns the minimum value.

Parameters

skipna
[boolean, True (default)] Excludes NaN values when computing the result.

Returns

min
[Python type corresponding to the column or numpy.nan]

nunique(*dropna=True*)
Returns the number of unique values.

Parameters

dropna
[bool, True (default)] If True, NULL values are not included in the count.

Returns

nunique
[int]

pull()
Pulls data represented by this object from Oracle Database into an in-memory Python object.

Returns

pulled_obj
[list of timezone and None]

sort_values(*ascending=True*, *na_position='last'*)
Sorts the values in the series data object.

Parameters

ascending
[bool, True (default)] If True, sorts in ascending order. Otherwise, sorts in descending order.

na_position
[{'first', 'last'}, 'last' (default)] `first` places NANs and Nones at the beginning; `last` places them at the end.

Returns

sorted_obj
[type of caller]

split(*ratio=(0.7, 0.3)*, *seed=12345*, *use_hash=True*, *nvl=None*)
Splits the series data object randomly into multiple sets.

Parameters

ratio
[a list of float values or (0.7, 0.3) (default)] All the numbers must be positive and the sum of them are no more than 1. Each number represents the ratio of split data in one set.

seed
[int or 12345 (default)] The seed to use for random splitting.

use_hash
[boolean, True (default)] If True, use hashing to randomly split the data. If False, use a random number to split the data.

nvl
[numeric value, str, or None (default)] If not None, the specified values are used to hash in place of Null.

Returns

split_data
[a list of series objects of the same type as caller] Each of which contains the portion of data by the specified ratio.

Raises

TypeError

- If `use_hash` is True, and the underlying database column type of `self` is a LOB.

tail(*n=5*)

Returns the last n elements.

Parameters

n
[int, 5 (default)] The number of elements to return.

Returns

obj_tail
[type of caller]

Special Method Examples

/ / $Header: Timezone.special.txt 07-oct-2022.02:01:11 ffeli Exp $ / / Timezone.special.txt / / Copyright (c) 2022, Oracle and/or its affiliates. / / NAME / Timezone.special.txt - <one-line expansion of the name> / / DESCRIPTION / <short description of component this file declares/defines> / / NOTES / <other useful comments, qualifications, etc.> / / MODIFIED (MM/DD/YY) / ffeli 10/07/22 - Creation /

```
>>> import oml
>>> import pandas as pd
>>> import datetime
>>> from datetime import datetime
>>> from datetime import timezone
```

```
>>> d1 = datetime.fromisoformat('2011-11-04 00:05:23+04:00')
>>> d11 = 2
>>> d2=pd.DataFrame({'timezone': [d1.tzinfo, d1.tzinfo], 'integer': [d11, d11]})
>>> d3=oml.push(d2)
>>> d3
    timezone  integer
0  UTC+04:00        2
1  UTC+04:00        2
```

```
>>> d3['timezone']
[datetime.timezone(datetime.timedelta(seconds=14400)), datetime.timezone(datetime.
→timedelta(seconds=14400))]
```

```
>>> d4=oml.push(d2, dbtypes = ['TIMEZONE', 'NUMBER'])
>>> d4
    timezone  integer
0  UTC+04:00        2
1  UTC+04:00        2
```

```
>>> d4['timezone']
[datetime.timezone(datetime.timedelta(seconds=14400)), datetime.timezone(datetime.
→timedelta(seconds=14400))]
```

(continues on next page)

(continued from previous page)

1.4 Access Control

OML functions described in this section control the read privilege for Python scripts or datastore.

`oml.grant`(name[, typ, user])	Grants read privilege for a Python script or datastore.
`oml.revoke`(name[, typ, user])	Revokes read privilege for a Python script or datastore.

`oml.grant`(*name*, *typ='datastore'*, *user=None*)

Grants read privilege for a Python script or datastore. Requires the user to have the *PYQADMIN* Oracle Database role.

Parameters

name
[str] The name of Python script in the Python script repository or the name of a datastore. The current user must be the owner of the Python script or datastore.

typ
['datastore' (default) or 'pyqscript'] A str specifying either 'datastore' or 'pyqscript' to grant the read privilege. 'pyqscript' requires Embedded Python.

user
[str or None (default)] The user to grant read privilege of the named Python script or datastore to. Treated as case-sensitive if wrapped in double quotes. Treated as case-insensitive otherwise. If None, grant read privilege to public.

Examples

See *oml.grant examples* in Section Datastore.

`oml.revoke`(*name*, *typ='datastore'*, *user=None*)

Revokes read privilege for a Python script or datastore. Requires the user to have the *PYQADMIN* Oracle Database role.

Parameters

name
[str] The name of Python script in the Python script repository or the name of a datastore. The current user must be the owner of the Python script or datastore.

typ
['datastore' (default) or 'pyqscript'] A str specifying either 'datastore' or 'pyqscript' to revoke the read privilege. 'pyqscript' requires Embedded Python.

user
[str or None (default)] The user to revoke read privilege of the named Python script or

datastore from. Treated as case-sensitive if wrapped in double quotes. Treated as case-insensitive otherwise. If None, revoke read privilege from public.

Examples

See *oml.revoke examples* in Section Datastore.

1.5 Graphic Rendering

Graphic display functionality to render boxplot and histogram.

Matplotlib is used to render the output. Methods in `matplotlib.pyplot` can be used to customize the created images, and `matplotlib.pyplot.show()` will show the images. By default, rendered graphics will have the same properties as those stored in `matplotlib.rcParams`.

`oml.boxplot`(x[, notch, sym, vert, whis, ...])	Makes a box and whisker plot.
`oml.hist`(x[, bins, range, density, weights, ...])	Plots a histogram.

`oml.`**`boxplot`**(*x*, *notch=None*, *sym=None*, *vert=None*, *whis=None*, *positions=None*, *widths=None*, *patch_artist=None*, *usermedians=None*, *conf_intervals=None*, *meanline=None*, *showmeans=None*, *showcaps=None*, *showbox=None*, *showfliers=None*, *boxprops=None*, *labels=None*, *flierprops=None*, *medianprops=None*, *meanprops=None*, *capprops=None*, *whiskerprops=None*, *manage_ticks=True*, *autorange=False*, *zorder=None*)

Makes a box and whisker plot.

For every column of `x` or for every column object in `x`, makes a box and whisker plot.

Parameters

x [oml.DataFrame or oml.Integer or oml.Float or list of oml.Integer or oml.Float] The data to plot.

notch
[bool, False (default), optional] If True, produces a notched box plot. Otherwise, a rectangular boxplot is produced. By default, the confidence intervals are approximated as `median +/-1.57 IQR/sqrt(n)` where `n` is the number of not-null/NA values in the column.

conf_intervals
[array-like, optional] Array or sequence whose first dimension is equal to the number of columns in `x` and whose second dimension is 2.

labels
[sequence, optional] Length must be equal to the number of columns in `x`. When an element of `labels` is not None, the default label of the column, which is the name of the column, is overridden.

Returns

ax [matplotlib.axes.Axes] The matplotlib.axes.Axes instance of the boxplot figure.

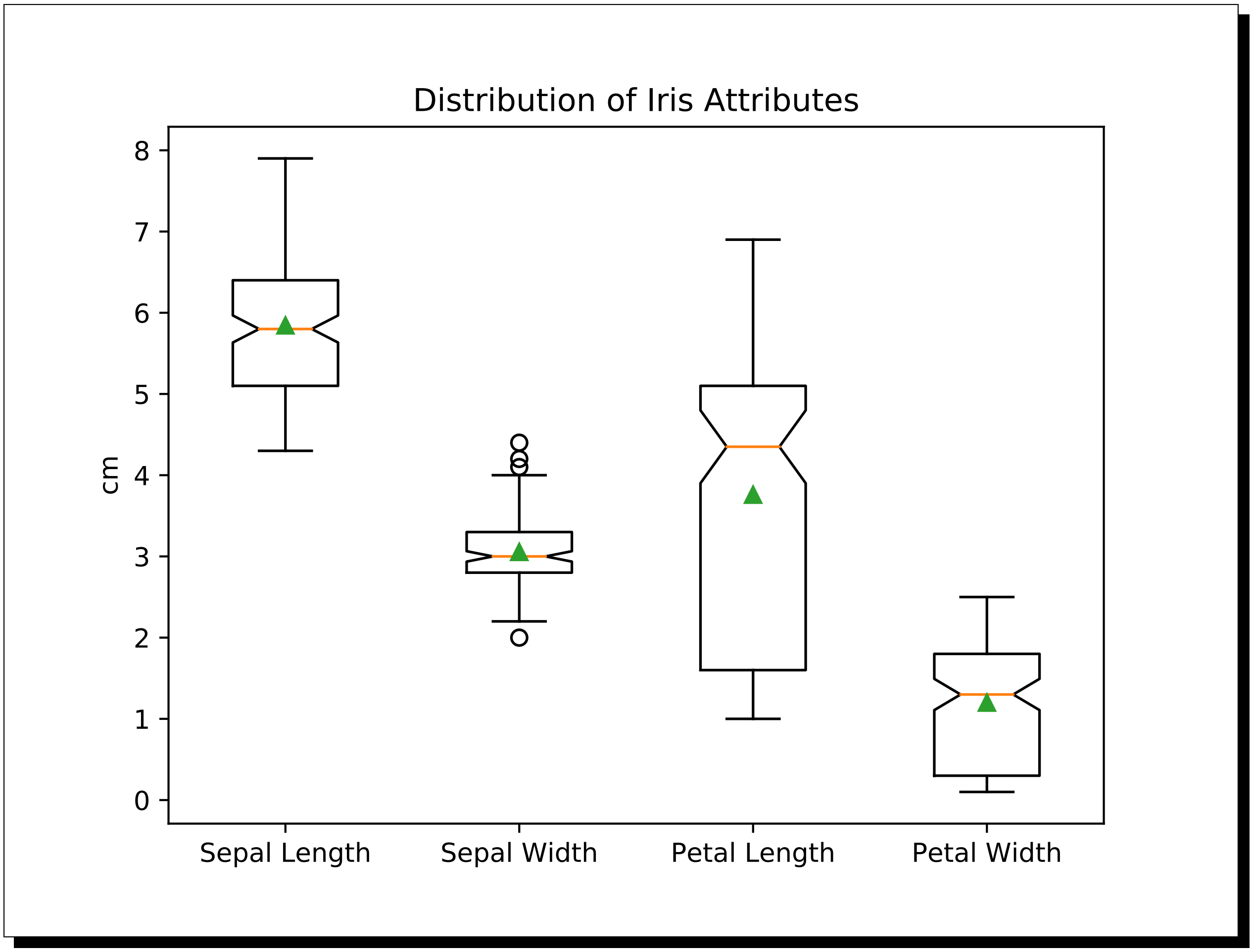
result
[dict] A dict mapping each component of the boxplot to the corresponding list of matplotlib.lines.Lines2D instances created.

Notes

For information on the other parameters, see documentation for matplotlib.pyplot.boxplot().

Examples

```
import oml
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
pd_iris = pd.DataFrame(iris.data, columns = iris.feature_names)
pd_iris = pd.concat([pd_iris,
                    pd.Series(iris.target, name = 'target')],
                    axis = 1)
oml_iris = oml.push(pd_iris)
oml.graphics.boxplot(oml_iris[:, :4], notch=True, showmeans = True,
                     labels=['Sepal Length', 'Sepal Width',
                     'Petal Length', 'Petal Width'])
plt.title('Distribution of Iris Attributes')
plt.ylabel('cm')
```



`oml.hist` (*x*, *bins=None*, *range=None*, *density=False*, *weights=None*, *cumulative=False*, *bottom=None*, *align='mid'*, *orientation='vertical'*, *rwidth=None*, *log=False*, *color=None*, *label=None*, ***kwargs*)

Plots a histogram.

Computes and draws a histogram for every data set column contained in `x`.

Parameters

x [oml.Integer or oml.Float]

bins
[int, strictly monotonic increasing sequence, 'auto', 'doane', 'fd', 'rice', 'scott', 'sqrt', or 'sturges', optional]

- If an integer, denotes the number of equal width bins to generate.
- If a sequence, denotes bin edges and overrides the values of `range`.
- If a string, denotes the estimator to use calculate the optimal number of bins. 'auto' is the maximum of the 'fd' and 'sturges' estimators.
- Default is taken from the matplotlib rcParam `hist.bins`.

weights
[oml.Integer or oml.Float] Must come from the same table as `x`.

cumulative
[int, float, or boolean, False (Default)] If greater than zero, then a histogram is computed where each bin gives the counts in that bin plus all bins for smaller values. If `density` is also True, then the histogram is normalized so the last bin equals 1. If less than zero, the direction of accumulation is reversed. In this case, if `density` is True, then the histogram is normalized so that the first bin equals 1.

rwidth
[int, float, or None (default)] Ratio of the width of the bars to the bin widths. Values less than 0 is treated as 0. Values more than 1 is treated as 1. If None, defaults to 1.

color
[str that indicates a color spec or None (default)] If None, use the standard line color sequence.

label
[str or None (default)] The label that is applied to the first patch of the histogram.

Returns

n [`numpy.ndarray`] The values of the histogram bins.

bins
[`numpy.ndarray`] The edges of the bins. An array of length #bins + 1.

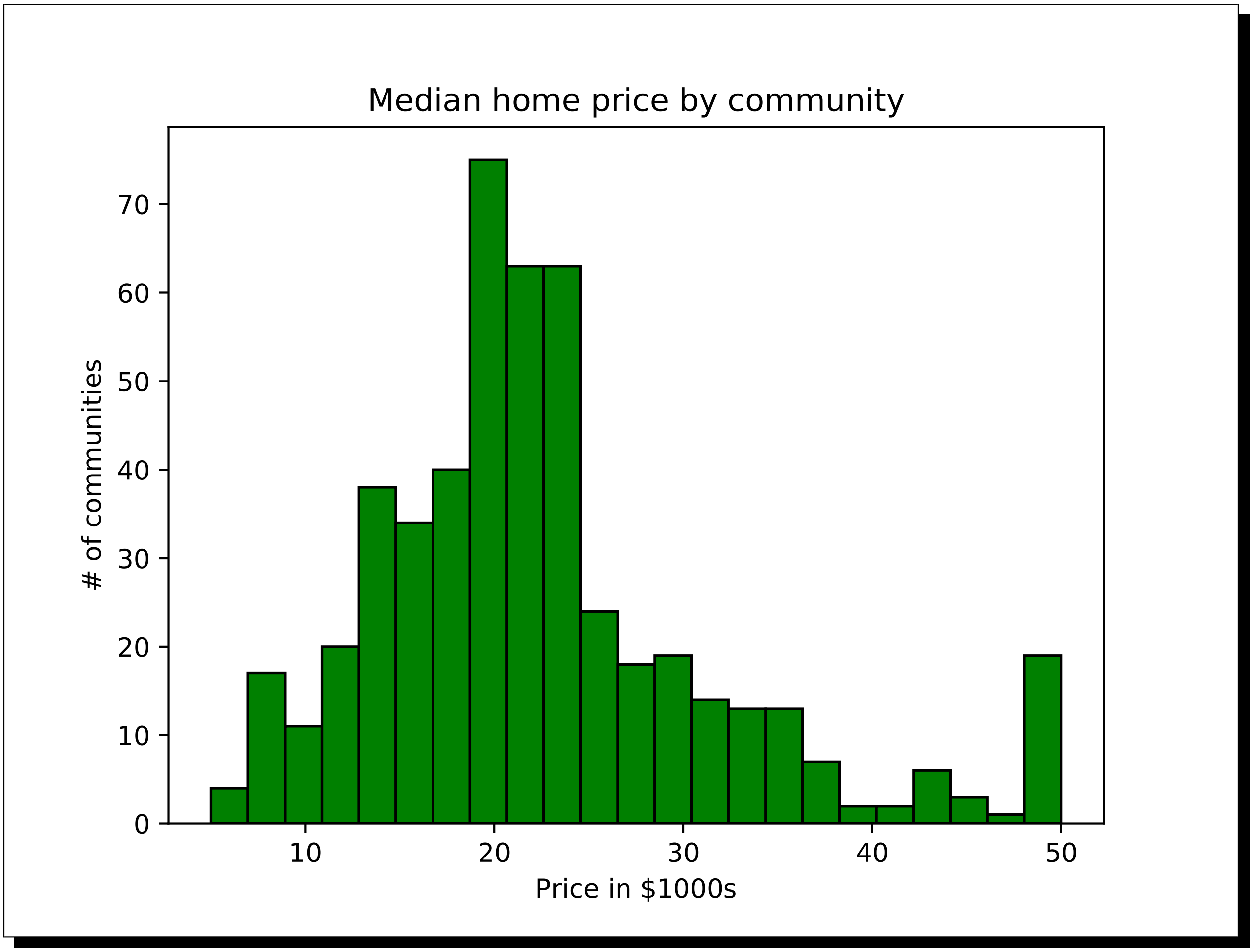
patches
[list of `matplotlib.patches.Rectangle`] Individual patches used to create the histogram.

Notes

For information on the other parameters, see documentation for `matplotlib.pyplot.hist()`.

Examples

```
import oml
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
boston = load_boston()
pd_boston = pd.DataFrame({'target':boston.target})
oml_boston = oml.push(pd_boston)
oml.graphics.hist(oml_boston['target'], 'auto', color='green',
                  linestyle='solid', edgecolor='black')
plt.title('Median home price by community')
plt.ylabel('# of communities')
plt.xlabel('Price in $1000s')
```


Median home price by community
# of communities
0
10
20
30
40
50
60
70
10
20
30
40
50
Price in $1000s

CHAPTER

TWO

ALGORITHMS

Oracle Machine Learning for Python Algorithms

Provides in-database machine learning. The algorithms cover classification, regression, clustering, feature extraction, attribute importance, and anomaly detection.

2.1 Attribute Importance

class oml.**ai**(*model_name=None*, *model_owner=None*, ***params*)

In-database Attribute Importance Model

Computes the relative importance of variables (aka attributes or columns) when predicting a target variable (numeric or categorical column). This function exposes the corresponding Oracle Machine Learning in-database algorithm. Oracle Machine Learning does not support the prediction functions for attribute importance. The results of attribute importance are the attributes of the build data ranked according to their predictive influence. The ranking and the measure of importance can be used for selecting attributes.

Attributes

importance : oml.DataFrame

Relative importance of predictor variables for predicting a response variable. It includes the following components:

- variable: The name of the predictor variable
- importance: The importance of the predictor variable
- rank: The predictor variable rank based on the importance value.

See also:

glm, *svm*, *km*

__init__(*model_name=None*, *model_owner=None*, ***params*)

Initializes an instance of Attribute Importance object.

Parameters

model_name

[string or None (default)] The name of an existing database Attribute Importance model to create an oml.ai object from. The specified database model is not dropped when the oml.ai object is deleted.

model_owner: string or None (default)

The owner name of the existing Attribute Importance model The current database user by default

params
[key-value pairs or dict] Oracle Machine Learning parameter settings. Each list element's name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 4-10 apply generally to the model. Mining Function Settings or Algorithm-specific Settings are not applicable to Attribute Importance model.

export_sermodel(*table=None*, *partition=None*)
Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

fit(*x*, *y*, *model_name=None*, *case_id=None*)
Fits an Attribute Importance Model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object, or string] Target values. If y is a single column OML object, target values specified by y must be combinable with x. If y is a string, y is the name of the column in x that specifies the target values.

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.ai object is deleted. If None, a system-generated model name will be used. The system-generated model is dropped when oml.ai object is deleted unless oml.ai object is saved into a datastore.

case_id
[string or None (default)] The column name used as case id for building the model.

get_params(*params=None*, *deep=False*)
Fetches parameters of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If `params` is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

set_params(***params*)
Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

References

- Oracle R Enterprise
- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

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

Create table containing dataset iris from sklearn.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width', 'Petal_
↪Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:'virginica'}
↪[x], iris.target)), columns = ['Species'])
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create training data and test data.

```
>>> dat = oml.sync(table = "IRIS").split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
```

User specified settings.

```
>>> setting = {'ODMS_SAMPLING':'ODMS_SAMPLING_DISABLE'}
```

Create AI model object.

```
>>> ai_mod = oml.ai(**setting)
```

Fit the AI Model according to the training data and parameter settings.

```
>>> ai_mod = ai_mod.fit(train_x, train_y)
```

Show model details.

```
>>> ai_mod

Algorithm Name: Attribute Importance

Mining Function: ATTRIBUTE_IMPORTANCE

Settings:
                   setting name            setting value
0                     ALGO_NAME              ALGO_AI_MDL
1                  ODMS_DETAILS              ODMS_ENABLE
2  ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
3                 ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
4                     PREP_AUTO                       ON

Global Statistics:
  attribute name  attribute value
0       NUM_ROWS              104

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

Importance:

       variable  importance  rank
0   Petal_Width    0.909653     1
1  Petal_Length    0.902137     2
2  Sepal_Length    0.374937     3
3   Sepal_Width    0.196127     4
```

Change parameter setting and refit model.

```
>>> new_setting = {'ODMS_SAMPLING':'ODMS_SAMPLING_ENABLE'}
>>> ai_mod.set_params(**new_setting).fit(train_x, train_y)

Algorithm Name: Attribute Importance

Mining Function: ATTRIBUTE_IMPORTANCE

Settings:
                   setting name            setting value
0                     ALGO_NAME              ALGO_AI_MDL
1                  ODMS_DETAILS              ODMS_ENABLE
2  ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
3                 ODMS_SAMPLING     ODMS_SAMPLING_ENABLE
4                     PREP_AUTO                       ON

Computed Settings:
       setting name setting value
0  ODMS_SAMPLE_SIZE           104

Global Statistics:
  attribute name  attribute value
0       NUM_ROWS              104

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

Importance:

       variable  importance  rank
0   Petal_Width    0.909653     1
1  Petal_Length    0.902137     2
2  Sepal_Length    0.374937     3
3   Sepal_Width    0.196127     4
```

```
>>> oml.drop('IRIS')
```

2.2 Association Rules

class oml.**ar**(*model_name=None*, *model_owner=None*, ***params*)

In-database Association Rules Model

Builds an Association Rules Model used to discover the probability of item co-occurrence in a collection. This function exposes the corresponding Oracle Machine Learning in-database algorithm. The relationships between co-occurring items are expressed as association rules. Oracle Machine Learning does not support the prediction functions for association modeling. The results of an association model are the rules that identify patterns of association within the data. Association rules can be ranked by support (How often do these items occur together in the data?) and confidence (How likely are these items to occur together in the data?).

Attributes

rules : oml.DataFrame

Details of each rule that shows how the appearance of a set of items in a transactional record implies the existence of another set of items. It includes the following components:

- rule_id: The identifier of the rule
- number_of_items: The total number of attributes referenced in the antecedent and consequent of the rule
- lhs_name: The name of the antecedent.
- lhs_value: The value of the antecedent.
- rhs_name: The name of the consequent.
- rhs_value: The value of the consequent.
- support: The number of transactions that satisfy the rule.
- confidence: The likelihood of a transaction satisfying the rule.
- revconfidence: The number of transactions in which the rule occurs divided by the number of transactions in which the consequent occurs.
- lift: The degree of improvement in the prediction over random chance when the rule is satisfied.

itemsets : oml.DataFrame

Description of the item sets from the model built. It includes the following components:

- itemset_id: The itemset identifier
- support: The support of the itemset
- number_of_items: The number of items in the itemset
- item_name: The name of the item
- item_value: The value of the item

See also:

`glm`, `svm`, `km`

`__init__`(*model_name=None*, *model_owner=None*, ***params*)

Initializes an instance of Association Rules object.

Parameters

model_name

[string or None (default)] The name of an existing database Association Rules model to create an oml.ar object from. The specified database model is not dropped when the oml.ar object is deleted.

model_owner: string or None (default)

The owner name of the existing Association Rules model The current database user by default

params

[key-value pairs or dict] Oracle Machine Learning parameter settings. Each list element's name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 4-10 apply generally to the model. Mining Function Settings for Association are applicable. No algorithm-specific Settings are applicable to Association model.

export_sermodel(*table=None*, *partition=None*)
Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

fit(*x*, *model_name=None*, *case_id=None*)
Fits an Association Rules Model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.ar object is deleted. If None, a system-generated model name will be used. The system-generated model is dropped when oml.ar object is deleted unless oml.ar object is saved into a datastore.

case_id
[string or None (default)] The column name used as case id for building the model.

get_params(*params=None*, *deep=False*)
Fetches parameters of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If `params` is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

set_params (***params*)
Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

References

- Oracle R Enterprise
- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

```
>>> import oml
```

Create training data.

```
>>> train_dat = oml.push(pd.DataFrame({'ID':[1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3], 'ITEM
↪':["b", "d", "e", "a", "b", "c", "e", "b", "c", "d", "e"]}))
```

User specified settings.

```
>>> setting = {'asso_min_support':'0.6', 'asso_min_confidence':'0.6', 'ODMS_ITEM_ID_
↪COLUMN_NAME':'ITEM'}
```

Create AR model object.

```
>>> ar_mod = oml.ar(**setting)
```

Fit the AR Model according to the training data and parameter settings.

```
>>> ar_mod = ar_mod.fit(train_dat, case_id = 'ID')
```

Show model details.

```
>>> ar_mod

Algorithm Name: Association Rules

Mining Function: ASSOCIATION

Settings:
                   setting name                   setting value
0                     ALGO_NAME  ALGO_APRIORI_ASSOCIATION_RULES
1          ASSO_MAX_RULE_LENGTH                               4
2           ASSO_MIN_CONFIDENCE                             0.6
3       ASSO_MIN_REV_CONFIDENCE                               0
4              ASSO_MIN_SUPPORT                             0.6
5          ASSO_MIN_SUPPORT_INT                               1
6                  ODMS_DETAILS                     ODMS_ENABLE
7       ODMS_ITEM_ID_COLUMN_NAME                            ITEM
8   ODMS_MISSING_VALUE_TREATMENT         ODMS_MISSING_VALUE_AUTO
9                 ODMS_SAMPLING           ODMS_SAMPLING_DISABLE
10                    PREP_AUTO                              ON

Global Statistics:
     attribute name  attribute value
0     ITEMSET_COUNT               11
1       MAX_SUPPORT                1
2          NUM_ROWS                3
3        RULE_COUNT               16
4  TRANSACTION_COUNT               3

Attributes:
ITEM

Partition: NO

Itemsets:

     ITEMSET_ID   SUPPORT  NUMBER_OF_ITEMS ITEM_NAME ITEM_VALUE
 0            1  1.000000                1         b       None
 1            2  0.666667                1         c       None
 2            3  0.666667                1         d       None
 3            4  1.000000                1         e       None
...         ...       ...              ...       ...        ...
 16          10  0.666667                3         c       None
 17          11  0.666667                3         d       None
 18          11  0.666667                3         b       None
 19          11  0.666667                3         e       None

Rules:

     RULE_ID  NUMBER_OF_ITEMS LHS_NAME LHS_VALUE RHS_NAME RHS_VALUE   SUPPORT  \
 0         1                2        c      None        b      None  0.666667
 1         2                2        b      None        c      None  0.666667
 2         3                2        d      None        b      None  0.666667
 3         4                2        b      None        d      None  0.666667
...      ...              ...      ...       ...      ...       ...       ...
 18       15                3        b      None        d      None  0.666667
 19       15                3        e      None        d      None  0.666667
 20       16                3        b      None        e      None  0.666667
```

(continues on next page)

(continued from previous page)

```
21       16                 3        d      None        e      None  0.666667

    CONFIDENCE  REVCONFIDENCE  LIFT
0     1.000000       0.666667     1
1     0.666667       1.000000     1
2     1.000000       0.666667     1
3     0.666667       1.000000     1
...        ...            ...   ...
18    0.666667       1.000000     1
19    0.666667       1.000000     1
20    1.000000       0.666667     1
21    1.000000       0.666667     1
```

Change parameter setting and refit model.

```
>>> new_setting = {'asso_min_support':'0.05', 'asso_min_confidence':'0.05'}
>>> ar_mod.set_params(**new_setting).fit(train_dat, case_id = 'ID')

Algorithm Name: Association Rules

Mining Function: ASSOCIATION

Settings:
                    setting name                    setting value
0                      ALGO_NAME  ALGO_APRIORI_ASSOCIATION_RULES
1           ASSO_MAX_RULE_LENGTH                               4
2            ASSO_MIN_CONFIDENCE                            0.05
3        ASSO_MIN_REV_CONFIDENCE                               0
4               ASSO_MIN_SUPPORT                            0.05
5           ASSO_MIN_SUPPORT_INT                               1
6                   ODMS_DETAILS                     ODMS_ENABLE
7       ODMS_ITEM_ID_COLUMN_NAME                            ITEM
8   ODMS_MISSING_VALUE_TREATMENT         ODMS_MISSING_VALUE_AUTO
9                  ODMS_SAMPLING           ODMS_SAMPLING_DISABLE
10                     PREP_AUTO                              ON

Global Statistics:
      attribute name  attribute value
0      ITEMSET_COUNT               23
1        MAX_SUPPORT                1
2           NUM_ROWS                3
3         RULE_COUNT               47
4   TRANSACTION_COUNT                3

Attributes:
ITEM

Partition: NO

Itemsets:

     ITEMSET_ID   SUPPORT  NUMBER_OF_ITEMS ITEM_NAME ITEM_VALUE
 0            1  0.333333                1         a       None
 1            2  1.000000                1         b       None
 2            3  0.666667                1         c       None
 3            4  0.666667                1         d       None
```

(continues on next page)

(continued from previous page)

```
...         ...        ...              ...      ...       ...
 48          23   0.333333                4        b       None
 49          23   0.333333                4        d       None
 50          23   0.333333                4        c       None
 51          23   0.333333                4        e       None

Rules:

    RULE_ID  NUMBER_OF_ITEMS LHS_NAME LHS_VALUE RHS_NAME RHS_VALUE   SUPPORT  \
 0        1                2        b      None        a      None  0.333333
 1        2                2        a      None        b      None  0.333333
 2        3                2        c      None        a      None  0.333333
 3        4                2        a      None        c      None  0.333333
...     ...              ...      ...       ...      ...       ...       ...
 80      46                4        e      None        d      None  0.333333
 81      47                4        b      None        e      None  0.333333
 82      47                4        c      None        e      None  0.333333
 83      47                4        d      None        e      None  0.333333

    CONFIDENCE  REVCONFIDENCE  LIFT
 0    0.333333       1.000000  1.00
 1    1.000000       0.333333  1.00
 2    0.500000       1.000000  1.50
 3    1.000000       0.500000  1.50
...        ...            ...   ...
 80   0.500000       0.500000  0.75
 81   1.000000       0.333333  1.00
 82   1.000000       0.333333  1.00
 83   1.000000       0.333333  1.00

[84 rows x 10 columns]
```

2.3 Decision Tree

class oml.**dt**(*model_name=None*, *model_owner=None*, ***params*)

In-database Decision Tree Model

Builds a Decision Tree Model used to generate rules (conditional statements that can easily be understood by humans and be used within a database to identify a set of records) to predict a target value (numeric or categorical column). This function exposes the corresponding Oracle Machine Learning in-database algorithm. A decision tree predicts a target value by asking a sequence of questions. At a given stage in the sequence, the question that is asked depends upon the answers to the previous questions. The goal is to ask questions that, taken together, uniquely identify specific target values. Graphically, this process forms a tree structure. During the training process, the Decision Tree algorithm must repeatedly find the most efficient way to split a set of cases (records) into two child nodes. The model offers two homogeneity metrics, gini and entropy, for calculating the splits. The default metric is gini.

Attributes

nodes : oml.DataFrame

The node summary information with tree node details. It includes the following components:

- parent: The node ID of the parent

- node.id: The node ID
- row.count: The number of records in the training set that belong to the node
- prediction: The predicted Target value
- split: The main split
- surrogate: The surrogate split
- full.splits: The full splitting criterion

distributions : oml.DataFrame

The target class distributions at each tree node. It includes the following components:

- node_id: The node ID
- target_value: The target value
- target_count: The number of rows for a given target_value

See also:

`glm`, `svm`, `km`

`__init__`(*model_name=None*, *model_owner=None*, ***params*)

Initializes an instance of dt object.

Parameters

model_name
[string or None (default)] The name of an existing database Decision Tree model to create an oml.dt object from. The specified database model is not dropped when the oml.dt object is deleted.

model_owner: string or None (default)
The owner name of the existing Decision Tree model The current database user by default

params
[key-value pairs or dict] Oracle Machine Learning parameter settings. Each dict element's name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 4-10 apply generally to the model. Mining Function Settings for Classification and Algorithm-specific Settings are applicable to Decision Tree model.

`export_sermodel`(*table=None*, *partition=None*)

Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

fit (*x*, *y*, *model_name=None*, *cost_matrix=None*, *case_id=None*)
Fits a decision tree model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object, or string] Target values. If y is a single column OML object, target values specified by y must be combinable with x. If y is a string, y is the name of the column in x that specifies the target values.

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.dt object is deleted. If None, a system-generated model name will be used. The system-generated model is dropped when oml.dt object is deleted unless oml.dt object is saved into a datastore.

cost_matrix
[OML DataFrame, list of ints, floats or None (default)] An optional numerical matrix that specifies the costs for incorrectly predicting the target values. The first value represents the actual target value. The second value represents the predicted target value. The third value is the cost. In general, the diagonal entries of the matrix are zeros. Refer to Oracle Data Mining User's Guide for more details about cost matrix.

case_id
[string or None (default)] The column name used as case id for building the model.

get_params (*params=None*, *deep=False*)
Fetches parameters of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If `params` is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

predict (*x*, *supplemental_cols=None*, *proba=False*, *topN_attrs=False*)
Makes predictions on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

proba
[boolean, False (default)] Returns prediction probability if proba is True.

topN_attrs
[boolean, positive integer, False (default)] Returns the top N most influence attributes of the predicted target value for regression if topN_attrs is not False. Returns the top N most influence attributes of the highest probability class for classification if topN_attrs is not False. N is equal to the specified positive integer or 5 if topN_attrs is True.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols`, and the results. The results include the most likely target class.

predict_proba (*x*, *supplemental_cols=None*, *topN=None*)
Makes predictions and returns probability for each class on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN
[positive integer or None (default)] A positive integer that restricts the returned target classes to the specified number of those that have the highest probability.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. The results include for each target class, the probability belonging to that class.

score (*x*, *y*)
Makes predictions on new data and returns the mean accuracy.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object] Target values.

Returns

score
[float] Mean accuracy for classifications.

set_params(***params*)

Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

References

- Oracle R Enterprise
- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

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

Create table containing dataset iris from sklearn.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width', 'Petal_
↪Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:'virginica'}
↪[x], iris.target)), columns = ['Species'])
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create training data and test data.

```
>>> dat = oml.sync(table = "IRIS").split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
```

Create a cost matrix table in the Oracle Database.

```
>>> cost_matrix = [['setosa', 'setosa', 0],
...                ['setosa', 'virginica', 0.2],
...                ['setosa', 'versicolor', 0.8],
...                ['virginica', 'virginica', 0],
...                ['virginica', 'setosa', 0.5],
...                ['virginica', 'versicolor', 0.5],
...                ['versicolor', 'versicolor', 0],
...                ['versicolor', 'setosa', 0.4],
```

(continues on next page)

(continued from previous page)

```
...                  ['versicolor', 'virginica', 0.6]]
>>> cost_matrix = oml.create(pd.DataFrame(cost_matrix, columns = ['ACTUAL_TARGET_VALUE
↪', 'PREDICTED_TARGET_VALUE', 'COST']), 'COST_MATRIX')
```

User specified settings.

```
>>> setting = {'TREE_TERM_MAX_DEPTH':'2'}
```

Create DT model object.

```
>>> dt_mod = oml.dt(**setting)
```

Fit the DT Model according to the training data and parameter settings.

```
>>> dt_mod.fit(train_x, train_y, cost_matrix = cost_matrix)

Algorithm Name: Decision Tree

Mining Function: CLASSIFICATION

Target: Species

Settings:
                    setting name             setting value
0                      ALGO_NAME        ALGO_DECISION_TREE
1           CLAS_COST_TABLE_NAME  "PYQUSER"."COST_MATRIX"
2              CLAS_MAX_SUP_BINS                        32
3          CLAS_WEIGHTS_BALANCED                       OFF
4                   ODMS_DETAILS               ODMS_ENABLE
5   ODMS_MISSING_VALUE_TREATMENT   ODMS_MISSING_VALUE_AUTO
6                  ODMS_SAMPLING     ODMS_SAMPLING_DISABLE
7                      PREP_AUTO                        ON
8           TREE_IMPURITY_METRIC        TREE_IMPURITY_GINI
9            TREE_TERM_MAX_DEPTH                         2
10         TREE_TERM_MINPCT_NODE                       .05
11        TREE_TERM_MINPCT_SPLIT                        .1
12         TREE_TERM_MINREC_NODE                        10
13        TREE_TERM_MINREC_SPLIT                        20

Global Statistics:
  attribute name  attribute value
0       NUM_ROWS              104

Attributes:
Petal_Length
Petal_Width

Partition: NO

Distributions:

    NODE_ID TARGET_VALUE  TARGET_COUNT
 0        0       setosa            36
 1        0   versicolor            35
 2        0    virginica            33
 3        1       setosa            36
 4        2   versicolor            35
```

(continues on next page)

(continued from previous page)

```
 5        2    virginica           33

Nodes:

    parent  node.id  row.count  prediction  \
 0     0.0        1         36      setosa
 1     0.0        2         68  versicolor
 2     NaN        0        104      setosa

                                          split  \
 0  (Petal_Length <=(2.4500000000000002E+000))
 1   (Petal_Length >(2.4500000000000002E+000))
 2                                         None

                                     surrogate  \
 0  Petal_Width <=(8.0000000000000004E-001))
 1   Petal_Width >(8.0000000000000004E-001))
 2                                      None

                                   full.splits
 0  (Petal_Length <=(2.4500000000000002E+000))
 1   (Petal_Length >(2.4500000000000002E+000))
 2                                          (
```

Or fit the DT Model using a single table dat[0] and target column name

```
>>> dt_mod.fit(dat[0], "Species", cost_matrix = cost_matrix)

Algorithm Name: Decision Tree

Mining Function: CLASSIFICATION

Target: Species

Settings:
                    setting name            setting value
0                      ALGO_NAME       ALGO_DECISION_TREE
1           CLAS_COST_TABLE_NAME  "PYQUSER"."COST_MATRIX"
2              CLAS_MAX_SUP_BINS                       32
3          CLAS_WEIGHTS_BALANCED                      OFF
4                   ODMS_DETAILS              ODMS_ENABLE
5   ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
6                  ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
7                      PREP_AUTO                       ON
8           TREE_IMPURITY_METRIC       TREE_IMPURITY_GINI
9             TREE_TERM_MAX_DEPTH                        2
10          TREE_TERM_MINPCT_NODE                      .05
11         TREE_TERM_MINPCT_SPLIT                       .1
12          TREE_TERM_MINREC_NODE                       10
13         TREE_TERM_MINREC_SPLIT                       20

Global Statistics:
  attribute name  attribute value
0       NUM_ROWS              104

Attributes:
Petal_Length
```

(continues on next page)

(continued from previous page)

```
Petal_Width

Partition: NO

Distributions:

   NODE_ID TARGET_VALUE  TARGET_COUNT
 0        0       setosa            36
 1        0   versicolor            35
 2        0    virginica            33
 3        1       setosa            36
 4        2   versicolor            35
 5        2    virginica            33

Nodes:

   parent  node.id  row.count  prediction  \
 0    0.0        1         36      setosa
 1    0.0        2         68  versicolor
 2    NaN        0        104      setosa

                                     split  \
 0  (Petal_Length <=(2.4500000000000002E+000))
 1   (Petal_Length >(2.4500000000000002E+000))
 2                                     None

                               surrogate  \
 0  Petal_Width <=(8.0000000000000004E-001))
 1   Petal_Width >(8.0000000000000004E-001))
 2                                     None

                               full.splits
 0  (Petal_Length <=(2.4500000000000002E+000))
 1   (Petal_Length >(2.4500000000000002E+000))
 2                                          (
```

Use the model to make predictions on test data.

```
>>> dt_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Petal_Length', 'Species']])
    Sepal_Length  Sepal_Width  Petal_Length     Species  PREDICTION
 0           4.9          3.0           1.4      setosa      setosa
 1           4.9          3.1           1.5      setosa      setosa
 2           4.8          3.4           1.6      setosa      setosa
 3           5.8          4.0           1.2      setosa      setosa
...          ...          ...           ...         ...         ...
 42          6.7          3.3           5.7   virginica  versicolor
 43          6.7          3.0           5.2   virginica  versicolor
 44          6.5          3.0           5.2   virginica  versicolor
 45          5.9          3.0           5.1   virginica  versicolor
```

```
>>> dt_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Species']], proba = True)
    Sepal_Length  Sepal_Width     Species  PREDICTION  PROBABILITY
 0           4.9          3.0      setosa      setosa     1.000000
 1           4.9          3.1      setosa      setosa     1.000000
```

(continues on next page)

(continued from previous page)

```
 2           4.8          3.4      setosa      setosa     1.000000
 3           5.8          4.0      setosa      setosa     1.000000
...          ...          ...         ...         ...          ...
 42          6.7          3.3   virginica  versicolor     0.514706
 43          6.7          3.0   virginica  versicolor     0.514706
 44          6.5          3.0   virginica  versicolor     0.514706
 45          5.9          3.0   virginica  versicolor     0.514706
```

```
>>> dt_mod.predict_proba(test_dat.drop('Species'), supplemental_cols = test_dat[:, [
↪'Sepal_Length', 'Species']]).sort_values(by = ['Sepal_Length', 'Species'])
     Sepal_Length      Species  PROBABILITY_OF_setosa   \
 0            4.4       setosa                    1.0
 1            4.4       setosa                    1.0
 2            4.5       setosa                    1.0
 3            4.8       setosa                    1.0
...           ...          ...                    ...
 42           6.7    virginica                    0.0
 43           6.9   versicolor                    0.0
 44           6.9    virginica                    0.0
 45           7.0   versicolor                    0.0

     PROBABILITY_OF_versicolor  PROBABILITY_OF_virginica
 0                    0.000000                  0.000000
 1                    0.000000                  0.000000
 2                    0.000000                  0.000000
 3                    0.000000                  0.000000
...                        ...                       ...
 42                   0.514706                  0.485294
 43                   0.514706                  0.485294
 44                   0.514706                  0.485294
 45                   0.514706                  0.485294
```

```
>>> dt_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.630435
```

Export serialized model to a table.

```
>>> dt_export = dt_mod.export_sermodel(table='dt_sermod')
>>> type(dt_export)
<class 'oml.core.bytes.Bytes'>
```

Show the first 100 characters of the BLOB content from the model export.

```
>>> dt_export.pull()[0][1:5]
b'\xff\xfc|\x00'
```

Reset TREE_TERM_MAX_DEPTH and refit model.

```
>>> dt_mod.set_params(TREE_TERM_MAX_DEPTH = '4').fit(train_x, train_y, cost_matrix =␣
↪cost_matrix)

Algorithm Name: Decision Tree

Mining Function: CLASSIFICATION

Target: Species
```

(continues on next page)

(continued from previous page)

```
Settings:
                   setting name            setting value
0                     ALGO_NAME       ALGO_DECISION_TREE
1          CLAS_COST_TABLE_NAME  "PYQUSER"."COST_MATRIX"
2              CLAS_MAX_SUP_BINS                       32
3         CLAS_WEIGHTS_BALANCED                      OFF
4                  ODMS_DETAILS              ODMS_ENABLE
5   ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
6                 ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
7                     PREP_AUTO                       ON
8          TREE_IMPURITY_METRIC       TREE_IMPURITY_GINI
9           TREE_TERM_MAX_DEPTH                        4
10        TREE_TERM_MINPCT_NODE                      .05
11       TREE_TERM_MINPCT_SPLIT                       .1
12        TREE_TERM_MINREC_NODE                       10
13       TREE_TERM_MINREC_SPLIT                       20

Global Statistics:
  attribute name  attribute value
0       NUM_ROWS              104

Attributes:
Petal_Length
Petal_Width

Partition: NO

Distributions:

   NODE_ID TARGET_VALUE  TARGET_COUNT
0        0       setosa            36
1        0   versicolor            35
2        0    virginica            33
3        1   versicolor            35
4        1    virginica            33
5        2       setosa            36
6        3   versicolor            35
7        3    virginica             5
8        4    virginica            28

Nodes:

   parent  node.id  row.count  prediction  \
0     0.0        1         68  versicolor
1     0.0        2         36      setosa
2     1.0        3         40  versicolor
3     1.0        4         28   virginica
4     NaN        0        104      setosa

                                        split  \
0   (Petal_Length >(2.4500000000000002E+000))
1  (Petal_Length <=(2.4500000000000002E+000))
2                (Petal_Width <=(1.75E+000))
3                 (Petal_Width >(1.75E+000))
4                                        None
```

(continues on next page)

(continued from previous page)

```
                                   surrogate  \
0    Petal_Width >(8.0000000000000004E-001))
1   Petal_Width <=(8.0000000000000004E-001))
2  Petal_Length <=(5.2000000000000002E+000))
3   Petal_Length >(5.2000000000000002E+000))
4                                       None

                                         full.splits
0          (Petal_Length >(2.4500000000000002E+000))
1         (Petal_Length <=(2.4500000000000002E+000))
2  (Petal_Length >(2.4500000000000002E+000)) AND ...
3  (Petal_Length >(2.4500000000000002E+000)) AND ...
4                                                  (
```

Fit the model with user-specified model name

```
>>> dt_mod=oml.dt()
>>> dt_mod=dt_mod.fit(train_x, train_y, model_name = 'DT_MODEL')
```

Show model name

```
>>> dt_mod.model_name
'DT_MODEL'
```

Show model details

```
>>> dt_mod

Model Name: DT_MODEL

Model Owner: PYQUSER

Algorithm Name: Decision Tree

Mining Function: CLASSIFICATION

Target: Species

Settings:
                    setting name            setting value
0                      ALGO_NAME       ALGO_DECISION_TREE
1              CLAS_MAX_SUP_BINS                       32
2          CLAS_WEIGHTS_BALANCED                      OFF
3                   ODMS_DETAILS              ODMS_ENABLE
4   ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
5                  ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
6                      PREP_AUTO                       ON
7           TREE_IMPURITY_METRIC       TREE_IMPURITY_GINI
8            TREE_TERM_MAX_DEPTH                        7
9          TREE_TERM_MINPCT_NODE                      .05
10        TREE_TERM_MINPCT_SPLIT                       .1
11         TREE_TERM_MINREC_NODE                       10
12        TREE_TERM_MINREC_SPLIT                       20

Global Statistics:
  attribute name  attribute value
0       NUM_ROWS              104
```

(continues on next page)

(continued from previous page)

```
Attributes:
Petal_Length
Petal_Width

Partition: NO

Distributions:

   NODE_ID TARGET_VALUE  TARGET_COUNT
0        0       setosa            36
1        0   versicolor            35
2        0    virginica            33
3        1   versicolor            35
4        1    virginica            33
5        2       setosa            36
6        3   versicolor            35
7        3    virginica             5
8        4    virginica            28

Nodes:

   parent  node.id  row.count  prediction  \
0     0.0        1         68  versicolor
1     0.0        2         36      setosa
2     1.0        3         40  versicolor
3     1.0        4         28   virginica
4     NaN        0        104      setosa

                                      split  \
0    (Petal_Length >(2.4500000000000002E+000))
1   (Petal_Length <=(2.4500000000000002E+000))
2                 (Petal_Width <=(1.75E+000))
3                  (Petal_Width >(1.75E+000))
4                                        None

                                  surrogate  \
0    Petal_Width >(8.0000000000000004E-001))
1   Petal_Width <=(8.0000000000000004E-001))
2  Petal_Length <=(5.2000000000000002E+000))
3   Petal_Length >(5.2000000000000002E+000))
4                                       None

                                         full.splits
0          (Petal_Length >(2.4500000000000002E+000))
1         (Petal_Length <=(2.4500000000000002E+000))
2  (Petal_Length >(2.4500000000000002E+000)) AND ...
3  (Petal_Length >(2.4500000000000002E+000)) AND ...
4                                                  (
```

Refit the model using a different user-specified model name

```
>>> dt_mod=dt_mod.fit(train_x, train_y, model_name = 'NEW_DT_MODEL')
>>> dt_mod.model_name
'NEW_DT_MODEL'
```

Refit the model without user-specified model name

```
>>> dt_mod=dt_mod.fit(train_x, train_y)
>>> dt_mod.model_name
''
```

Rename the model with a new user-specified model name

```
>>> dt_mod.model_name = 'DT_MODEL_RENAME'
>>> dt_mod.model_name
'DT_MODEL_RENAME'
```

Sync an existing database model

```
>>> dt_mod2 = oml.dt(model_name = 'DT_MODEL')
>>> dt_mod2.model_name
'DT_MODEL'
```

```
>>> oml.drop(model = 'DT_MODEL')
>>> oml.drop(model = 'NEW_DT_MODEL')
>>> oml.drop(model = 'DT_MODEL_RENAME')
>>> oml.drop(table = 'COST_MATRIX')
>>> oml.drop(table = 'dt_sermod')
>>> oml.drop('IRIS')
```

2.4 Expectation Maximization

class oml.**em**(*mining_function='CLUSTERING'*, *n_clusters=None*, *model_name=None*, *model_owner=None*, ***params*)

In-database Expectation Maximization Model

Builds an Expectation Maximization (EM) Model used to performs probabilistic clustering based on a density estimation algorithm. This function exposes the corresponding Oracle Machine Learning in-database algorithm. In density estimation, the goal is to construct a density function that captures how a given population is distributed. The density estimate is based on observed data that represents a sample of the population.

Attributes

clusters : oml.DataFrame

The general per-cluster information. It includes the following components:

- cluster_id: The ID of a cluster in the model
- cluster_name: The name of a cluster in the model
- record_count: The number of rows used in the build
- parent: The ID of the parent
- tree_level: The number of splits from the root
- left_child_id: The ID of the left child
- right_child_id: The ID of the right child

taxonomy: oml.DataFrame

The parent/child cluster relationship. It includes the following components:

- parent_cluster_id: The ID of the parent cluster
- child_cluster_id: The ID of the child cluster

centroids: oml.DataFrame

Per cluster-attribute center (centroid) information. It includes the following components:

- cluster_id: The ID of a cluster in the model
- attribute_name: The attribute name
- mean: The average value of a numeric attribute
- mode_value: The most frequent value of a categorical attribute
- variance: The variance of a numeric attribute

leaf_cluster_counts: pandas.DataFrame

Leaf clusters with support. It includes the following components:

- cluster_id: The ID of a leaf cluster in the model
- cnt: The number of records in a leaf cluster

attribute_importance: oml.DataFrame

Attribute importance of the fitted model. It includes the following components:

- attribute_name: The attribute name
- attribute_importance_value: The attribute importance for an attribute
- attribute_rank: The rank of the attribute based on importance

projection: oml.DataFrame

The coefficients used by random projections to map nested columns to a lower dimensional space. It exists only when nested or text data is present in the build data. It includes the following components:

- feature_name: The name of feature
- attribute_name: The attribute name
- attribute_value: The attribute value
- coefficient: The projection coefficient for an attribute

components: oml.DataFrame

EM components information about their prior probabilities and what cluster they map to. It includes the following components:

- component_id: The unique identifier of a component
- cluster_id: The ID of a cluster in the model
- prior_probability: The component prior probability

cluster_hists: oml.DataFrame

Cluster histogram information. It includes the following components:

- cluster.id: The ID of a cluster in the model
- variable: The attribute name
- bin.id: The ID of a bin
- lower.bound: The numeric lower bin boundary

- upper.bound: The numeric upper bin boundary
- label: The label of the cluster
- count: The histogram count

rules: oml.DataFrame

Conditions for a case to be assigned with some probability to a cluster. It includes the following components:

- cluster.id: The ID of a cluster in the model
- rhs.support: The record count
- rhs.conf: The record confidence
- lhr.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

See also:

glm, *svm*, *km*

__init__(*mining_function='CLUSTERING'*, *n_clusters=None*, *model_name=None*, *model_owner=None*, ***params*)

Initializes an instance of em object.

Parameters

mining_function
['CLUSTERING' (default) or 'ANOMALY_DETECTION'] Type of model mining functionality

n_clusters
[positive integer, None (default)] The number of clusters. If n_clusters is None, the number of clusters will be determined either by current setting parameters or automatically by the algorithm.

model_name
[string or None (default)] The name of an existing database Expectation Maximization model to create an oml.em object from. The specified database model is not dropped when the oml.em object is deleted.

model_owner: string or None (default)
The owner name of the existing Expectation Maximization model The current database user by default

params
[key-value pairs or dict] Oracle Machine Learning parameter settings. Each list element's name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 4-10 apply generally to the model. Mining Function Settings for Clustering and Algorithm-specific Settings are applicable to Expectation Maximization model.

export_sermodel(*table=None*, *partition=None*)
Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

fit(*x*, *model_name=None*, *case_id=None*)
Fits an Expectation Maximization Model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model.

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.em object is deleted. If None, a system-generated model name will be used. The system-generated model is dropped when oml.em object is deleted unless oml.em object is saved into a datastore.

case_id
[string or None (default)] The name of a column that contains unique case identifiers.

get_params(*params=None*, *deep=False*)
Fetches parameters of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If `params` is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

predict (*x*, *supplemental_cols=None*, *topN_attrs=False*, *proba=False*)
Makes predictions on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN_attrs
[boolean, positive integer, False (default)] Returns the top N most likely clusters with the corresponding weights. N is equal to the specified positive integer or 5 if topN_attrs is True.

proba
[boolean, False (default)] Returns prediction probability if `proba` is True.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. If the mode is 'class', the results include the most likely target class and its probability. If mode is 'raw', the results include for each target class, the probability belonging to that class.

predict_proba (*x*, *supplemental_cols=None*, *topN=None*)
Makes predictions and returns probability for each cluster on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN
[positive integer or None (default)] A positive integer that restricts the returned clusters to the specified number of those that have the highest probability.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. The results include for each cluster, the probability belonging to that cluster.

set_params (***params*)
Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

References

- Oracle R Enterprise
- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

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

Create table containing dataset iris from sklearn

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width', 'Petal_
↪Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:'virginica'}
↪[x], iris.target)), columns = ['Species'])
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create training data and test data

```
>>> dat = oml.sync(table = "IRIS").split()
>>> train_dat = dat[0]
>>> test_dat = dat[1]
```

User specified settings

```
>>> setting = {'emcs_num_iterations': 100}
```

Create EM model object

```
>>> em_mod = oml.em(n_clusters = 2, **setting)
```

Fit the EM Model according to the training data and parameter settings.

```
>>> em_mod = em_mod.fit(train_dat)
```

Show model details

```
>>> em_mod

Algorithm Name: Expectation Maximization
```

(continues on next page)

(continued from previous page)

```
Mining Function: CLUSTERING

Settings:
                   setting name                  setting value
0                     ALGO_NAME  ALGO_EXPECTATION_MAXIMIZATION
1             CLUS_NUM_CLUSTERS                              2
2       EMCS_CLUSTER_COMPONENTS       EMCS_CLUSTER_COMP_ENABLE
3       EMCS_CLUSTER_STATISTICS         EMCS_CLUS_STATS_ENABLE
4           EMCS_CLUSTER_THRESH                              2
5         EMCS_LINKAGE_FUNCTION            EMCS_LINKAGE_SINGLE
6      EMCS_LOGLIKE_IMPROVEMENT                           .001
7          EMCS_MAX_NUM_ATTR_2D                             50
8     EMCS_MIN_PCT_ATTR_SUPPORT                             .1
9             EMCS_MODEL_SEARCH      EMCS_MODEL_SEARCH_DISABLE
10          EMCS_NUM_COMPONENTS                             20
11        EMCS_NUM_DISTRIBUTION          EMCS_NUM_DISTR_SYSTEM
12      EMCS_NUM_EQUIWIDTH_BINS                             11
13          EMCS_NUM_ITERATIONS                            100
14         EMCS_NUM_PROJECTIONS                             50
15             EMCS_RANDOM_SEED                              0
16       EMCS_REMOVE_COMPONENTS       EMCS_REMOVE_COMPS_ENABLE
17                 ODMS_DETAILS                    ODMS_ENABLE
18  ODMS_MISSING_VALUE_TREATMENT        ODMS_MISSING_VALUE_AUTO
19                ODMS_SAMPLING          ODMS_SAMPLING_DISABLE
20                    PREP_AUTO                             ON

Computed Settings:
                 setting name            setting value
0       EMCS_ATTRIBUTE_FILTER  EMCS_ATTR_FILTER_DISABLE
1  EMCS_CONVERGENCE_CRITERION        EMCS_CONV_CRIT_BIC
2      EMCS_NUM_QUANTILE_BINS                         3
3          EMCS_NUM_TOPN_BINS                         3

Global Statistics:
       attribute name attribute value
0           CONVERGED             YES
1       LOGLIKELIHOOD        -2.10044
2        NUM_CLUSTERS               2
3      NUM_COMPONENTS               8
4            NUM_ROWS             104
5         RANDOM_SEED               0
6  REMOVED_COMPONENTS              12

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Clusters:

   CLUSTER_ID  CLUSTER_NAME  RECORD_COUNT  PARENT  TREE_LEVEL  LEFT_CHILD_ID  \
0           1             1           104     NaN           1            2.0
```

(continues on next page)

(continued from previous page)

```
1          2             2             68     1.0           2             NaN
2          3             3             36     1.0           2             NaN

   RIGHT_CHILD_ID
0             3.0
1             NaN
2             NaN

Taxonomy:

   PARENT_CLUSTER_ID  CHILD_CLUSTER_ID
0                  1               2.0
1                  1               3.0
2                  2               NaN
3                  3               NaN

Centroids:

    CLUSTER_ID ATTRIBUTE_NAME      MEAN  MODE_VALUE  VARIANCE
0            1   Petal_Length  3.721154        None  3.234694
1            1    Petal_Width  1.155769        None  0.567539
2            1   Sepal_Length  5.831731        None  0.753255
3            1    Sepal_Width  3.074038        None  0.221358
4            1        Species       NaN      setosa       NaN
5            2   Petal_Length  4.902941        None  0.860588
6            2    Petal_Width  1.635294        None  0.191572
7            2   Sepal_Length  6.266176        None  0.545555
8            2    Sepal_Width  2.854412        None  0.128786
9            2        Species       NaN  versicolor       NaN
10           3   Petal_Length  1.488889        None  0.033016
11           3    Petal_Width  0.250000        None  0.012857
12           3   Sepal_Length  5.011111        None  0.113016
13           3    Sepal_Width  3.488889        None  0.134159
14           3        Species       NaN      setosa       NaN

Leaf Cluster Counts:

   CLUSTER_ID  CNT
0           2   68
1           3   36

Attribute Importance:

  ATTRIBUTE_NAME  ATTRIBUTE_IMPORTANCE_VALUE  ATTRIBUTE_RANK
0   Petal_Length                    0.558311               2
1    Petal_Width                    0.556300               3
2   Sepal_Length                    0.469978               4
3    Sepal_Width                    0.196211               5
4        Species                    0.612463               1

Components:

   COMPONENT_ID  CLUSTER_ID  PRIOR_PROBABILITY
0             1           2           0.115366
1             2           2           0.079158
2             3           3           0.113448
3             4           2           0.148059
```

(continues on next page)

(continued from previous page)

```
4            5          3          0.126979
5            6          2          0.134402
6            7          3          0.105727
7            8          2          0.176860

Cluster Hists:

     cluster.id            variable  bin.id  lower.bound  upper.bound  \
0             1        Petal_Length       1         1.00         1.59
1             1        Petal_Length       2         1.59         2.18
2             1        Petal_Length       3         2.18         2.77
3             1        Petal_Length       4         2.77         3.36
...         ...                 ...     ...          ...          ...
137           3         Sepal_Width      11          NaN          NaN
138           3      Species:'Other'      1          NaN          NaN
139           3       Species:setosa      2          NaN          NaN
140           3   Species:versicolor      3          NaN          NaN

         label  count
0       1:1.59     25
1    1.59:2.18     11
2    2.18:2.77      0
3    2.77:3.36      3
...        ...    ...
137          :      0
138          :      0
139          :     36
140          :      0

[141 rows x 7 columns]

Rules:

     cluster.id  rhs.support  rhs.conf  lhr.support  lhs.conf       lhs.var  \
 0            1          104  1.000000           93  0.894231   Sepal_Width
 1            1          104  1.000000           93  0.894231   Sepal_Width
 2            1          104  1.000000           99  0.894231  Petal_Length
 3            1          104  1.000000           99  0.894231  Petal_Length
...         ...          ...       ...          ...       ...           ...
 26           3           36  0.346154           36  0.972222  Petal_Length
 27           3           36  0.346154           36  0.972222  Sepal_Length
 28           3           36  0.346154           36  0.972222  Sepal_Length
 29           3           36  0.346154           36  0.972222       Species

     lhs.var.support  lhs.var.conf             predicate
 0                93      0.400000    Sepal_Width <= 3.92
 1                93      0.400000     Sepal_Width > 2.48
 2                93      0.222222   Petal_Length <= 6.31
 3                93      0.222222      Petal_Length >= 1
...              ...           ...                   ...
 26               35      0.134398      Petal_Length >= 1
 27               35      0.094194   Sepal_Length <= 5.74
 28               35      0.094194    Sepal_Length >= 4.3
 29               35      0.281684       Species = setosa
```

```
>>> em_mod.predict(test_dat)
    CLUSTER_ID
 0           3
 1           3
 2           3
 3           3
...        ...
 42          2
 43          2
 44          2
 45          2
```

```
>>> em_mod.predict_proba(test_dat, supplemental_cols = test_dat[:, ['Sepal_Length',
↪'Sepal_Width', 'Petal_Length']]).sort_values(by = ['Sepal_Length', 'Sepal_Width',
↪'Petal_Length', 'PROBABILITY_OF_2', 'PROBABILITY_OF_3'])
    Sepal_Length  Sepal_Width  Petal_Length  PROBABILITY_OF_2  \
 0           4.4          3.0           1.3      4.680788e-20
 1           4.4          3.2           1.3      1.052071e-20
 2           4.5          2.3           1.3      7.751240e-06
 3           4.8          3.4           1.6      5.363418e-19
...          ...          ...           ...               ...
 43          6.9          3.1           4.9      1.000000e+00
 44          6.9          3.1           5.4      1.000000e+00
 45          7.0          3.2           4.7      1.000000e+00

    PROBABILITY_OF_3
 0      1.000000e+00
 1      1.000000e+00
 2      9.999922e-01
 3      1.000000e+00
...              ...
 43     3.295578e-97
 44    6.438740e-137
 45     3.853925e-89
```

Change random seed and refit model.

```
>>> em_mod.set_params(EMCS_RANDOM_SEED = '5').fit(train_dat)

Algorithm Name: Expectation Maximization

Mining Function: CLUSTERING

Settings:
                    setting name                setting value
0                      ALGO_NAME  ALGO_EXPECTATION_MAXIMIZATION
1              CLUS_NUM_CLUSTERS                              2
2        EMCS_CLUSTER_COMPONENTS        EMCS_CLUSTER_COMP_ENABLE
3        EMCS_CLUSTER_STATISTICS         EMCS_CLUS_STATS_ENABLE
4            EMCS_CLUSTER_THRESH                              2
5          EMCS_LINKAGE_FUNCTION            EMCS_LINKAGE_SINGLE
6       EMCS_LOGLIKE_IMPROVEMENT                           .001
7           EMCS_MAX_NUM_ATTR_2D                             50
8      EMCS_MIN_PCT_ATTR_SUPPORT                             .1
9              EMCS_MODEL_SEARCH       EMCS_MODEL_SEARCH_DISABLE
10           EMCS_NUM_COMPONENTS                             20
11         EMCS_NUM_DISTRIBUTION          EMCS_NUM_DISTR_SYSTEM
```

(continues on next page)

(continued from previous page)

```
12       EMCS_NUM_EQUIWIDTH_BINS                               11
13           EMCS_NUM_ITERATIONS                              100
14          EMCS_NUM_PROJECTIONS                               50
15              EMCS_RANDOM_SEED                                5
16        EMCS_REMOVE_COMPONENTS       EMCS_REMOVE_COMPS_ENABLE
17                  ODMS_DETAILS                    ODMS_ENABLE
18  ODMS_MISSING_VALUE_TREATMENT        ODMS_MISSING_VALUE_AUTO
19                 ODMS_SAMPLING          ODMS_SAMPLING_DISABLE
20                     PREP_AUTO                             ON

Computed Settings:
                 setting name             setting value
0       EMCS_ATTRIBUTE_FILTER  EMCS_ATTR_FILTER_DISABLE
1  EMCS_CONVERGENCE_CRITERION        EMCS_CONV_CRIT_BIC
2      EMCS_NUM_QUANTILE_BINS                         3
3          EMCS_NUM_TOPN_BINS                         3

Global Statistics:
       attribute name attribute value
0           CONVERGED             YES
1       LOGLIKELIHOOD        -1.75777
2        NUM_CLUSTERS               2
3      NUM_COMPONENTS               9
4            NUM_ROWS             104
5         RANDOM_SEED               5
6  REMOVED_COMPONENTS              11

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Clusters:

   CLUSTER_ID  CLUSTER_NAME  RECORD_COUNT  PARENT  TREE_LEVEL  LEFT_CHILD_ID  \
0           1             1           104     NaN           1            2.0
1           2             2            36     1.0           2            NaN
2           3             3            68     1.0           2            NaN

   RIGHT_CHILD_ID
0             3.0
1             NaN
2             NaN

Taxonomy:

   PARENT_CLUSTER_ID  CHILD_CLUSTER_ID
0                  1               2.0
1                  1               3.0
2                  2               NaN
3                  3               NaN

Centroids:
```

(continues on next page)

(continued from previous page)

```
   CLUSTER_ID ATTRIBUTE_NAME      MEAN  MODE_VALUE  VARIANCE
0           1   Petal_Length  3.721154        None  3.234694
1           1    Petal_Width  1.155769        None  0.567539
2           1   Sepal_Length  5.831731        None  0.753255
3           1    Sepal_Width  3.074038        None  0.221358
4           1        Species       NaN      setosa       NaN
5           2   Petal_Length  1.488889        None  0.033016
6           2    Petal_Width  0.250000        None  0.012857
7           2   Sepal_Length  5.011111        None  0.113016
8           2    Sepal_Width  3.488889        None  0.134159
9           2        Species       NaN      setosa       NaN
10          3   Petal_Length  4.902941        None  0.860588
11          3    Petal_Width  1.635294        None  0.191572
12          3   Sepal_Length  6.266176        None  0.545555
13          3    Sepal_Width  2.854412        None  0.128786
14          3        Species       NaN  versicolor       NaN

Leaf Cluster Counts:

   CLUSTER_ID  CNT
0           2   36
1           3   68

Attribute Importance:

  ATTRIBUTE_NAME  ATTRIBUTE_IMPORTANCE_VALUE  ATTRIBUTE_RANK
0   Petal_Length                    0.558311               2
1    Petal_Width                    0.556300               3
2   Sepal_Length                    0.469978               4
3    Sepal_Width                    0.196211               5
4        Species                    0.612463               1

Components:

   COMPONENT_ID  CLUSTER_ID  PRIOR_PROBABILITY
0             1           2           0.113452
1             2           2           0.105727
2             3           3           0.114202
3             4           3           0.086285
4             5           3           0.067294
5             6           3           0.124365
6             7           2           0.126975
7             8           3           0.105761
8             9           3           0.155939

Cluster Hists:

     cluster.id            variable  bin.id  lower.bound  upper.bound  \
0             1        Petal_Length       1         1.00         1.59
1             1        Petal_Length       2         1.59         2.18
2             1        Petal_Length       3         2.18         2.77
3             1        Petal_Length       4         2.77         3.36
...         ...                 ...     ...          ...          ...
137           3         Sepal_Width      11          NaN          NaN
138           3      Species:'Other'       1          NaN          NaN
139           3      Species:setosa        2          NaN          NaN
```

(continues on next page)

(continued from previous page)

```
140           3  Species:versicolor       3          NaN          NaN

         label  count
0       1:1.59     25
1    1.59:2.18     11
2    2.18:2.77      0
3    2.77:3.36      3
...        ...    ...
137          :      0
138          :     33
139          :      0
140          :     35

[141 rows x 7 columns]

Rules:

     cluster.id  rhs.support  rhs.conf  lhr.support  lhs.conf        lhs.var  \
 0            1          104  1.000000           93  0.894231    Sepal_Width
 1            1          104  1.000000           93  0.894231    Sepal_Width
 2            1          104  1.000000           99  0.894231   Petal_Length
 3            1          104  1.000000           99  0.894231   Petal_Length
...         ...          ...       ...          ...       ...            ...
 26           3           68  0.653846           68  0.955882   Sepal_Length
 27           3           68  0.653846           68  0.955882   Sepal_Length
 28           3           68  0.653846           68  0.955882        Species
 29           3           68  0.653846           68  0.955882        Species

     lhs.var.support  lhs.var.conf             predicate
 0                93      0.400000    Sepal_Width <= 3.92
 1                93      0.400000     Sepal_Width > 2.48
 2                93      0.222222   Petal_Length <= 6.31
 3                93      0.222222      Petal_Length >= 1
...              ...           ...                   ...
 26               65      0.026013    Sepal_Length <= 7.9
 27               65      0.026013    Sepal_Length > 4.66
 28               65      0.125809     Species IN 'Other'
 29               65      0.125809  Species IN versicolor
```

EM Anomaly Detection

```
>>> em_mod = oml.em(mining_function = "anomaly_detection").fit(train_dat)
```

Show model details

```
>>> em_mod

Algorithm Name: Expectation Maximization

Mining Function: ANOMALY_DETECTION

Settings:
                    setting name                  setting value
0                      ALGO_NAME  ALGO_EXPECTATION_MAXIMIZATION
1          EMCS_ATTRIBUTE_FILTER        EMCS_ATTR_FILTER_DISABLE
2       EMCS_LOGLIKE_IMPROVEMENT                            .001
```

(continues on next page)

(continued from previous page)

```
3           EMCS_MAX_NUM_ATTR_2D                            50
4              EMCS_MODEL_SEARCH      EMCS_MODEL_SEARCH_DISABLE
5            EMCS_NUM_COMPONENTS                            20
6          EMCS_NUM_DISTRIBUTION          EMCS_NUM_DISTR_SYSTEM
7        EMCS_NUM_EQUIWIDTH_BINS                            11
8            EMCS_NUM_ITERATIONS                           100
9           EMCS_NUM_PROJECTIONS                            50
10             EMCS_OUTLIER_RATE                           .05
11              EMCS_RANDOM_SEED                             0
12        EMCS_REMOVE_COMPONENTS     EMCS_REMOVE_COMPS_ENABLE
13                  ODMS_DETAILS                   ODMS_ENABLE
14  ODMS_MISSING_VALUE_TREATMENT       ODMS_MISSING_VALUE_AUTO
15                 ODMS_SAMPLING         ODMS_SAMPLING_DISABLE
16                     PREP_AUTO                            ON

Computed Settings:
                 setting name       setting value
0  EMCS_CONVERGENCE_CRITERION  EMCS_CONV_CRIT_BIC
1      EMCS_NUM_QUANTILE_BINS                   3
2          EMCS_NUM_TOPN_BINS                   3

Global Statistics:
       attribute name attribute value
0           CONVERGED             YES
1       LOGLIKELIHOOD          -8.488
2      NUM_COMPONENTS               8
3            NUM_ROWS             104
4         RANDOM_SEED               0
5  REMOVED_COMPONENTS              12

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO
```

```
>>> em_mod.predict(test_dat)
     PREDICTION
 0            1
 1            1
 2            1
 3            1
...         ...
 42           1
 43           1
 44           1
 45           1
```

```
>>> em_mod.predict_proba(test_dat, supplemental_cols = test_dat[:, ['Sepal_Length',
↪'Sepal_Width', 'Petal_Length']]).sort_values(by = ['Sepal_Length', 'Sepal_Width',
↪'Petal_Length'])
     Sepal_Length  Sepal_Width  Petal_Length  PROBABILITY_OF_0  \
```

(continues on next page)

(continued from previous page)

```
 0            4.4          3.0           1.3      3.842800e-09
 1            4.4          3.2           1.3      2.853427e-08
 2            4.5          2.3           1.3      1.000000e+00
 3            4.8          3.4           1.6      1.296925e-08
...           ...          ...           ...               ...
 43           6.9          3.1           4.9      4.703019e-01
 44           6.9          3.1           5.4      2.173003e-03
 45           7.0          3.2           4.7      8.308335e-02

    PROBABILITY_OF_1
 0      1.000000e+00
 1      1.000000e+00
 2      7.360312e-35
 3      1.000000e+00
...              ...
 43     5.296981e-01
 44     9.978270e-01
 45     9.169166e-01
```

```
>>> oml.drop('IRIS')
```

2.5 Explicit Semantic Analysis

class oml.**esa**(*model_name=None*, *model_owner=None*, ***params*)

In-database Explicit Semantic Analysis Model

Builds an Explicit Semantic Analysis (ESA) Model to be used for feature extraction. This function exposes the corresponding Oracle Machine Learning in-database algorithm. ESA uses concepts of an existing knowledge base as features rather than latent features derived by latent semantic analysis methods such as Singular Value Decomposition and Latent Dirichlet Allocation. Each row, for example, a document in the training data maps to a feature, that is, a concept. ESA works best with concepts represented by text documents. It has multiple applications in the area of text processing, most notably semantic relatedness (similarity) and explicit topic modeling. Text similarity use cases might involve, for example, resume matching, searching for similar blog postings, and so on.

Attributes

features : oml.DataFrame

Description of each feature extracted. It includes the following components:

- feature_id: The unique identifier of a feature as it appears in the training data
- attribute_name: The attribute name
- attribute_value: The attribute value
- coefficient: The coefficient (weight) associated with the attribute in a particular feature.

__init__(*model_name=None*, *model_owner=None*, ***params*)

Initializes an instance of esa object.

Parameters

model_name

[string or None (default)] The name of an existing database Explicit Semantic Analysis model to create an oml.esa object from. The specified database model is not dropped when the oml.esa object is deleted.

model_owner: string or None (default)
The owner name of the existing Explicit Semantic Analysis model The current database user by default

params
[key-value pairs or dict] Oracle Machine Learning parameter settings. Each list element's name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 4-10 apply generally to the model. Mining Function Settings for Feature Extraction and Algorithm-specific Settings are applicable to Explicit Semantic Analysis model.

export_sermodel (*table=None*, *partition=None*)
Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

feature_compare (*x*, *compare_cols=None*, *supplemental_cols=None*)
Compares features of data and generates relatedness.

Parameters

x [an OML object] The data used to measure relatedness.

compare_cols
[str, a list of str or None (default)] The column(s) used to measure data relatedness. If None, all the columns of `x` are compared to measure relatedness.

supplemental_cols
[a list of str or None (default)] A list of columns to display along with the resulting 'SIMILARITY' column.

Returns

pred
[oml.DataFrame] Contains a 'SIMILARITY' column that measures relatedness and supplementary columns if specified.

fit (*x*, *model_name=None*, *case_id=None*, *ctx_settings=None*)
Fits an ESA Model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model.

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.esa object is deleted. If None, a system-generated model name will be used. The system-generated model is dropped when oml.esa object is deleted unless oml.esa object is saved into a datastore.

case_id
[string or None (default)] The column name used as case id for building the model.

ctx_settings
[dict or None (default)] A list to specify Oracle Text attribute-specific settings. This argument is applicable to building models in Oracle Database 12.2 or later. The name of each list element refers to the text column while the list value is a scalar string specifying the attribute-specific text transformation. The valid entries in the string include TEXT, POLICY_NAME, TOKEN_TYPE, and MAX_FEATURES.

get_params (*params=None*, *deep=False*)
Fetches parameters of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If `params` is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

predict (*x*, *supplemental_cols=None*, *topN_attrs=False*)
Makes predictions on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN_attrs
[boolean, positive integer, False (default)] Returns the top N most important features with the corresponding weights. N is equal to the specified positive integer or 5 if topN_attrs is True.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. The results include the most likely feature and its probability.

set_params(***params*)
Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

transform(*x*, *supplemental_cols=None*, *topN=None*)
Make predictions and return relevancy for each feature on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN
[positive integer or None (default)] A positive integer that restricts the returned values to the specified number of features that have the highest topN values.

Returns

pred
[oml.DataFrame] Contains the relevancy for each feature on new data and the specified `supplemental_cols`.

References

- Oracle R Enterprise
- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

```
>>> import oml
>>> from oml import cursor
>>> import pandas as pd
```

Create training data and test data.

```
>>> dat = oml.push(pd.DataFrame({'COMMENTS':['Aids in Africa: Planning for a long war
↪',
...                                           'Mars rover maneuvers for rim shot',
...                                           'Mars express confirms presence of␣
↪water at Mars south pole',
...                                           'NASA announces major Mars rover finding
↪',
...                                           'Drug access, Asia threat in focus at␣
↪AIDS summit',
...                                           'NASA Mars Odyssey THEMIS image:␣
↪typical crater',
...                                           'Road blocks for Aids'],
...                              'YEAR':['2017', '2018', '2017', '2017', '2018',
↪'2018', '2018'],
...                              'ID':[1,2,3,4,5,6,7]})).split(ratio=(0.7,0.3), seed␣
↪= 1234)
>>> train_dat = dat[0]
>>> test_dat = dat[1]
```

User specified settings.

```
>>> cur = cursor()
>>> cur.execute("Begin ctx_ddl.create_policy('DMDEMO_ESA_POLICY'); End;")
>>> cur.close()
```

```
>>> odm_settings = {'odms_text_policy_name': 'DMDEMO_ESA_POLICY',
...                 '"ODMS_TEXT_MIN_DOCUMENTS"': 1,
...                 '"ESAS_MIN_ITEMS"': 1}
```

```
>>> ctx_settings = {'COMMENTS': 'TEXT(POLICY_NAME:DMDEMO_ESA_POLICY)(TOKEN_TYPE:STEM)
↪'}
```

Create ESA model object.

```
>>> esa_mod = oml.esa(**odm_settings)
```

Fit the ESA Model according to the training data and parameter settings.

```
>>> esa_mod = esa_mod.fit(train_dat, case_id = 'ID', ctx_settings = ctx_settings)
```

Show model details.

```
>>> esa_mod

Algorithm Name: Explicit Semantic Analysis

Mining Function: FEATURE_EXTRACTION
```

(continues on next page)

(continued from previous page)

```
Settings:
                     setting name                  setting value
0                       ALGO_NAME  ALGO_EXPLICIT_SEMANTIC_ANALYS
1                  ESAS_MIN_ITEMS                              1
2              ESAS_TOPN_FEATURES                           1000
3            ESAS_VALUE_THRESHOLD                      .00000001
4                    ODMS_DETAILS                    ODMS_ENABLE
5    ODMS_MISSING_VALUE_TREATMENT        ODMS_MISSING_VALUE_AUTO
6                   ODMS_SAMPLING          ODMS_SAMPLING_DISABLE
7           ODMS_TEXT_MAX_FEATURES                         300000
8          ODMS_TEXT_MIN_DOCUMENTS                              1
9           ODMS_TEXT_POLICY_NAME               DMDEMO_ESA_POLICY
10                      PREP_AUTO                             ON

Computed Settings:
              setting name setting value
0   ODMS_EXPLOSION_MIN_SUPP             1

Global Statistics:
  attribute name  attribute value
0       NUM_ROWS                4

Attributes:
COMMENTS
YEAR

Partition: NO

Features:

     FEATURE_ID      ATTRIBUTE_NAME ATTRIBUTE_VALUE  COEFFICIENT
 0            1     COMMENTS.AFRICA            None     0.342997
 1            1       COMMENTS.AIDS            None     0.171499
 2            1       COMMENTS.LONG            None     0.342997
 3            1   COMMENTS.PLANNING            None     0.342997
...         ...                 ...             ...          ...
 24           5      COMMENTS.FOCUS            None     0.282843
 25           5     COMMENTS.SUMMIT            None     0.282843
 26           5     COMMENTS.THREAT            None     0.282843
 27           5                YEAR            2018     0.707107
```

Use the model to make predictions on test data.

```
>>> esa_mod.predict(test_dat, supplemental_cols = test_dat[:, ['ID', 'COMMENTS']])
   ID                                       COMMENTS  FEATURE_ID
0   4          NASA announces major Mars rover finding           3
1   6   NASA Mars Odyssey THEMIS image: typical crater           2
2   7                             Road blocks for Aids           5
```

```
>>> esa_mod.transform(test_dat, supplemental_cols = test_dat[:, ['ID', 'COMMENTS']],
↪topN = 2).sort_values(by = ['ID'])
    ID                                      COMMENTS  TOP_1  TOP_1_VAL   \
 0   4          NASA announces major Mars rover finding      3   0.667889
 1   6   NASA Mars Odyssey THEMIS image: typical crater      2   0.766237
 2   7                             Road blocks for Aids      5   0.759125
```

(continues on next page)

(continued from previous page)

```
   TOP_2   TOP_2_VAL
0      1    0.590565
1      5    0.616672
2      2    0.632604
```

```
>>> esa_mod.feature_compare(test_dat, compare_cols = 'COMMENTS', supplemental_cols = [
↪'ID'])
   ID_A  ID_B  SIMILARITY
0     4     6    0.946469
1     4     7    0.871994
2     6     7    0.954565
```

```
>>> esa_mod.feature_compare(test_dat, compare_cols = ['COMMENTS', 'YEAR'],␣
↪supplemental_cols = ['ID'])
   ID_A  ID_B  SIMILARITY
0     4     6    0.467644
1     4     7    0.377144
2     6     7    0.952857
```

Change parameter setting and refit model.

```
>>> new_setting = {'ESAS_VALUE_THRESHOLD': '0.01', 'ODMS_TEXT_MAX_FEATURES': '2',
↪'ESAS_TOPN_FEATURES': '2'}
>>> esa_mod.set_params(**new_setting).fit(train_dat, case_id = 'ID', ctx_settings =␣
↪ctx_settings)

Algorithm Name: Explicit Semantic Analysis

Mining Function: FEATURE_EXTRACTION

Settings:
                    setting name                 setting value
0                      ALGO_NAME  ALGO_EXPLICIT_SEMANTIC_ANALYS
1                 ESAS_MIN_ITEMS                             1
2             ESAS_TOPN_FEATURES                             2
3           ESAS_VALUE_THRESHOLD                          0.01
4                   ODMS_DETAILS                   ODMS_ENABLE
5   ODMS_MISSING_VALUE_TREATMENT       ODMS_MISSING_VALUE_AUTO
6                  ODMS_SAMPLING         ODMS_SAMPLING_DISABLE
7         ODMS_TEXT_MAX_FEATURES                             2
8        ODMS_TEXT_MIN_DOCUMENTS                             1
9          ODMS_TEXT_POLICY_NAME               DMDEMO_ESA_POLICY
10                     PREP_AUTO                            ON

Computed Settings:
              setting name setting value
0  ODMS_EXPLOSION_MIN_SUPP             1

Global Statistics:
  attribute name  attribute value
0       NUM_ROWS                4

Attributes:
COMMENTS
YEAR
```

(continues on next page)

(continued from previous page)

```
Partition: NO

Features:

   FEATURE_ID ATTRIBUTE_NAME ATTRIBUTE_VALUE  COEFFICIENT
 0          1  COMMENTS.AIDS            None     0.707107
 1          1           YEAR            2017     0.707107
 2          2  COMMENTS.MARS            None     0.707107
 3          2           YEAR            2018     0.707107
 4          3  COMMENTS.MARS            None     0.707107
 5          3           YEAR            2017     0.707107
 6          5  COMMENTS.AIDS            None     0.707107
 7          5           YEAR            2018     0.707107
```

```
>>> cur = cursor()
>>> cur.execute("Begin ctx_ddl.drop_policy('DMDEMO_ESA_POLICY'); End;")
>>> cur.close()
```

2.6 Exponential Smoothing

class oml.**esm**(*model_name=None*, *model_owner=None*, ***params*)

In-database Exponential Smoothing Model

Exponential Smoothing methods have been widely used in forecasting for over half a century. It has applications at the strategic, tactical, and operation level. For example, at a strategic level, forecasting is used for projecting return on investment, growth and the effect of innovations. At a tactical level, forecasting is used for projecting costs, inventory requirements, and customer satisfaction. At an operational level, forecasting is used for setting targets and predicting quality and conformance with standards.

In its simplest form, Exponential Smoothing is a moving average method with a single parameter which models an exponentially decreasing effect of past levels on future values. With a variety of extensions, Exponential Smoothing covers a broader class of models than competitors, such as the Box-Jenkins auto-regressive integrated moving average (ARIMA) approach. Oracle Data Mining implements Exponential Smoothing using a state of the art state space method that incorporates a single source of error (SSOE) assumption which provides theoretical and performance advantages.

Attributes

prediction : oml.DataFrame

The prediction information. It includes the following components:

- time_seq: The time sequence used as case id
- value: The real value in the training data
- prediction: The predicted value by the model
- lower: The lower bound of the predicted value
- upper: The upper bound of the predicted value

See also:

dt, *km*, *svm*

`__init__`(*model_name=None*, *model_owner=None*, ***params*)
Initializes an instance of esm object.

Parameters

model_name
[string or None (default)] The name of an existing database Decision Tree model to create an oml.esm object from. The specified database model is not dropped when the oml.esm object is deleted.

model_owner: string or None (default)
The owner name of the existing Decision Tree model The current database user by default

params
[key-value pairs or dict] Oracle Machine Learning parameter settings. Each dict element's name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 4-10 apply generally to the model. Mining Function Settings for Classification and Algorithm-specific Settings are applicable to Decision Tree model.

`export_sermodel`(*table=None*, *partition=None*)
Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

`fit`(*x*, *y*, *time_seq*, *model_name=None*)
Fits a exponential smoothing model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object, or string] Target values. If y is a single column OML object, target values specified by y must be combinable with x. If y is a string, y is the name of the column in x that specifies the target values.

time_seq: string
The name of the column in x used as time sequence to build the model. The column must be oml.Integer, oml.Float (Number based), or oml.Datetime

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.esm object is deleted. If None, a system-generated model name

will be used. The system-generated model is dropped when oml.esm object is deleted unless oml.esm object is saved into a datastore.

get_params (*params=None*, *deep=False*)
Fetches parameters of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If `params` is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

set_params (***params*)
Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

References

- Oracle R Enterprise
- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

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

Create table containing dataset iris from sklearn.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame([round(i) for i in iris.data[:,0].tolist()], columns = ['val'])
>>> y = pd.DataFrame([round(i) for i in iris.data[:,1].tolist()], columns = ['target
↪'])
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create training data and test data.

```
>>> dat = oml.sync(table = "IRIS")
>>> train_x = dat[:, 0]
>>> train_y = dat[:, 1]
```

Create ESM model object.

```
>>> esm_mod = oml.esm(ODMS_DETAILS = 'ODMS_ENABLE')
```

Fit the ESM Model according to the training data and parameter settings.

```
>>> esm_mod = esm_mod.fit(train_x, train_y, time_seq = 'val')
```

Show model details.

```
>>> esm_mod

Algorithm Name: Exponential Smoothing

Mining Function: TIME_SERIES

Target: target

Settings:
                    setting name                setting value
0                      ALGO_NAME   ALGO_EXPONENTIAL_SMOOTHING
1           EXSM_BACKCAST_OUTPUT  EXSM_BACKCAST_OUTPUT_ENABLE
2          EXSM_CONFIDENCE_LEVEL                          .95
3           EXSM_INITVL_OPTIMIZE  EXSM_INITVL_OPTIMIZE_ENABLE
4                      EXSM_NMSE                            3
5         EXSM_OPTIMIZATION_CRIT            EXSM_OPT_CRIT_LIK
6           EXSM_PREDICTION_STEP                            1
7               EXSM_SETMISSING               EXSM_MISS_AUTO
8                   ODMS_BOXCOX           ODMS_BOXCOX_ENABLE
9                  ODMS_DETAILS                  ODMS_ENABLE
10  ODMS_MISSING_VALUE_TREATMENT       ODMS_MISSING_VALUE_AUTO
11                ODMS_SAMPLING        ODMS_SAMPLING_DISABLE
12                     PREP_AUTO                           ON

Computed Settings:
  setting name setting value
0   EXSM_MODEL   EXSM_SIMPLE

Global Statistics:
```

(continues on next page)

(continued from previous page)

```
       attribute name attribute value
0   -2 LOG-LIKELIHOOD        -277.972
1                 AIC         561.944
2                AICC         562.109
3               ALPHA       0.0998743
4                AMSE        0.281354
5                 BIC         570.976
6           CONVERGED             YES
7       INITIAL LEVEL         3.16632
8                 MAE        0.377722
9                 MSE        0.271351
10           NUM_ROWS             150
11              SIGMA        0.524422
12                STD        0.524422

Attributes:

Partition: NO

Prediction:

     TIME_SEQ  VALUE  PREDICTION     LOWER     UPPER
0           4    2.0    3.121295       NaN       NaN
1           4    3.0    3.009307       NaN       NaN
2           4    3.0    3.134754       NaN       NaN
3           4    3.0    3.149705       NaN       NaN
4           4    3.0    3.166316       NaN       NaN
..        ...    ...         ...       ...       ...
146         8    3.0    3.094874       NaN       NaN
147         8    3.0    3.169066       NaN       NaN
148         8    4.0    2.994444       NaN       NaN
149         8    4.0    3.076869       NaN       NaN
150         9    NaN    3.152181  2.124333  4.180029

[151 rows x 5 columns]
```

```
>>> oml.drop('IRIS')
```

2.7 Generalized Linear Model

class oml.**glm**(*mining_function='CLASSIFICATION'*, *model_name=None*, *model_owner=None*, ***params*)

In-database Generalized Linear Models

Builds Generalized Linear Models (GLM), which include and extend the class of linear models (linear regression), to be used for classification or regression. This function exposes the corresponding Oracle Machine Learning in-database algorithm. Generalized linear models relax the restrictions on linear models, which are often violated in practice. For example, binary (yes/no or 0/1) responses do not have same variance across classes. This model uses a parametric modeling technique. Parametric models make assumptions about the distribution of the data. When the assumptions are met, parametric models can be more efficient than non-parametric models.

Attributes

coef : oml.DataFrame

The coefficients of the GLM model, one for each predictor variable. It includes the following components:

- nonreference: The target value used as nonreference
- attribute name: The attribute name
- attribute value: The attribute value
- coefficient: The estimated coefficient
- std error: The standard error
- t value: The test statistics
- p value: The statistical significance

fit_details: oml.DataFrame

The model fit details such as adjusted_r_square, error_mean_square and so on. It includes the following components:

- name: The fit detail name
- value: The fit detail value

deviance: float

Minus twice the maximized log-likelihood, up to a constant.

null_deviance: float

The deviance for the null (intercept only) model.

aic: float

Akaike information criterion.

rank: integer

The numeric rank of the fitted model.

df_residual: float

The residual degrees of freedom.

df_null: float

The residual degrees of freedom for the null model.

converged: bool

The indicator for whether the model converged.

nonreference: int or str

For logistic regression, the response values that represents success.

See also:

svm, *dt*, *km*

__init__(*mining_function='CLASSIFICATION'*, *model_name=None*, *model_owner=None*, ***params*)

Initializes an instance of glm object.

Parameters

mining_function
[‘CLASSIFICATION’ or ‘REGRESSION’, ‘CLASSIFICATION’ (default)] Type of model mining functionality

model_name
[string or None (default)] The name of an existing database Generalized Linear Model to create an oml.glm object from. The specified database model is not dropped when the oml.glm object is deleted.

model_owner: string or None (default)
The owner name of the existing Generalized Linear Model The current database user by default

params
[key-value pairs or dict] Oracle Machine Learning parameter settings. Each list element’s name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 4-10 apply generally to the model. Mining Function Settings for Classification and Algorithm-specific Settings are applicable to Generalized Linear Model model.

export_sermodel(*table=None*, *partition=None*)
Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

fit(*x*, *y*, *model_name=None*, *case_id=None*, *ctx_settings=None*)
Fits a GLM Model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object, or string] Target values. If y is a single column OML object, target values specified by y must be combinable with x. If y is a string, y is the name of the column in x that specifies the target values.

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.glm object is deleted. If None, a system-generated model name will be used. The system-generated model is dropped when oml.glm object is deleted unless oml.glm object is saved into a datastore.

case_id
[string or None (default)] The name of a column that contains unique case identifiers.

ctx_settings
[dict or None (default)] A list to specify Oracle Text attribute-specific settings. This argument is applicable to building models in Oracle Database 12.2 or later. The name of each list element refers to the text column while the list value is a scalar string specifying the attribute-specific text transformation. The valid entries in the string include TEXT, POLICY_NAME, TOKEN_TYPE, and MAX_FEATURES.

get_params (*params=None*, *deep=False*)
Fetches parameters of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If `params` is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

predict (*x*, *supplemental_cols=None*, *confint=None*, *level=None*, *proba=False*, *topN_attrs=False*)
Makes predictions on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

confint
[bool, False (default)] A logical indicator for whether to produce confidence intervals. for the predicted values.

level
[float between 0 and 1 or None (default)] A numeric value within [0, 1] to use for the confidence level.

proba
[boolean, False (default)] Returns prediction probability if `proba` is True for classification.

topN_attrs
[boolean, positive integer, False (default)] Returns the top N most influence attributes of

the predicted target value for regression if topN_attrs is not False. Returns the top N most influence attributes of the highest probability class for classification if topN_attrs is not False. N is equal to the specified positive integer or 5 if topN_attrs is True.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. For a classification model, the results include the most likely target class and optionally its probability and confidence intervals. For a linear regression model, the results consist of a column for the prediction and optionally its confidence intervals.

predict_proba (*x*, *supplemental_cols=None*, *topN=None*)
Makes predictions and returns probability for each class on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN
[positive integer or None (default)] A positive integer that restricts the returned target classes to the specified number of those that have the highest probability.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. The results include for each target class, the probability belonging to that class.

residuals (*x*, *y*)

Return the deviance residuals, which includes the following components:

- deviance: The deviance residual
- pearson: The Pearson residual
- response: The residual of the working response.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object, or string] Target values. If y is a single column OML object, target values specified by y must be combinable with x. If y is a string, y is the name of the column in x that specifies the target values.

Return: oml.DataFrame

score (*x*, *y*)
Makes predictions on new data, returns the mean accuracy for classifications or the coefficient of determination R^2 of the prediction for regressions.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object] Target values.

Returns

score
[float] Mean accuracy for classifications or the coefficient of determination R^2 of the prediction for regressions.

set_params(***params*)
Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

References

- Oracle R Enterprise
- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

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

Create table containing dataset iris from sklearn.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width', 'Petal_
↪Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:'virginica'}
↪[x], iris.target)), columns = ['Species'])
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create training data and test data.

```
>>> dat = oml.sync(table = "IRIS").split()
>>> train_x = dat[0].drop('Petal_Width')
>>> train_y = dat[0]['Petal_Width']
>>> test_dat = dat[1]
```

User specified settings.

```
>>> setting = {'GLMS_SOLVER': 'dbms_data_mining.GLMS_SOLVER_QR'}
```

Create GLM model object.

```
>>> glm_mod = oml.glm("regression", **setting)
```

Fit the GLM Model according to the training data and parameter settings.

```
>>> glm_mod = glm_mod.fit(train_x, train_y)
```

Show model details.

```
>>> glm_mod

Algorithm Name: Generalized Linear Model

Mining Function: REGRESSION

Target: Petal_Width

Settings:
                   setting name                    setting value
0                     ALGO_NAME  ALGO_GENERALIZED_LINEAR_MODEL
1               GLMS_CONF_LEVEL                            .95
2           GLMS_FTR_GENERATION     GLMS_FTR_GENERATION_DISABLE
3            GLMS_FTR_SELECTION      GLMS_FTR_SELECTION_DISABLE
4            GLMS_LINK_FUNCTION              GLMS_IDENTITY_LINK
5                   GLMS_SOLVER                  GLMS_SOLVER_QR
6                  ODMS_DETAILS                     ODMS_ENABLE
7  ODMS_MISSING_VALUE_TREATMENT         ODMS_MISSING_VALUE_AUTO
8                 ODMS_SAMPLING           ODMS_SAMPLING_DISABLE
9                     PREP_AUTO                              ON

Computed Settings:
              setting name            setting value
0      GLMS_CONV_TOLERANCE  .0000050000000000000004
1      GLMS_NUM_ITERATIONS                       30
2    GLMS_RIDGE_REGRESSION    GLMS_RIDGE_REG_DISABLE
3  ODMS_EXPLOSION_MIN_SUPP                        1

Global Statistics:
             attribute name attribute value
0         ADJUSTED_R_SQUARE        0.949634
1                       AIC        -363.888
2                 COEFF_VAR         14.6284
3                 CONVERGED             YES
4        CORRECTED_TOTAL_DF             103
5          CORRECTED_TOT_SS         58.4565
6            DEPENDENT_MEAN         1.15577
7                  ERROR_DF              98
8         ERROR_MEAN_SQUARE       0.0285848
9         ERROR_SUM_SQUARES         2.80131
10                  F_VALUE         389.405
11                    GMSEP       0.0303473
12               HOCKING_SP     0.000294689
13                      J_P       0.0302339
```

(continues on next page)

(continued from previous page)

```
14                 MODEL_DF               5
15          MODEL_F_P_VALUE               0
16        MODEL_MEAN_SQUARE          11.131
17        MODEL_SUM_SQUARES         55.6552
18               NUM_PARAMS               6
19                 NUM_ROWS             104
20          RANK_DEFICIENCY               0
21             ROOT_MEAN_SQ         0.16907
22                     R_SQ        0.952079
23                     SBIC        -348.021
24  VALID_COVARIANCE_MATRIX             YES

Attributes:
Petal_Length
Sepal_Length
Sepal_Width
Species

Partition: NO

Coefficients:

  attribute name attribute value  coefficient  std error   t value  \
0    (Intercept)            None    -0.600603   0.223742 -2.684354
1   Petal_Length            None     0.239775   0.058393  4.106194
2   Sepal_Length            None    -0.078338   0.055799 -1.403943
3    Sepal_Width            None     0.253996   0.055936  4.540857
4        Species      versicolor     0.652420   0.141010  4.626767
5        Species       virginica     1.010438   0.191501  5.276426

        p value significance code
0  8.533328e-03                **
1  8.339409e-05               ***
2  1.634970e-01
3  1.595696e-05               ***
4  1.137748e-05               ***
5  7.897414e-07               ***

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 '
Fit Details:


                       name         value
 0         ADJUSTED_R_SQUARE  9.496338e-01
 1                       AIC -3.638876e+02
 2                 COEFF_VAR  1.462838e+01
 3        CORRECTED_TOTAL_DF  1.030000e+02
...                     ...           ...
 21             ROOT_MEAN_SQ  1.690704e-01
 22                     R_SQ  9.520788e-01
 23                     SBIC -3.480213e+02
 24  VALID_COVARIANCE_MATRIX  1.000000e+00

Rank:

 6
```

(continues on next page)

(continued from previous page)

```
Deviance:

 2.801309

AIC:

 -364

Null Deviance:

 58.456538

DF Residual:

 98.0

DF Null:

 103.0

Converged:

 True
```

Use the model to make predictions on test data.

```
>>> glm_mod.predict(test_dat.drop('Petal_Width'), supplemental_cols = test_dat[:, [
↪'Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Species']])
     Sepal_Length  Sepal_Width  Petal_Length    Species  PREDICTION
 0            4.9          3.0           1.4     setosa    0.113215
 1            4.9          3.1           1.5     setosa    0.162592
 2            4.8          3.4           1.6     setosa    0.270602
 3            5.8          4.0           1.2     setosa    0.248752
...           ...          ...           ...        ...         ...
 42           6.7          3.3           5.7  virginica    2.089876
 43           6.7          3.0           5.2  virginica    1.893790
 44           6.5          3.0           5.2  virginica    1.909457
 45           5.9          3.0           5.1  virginica    1.932483
```

```
>>> glm_mod.predict(test_dat.drop('Petal_Width'), supplemental_cols = test_dat[:, [
↪'Sepal_Length', 'Sepal_Width', 'Species']], proba = True)
     Sepal_Length  Sepal_Width    Species  PREDICTION
 0            4.9          3.0     setosa    0.113215
 1            4.9          3.1     setosa    0.162592
 2            4.8          3.4     setosa    0.270602
 3            5.8          4.0     setosa    0.248752
...           ...          ...        ...         ...
 42           6.7          3.3  virginica    2.089876
 43           6.7          3.0  virginica    1.893790
 44           6.5          3.0  virginica    1.909457
 45           5.9          3.0  virginica    1.932483
```

```
>>> glm_mod.score(test_dat.drop('Petal_Width'), test_dat[:, ['Petal_Width']])
 0.951252
```

Change parameter setting and refit model.

```
>>> new_setting = {'GLMS_SOLVER': 'GLMS_SOLVER_SGD'}
>>> glm_mod.set_params(**new_setting).fit(train_x, train_y)

Algorithm Name: Generalized Linear Model

Mining Function: REGRESSION

Target: Petal_Width

Settings:
                   setting name                    setting value
0                     ALGO_NAME  ALGO_GENERALIZED_LINEAR_MODEL
1               GLMS_CONF_LEVEL                            .95
2           GLMS_FTR_GENERATION    GLMS_FTR_GENERATION_DISABLE
3            GLMS_FTR_SELECTION     GLMS_FTR_SELECTION_DISABLE
4            GLMS_LINK_FUNCTION             GLMS_IDENTITY_LINK
5                   GLMS_SOLVER                GLMS_SOLVER_SGD
6                  ODMS_DETAILS                    ODMS_ENABLE
7   ODMS_MISSING_VALUE_TREATMENT        ODMS_MISSING_VALUE_AUTO
8                 ODMS_SAMPLING          ODMS_SAMPLING_DISABLE
9                     PREP_AUTO                             ON

Computed Settings:
              setting name          setting value
0          GLMS_BATCH_ROWS                   2000
1      GLMS_CONV_TOLERANCE                  .0001
2      GLMS_NUM_ITERATIONS                    500
3    GLMS_RIDGE_REGRESSION  GLMS_RIDGE_REG_ENABLE
4         GLMS_RIDGE_VALUE                    .01
5   ODMS_EXPLOSION_MIN_SUPP                     1

Global Statistics:
             attribute name attribute value
0         ADJUSTED_R_SQUARE         0.94175
1                       AIC        -348.764
2                 COEFF_VAR         15.7316
3                 CONVERGED              NO
4        CORRECTED_TOTAL_DF             103
5          CORRECTED_TOT_SS         58.4565
6            DEPENDENT_MEAN         1.15577
7                  ERROR_DF              98
8         ERROR_MEAN_SQUARE       0.0330591
9         ERROR_SUM_SQUARES         3.23979
10                  F_VALUE         324.347
11                    GMSEP       0.0350974
12               HOCKING_SP     0.000340815
13                      J_P       0.0349663
14                 MODEL_DF               5
15           MODEL_F_P_VALUE               0
16        MODEL_MEAN_SQUARE         10.7226
17        MODEL_SUM_SQUARES          53.613
18               NUM_PARAMS               6
19                 NUM_ROWS             104
20          RANK_DEFICIENCY               0
21             ROOT_MEAN_SQ        0.181821
22                     R_SQ        0.944578
23                     SBIC        -332.898
```

(continues on next page)

(continued from previous page)

```
24  VALID_COVARIANCE_MATRIX              NO

Attributes:
Petal_Length
Sepal_Length
Sepal_Width
Species

Partition: NO

Coefficients:

  attribute name attribute value  coefficient std error t value p value  \
0    (Intercept)            None    -0.338046      None    None    None
1   Petal_Length            None     0.378658      None    None    None
2   Sepal_Length            None    -0.084440      None    None    None
3    Sepal_Width            None     0.137150      None    None    None
4        Species      versicolor     0.151916      None    None    None
5        Species       virginica     0.337535      None    None    None

  significance code
0
1
2
3
4
5

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 '

Fit Details:

                        name         value
 0         ADJUSTED_R_SQUARE  9.417502e-01
 1                       AIC -3.487639e+02
 2                 COEFF_VAR  1.573164e+01
 3        CORRECTED_TOTAL_DF  1.030000e+02
...                      ...           ...
 21             ROOT_MEAN_SQ  1.818215e-01
 22                     R_SQ  9.445778e-01
 23                     SBIC -3.328975e+02
 24  VALID_COVARIANCE_MATRIX  0.000000e+00

Rank:

 6

Deviance:

 3.239787

AIC:

 -349

Null Deviance:
```

(continues on next page)

(continued from previous page)

```
 58.456538

DF Residual:

 98.0

DF Null:

 103.0

Converged:

 False
```

```
>>> oml.drop('IRIS')
```

2.8 K-Means

class oml.**km**(*n_clusters=None*, *model_name=None*, *model_owner=None*, ***params*)

In-database k-means Model

Builds a K-Means (KM) Model that uses a distance-based clustering algorithm to partition data into a specified number of clusters. This function exposes the corresponding Oracle Machine Learning in-database algorithm. Distance-based algorithms rely on a distance function to measure the similarity between cases. Cases are assigned to the nearest cluster according to the distance function used.

Attributes

clusters : oml.DataFrame

The general per-cluster information. It includes the following components:

- cluster_id: The ID of a cluster in the model
- row_cnt: The number of rows used in the build
- parent_cluster_id: The ID of the parent
- tree_level: The number of splits from the root
- dispersion: The measure of the quality of the cluster, and computationally, the sum of square errors

taxonomy: oml.DataFrame

The parent/child cluster relationship. It includes the following components:

- parent_cluster_id: The ID of the parent cluster
- child_cluster_id: The ID of the child cluster

centroids: oml.DataFrame

Per cluster-attribute center (centroid) information. It includes the following components:

- cluster_id: The ID of a cluster in the model
- attribute_name: The attribute name

- mean: The average value of a numeric attribute
- mode_value: The most frequent value of a categorical attribute
- variance: The variance of a numeric attribute

leaf_cluster_counts: pandas.DataFrame

Leaf clusters with support. It includes the following components:

- cluster_id: The ID of a leaf cluster in the model
- cnt: The number of records in a leaf cluster

cluster_hists: oml.DataFrame

Cluster histogram information. It includes the following components:

- cluster.id: The ID of a cluster in the model
- variable: The attribute name
- bin.id: The ID of a bin
- lower.bound: The numeric lower bin boundary
- upper.bound: The numeric upper bin boundary
- label: The label of the cluster
- count: The histogram count

rules: oml.DataFrame

Conditions for a case to be assigned with some probability to a cluster. It includes the following components:

- cluster.id: The ID of a cluster in the model
- rhs.support: The record count
- rhs.conf: The record confidence
- lhr.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

See also:

dt, *svd*, *svm*

__init__(*n_clusters=None*, *model_name=None*, *model_owner=None*, ***params*)

Initializes an instance of km object.

Parameters

n_clusters

[positive integer, default None] Number of clusters. If n_clusters is None, the number of clusters will be determined either by current setting parameters or automatically by the internal algorithm.

model_name
[string or None (default)] The name of an existing database K-Means model to create an oml.km object from. The specified database model is not dropped when the oml.km object is deleted.

model_owner: string or None (default)
The owner name of the existing K-Means model The current database user by default

params
[key-value pairs or dict] Oracle Machine Learning parameter settings. Each list element's name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 5-5 apply generally to the model. Mining Function Settings for Clustering and Algorithm-specific Settings are applicable to K-Means model.

export_sermodel (*table=None*, *partition=None*)
Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

fit (*x*, *model_name=None*, *case_id=None*, *ctx_settings=None*)
Fits a K-Means Model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model.

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.km object is deleted. If None, a system-generated model name will be used. The system-generated model is dropped when oml.km object is deleted unless oml.km object is saved into a datastore.

case_id
[string or None (default)] The name of a column that contains unique case identifiers.

ctx_settings
[dict or None(default)] A list to specify Oracle Text attribute-specific settings. This argument is applicable to building models in Oracle Database 12.2 or later. The name of each list element refers to the text column while the list value is a scalar string specifying the attribute-specific text transformation. The valid entries in the string include TEXT, POLICY_NAME, TOKEN_TYPE, and MAX_FEATURES.

get_params (*params=None*, *deep=False*)
Fetches parameters of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If `params` is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

predict (*x*, *supplemental_cols=None*, *topN_attrs=False*, *proba=False*)
Makes predictions on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN_attrs
[boolean, positive integer, False (default)] Returns the top N most likely clusters with the corresponding weights. N is equal to the specified positive integer or 5 if topN_attrs is True.

proba
[boolean, False (default)] Returns prediction probability if `proba` is True.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results.

predict_proba (*x*, *supplemental_cols=None*, *topN=None*)
Makes predictions and returns probability for each cluster on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN
[positive integer] A positive integer that restricts the returned clusters to the specified number of those that have the highest probability.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. The results include for each cluster, the probability belonging to that cluster.

score(*x*)
Calculates the score value based on the input data `x`.

Parameters

x [an OML object] A new data set used to calculate score value.

Returns

pred
[float] Score values, that is, opposite of the value of `x` on the K-means objective.

set_params(***params*)
Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

transform(*x*)
Transforms `x` to a cluster-distance space.

Parameters

x [an OML object] Attribute values used by the model.

Returns

pred
[oml.DataFrame] Contains the distance to each cluster.

References

- Oracle R Enterprise

- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

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

Create table containing dataset iris from sklearn.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width', 'Petal_
↪Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:'virginica'}
↪[x], iris.target)), columns = ['Species'])
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create training data and test data.

```
>>> dat = oml.sync(table = "IRIS").split()
>>> train_dat = dat[0]
>>> test_dat = dat[1]
```

User specified settings.

```
>>> setting = {'kmns_iterations': 20}
```

Create KM model object and fit.

```
>>> km_mod = oml.km(n_clusters = 3, **setting).fit(train_dat)
```

Show model details.

```
>>> km_mod

Algorithm Name: K-Means

Mining Function: CLUSTERING

Settings:
                    setting name            setting value
0                      ALGO_NAME              ALGO_KMEANS
1              CLUS_NUM_CLUSTERS                        3
2            KMNS_CONV_TOLERANCE                     .001
3                   KMNS_DETAILS   KMNS_DETAILS_HIERARCHY
4                  KMNS_DISTANCE           KMNS_EUCLIDEAN
5                KMNS_ITERATIONS                       20
6        KMNS_MIN_PCT_ATTR_SUPPORT                      .1
7                  KMNS_NUM_BINS                       11
8               KMNS_RANDOM_SEED                        0
9           KMNS_SPLIT_CRITERION            KMNS_VARIANCE
10                KMNS_WINSORIZE   KMNS_WINSORIZE_DISABLE
11                  ODMS_DETAILS              ODMS_ENABLE
12  ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
13                 ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
```

(continues on next page)

(continued from previous page)

```
14                      PREP_AUTO                         ON

Computed Settings:
               setting name setting value
0   ODMS_EXPLOSION_MIN_SUPP             1

Global Statistics:
   attribute name attribute value
0       CONVERGED             YES
1  NUM_ITERATIONS               4
2        NUM_ROWS             104

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Clusters:

   CLUSTER_ID  ROW_CNT  PARENT_CLUSTER_ID  TREE_LEVEL  DISPERSION
0           1      104                NaN           1    0.986153
1           2       68                1.0           2    1.102147
2           3       36                1.0           2    0.767052
3           4       37                2.0           3    1.015669
4           5       31                2.0           3    1.205363

Taxonomy:

   PARENT_CLUSTER_ID  CHILD_CLUSTER_ID
0                  1               2.0
1                  1               3.0
2                  2               4.0
3                  2               5.0
4                  3               NaN
5                  4               NaN
6                  5               NaN

Leaf Cluster Counts:

   CLUSTER_ID  CNT
0           3   36
1           4   37
2           5   31
```

Use the model to make predictions on test data.

```
>>> km_mod.predict(test_dat, supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_
↪Width', 'Petal_Length', 'Species']])
     Sepal_Length  Sepal_Width  Petal_Length     Species  CLUSTER_ID
 0            4.9          3.0           1.4      setosa           3
 1            4.9          3.1           1.5      setosa           3
 2            4.8          3.4           1.6      setosa           3
```

(continues on next page)

(continued from previous page)

```
 3           5.8          4.0          1.2     setosa           3
...          ...          ...          ...        ...         ...
 42          6.7          3.3          5.7  virginica           5
 43          6.7          3.0          5.2  virginica           5
 44          6.5          3.0          5.2  virginica           5
 45          5.9          3.0          5.1  virginica           5
```

```
>>> km_mod.predict_proba(test_dat, supplemental_cols = test_dat[:, ['Species']]).sort_
↪values(by = ['Species', 'PROBABILITY_OF_3'])
       Species  PROBABILITY_OF_3  PROBABILITY_OF_4  PROBABILITY_OF_5
 0      setosa          0.791267          0.208494          0.000240
 1      setosa          0.971498          0.028350          0.000152
 2      setosa          0.981020          0.018499          0.000481
 3      setosa          0.981907          0.017989          0.000104
...        ...               ...               ...               ...
 42  virginica          0.000655          0.316671          0.682674
 43  virginica          0.001036          0.413744          0.585220
 44  virginica          0.001036          0.413744          0.585220
 45  virginica          0.002452          0.305021          0.692527
```

```
>>> km_mod.transform(test_dat)
     CLUSTER_DISTANCE
 0           1.050234
 1           0.859817
 2           0.321065
 3           1.427080
...               ...
 42          0.837757
 43          0.479313
 44          0.448562
 45          1.123587
```

```
>>> km_mod.score(test_dat)
 -47.487712
```

Change parameter setting and refit model.

```
>>> new_setting = {'KMNS_RANDOM_SEED':'25'}
>>> km_mod.set_params(**new_setting).fit(train_dat)

Algorithm Name: K-Means

Mining Function: CLUSTERING

Settings:
                   setting name            setting value
0                     ALGO_NAME              ALGO_KMEANS
1             CLUS_NUM_CLUSTERS                        3
2           KMNS_CONV_TOLERANCE                     .001
3                  KMNS_DETAILS   KMNS_DETAILS_HIERARCHY
4                 KMNS_DISTANCE           KMNS_EUCLIDEAN
5               KMNS_ITERATIONS                       20
6       KMNS_MIN_PCT_ATTR_SUPPORT                       .1
7                 KMNS_NUM_BINS                       11
8              KMNS_RANDOM_SEED                       25
```

(continues on next page)

(continued from previous page)

```
9           KMNS_SPLIT_CRITERION            KMNS_VARIANCE
10                KMNS_WINSORIZE   KMNS_WINSORIZE_DISABLE
11                  ODMS_DETAILS              ODMS_ENABLE
12  ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
13                 ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
14                     PREP_AUTO                       ON

Computed Settings:
              setting name setting value
0  ODMS_EXPLOSION_MIN_SUPP             1

Global Statistics:
   attribute name attribute value
0       CONVERGED             YES
1  NUM_ITERATIONS               3
2        NUM_ROWS             104

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Clusters:

   CLUSTER_ID  ROW_CNT  PARENT_CLUSTER_ID  TREE_LEVEL  DISPERSION
0           1      104                NaN           1    0.986153
1           2       36                1.0           2    0.767052
2           3       68                1.0           2    1.102147
3           4       31                3.0           3    1.205363
4           5       37                3.0           3    1.015669

Taxonomy:

   PARENT_CLUSTER_ID  CHILD_CLUSTER_ID
0                  1               2.0
1                  1               3.0
2                  2               NaN
3                  3               4.0
4                  3               5.0
5                  4               NaN
6                  5               NaN

Leaf Cluster Counts:

   CLUSTER_ID  CNT
0           2   36
1           4   31
2           5   37
```

```
>>> oml.drop('IRIS')
```

2.9 Naive Bayes

class `oml.nb`(*model_name=None*, *model_owner=None*, ***params*)

In-database Naive Bayes Model

Builds a Naive Bayes Model that uses conditional probabilities to predict a target variable (numeric or categorical column). Naive Bayes looks at the historical data and calculates conditional probabilities for the target values by observing the frequency of attribute values and of combinations of attribute values. Naive Bayes assumes that each predictor is conditionally independent of the others. (Bayes' Theorem requires that the predictors be independent.)

Attributes

priors : oml.DataFrame

An optional named numerical vector that specifies the priors for the target classes. It includes the following components:

- target_name: The name of the target column
- target_value: The target value
- prior_probability: The prior probability for a given target_value
- count: The number of rows for a given target_value

conditionals : oml.DataFrame

Conditional probabilities for each predictor variable. It includes the following components:

- target_name: The name of the target column
- target_value: The target value
- attribute_name: The column name
- attribute_subname: The nested column subname.
- attribute_value: The mining attribute value
- conditional_probability: The conditional probability of a mining attribute for a given target
- count: The number of rows for a given mining attribute and a given target

See also:

glm, *svm*, *km*

`__init__`(*model_name=None*, *model_owner=None*, ***params*)

Initializes an instance of nb object.

Parameters

model_name

[string or None (default)] The name of an existing database Naive Bayes model to create an oml.nb object from. The specified database model is not dropped when the oml.nb object is deleted.

model_owner: string or None (default)

The owner name of the existing Naive Bayes model The current database user by default

params
[key-value pairs or dict] Oracle Machine Learning parameter settings. Each dict element's name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 4-10 apply generally to the model. Mining Function Settings for Classification and Algorithm-specific Settings are applicable to Naive Bayes model.

export_sermodel(*table=None*, *partition=None*)
Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

fit(*x*, *y*, *model_name=None*, *priors=None*, *case_id=None*)
Fits a Naive Bayes Model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object, or string] Target values. If y is a single column OML object, target values specified by y must be combinable with x. If y is a string, y is the name of the column in x that specifies the target values.

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.nb object is deleted. If None, a system-generated model name will be used. The system-generated model is dropped when oml.nb object is deleted unless oml.nb object is saved into a datastore.

priors
[OML DataFrame or dict or list of ints or floats or None (default)] The priors represent the overall distribution of the target in the population. By default, the priors are computed from the sample. If the sample is known to be a distortion of the population target distribution, then the user can override the default by providing a priors table as a setting for model creation. For OML DataFrame input, the first value represents the target value. The second value represents the prior probability. For dictionary type input, the key represents the target value. The value represents the prior probability. For list type input, the first value represents target value. The second value represents the prior probability. See Oracle Data Mining Concepts Guide for more details.

case_id
[string or None (default)] The column name used as case id for building the model.

get_params (*params=None*, *deep=False*)
Fetches parameters of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If `params` is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

predict (*x*, *supplemental_cols=None*, *proba=False*, *topN_attrs=False*)
Makes predictions on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

proba
[boolean, False (default)] Returns prediction probability if `proba` is True.

topN_attrs
[boolean, positive integer, False (default)] Returns the top N most influence attributes of the predicted target value for regression if topN_attrs is not False. Returns the top N most influence attributes of the highest probability class for classification if topN_attrs is not False. N is equal to the specified positive integer or 5 if topN_attrs is True.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. The results include the most likely target class.

predict_proba (*x*, *supplemental_cols=None*, *topN=None*)
Makes predictions and returns probability for each class on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

TopN
[positive integer or None (default)] A positive integer that restricts the returned target classes to the specified number of those that have the highest probability.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. The results include for each target class, the probability belonging to that class.

score (*x*, *y*)
Makes predictions on new data and returns the mean accuracy.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object] Target values.

Returns

score
[float] Mean accuracy for classifications.

set_params (***params*)
Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

References

- Oracle R Enterprise
- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

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

Create table containing dataset iris from sklearn.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width', 'Petal_
↪Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:'virginica'}
↪[x], iris.target)), columns = ['Species'])
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create training data and test data.

```
>>> dat = oml.sync(table = "IRIS").split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
```

User specified settings.

```
>>> setting = {'CLAS_WEIGHTS_BALANCED': 'ON'}
```

Create NB model object.

```
>>> nb_mod = oml.nb(**setting)
```

Fit the NB Model according to the training data and parameter settings.

```
>>> nb_mod = nb_mod.fit(train_x, train_y)
```

Show model details.

```
>>> nb_mod

Algorithm Name: Naive Bayes

Mining Function: CLASSIFICATION

Target: Species

Settings:
                   setting name            setting value
0                     ALGO_NAME         ALGO_NAIVE_BAYES
1         CLAS_WEIGHTS_BALANCED                       ON
2       NABS_PAIRWISE_THRESHOLD                        0
3      NABS_SINGLETON_THRESHOLD                        0
4                  ODMS_DETAILS              ODMS_ENABLE
5   ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
6                 ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
7                     PREP_AUTO                       ON

Global Statistics:
  attribute name  attribute value
0       NUM_ROWS              104

```

(continues on next page)

(continued from previous page)

```
Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

Priors:

   TARGET_NAME TARGET_VALUE  PRIOR_PROBABILITY  COUNT
 0     Species       setosa           0.333333     36
 1     Species   versicolor           0.333333     35
 2     Species    virginica           0.333333     33

Conditionals:

     TARGET_NAME TARGET_VALUE ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE  \
 0       Species       setosa   Petal_Length              None       ( ; 1.05]
 1       Species       setosa   Petal_Length              None     (1.05; 1.2]
 2       Species       setosa   Petal_Length              None     (1.2; 1.35]
 3       Species       setosa   Petal_Length              None    (1.35; 1.45]
 ...         ...          ...            ...               ...             ...
 152     Species    virginica    Sepal_Width              None    (3.25; 3.35]
 153     Species    virginica    Sepal_Width              None    (3.35; 3.45]
 154     Species    virginica    Sepal_Width              None    (3.55; 3.65]
 155     Species    virginica    Sepal_Width              None    (3.75; 3.85]

      CONDITIONAL_PROBABILITY  COUNT
 0                   0.027778      1
 1                   0.027778      1
 2                   0.083333      3
 3                   0.277778     10
 ...                      ...    ...
 152                 0.030303      1
 153                 0.060606      2
 154                 0.030303      1
 155                 0.060606      2

[156 rows x 7 columns]
```

Create a priors table in the Oracle Database.

```
>>> priors = {'setosa': 0.2, 'versicolor': 0.3, 'virginica': 0.5}
>>> priors = oml.create(pd.DataFrame(list(priors.items()), columns = ['TARGET_VALUE',
↪'PRIOR_PROBABILITY']), 'NB_PRIOR_PROBABILITY_DEMO')
```

Change parameter setting and refit model with a user-defined prior table.

```
>>> new_setting = {'CLAS_WEIGHTS_BALANCED': 'OFF'}
>>> nb_mod = nb_mod.set_params(**new_setting).fit(train_x, train_y, priors = priors)
>>> nb_mod

Algorithm Name: Naive Bayes

Mining Function: CLASSIFICATION
```

(continues on next page)

(continued from previous page)

```
Target: Species

Settings:
                   setting name                            setting value
0                     ALGO_NAME                         ALGO_NAIVE_BAYES
1        CLAS_PRIORS_TABLE_NAME  "PYQUSER"."NB_PRIOR_PROBABILITY_DEMO"
2         CLAS_WEIGHTS_BALANCED                                      OFF
3       NABS_PAIRWISE_THRESHOLD                                        0
4      NABS_SINGLETON_THRESHOLD                                        0
5                  ODMS_DETAILS                              ODMS_ENABLE
6   ODMS_MISSING_VALUE_TREATMENT                 ODMS_MISSING_VALUE_AUTO
7                 ODMS_SAMPLING                    ODMS_SAMPLING_DISABLE
8                     PREP_AUTO                                       ON

Global Statistics:
  attribute name  attribute value
0       NUM_ROWS              104

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

Priors:

   TARGET_NAME TARGET_VALUE  PRIOR_PROBABILITY  COUNT
 0     Species       setosa                0.2     36
 1     Species   versicolor                0.3     35
 2     Species    virginica                0.5     33

 Conditionals:

     TARGET_NAME TARGET_VALUE ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE  \
 0       Species       setosa   Petal_Length              None       ( ; 1.05]
 1       Species       setosa   Petal_Length              None     (1.05; 1.2]
 2       Species       setosa   Petal_Length              None     (1.2; 1.35]
 3       Species       setosa   Petal_Length              None    (1.35; 1.45]
 ...         ...          ...            ...               ...             ...
 152     Species    virginica    Sepal_Width              None    (3.25; 3.35]
 153     Species    virginica    Sepal_Width              None    (3.35; 3.45]
 154     Species    virginica    Sepal_Width              None    (3.55; 3.65]
 155     Species    virginica    Sepal_Width              None    (3.75; 3.85]

      CONDITIONAL_PROBABILITY  COUNT
 0                   0.027778      1
 1                   0.027778      1
 2                   0.083333      3
 3                   0.277778     10
 ...                      ...    ...
 152                 0.030303      1
 153                 0.060606      2
 154                 0.030303      1
 155                 0.060606      2
```

(continues on next page)

(continued from previous page)

```
[156 rows x 7 columns]
```

Use the model to make predictions on test data.

```
>>> nb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Petal_Length', 'Species']])
     Sepal_Length  Sepal_Width  Petal_Length     Species  PREDICTION
 0            4.9          3.0           1.4      setosa      setosa
 1            4.9          3.1           1.5      setosa      setosa
 2            4.8          3.4           1.6      setosa      setosa
 3            5.8          4.0           1.2      setosa      setosa
...           ...          ...           ...         ...         ...
 42           6.7          3.3           5.7   virginica   virginica
 43           6.7          3.0           5.2   virginica   virginica
 44           6.5          3.0           5.2   virginica   virginica
 45           5.9          3.0           5.1   virginica   virginica
```

```
>>> nb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Species']], proba = True)
     Sepal_Length  Sepal_Width     Species  PREDICTION  PROBABILITY
 0            4.9          3.0      setosa      setosa     1.000000
 1            4.9          3.1      setosa      setosa     1.000000
 2            4.8          3.4      setosa      setosa     1.000000
 3            5.8          4.0      setosa      setosa     1.000000
...           ...          ...         ...         ...          ...
 42           6.7          3.3   virginica   virginica     1.000000
 43           6.7          3.0   virginica   virginica     0.953848
 44           6.5          3.0   virginica   virginica     1.000000
 45           5.9          3.0   virginica   virginica     0.932334
```

```
>>> nb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Petal_Length', 'Species']], topN_attrs = 2).sort_values(by␣
↪= ['Species', 'Sepal_Length', 'Sepal_Width', 'Petal_Length'])
    Sepal_Length  Sepal_Width  Petal_Length     Species  PREDICTION   \
0            4.4          3.0           1.3      setosa      setosa
1            4.4          3.2           1.3      setosa      setosa
2            4.5          2.3           1.3      setosa      setosa
3            4.8          3.4           1.6      setosa      setosa
4            4.9          3.0           1.4      setosa      setosa
5            4.9          3.1           1.5      setosa      setosa
6            5.0          3.2           1.2      setosa      setosa
7            5.0          3.3           1.4      setosa      setosa
8            5.0          3.4           1.6      setosa      setosa
9            5.1          3.5           1.4      setosa      setosa
10           5.2          3.5           1.5      setosa      setosa
11           5.4          3.4           1.5      setosa      setosa
12           5.5          3.5           1.3      setosa      setosa
13           5.8          4.0           1.2      setosa      setosa
14           5.5          2.3           4.0  versicolor  versicolor
15           5.5          2.4           3.8  versicolor  versicolor
16           5.5          2.5           4.0  versicolor  versicolor
17           5.6          2.7           4.2  versicolor  versicolor
18           5.6          3.0           4.5  versicolor  versicolor
19           5.9          3.0           4.2  versicolor  versicolor
20           5.9          3.2           4.8  versicolor   virginica
21           6.1          3.0           4.6  versicolor  versicolor
```

(continues on next page)

(continued from previous page)

```
22          6.3         3.3          4.7  versicolor  versicolor
23          6.4         2.9          4.3  versicolor  versicolor
24          6.5         2.8          4.6  versicolor  versicolor
25          6.6         2.9          4.6  versicolor  versicolor
26          6.7         3.1          4.4  versicolor  versicolor
27          6.9         3.1          4.9  versicolor   virginica
28          7.0         3.2          4.7  versicolor   virginica
29          5.7         2.5          5.0   virginica   virginica
30          5.8         2.7          5.1   virginica   virginica
31          5.8         2.7          5.1   virginica   virginica
32          5.8         2.8          5.1   virginica   virginica
33          5.9         3.0          5.1   virginica   virginica
34          6.3         2.7          4.9   virginica   virginica
35          6.4         2.8          5.6   virginica   virginica
36          6.5         3.0          5.2   virginica   virginica
37          6.5         3.0          5.5   virginica   virginica
38          6.5         3.0          5.8   virginica   virginica
39          6.5         3.2          5.1   virginica   virginica
40          6.7         2.5          5.8   virginica   virginica
41          6.7         3.0          5.2   virginica   virginica
42          6.7         3.1          5.6   virginica   virginica
43          6.7         3.3          5.7   virginica   virginica
44          6.7         3.3          5.7   virginica   virginica
45          6.9         3.1          5.4   virginica   virginica

          NAME_1 VALUE_1 WEIGHT_1         NAME_2 VALUE_2 WEIGHT_2
0    Petal_Width      .2    1.401  Petal_Length     1.3      .947
1    Petal_Width      .2    1.401  Petal_Length     1.3      .947
2    Petal_Width      .3    1.117  Petal_Length     1.3      .947
3    Petal_Width      .2    1.401  Petal_Length     1.6     1.117
4    Petal_Width      .2    1.401  Petal_Length     1.4     1.291
5   Petal_Length     1.5    1.291   Petal_Width      .1     1.047
6    Petal_Width      .2    1.401  Sepal_Length       5      .795
7    Petal_Width      .2    1.401  Petal_Length     1.4     1.291
8    Petal_Width      .4    1.117  Petal_Length     1.6     1.117
9   Petal_Length     1.4    1.291   Petal_Width      .3     1.117
10   Petal_Width      .2    1.401  Petal_Length     1.5     1.291
11  Petal_Length     1.5    1.291   Petal_Width      .4     1.117
12   Petal_Width      .2    1.401   Sepal_Width     3.5      .947
13   Petal_Width      .2    1.401  Sepal_Length     5.8     1.053
14   Petal_Width     1.3    1.282  Petal_Length       4      .987
15  Petal_Length     3.8     .987   Sepal_Width     2.4      .835
16   Petal_Width     1.3    1.282  Petal_Length       4      .987
17   Petal_Width     1.3    1.282   Sepal_Width     2.7      .893
18  Petal_Length     4.5    1.057   Petal_Width     1.5      .919
19  Sepal_Length     5.9     .987   Petal_Width     1.5      .919
20   Petal_Width     1.8    1.332  Sepal_Length     5.9      .931
21  Petal_Length     4.6     .987   Petal_Width     1.4      .893
22  Petal_Length     4.7     .987   Sepal_Width     3.3       .75
23   Petal_Width     1.3    1.282   Sepal_Width     2.9      .697
24  Petal_Length     4.6     .987   Petal_Width     1.5      .919
25   Petal_Width     1.3    1.282  Petal_Length     4.6      .987
26  Petal_Length     4.4     .987   Petal_Width     1.4      .893
27  Sepal_Length     6.9     .917  Petal_Length     4.9      .654
28  Petal_Length     4.7     .931  Sepal_Length       7      .917
29  Sepal_Length     5.7    1.668   Petal_Width       2     1.069
30   Petal_Width     1.9    1.069  Sepal_Length     5.8      .931
```

(continues on next page)

(continued from previous page)

```
31   Petal_Width     1.9    1.069  Sepal_Length     5.8      .931
32  Sepal_Length     5.8     .931   Sepal_Width     2.8      .654
33   Petal_Width     1.8    1.332  Sepal_Length     5.9      .931
34   Petal_Width     1.8    1.332  Sepal_Length     6.3      .917
35  Petal_Length     5.6    1.169  Sepal_Length     6.4      .976
36   Petal_Width       2    1.069  Sepal_Length     6.5      .976
37   Petal_Width     1.8    1.332  Sepal_Length     6.5      .976
38  Sepal_Length     6.5     .976   Petal_Width     2.2      .654
39   Petal_Width       2    1.069  Sepal_Length     6.5      .976
40   Petal_Width     1.8    1.332  Sepal_Length     6.7      .668
41   Petal_Width     2.3    1.332  Sepal_Length     6.7      .668
42  Petal_Length     5.6    1.169  Sepal_Length     6.7      .668
43   Petal_Width     2.5     .917  Sepal_Length     6.7      .668
44   Petal_Width     2.1    1.169  Sepal_Length     6.7      .668
45   Petal_Width     2.1    1.169  Sepal_Length     6.9      .917
```

```
>>> nb_mod.predict_proba(test_dat.drop('Species'), supplemental_cols = test_dat[:, [
↪'Sepal_Length', 'Species']]).sort_values(by = ['Sepal_Length', 'Species',
↪'PROBABILITY_OF_setosa', 'PROBABILITY_OF_versicolor'])
     Sepal_Length      Species  PROBABILITY_OF_setosa  \
 0            4.4       setosa           1.000000e+00
 1            4.4       setosa           1.000000e+00
 2            4.5       setosa           1.000000e+00
 3            4.8       setosa           1.000000e+00
...           ...          ...                    ...
 42           6.7    virginica           1.412132e-13
 43           6.9   versicolor           5.295492e-20
 44           6.9    virginica           5.295492e-20
 45           7.0   versicolor           6.189014e-14

     PROBABILITY_OF_versicolor  PROBABILITY_OF_virginica
 0                9.327306e-21              7.868301e-20
 1                3.497737e-20              1.032715e-19
 2                2.238553e-13              2.360490e-19
 3                6.995487e-22              2.950617e-21
...                        ...                       ...
 42               4.741700e-13              1.000000e+00
 43               1.778141e-07              9.999998e-01
 44               2.963565e-20              1.000000e+00
 45               4.156340e-01              5.843660e-01
```

```
>>> nb_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
 0.934783
```

```
>>> oml.drop(table = 'NB_PRIOR_PROBABILITY_DEMO')
>>> oml.drop('IRIS')
```

2.10 Neural Network

class oml.**nn**(*mining_function='CLASSIFICATION'*, *model_name=None*, *model_owner=None*, ***params*)

In-database Neural Network Model

Builds a Neural Network (NN) Model that uses an algorithm inspired from biological neural network for classification and regression. Neural Network is used to to estimate or approximate functions that depend on a large number of generally unknown inputs. This function exposes the corresponding Oracle Machine Learning in-database algorithm. An artificial neural network is composed of a large number of interconnected neurons which exchange messages between each other to solve specific problems. They learn by examples and tune the weights of the connections among the neurons during the learning process. Neural Network is capable of solving a wide variety of tasks such as computer vision, speech recognition, and various complex business problems.

Attributes

weights : oml.DataFrame

Weights of fitted model between nodes in different layers. It includes the following components:

- layer: The layer ID, 0 as an input layer
- idx_from: The node index that the weight connects from (attribute id for input layer)
- idx_to: The node index that the weights connects to
- attribute_name: The attribute name (only for the input layer)
- attribute_subname: The attribute subname
- attribute_value: The attribute value
- target_value: The target value.
- weight: The value of weight

topology : oml.DataFrame

Topology of the fitted model including number of nodes and hidden layers. It includes the following components:

- hidden_layer_id: The id number of the hidden layer
- num_node: The number of nodes in each layer
- activation_function: The activation function in each layer

See also:

`glm`, `svm`, `km`

`__init__`(*mining_function='CLASSIFICATION'*, *model_name=None*, *model_owner=None*, ***params*)

Initializes an instance of nn object.

Parameters

mining_function
['CLASSIFICATION' or 'REGRESSION', 'CLASSIFICATION' (default)] Type of model mining functionality

model_name
[string or None (default)] The name of an existing database Neural Network model to create an oml.nn object from. The specified database model is not dropped when the oml.nn object is deleted.

model_owner: string or None (default)
The owner name of the existing Neural Network model The current database user by default

params
[key-value pairs or dict] Oracle Machine Learning parameter settings. Each dict element's name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 4-10 apply generally to the model. Mining Function Settings for Classification and Algorithm-specific Settings are applicable to Neural Network model.

export_sermodel(*table=None*, *partition=None*)
Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

fit(*x*, *y*, *model_name=None*, *case_id=None*, *class_weight=None*)
Fits a Neural Network Model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object, or string] Target values. If y is a single column OML object, target values specified by y must be combinable with x. If y is a string, y is the name of the column in x that specifies the target values.

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.nn object is deleted. If None, a system-generated model name will be used. The system-generated model is dropped when oml.nn object is deleted unless oml.nn object is saved into a datastore.

case_id
[string or None (default)] The column name used as case id for building the model.

class_weight
[OML DataFrame or dict or list of ints or floats or None (default)] An optional matrix that is used to influence the weighting of target classes during model creation. For OML DataFrame input, the first value represents the target value. The second value represents the class weight. For dictionary type input, the key represents the target value. The value represents the class weight. For list type input, the first value represents target value. The second value represents the predicted target value. Refer to Oracle Data Mining User's Guide for more details about class weights.

get_params(*params=None*, *deep=False*)
Fetches settings of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If params is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

predict (*x*, *supplemental_cols=None*, *proba=False*, *topN_attrs=False*)
Makes predictions on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

proba
[boolean, False (default)] Returns prediction probability if `proba` is True.

topN_attrs
[boolean, positive integer, False (default)] Returns the top N most influence attributes of the predicted target value for regression if topN_attrs is not False. Returns the top N most influence attributes of the highest probability class for classification if topN_attrs is not False. N is equal to the specified positive integer or 5 if topN_attrs is True.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. For a classification model, the results include the most likely target class and its probability. For a regression model, the results consist of a column for the prediction. For an anomaly detection model, the results include a prediction and its probability. If the prediction is 1, the case is considered typical. If the prediction is 0, the case is considered anomalous. This behavior reflects the fact that the model is trained with normal data.

predict_proba (*x*, *supplemental_cols=None*, *topN=None*)
Makes predictions and returns probability for each class on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN
[positive integer or None (default)] A positive integer that restricts the returned target classes to the specified number of those that have the highest probability.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. The results include for each target class, the probability belonging to that class.

score (*x*, *y*)
Makes predictions on new data, returns the mean accuracy for classifications or the coefficient of determination R^2 of the prediction for regressions.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object] Target values.

Returns

score
[float] Mean accuracy for classifications or the coefficient of determination R^2 of the prediction for regressions.

set_params (***params*)
Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

References

- Oracle R Enterprise
- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

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

Create table containing dataset iris from sklearn.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width', 'Petal_
↪Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:'virginica'}
↪[x], iris.target)), columns = ['Species'])
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create training data and test data.

```
>>> dat = oml.sync(table = "IRIS").split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
```

Create NN model object.

```
>>> nn_mod = oml.nn(nnet_hidden_layers = 1, nnet_activations= "'NNET_ACTIVATIONS_LOG_
↪SIG'", NNET_NODES_PER_LAYER= '30')
```

Fit the NN Model according to the training data and parameter settings.

```
>>> nn_mod = nn_mod.fit(train_x, train_y)
```

Show model details.

```
>>> nn_mod

Algorithm Name: Neural Network

Mining Function: CLASSIFICATION

Target: Species

Settings:
                    setting name                 setting value
0                      ALGO_NAME           ALGO_NEURAL_NETWORK
1           CLAS_WEIGHTS_BALANCED                          OFF
2                NNET_ACTIVATIONS  'NNET_ACTIVATIONS_LOG_SIG'
3              NNET_HIDDEN_LAYERS                            1
4            NNET_NODES_PER_LAYER                           30
5                  NNET_TOLERANCE                      .000001
6                    ODMS_DETAILS                  ODMS_ENABLE
7    ODMS_MISSING_VALUE_TREATMENT      ODMS_MISSING_VALUE_AUTO
8                ODMS_RANDOM_SEED                            0
9                   ODMS_SAMPLING        ODMS_SAMPLING_DISABLE
10                      PREP_AUTO                           ON

Computed Settings:
               setting name                setting value
```

(continues on next page)

(continued from previous page)

```
0  LBFGS_GRADIENT_TOLERANCE  .0000000010000000000000001
1       LBFGS_HISTORY_DEPTH                          20
2       LBFGS_SCALE_HESSIAN  LBFGS_SCALE_HESSIAN_ENABLE
3           NNET_ITERATIONS                         200
4          NNET_REGULARIZER       NNET_REGULARIZER_NONE
5               NNET_SOLVER           NNET_SOLVER_LBFGS

Global Statistics:
  attribute name attribute value
0      CONVERGED             YES
1     ITERATIONS              66
2     LOSS_VALUE               0
3       NUM_ROWS             104

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

Topology:

    HIDDEN_LAYER_ID  NUM_NODE       ACTIVATION_FUNCTION
 0                0        30  NNET_ACTIVATIONS_LOG_SIG

Weights:

      LAYER  IDX_FROM  IDX_TO ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE  \
 0        0       0.0       0   Petal_Length              None            None
 1        0       0.0       1   Petal_Length              None            None
 2        0       0.0       2   Petal_Length              None            None
 3        0       0.0       3   Petal_Length              None            None
 ...    ...       ...     ...            ...               ...             ...
 239      1      29.0       2           None              None            None
 240      1       NaN       0           None              None            None
 241      1       NaN       1           None              None            None
 242      1       NaN       2           None              None            None

     TARGET_VALUE     WEIGHT
 0           None -100.896828
 1           None   55.437420
 2           None    6.002858
 3           None    3.423386
 ...          ...        ...
 239    virginica  -32.203426
 240       setosa    4.609416
 241   versicolor   18.308834
 242    virginica  -22.362225

[243 rows x 8 columns]
```

Use the model to make predictions on test data.

```
>>> nn_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Petal_Length', 'Species']])
    Sepal_Length  Sepal_Width  Petal_Length    Species  PREDICTION
 0           4.9          3.0           1.4     setosa      setosa
 1           4.9          3.1           1.5     setosa      setosa
 2           4.8          3.4           1.6     setosa      setosa
 3           5.8          4.0           1.2     setosa      setosa
...          ...          ...           ...        ...         ...
 42          6.7          3.3           5.7  virginica   virginica
 43          6.7          3.0           5.2  virginica   virginica
 44          6.5          3.0           5.2  virginica   virginica
 45          5.9          3.0           5.1  virginica   virginica
```

```
>>> nn_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Species']], proba = True)
    Sepal_Length  Sepal_Width    Species  PREDICTION  PROBABILITY
 0           4.9          3.0     setosa      setosa     1.000000
 1           4.9          3.1     setosa      setosa     1.000000
 2           4.8          3.4     setosa      setosa     1.000000
 3           5.8          4.0     setosa      setosa     1.000000
...          ...          ...        ...         ...          ...
 42          6.7          3.3  virginica   virginica     1.000000
 43          6.7          3.0  virginica   virginica     1.000000
 44          6.5          3.0  virginica   virginica     1.000000
 45          5.9          3.0  virginica   virginica     1.000000
```

```
>>> nn_mod.predict_proba(test_dat.drop('Species'), supplemental_cols = test_dat[:, [
↪'Sepal_Length', 'Species']]).sort_values(by = ['Sepal_Length', 'Species',
↪'PROBABILITY_OF_setosa', 'PROBABILITY_OF_versicolor'])
    Sepal_Length     Species  PROBABILITY_OF_setosa  \
 0           4.4      setosa           1.000000e+00
 1           4.4      setosa           1.000000e+00
 2           4.5      setosa           1.000000e+00
 3           4.8      setosa           1.000000e+00
...          ...         ...                    ...
 42          6.7   virginica           0.000000e+00
 43          6.9  versicolor          7.460391e-213
 44          6.9   virginica           0.000000e+00
 45          7.0  versicolor          2.487811e-189

    PROBABILITY_OF_versicolor  PROBABILITY_OF_virginica
 0              2.110796e-112              0.000000e+00
 1              2.330673e-112              0.000000e+00
 2              2.110793e-112              0.000000e+00
 3               8.954877e-85              0.000000e+00
...                       ...                       ...
 42              1.237268e-39              1.000000e+00
 43              1.000000e+00             4.123744e-298
 44              4.982164e-01              5.017836e-01
 45              1.000000e+00             1.086105e-319
```

```
>>> nn_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
 0.934783
```

Change parameter setting and refit model.

```
>>> new_setting = {'NNET_NODES_PER_LAYER': '50'}
>>> nn_mod.set_params(**new_setting).fit(train_x, train_y)

Algorithm Name: Neural Network

Mining Function: CLASSIFICATION

Target: Species

Settings:
                    setting name                setting value
0                      ALGO_NAME          ALGO_NEURAL_NETWORK
1           CLAS_WEIGHTS_BALANCED                         OFF
2                NNET_ACTIVATIONS   'NNET_ACTIVATIONS_LOG_SIG'
3              NNET_HIDDEN_LAYERS                           1
4            NNET_NODES_PER_LAYER                          50
5                  NNET_TOLERANCE                     .000001
6                    ODMS_DETAILS                 ODMS_ENABLE
7    ODMS_MISSING_VALUE_TREATMENT     ODMS_MISSING_VALUE_AUTO
8                ODMS_RANDOM_SEED                           0
9                   ODMS_SAMPLING       ODMS_SAMPLING_DISABLE
10                      PREP_AUTO                          ON

Computed Settings:
               setting name                setting value
0  LBFGS_GRADIENT_TOLERANCE  .000000001000000000000000001
1       LBFGS_HISTORY_DEPTH                           20
2       LBFGS_SCALE_HESSIAN  LBFGS_SCALE_HESSIAN_ENABLE
3           NNET_ITERATIONS                          200
4          NNET_REGULARIZER        NNET_REGULARIZER_NONE
5               NNET_SOLVER            NNET_SOLVER_LBFGS

Global Statistics:
  attribute name attribute value
0      CONVERGED             YES
1     ITERATIONS              85
2     LOSS_VALUE               0
3       NUM_ROWS             104

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

Topology:

    HIDDEN_LAYER_ID  NUM_NODE       ACTIVATION_FUNCTION
 0                0        50  NNET_ACTIVATIONS_LOG_SIG

Weights:

      LAYER  IDX_FROM  IDX_TO ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE  \
 0        0       0.0       0   Petal_Length              None            None
 1        0       0.0       1   Petal_Length              None            None
```

(continues on next page)

(continued from previous page)

```
2        0       0.0       2   Petal_Length             None            None
3        0       0.0       3   Petal_Length             None            None
...    ...       ...     ...            ...              ...             ...
399      1      49.0       2           None             None            None
400      1       NaN       0           None             None            None
401      1       NaN       1           None             None            None
402      1       NaN       2           None             None            None

    TARGET_VALUE     WEIGHT
0           None -32.079338
1           None -53.977856
2           None  -7.649853
3           None -33.343208
...          ...        ...
399    virginica -49.988823
400       setosa   2.376528
401   versicolor  24.591450
402    virginica -27.208340

[403 rows x 8 columns]
```

```
>>> oml.drop('IRIS')
```

2.11 Random Forest

class oml.**rf**(*model_name=None*, *model_owner=None*, ***params*)

In-database Random Forest Model

Builds a Random Forest (RF) Model that uses an ensemble (also called forest) of trees for classification. This function exposes the corresponding Oracle Machine Learning in-database algorithm. Random Forest is a popular ensemble learning technique for classification. By combining the ideas of bagging and random selection of variables, the algorithm produces collection of decision trees with controlled variance, while avoiding overfitting - a common problem for decision trees.

Attributes

importance : oml.DataFrame

Attribute importance of the fitted model. It includes the following components:

- attribute_name: The attribute name
- attribute_subname: The attribute subname
- attribute_importance: The attribute importance for an attribute in the forest

See also:

glm, *svm*, *km*

__init__(*model_name=None*, *model_owner=None*, ***params*)

Initializes an instance of rf object.

Parameters

model_name

[string or None (default)] The name of an existing database Random Forest model to create an oml.rf object from. The specified database model is not dropped when the oml.rf object is deleted.

model_owner: string or None (default)
The owner name of the existing Random Forest model The current database user by default

params
[key-value pairs or dict] Oracle Machine Learning parameter settings. Each dict element's name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 4-10 apply generally to the model. Mining Function Settings for Classification and Algorithm-specific Settings are applicable to Random Forest model.

export_sermodel(*table=None*, *partition=None*)
Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

fit(*x*, *y*, *model_name=None*, *cost_matrix=None*, *case_id=None*)
Fits a Random Forest Model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object, or string] Target values. If y is a single column OML object, target values specified by y must be combinable with x. If y is a string, y is the name of the column in x that specifies the target values.

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.rf object is deleted. If None, a system-generated model name will be used. The system-generated model is dropped when oml.rf object is deleted unless oml.rf object is saved into a datastore.

cost_matrix
[OML DataFrame or list of ints or floats or None (default)] An optional numerical square matrix that specifies the costs for incorrectly predicting the target values. The first value represents the actual target value. The second value represents the predicted target value. The third value is the cost. In general, the diagonal entries of the matrix are zeros. Refer to Oracle Data Mining User's Guide for more details about cost matrix.

case_id
[string or None (default)] The column name used as case id for building the model.

get_params (*params=None*, *deep=False*)
Fetches parameters of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If `params` is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

predict (*x*, *supplemental_cols=None*, *proba=False*, *topN_attrs=False*)
Makes predictions on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

proba
[boolean, False (default)] Returns prediction probability if `proba` is True

topN_attrs
[boolean, positive integer, False (default)] Returns the top N most influence attributes of the predicted target value for regression if topN_attrs is not False. Returns the top N most influence attributes of the highest probability class for classification if topN_attrs is not False. N is equal to the specified positive integer or 5 if topN_attrs is True.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols`, and the most likely target class.

predict_proba (*x*, *supplemental_cols=None*, *topN=None*)
Makes predictions and returns probability for each class on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN
[positive integer or None (default)] A positive integer that restricts the returned target classes to the specified number of those that have the highest probability.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. The results include for each target class, the probability belonging to that class.

score (*x*, *y*)
Makes predictions on new data and returns the mean accuracy.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object] Target values.

Returns

score
[float] Mean accuracy for classifications.

set_params (*****params***)
Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

References

- Oracle R Enterprise
- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

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

Create table containing dataset iris from sklearn.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width', 'Petal_
↪Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:'virginica'}
↪[x], iris.target)), columns = ['Species'])
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create training data and test data.

```
>>> dat = oml.sync(table = "IRIS").split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
```

Create a cost matrix table in the Oracle Database.

```
>>> cost_matrix = [['setosa', 'setosa', 0],
...                ['setosa', 'virginica', 0.2],
...                ['setosa', 'versicolor', 0.8],
...                ['virginica', 'virginica', 0],
...                ['virginica', 'setosa', 0.5],
...                ['virginica', 'versicolor', 0.5],
...                ['versicolor', 'versicolor', 0],
...                ['versicolor', 'setosa', 0.4],
...                ['versicolor', 'virginica', 0.6]]
>>> cost_matrix = oml.create(pd.DataFrame(cost_matrix, columns = ['ACTUAL_TARGET_VALUE
↪', 'PREDICTED_TARGET_VALUE', 'COST']), 'RF_COST')
```

Create RF model object.

```
>>> rf_mod = oml.rf(tree_term_max_depth = '2')
```

Fit the RF Model according to the training data and parameter settings.

```
>>> rf_mod = rf_mod.fit(train_x, train_y, cost_matrix = cost_matrix)
```

Show model details.

```
>>> rf_mod

Algorithm Name: Random Forest

Mining Function: CLASSIFICATION

Target: Species

Settings:
                    setting name             setting value
0                      ALGO_NAME        ALGO_RANDOM_FOREST
1           CLAS_COST_TABLE_NAME       "PYQUSER"."RF_COST"
2              CLAS_MAX_SUP_BINS                        32
```

(continues on next page)

(continued from previous page)

```
3          CLAS_WEIGHTS_BALANCED                         OFF
4                   ODMS_DETAILS                 ODMS_ENABLE
5   ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
6               ODMS_RANDOM_SEED                           0
7                  ODMS_SAMPLING       ODMS_SAMPLING_DISABLE
8                      PREP_AUTO                          ON
9                 RFOR_NUM_TREES                          20
10           RFOR_SAMPLING_RATIO                          .5
11          TREE_IMPURITY_METRIC          TREE_IMPURITY_GINI
12           TREE_TERM_MAX_DEPTH                           2
13         TREE_TERM_MINPCT_NODE                         .05
14        TREE_TERM_MINPCT_SPLIT                          .1
15         TREE_TERM_MINREC_NODE                          10
16        TREE_TERM_MINREC_SPLIT                          20

Computed Settings:
  setting name setting value
0    RFOR_MTRY             2

Global Statistics:
  attribute name  attribute value
0      AVG_DEPTH                2
1  AVG_NODECOUNT                3
2      MAX_DEPTH                2
3  MAX_NODECOUNT                3
4      MIN_DEPTH                2
5  MIN_NODECOUNT                3
6       NUM_ROWS              104

Attributes:
Petal_Length
Petal_Width
Sepal_Length

Partition: NO

Importance:

  ATTRIBUTE_NAME ATTRIBUTE_SUBNAME  ATTRIBUTE_IMPORTANCE
0   Petal_Length              None              0.329971
1    Petal_Width              None              0.296799
2   Sepal_Length              None              0.037309
3    Sepal_Width              None              0.000000
```

Use the model to make predictions on test data.

```
>>> rf_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Petal_Length', 'Species']])
     Sepal_Length  Sepal_Width  Petal_Length    Species  PREDICTION
 0            4.9          3.0           1.4     setosa      setosa
 1            4.9          3.1           1.5     setosa      setosa
 2            4.8          3.4           1.6     setosa      setosa
 3            5.8          4.0           1.2     setosa      setosa
...           ...          ...           ...        ...         ...
 42           6.7          3.3           5.7  virginica   virginica
 43           6.7          3.0           5.2  virginica   virginica
```

(continues on next page)

(continued from previous page)

```
44            6.5           3.0          5.2   virginica   virginica
45            5.9           3.0          5.1   virginica   virginica
```

```
>>> rf_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Species']], proba = True)
     Sepal_Length  Sepal_Width     Species  PREDICTION  PROBABILITY
 0            4.9          3.0      setosa      setosa     0.989130
 1            4.9          3.1      setosa      setosa     0.989130
 2            4.8          3.4      setosa      setosa     0.989130
 3            5.8          4.0      setosa      setosa     0.950000
...           ...          ...         ...         ...          ...
 42           6.7          3.3   virginica   virginica     0.501016
 43           6.7          3.0   virginica   virginica     0.501016
 44           6.5          3.0   virginica   virginica     0.501016
 45           5.9          3.0   virginica   virginica     0.501016
```

```
>>> rf_mod.predict_proba(test_dat.drop('Species'), supplemental_cols = test_dat[:, [
↪'Sepal_Length', 'Species']], topN = 2).sort_values(by = ['Sepal_Length', 'Species'])
     Sepal_Length     Species       TOP_1  TOP_1_VAL       TOP_2  TOP_2_VAL
 0            4.4      setosa      setosa   0.989130  versicolor   0.010870
 1            4.4      setosa      setosa   0.989130  versicolor   0.010870
 2            4.5      setosa      setosa   0.989130  versicolor   0.010870
 3            4.8      setosa      setosa   0.989130  versicolor   0.010870
...           ...         ...         ...        ...         ...        ...
 42           6.7   virginica   virginica   0.501016  versicolor   0.498984
 43           6.9  versicolor   virginica   0.501016  versicolor   0.498984
 44           6.9   virginica   virginica   0.501016  versicolor   0.498984
 45           7.0  versicolor   virginica   0.501016  versicolor   0.498984
```

```
>>> rf_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.76087
```

Reset TREE_TERM_MAX_DEPTH and refit model.

```
>>> rf_mod.set_params(tree_term_max_depth = '3').fit(train_x, train_y, cost_matrix =␣
↪cost_matrix)

Algorithm Name: Random Forest

Mining Function: CLASSIFICATION

Target: Species

Settings:
                    setting name            setting value
0                      ALGO_NAME       ALGO_RANDOM_FOREST
1           CLAS_COST_TABLE_NAME      "PYQUSER"."RF_COST"
2              CLAS_MAX_SUP_BINS                       32
3          CLAS_WEIGHTS_BALANCED                      OFF
4                   ODMS_DETAILS              ODMS_ENABLE
5    ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
6               ODMS_RANDOM_SEED                        0
7                  ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
8                      PREP_AUTO                       ON
9                 RFOR_NUM_TREES                       20
```

(continues on next page)

(continued from previous page)

```
10          RFOR_SAMPLING_RATIO                         .5
11         TREE_IMPURITY_METRIC       TREE_IMPURITY_GINI
12          TREE_TERM_MAX_DEPTH                          3
13        TREE_TERM_MINPCT_NODE                        .05
14       TREE_TERM_MINPCT_SPLIT                         .1
15        TREE_TERM_MINREC_NODE                         10
16       TREE_TERM_MINREC_SPLIT                         20

Computed Settings:
  setting name setting value
0    RFOR_MTRY             2

Global Statistics:
  attribute name  attribute value
0      AVG_DEPTH                3
1   AVG_NODECOUNT               5
2      MAX_DEPTH                3
3   MAX_NODECOUNT               5
4      MIN_DEPTH                3
5   MIN_NODECOUNT               5
6       NUM_ROWS              104

Attributes:
Petal_Length
Petal_Width
Sepal_Length

Partition: NO

Importance:

  ATTRIBUTE_NAME ATTRIBUTE_SUBNAME  ATTRIBUTE_IMPORTANCE
0   Petal_Length              None              0.501022
1    Petal_Width              None              0.568170
2   Sepal_Length              None              0.091617
3    Sepal_Width              None              0.000000
```

```
>>> oml.drop(table = 'RF_COST')
>>> oml.drop('IRIS')
```

2.12 Singular Value Decomposition

class oml.**svd**(*model_name=None*, *model_owner=None*, ***params*)

In-database Singular Value Decomposition Model

Builds a Singular Value Decomposition (SVD) Model that can be used for feature extraction. SVD provides orthogonal linear transformations that capture the underlying variance of the data by decomposing a rectangular matrix into three matrixes: U, D, and V. Matrix D is a diagonal matrix and its singular values reflect the amount of data variance captured by the bases. Columns of matrix V contain the right singular vectors and columns of matrix U contain the left singular vectors.

Attributes

features : oml.DataFrame

Features extracted by the fitted model including feature id and associated coefficient. It includes the following components:

- feature_id: The ID of a feature in the model
- attribute_name: The attribute name
- attribute_value: The attribute value
- value: The matrix entry value

u : oml.DataFrame

A dataframe whose columns contain the left singular vectors. The column name is the corresponding feature id.

v : oml.DataFrame

A dataframe whose columns contain the right singular vectors. The column name is the corresponding feature id.

d : oml.DataFrame

A dataframe containing the singular values of the input data. It includes the following components:

- feature_id: The ID of a feature in the model
- value: The singular values of the input data

See also:

dt, *km*, *svm*

__init__(*model_name=None*, *model_owner=None*, ***params*)

Initializes an instance of svd object.

Parameters

model_name
[string or None (default)] The name of an existing database Singular Value Decomposition model to create an oml.svd object from. The specified database model is not dropped when the oml.svd object is deleted.

model_owner: string or None (default)
The owner name of the existing Singular Value Decomposition model The current database user by default

params
[key-value pairs or dict] Oracle Machine Learning parameter settings. Each list element's name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 4-10 apply generally to the model. Mining Function Settings for Feature Extraction and Algorithm-specific Settings are applicable to Singular Value Decomposition model.

export_sermodel(*table=None*, *partition=None*)

Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

feature_compare (*x*, *compare_cols=None*, *supplemental_cols=None*)
Compares features of data and generates relatedness.

Parameters

x [an OML object] The data used to measure relatedness.

compare_cols
[str, a list of str or None (default)] The column(s) used to measure data relatedness. If None, all the columns of `x` are compared to measure relatedness.

supplemental_cols
[a list of str or None (default)] A list of columns to display along with the resulting 'SIMILARITY' column.

Returns

pred
[oml.DataFrame] Contains a 'SIMILARITY' column that measures relatedness and supplementary columns if specified.

fit (*x*, *model_name=None*, *case_id=None*, *ctx_settings=None*)
Fits an SVD Model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model.

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.svd object is deleted. If None, a system-generated model name will be used. The system-generated model is dropped when oml.svd object is deleted unless oml.svd object is saved into a datastore.

case_id
[string or None (default)] The column name used as case id for building the model. `case_id` and SVDS_U_MATRIX_OUTPUT in `odm_settings` must be specified in order to produce matrix U.

ctx_settings
[dict or None (default)] A list to specify Oracle Text attribute-specific settings. This argument is applicable to building models in Oracle Database 12.2 or later. The name of each list element refers to the text column while the list value is a scalar string specifying the attribute-specific text transformation. The valid entries in the string include TEXT, POLICY_NAME, TOKEN_TYPE, and MAX_FEATURES.

get_params (*params=None*, *deep=False*)
Fetches parameters of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If `params` is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

predict (*x*, *supplemental_cols=None*, *topN_attrs=False*)
Makes predictions on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN_attrs
[boolean, positive integer, False (default)] Returns the top N most important features with the corresponding weights. N is equal to the specified positive integer or 5 if topN_attrs is True.

Returns

pred
[oml.DataFrame] Contains the predicted feature index on the new data and the specified `supplemental_cols`.

set_params (***params*)
Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

transform(*x*, *supplemental_cols=None*, *topN=None*)
Performs dimensionality reduction and returns value for each feature on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN
[positive integer or None (default)] A positive integer that restricts the returned values to the specified number of features that have the highest topN values. If None, all features will be returned.

Returns

pred
[oml.DataFrame] Contains the values of new data after the SVD transform and the specified `supplemental_cols`.

References

- Oracle R Enterprise
- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

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

Create table containing dataset iris from sklearn.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width', 'Petal_
↪Length', 'Petal_Width'])
>>> x.insert(0, "ID", range(1, len(x) + 1))
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:'virginica'}
↪[x], iris.target)), columns = ['Species'])
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create training data and test data.

```
>>> dat = oml.sync(table = "IRIS").split()
>>> train_dat = dat[0]
>>> test_dat = dat[1]
```

Create SVD model object.

```
>>> svd_mod = oml.svd(ODMS_DETAILS = 'ODMS_ENABLE')
```

Fit the SVD Model according to the training data and parameter settings.

```
>>> svd_mod = svd_mod.fit(train_dat)
```

Show model details.

```
>>> svd_mod

Algorithm Name: Singular Value Decomposition

Mining Function: FEATURE_EXTRACTION

Settings:
                   setting name                setting value
0                     ALGO_NAME  ALGO_SINGULAR_VALUE_DECOMP
1                  ODMS_DETAILS                 ODMS_ENABLE
2  ODMS_MISSING_VALUE_TREATMENT     ODMS_MISSING_VALUE_AUTO
3                 ODMS_SAMPLING       ODMS_SAMPLING_DISABLE
4                     PREP_AUTO                          ON
5             SVDS_SCORING_MODE            SVDS_SCORING_SVD
6          SVDS_U_MATRIX_OUTPUT       SVDS_U_MATRIX_DISABLE

Computed Settings:
              setting name                setting value
0        FEAT_NUM_FEATURES                            8
1  ODMS_EXPLOSION_MIN_SUPP                            1
2              SVDS_SOLVER          SVDS_SOLVER_TSEIGEN
3           SVDS_TOLERANCE  .000000000000024646951146678475

Global Statistics:
     attribute name  attribute value
0    NUM_COMPONENTS                8
1          NUM_ROWS              111
2  SUGGESTED_CUTOFF                1

Attributes:
ID
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Features:

     FEATURE_ID ATTRIBUTE_NAME ATTRIBUTE_VALUE     VALUE
 0            1             ID            None  0.996297
```

(continues on next page)

(continued from previous page)

```
 1             1   Petal_Length            None  0.046646
 2             1    Petal_Width            None  0.015917
 3             1   Sepal_Length            None  0.063312
...          ...            ...             ...       ...
 60            8    Sepal_Width            None -0.030620
 61            8        Species          setosa  0.431543
 62            8        Species      versicolor  0.566418
 63            8        Species       virginica  0.699261

[64 rows x 4 columns]

D:

    FEATURE_ID        VALUE
 0           1   886.737809
 1           2    32.736792
 2           3    10.043389
 3           4     5.270496
 4           5     2.708602
 5           6     1.652340
 6           7     0.938640
 7           8     0.452170

V:

         '1'       '2'       '3'       '4'       '5'       '6'       '7'  \
 0  0.001332  0.156581 -0.317375  0.113462 -0.154414 -0.113058  0.799390
 1  0.003692  0.052289  0.316295 -0.733040  0.190746  0.022285 -0.046406
 2  0.005267 -0.051498 -0.052111  0.527881 -0.066995  0.046461 -0.469396
 3  0.015917  0.008741  0.263614  0.244811  0.460445  0.767503  0.262966
 4  0.030208  0.550384 -0.358277  0.041807  0.689962 -0.261815 -0.143258
 5  0.046646  0.189325  0.766663  0.326363  0.079611 -0.479070  0.177661
 6  0.063312  0.790864  0.097964 -0.051230 -0.490804  0.312159 -0.131337
 7  0.996297 -0.076079 -0.035940 -0.017429 -0.000960 -0.001908  0.001755

         '8'
 0  0.431543
 1  0.566418
 2  0.699261
 3  0.005000
 4 -0.030620
 5 -0.016932
 6 -0.052185
 7 -0.001415
```

Use the model to make predictions on test data.

```
>>> svd_mod.predict(test_dat, supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_
↪Width', 'Petal_Length', 'Species']])
     Sepal_Length  Sepal_Width  Petal_Length     Species  FEATURE_ID
 0            5.0          3.6           1.4      setosa           2
 1            5.0          3.4           1.5      setosa           2
 2            4.4          2.9           1.4      setosa           8
 3            4.9          3.1           1.5      setosa           2
...           ...          ...           ...         ...         ...
 35           6.9          3.1           5.4   virginica           1
```

(continues on next page)

(continued from previous page)

```
36            5.8          2.7           5.1   virginica          1
37            6.2          3.4           5.4   virginica          5
38            5.9          3.0           5.1   virginica          1
```

```
>>> svd_mod.transform(test_dat, supplemental_cols = test_dat[:, ['Sepal_Length']],␣
↪topN = 2).sort_values(by = ['Sepal_Length', 'TOP_1', 'TOP_1_VAL'])
    Sepal_Length  TOP_1  TOP_1_VAL  TOP_2  TOP_2_VAL
0            4.4      7   0.153125      3  -0.130778
1            4.4      8   0.171819      2   0.147070
2            4.8      2   0.159324      6  -0.085194
3            4.8      7   0.157187      3  -0.141668
...          ...    ...        ...    ...        ...
35           7.2      6  -0.167688      1   0.142545
36           7.2      7  -0.176290      6  -0.175527
37           7.6      4   0.205779      3   0.141533
38           7.9      8  -0.253194      7  -0.166967
```

```
>>> svd_mod.feature_compare(test_dat, compare_cols = "Sepal_Length", supplemental_
↪cols = ["Species"])
      Species_A   Species_B  SIMILARITY
0        setosa      setosa    1.000000
1        setosa      setosa    0.808191
2        setosa      setosa    0.968032
3        setosa      setosa    0.936064
...         ...         ...         ...
737   virginica   virginica    0.680319
738   virginica   virginica    0.872128
739   virginica   virginica    0.968032
740   virginica   virginica    0.904096

[741 rows x 3 columns]
```

```
>>> svd_mod.feature_compare(test_dat, compare_cols = ["Sepal_Length", "Petal_Length"],
↪ supplemental_cols = ["Species"])
      Species_A   Species_B  SIMILARITY
0        setosa      setosa    0.963689
1        setosa      setosa    0.808191
2        setosa      setosa    0.936912
3        setosa      setosa    0.873825
...         ...         ...         ...
737   virginica   virginica    0.745295
738   virginica   virginica    0.907574
739   virginica   virginica    0.968032
740   virginica   virginica    0.920580

[741 rows x 3 columns]
```

Set parameters and refit model to produce U matrix output.

```
>>> new_setting = {'SVDS_U_MATRIX_OUTPUT':'SVDS_U_MATRIX_ENABLE'}
>>> svd_mod2 = svd_mod.set_params(**new_setting).fit(train_dat, case_id = "ID")
>>> svd_mod2

Algorithm Name: Singular Value Decomposition
```

(continues on next page)

(continued from previous page)

```
Mining Function: FEATURE_EXTRACTION

Settings:
                   setting name              setting value
0                     ALGO_NAME  ALGO_SINGULAR_VALUE_DECOMP
1                  ODMS_DETAILS                 ODMS_ENABLE
2  ODMS_MISSING_VALUE_TREATMENT     ODMS_MISSING_VALUE_AUTO
3                 ODMS_SAMPLING       ODMS_SAMPLING_DISABLE
4                     PREP_AUTO                          ON
5             SVDS_SCORING_MODE            SVDS_SCORING_SVD
6          SVDS_U_MATRIX_OUTPUT        SVDS_U_MATRIX_ENABLE

Computed Settings:
              setting name                     setting value
0        FEAT_NUM_FEATURES                                 7
1  ODMS_EXPLOSION_MIN_SUPP                                 1
2              SVDS_SOLVER               SVDS_SOLVER_TSEIGEN
3           SVDS_TOLERANCE  .0000000000000024646951146678475

Global Statistics:
     attribute name  attribute value
0    NUM_COMPONENTS                7
1          NUM_ROWS              111
2  SUGGESTED_CUTOFF                1

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Features:

     FEATURE_ID ATTRIBUTE_NAME ATTRIBUTE_VALUE     VALUE
 0            1   Petal_Length            None  0.502473
 1            1    Petal_Width            None  0.163443
 2            1   Sepal_Length            None  0.754196
 3            1    Sepal_Width            None  0.382870
...         ...            ...             ...       ...
 45           7    Sepal_Width            None -0.039245
 46           7        Species          setosa  0.519357
 47           7        Species      versicolor  0.569289
 48           7        Species       virginica  0.631702

D:

    FEATURE_ID      VALUE
 0           1  81.794768
 1           2  16.665776
 2           3   6.233147
 3           4   2.713624
 4           5   1.688730
 5           6   0.990827
 6           7   0.508358
```

(continues on next page)

(continued from previous page)

```
V:

        '1'       '2'       '3'       '4'       '5'       '6'       '7'
 0  0.034567 -0.321783  0.189861 -0.158983 -0.139476  0.738073  0.519357
 1  0.045714  0.160442  0.542867 -0.071005  0.075494 -0.517374  0.631702
 2  0.046062  0.051656 -0.779916  0.208635  0.066414 -0.122111  0.569289
 3  0.163443  0.319041  0.163301  0.458382  0.726772  0.326371 -0.001224
 4  0.382870 -0.514653  0.146120  0.689289 -0.259206 -0.152788 -0.039245
 5  0.502473  0.663275  0.014142  0.055890 -0.522278  0.177428 -0.002961
 6  0.754196 -0.247904 -0.112963 -0.487644  0.319807 -0.106385 -0.074700

U:

      CASE_ID       '1'       '2'       '3'       '4'       '5'       '6'  \
 0          1  0.072831 -0.143707  0.028497 -0.023413 -0.000896 -0.025813
 1          2  0.068646 -0.125291  0.020401 -0.114478  0.037974  0.072763
 2          3  0.067124 -0.132472  0.028487 -0.029795  0.000328  0.045489
 3          4  0.066962 -0.119937  0.028409 -0.033107 -0.065116  0.107460
 ...      ...       ...       ...       ...       ...       ...       ...
 107      145  0.117795  0.082769  0.121459  0.147758  0.120074  0.093773
 108      146  0.112919  0.068305  0.108052  0.027473  0.234684 -0.015380
 109      147  0.104863  0.074078  0.092647 -0.099338  0.125387 -0.062902
 110      148  0.110476  0.065537  0.103817  0.012738  0.067699 -0.092724

          '7'
 0   -0.006612
 1    0.061376
 2    0.075908
 3    0.097157
 ...       ...
 107 -0.035875
 108 -0.009321
 109  0.090184
 110  0.020790

[111 rows x 8 columns]
```

```
>>> oml.drop('IRIS')
```

2.13 Support Vector Machine

class oml.**svm**(*mining_function='CLASSIFICATION'*, *model_name=None*, *model_owner=None*, ***params*)

In-database Support Vector Machine Model

Builds a Support Vector Machine (SVM) Model to be used for regression, classification, or anomaly detection. This function exposes the corresponding Oracle Machine Learning in-database algorithm. SVM is a powerful, state-of-the-art algorithm with strong theoretical foundations based on the Vapnik-Chervonenkis theory. SVM has strong regularization properties. Regularization refers to the generalization of the model to new data.

Attributes

coef : oml.DataFrame

The coefficients of the SVM model, one for each predictor variable. It includes the following components:

- target_value: The target value
- attribute_name: The attribute name
- attribute_subname: The attribute subname
- attribute_value: The attribute value
- coef: The projection coefficient value

See also:

dt, *glm*, *km*

`__init__`(*mining_function='CLASSIFICATION'*, *model_name=None*, *model_owner=None*, ***params*)

Initializes an instance of svm object.

Parameters

mining_function
['CLASSIFICATION' (default), 'REGRESSION' or 'ANOMALY_DETECTION'] Type of model mining functionality.

model_name
[string or None (default)] The name of an existing database Support Vector Machine model to create an oml.svm object from. The specified database model is not dropped when the oml.svm object is deleted.

model_owner: string or None (default)
The owner name of the existing Support Vector Machine model The current database user by default

params
[key-value pairs or dict] Oracle Machine Learning parameter settings. Each list element's name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 4-10 apply generally to the model. Mining Function Settings for Classification and Algorithm-specific Settings are applicable to Support Vector Machine model.

`export_sermodel`(*table=None*, *partition=None*)

Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

fit (*x*, *y*, *model_name=None*, *case_id=None*, *ctx_settings=None*)
Fits an SVM Model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object or None, or string] Target values. Must be specified when SVM algorithm is used for classification or regression and must be None when used for anomaly detection. If y is a single column OML object, target values specified by y must be combinable with x. If y is a string, y is the name of the column in x that specifies the target values.

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.svm object is deleted. If None, a system-generated model name will be used. The system-generated model is dropped when oml.svm object is deleted unless oml.svm object is saved into a datastore.

case_id
[string or None (default)] The name of a column that contains unique case identifiers.

ctx_settings
[dict or None (default)] A list to specify Oracle Text attribute-specific settings. This argument is applicable to building models in Oracle Database 12.2 or later. The name of each list element refers to the text column while the list value is a scalar string specifying the attribute-specific text transformation. The valid entries in the string include TEXT, POLICY_NAME, TOKEN_TYPE, and MAX_FEATURES.

get_params (*params=None*, *deep=False*)
Fetches parameters of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If `params` is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

predict (*x*, *supplemental_cols=None*, *proba=False*, *topN_attrs=False*)
Makes predictions on new data.

Parameters

x [oml.DataFrame] Predictor values used by the model to generate scores

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

proba
[boolean, False (default)] Returns prediction probability if `proba` is True.

topN_attrs
[boolean, positive integer, False (default)] Returns the top N most influence attributes of the predicted target value for regression if topN_attrs is not False. Returns the top N most influence attributes of the highest probability class for classification if topN_attrs is not False. N is equal to the specified positive integer or 5 if topN_attrs is True.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. For a classification model, the results include the most likely target class and its probability. For a regression model, the results consist of a column for the prediction. For an anomaly detection model, the results include a prediction and its probability. If the prediction is 1, the case is considered normal. If the prediction is 0, the case is considered anomalous. This behavior reflects the fact that the model is trained with normal data.

`predict_proba` (*x*, *supplemental_cols=None*, *topN=None*)
Makes predictions and returns probability for each class on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN
[positive integer or None (default)] A positive integer that restricts the returned target classes to the specified number of those that have the highest probability.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. The results include for each target class, the probability belonging to that class.

`score` (*x*, *y*)
Makes predictions on new data, returns the mean accuracy for classifications or the coefficient of determination R^2 of the prediction for regressions.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object] Target values.

Returns

score
[float] Mean accuracy for classifications or the coefficient of determination R^2 of the prediction for regressions.

set_params(***params*)
Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

References

- B. L. Milenova, J. S. Yarmus, and M. M. Campos. SVM in oracle database 10g: removing the barriers to widespread adoption of support vector machines. In Proceedings of the ''31st international Conference on Very Large Data Bases" (Trondheim, Norway, August 30 - September 02, 2005). pp1152-1163, ISBN:1-59593-154-6.
- Milenova, B.L. Campos, M.M., Mining high-dimensional data for information fusion: a database-centric approach 8th International Conference on Information Fusion, 2005. Publication Date: 25-28 July 2005. ISBN: 0-7803-9286-8. John Shawe-Taylor & Nello Cristianini. Support Vector Machines and other kernel-based learning methods. Cambridge University Press, 2000.
- Oracle R Enterprise
- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

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

Create table containing dataset iris from sklearn.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width', 'Petal_
↪Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:'virginica'}
↪[x], iris.target)), columns = ['Species'])
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create training data and test data.

```
>>> dat = oml.sync(table = "IRIS").split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
```

Create SVM model object.

```
>>> svm_mod = oml.svm('classification', svms_kernel_function = 'dbms_data_mining.svms_
↪linear')
```

Fit the SVM Model according to the training data and parameter settings.

```
>>> svm_mod.fit(train_x, train_y)

Algorithm Name: Support Vector Machine

Mining Function: CLASSIFICATION

Target: Species

Settings:
                   setting name                  setting value
0                     ALGO_NAME  ALGO_SUPPORT_VECTOR_MACHINES
1          CLAS_WEIGHTS_BALANCED                           OFF
2                  ODMS_DETAILS                   ODMS_ENABLE
3   ODMS_MISSING_VALUE_TREATMENT       ODMS_MISSING_VALUE_AUTO
4                 ODMS_SAMPLING         ODMS_SAMPLING_DISABLE
5                     PREP_AUTO                            ON
6           SVMS_CONV_TOLERANCE                         .0001
7          SVMS_KERNEL_FUNCTION                   SVMS_LINEAR

Computed Settings:
             setting name    setting value
0  SVMS_COMPLEXITY_FACTOR               10
1     SVMS_NUM_ITERATIONS               30
2             SVMS_SOLVER  SVMS_SOLVER_IPM

Global Statistics:
  attribute name attribute value
0      CONVERGED             YES
1     ITERATIONS              14
2       NUM_ROWS             104

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

COEFFICIENTS:

    TARGET_VALUE ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE    COEF
 0        setosa   Petal_Length              None            None -0.5809
 1        setosa    Petal_Width              None            None -0.7736
 2        setosa   Sepal_Length              None            None -0.1653
```

(continues on next page)

(continued from previous page)

```
3        setosa    Sepal_Width             None          None  0.5689
4        setosa           None             None          None -0.7355
5    versicolor   Petal_Length             None          None  1.1304
6    versicolor    Petal_Width             None          None -0.3323
7    versicolor   Sepal_Length             None          None -0.8877
8    versicolor    Sepal_Width             None          None -1.2582
9    versicolor           None             None          None -0.9091
10    virginica   Petal_Length             None          None  4.6042
11    virginica    Petal_Width             None          None  4.0681
12    virginica   Sepal_Length             None          None -0.7985
13    virginica    Sepal_Width             None          None -0.4328
14    virginica           None             None          None -5.3180
```

```
>>> svm_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Petal_Length', 'Species']])
    Sepal_Length  Sepal_Width  Petal_Length    Species  PREDICTION
0            4.9          3.0           1.4     setosa      setosa
1            4.9          3.1           1.5     setosa      setosa
2            4.8          3.4           1.6     setosa      setosa
3            5.8          4.0           1.2     setosa      setosa
...          ...          ...           ...        ...         ...
42           6.7          3.3           5.7  virginica   virginica
43           6.7          3.0           5.2  virginica   virginica
44           6.5          3.0           5.2  virginica   virginica
45           5.9          3.0           5.1  virginica   virginica
```

```
>>> svm_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Species']], proba = True)
    Sepal_Length  Sepal_Width    Species  PREDICTION  PROBABILITY
0            4.9          3.0     setosa      setosa     0.782169
1            4.9          3.1     setosa      setosa     0.819041
2            4.8          3.4     setosa      setosa     0.919891
3            5.8          4.0     setosa      setosa     0.998816
...          ...          ...        ...         ...          ...
42           6.7          3.3  virginica   virginica     0.920812
43           6.7          3.0  virginica   virginica     0.857884
44           6.5          3.0  virginica   virginica     0.794518
45           5.9          3.0  virginica   virginica     0.660109
```

```
>>> svm_mod.predict_proba(test_dat.drop('Species'), supplemental_cols = test_dat[:, [
↪'Sepal_Length', 'Sepal_Width', 'Species']], topN = 1).sort_values(by = ['Sepal_
↪Length', 'Sepal_Width'])
    Sepal_Length  Sepal_Width     Species       TOP_1  TOP_1_VAL
0            4.4          3.0      setosa      setosa   0.697652
1            4.4          3.2      setosa      setosa   0.810250
2            4.5          2.3      setosa  versicolor   0.600686
3            4.8          3.4      setosa      setosa   0.919891
...          ...          ...         ...         ...        ...
42           6.7          3.3   virginica   virginica   0.920812
43           6.9          3.1  versicolor   virginica   0.367307
44           6.9          3.1   virginica   virginica   0.883710
45           7.0          3.2  versicolor      setosa   0.618889
```

```
>>> svm_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.869565
```

Change parameter setting and refit model.

```
>>> new_setting = {'SVMS_CONV_TOLERANCE': '0.01'}
>>> svm_mod.set_params(**new_setting).fit(train_x, train_y)

Algorithm Name: Support Vector Machine

Mining Function: CLASSIFICATION

Target: Species

Settings:
                   setting name                  setting value
0                     ALGO_NAME  ALGO_SUPPORT_VECTOR_MACHINES
1          CLAS_WEIGHTS_BALANCED                           OFF
2                  ODMS_DETAILS                   ODMS_ENABLE
3  ODMS_MISSING_VALUE_TREATMENT       ODMS_MISSING_VALUE_AUTO
4                 ODMS_SAMPLING         ODMS_SAMPLING_DISABLE
5                     PREP_AUTO                            ON
6           SVMS_CONV_TOLERANCE                          0.01
7          SVMS_KERNEL_FUNCTION                   SVMS_LINEAR

Computed Settings:
             setting name    setting value
0  SVMS_COMPLEXITY_FACTOR               10
1     SVMS_NUM_ITERATIONS               30
2             SVMS_SOLVER  SVMS_SOLVER_IPM

Global Statistics:
  attribute name attribute value
0      CONVERGED             YES
1     ITERATIONS              13
2       NUM_ROWS             104

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

COEFFICIENTS:

   TARGET_VALUE ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE    COEF
0        setosa   Petal_Length              None            None -0.5818
1        setosa    Petal_Width              None            None -0.7731
2        setosa   Sepal_Length              None            None -0.1648
3        setosa    Sepal_Width              None            None  0.5689
4        setosa           None              None            None -0.7354
5    versicolor   Petal_Length              None            None  1.1304
6    versicolor    Petal_Width              None            None -0.3323
7    versicolor   Sepal_Length              None            None -0.8877
8    versicolor    Sepal_Width              None            None -1.2582
9    versicolor           None              None            None -0.9091
10    virginica   Petal_Length              None            None  4.6042
11    virginica    Petal_Width              None            None  4.0682
12    virginica   Sepal_Length              None            None -0.7985
```

(continues on next page)

(continued from previous page)

```
13    virginica    Sepal_Width                 None            None -0.4329
14    virginica           None                 None            None -5.3180
```

Anomaly detection (1-class SVM) example

```
>>> dat = oml.sync(table = "IRIS").split()
>>> train_x = dat[0]
>>> test_dat = dat[1]
>>> new_dat = oml.push(pd.DataFrame([(100.0, 3.5, 1.5, 2.0, "setosa"),
...                                   (1.0, 3.5, 1.5, 0, "setosa"),
...                                   (2.5, 3.8, 5.3, 10.0, "versicolor"),
...                                   (2.5, -2.8, 5.3, 1.4, "versicolor"),
...                                   (6.6, 4.1, 2.2, 1.2, "unknown")], columns =
↪test_dat.columns))
>>> test_dat_anomaly = test_dat.append(new_dat)
```

User specified settings.

```
>>> odm_settings = {'svms_outlier_rate' : 0.01}
```

Create SVM model object.

```
>>> svm_mod = oml.svm("anomaly_detection", **odm_settings)
```

Fit the SVM Model according to the training data and parameter settings.

```
>>> svm_mod.fit(train_x, None)

Algorithm Name: Support Vector Machine

Mining Function: ANOMALY_DETECTION

Settings:
                   setting name                  setting value
0                     ALGO_NAME  ALGO_SUPPORT_VECTOR_MACHINES
1                  ODMS_DETAILS                   ODMS_ENABLE
2  ODMS_MISSING_VALUE_TREATMENT       ODMS_MISSING_VALUE_AUTO
3                 ODMS_SAMPLING         ODMS_SAMPLING_DISABLE
4                     PREP_AUTO                            ON
5           SVMS_CONV_TOLERANCE                         .0001
6          SVMS_KERNEL_FUNCTION                   SVMS_LINEAR
7             SVMS_OUTLIER_RATE                          0.01

Computed Settings:
              setting name    setting value
0  ODMS_EXPLOSION_MIN_SUPP                1
1   SVMS_COMPLEXITY_FACTOR               10
2      SVMS_NUM_ITERATIONS               30
3              SVMS_SOLVER  SVMS_SOLVER_IPM

Global Statistics:
  attribute name attribute value
0      CONVERGED             YES
1     ITERATIONS              12
2       NUM_ROWS             104
```

(continues on next page)

(continued from previous page)

```
Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

COEFFICIENTS:

   TARGET_VALUE ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE    COEF
0             1   Petal_Length              None            None  1.0946
1             1    Petal_Width              None            None  1.0293
2             1   Sepal_Length              None            None  0.9284
3             1    Sepal_Width              None            None  1.0978
4             1        Species              None          setosa  1.2018
5             1        Species              None      versicolor  0.8320
6             1        Species              None       virginica  0.6450
7             1           None              None            None -1.0000
```

```
>>> svm_pred = svm_mod.predict(test_dat_anomaly, supplemental_cols = test_dat_
↪anomaly[:,['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width', 'Species
↪']], proba = True)
>>> svm_pred[svm_pred["PREDICTION"] == 0, :].sort_values(by = ['Sepal_Length', 'Sepal_
↪Width'])
    Sepal_Length  Sepal_Width  Petal_Length  Petal_Width     Species  \
 0           1.0          3.5           1.5          0.0      setosa
 1           2.5         -2.8           5.3          1.4  versicolor
 2           2.5          3.8           5.3         10.0  versicolor
 3           6.6          4.1           2.2          1.2     unknown
 4         100.0          3.5           1.5          2.0      setosa

    PREDICTION  PROBABILITY
 0           0     0.705993
 1           0     0.938090
 2           0     0.689538
 3           0     0.727253
 4           0     0.931758
```

```
>>> oml.drop('IRIS')
```

2.14 XGBoost

class oml.**xgb**(*mining_function='CLASSIFICATION'*, *model_name=None*, *model_owner=None*, ***params*)

In-database XGBoost Model

XGBoost is a highly-efficient, scalable gradient tree boosting machine learning algorithm for regression and classification. The XGBoost algorithm prepares training data, builds and persists a model, and applies the model for prediction. It can be used as a stand-alone predictor or be incorporated 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.

Attributes

importance : oml.DataFrame

The coefficients of the XGBoost model, one for each predictor variable. It includes the following components:

For classification or ranking:

- pname: The partition name
- attribute_name: The attribute name
- attribute_subname: The attribute subname
- attribute_value: The attribute value
- gain: The relative contribution of the corresponding feature to the model
- cover: The relative number of observations related to this feature
- frequency: The percentage representing the relative number of times a feature occurs in the trees of the model

For regression:

- pname: The partition name
- attribute_name: The attribute name
- attribute_subname: The attribute subname
- attribute_value: The attribute value
- weight: The importance of the corresponding feature to the model
- class: The class name

See also:

`svm`, `rf`, `glm`

`__init__`(*mining_function='CLASSIFICATION'*, *model_name=None*, *model_owner=None*, ***params*)

Initializes an instance of XGBoost object.

Parameters

mining_function
['CLASSIFICATION' (default) or 'REGRESSION'] Type of model mining functionality. Should be compatible with xgboost_objective parameter setting. For example, mining_function must be 'classification' when setting objective is 'binary:logistic' and mining_function must be 'regression' when setting objective is 'rank:pairwise'

model_name
[string or None (default)] The name of an existing database XGBoost model to create an oml.xgb object from. The specified database model is not dropped when the oml.xgb object is deleted.

model_owner: string or None (default)
The owner name of the existing XGBoost model The current database user by default

params
[key-value pairs or dict] Oracle Machine Learning parameter settings. Each list element's name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 4-10 apply generally to the model. Mining Function Settings for Classification and Algorithm-specific Settings are applicable to XGBoost model.

export_sermodel(*table=None*, *partition=None*)
Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

fit(*x*, *y*, *model_name=None*, *case_id=None*, *ctx_settings=None*)
Fits an XGBoost Model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model.

y [a single column OML object or None, or string] Target values. Must be specified when XGBoost algorithm is used for classification or regression and must be None when used for anomaly detection. If y is a single column OML object, target values specified by y must be combinable with x. If y is a string, y is the name of the column in x that specifies the target values.

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.xgb object is deleted. If None, a system-generated model name will be used. The system-generated model is dropped when oml.xgb object is deleted unless oml.xgb object is saved into a datastore.

case_id
[string or None (default)] The name of a column that contains unique case identifiers.

ctx_settings
[dict or None (default)] A list to specify Oracle Text attribute-specific settings. This argument is applicable to building models in Oracle Database 12.2 or later. The name of each list element refers to the text column while the list value is a scalar string specifying the attribute-specific text transformation. The valid entries in the string include TEXT, POLICY_NAME, TOKEN_TYPE, and MAX_FEATURES.

get_params(*params=None*, *deep=False*)
Fetches parameters of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If `params` is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

predict (*x*, *supplemental_cols=None*, *proba=False*, *topN_attrs=False*)
Makes predictions on new data.

Parameters

x [oml.DataFrame] Predictor values used by the model to generate scores

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

proba
[boolean, False (default)] Returns prediction probability if `proba` is True.

topN_attrs
[boolean, positive integer, False (default)] Returns the top N most influence attributes of the predicted target value for regression if topN_attrs is not False. Returns the top N most influence attributes of the highest probability class for classification if topN_attrs is not False. N is equal to the specified positive integer or 5 if topN_attrs is True.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. For a classification model, the results include the most likely target class and its probability. For a regression model, the results consist of a column for the prediction. For an anomaly detection model, the results include a prediction and its probability. If the prediction is 1, the case is considered normal. If the prediction is 0, the case is considered anomalous. This behavior reflects the fact that the model is trained with normal data.

predict_proba (*x*, *supplemental_cols=None*, *topN=None*)
Makes predictions and returns probability for each class on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN
[positive integer or None (default)] A positive integer that restricts the returned target classes to the specified number of those that have the highest probability.

Returns

pred
[oml.DataFrame] Contains the features specified by `supplemental_cols` and the results. The results include for each target class, the probability belonging to that class.

set_params(***params*)
Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

References

- Oracle R Enterprise
- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

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

Create table containing dataset iris from sklearn.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width', 'Petal_
↪Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:'virginica'}
↪[x], iris.target)), columns = ['Species'])
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create training data and test data.

```
>>> dat = oml.sync(table = "IRIS").split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
```

Classification Example:

Create XGB model object.

```
>>> setting = {'xgboost_max_depth': '3',
...             'xgboost_eta': '1',
...             'xgboost_num_round': '10'}
>>> xgb_mod = oml.xgb('classification', **setting)
```

Fit the XGB Model according to the training data and parameter settings.

```
>>> xgb_mod.fit(train_x, train_y)

Algorithm Name: XGBOOST

Mining Function: CLASSIFICATION

Target: Species

Settings:
                    setting name            setting value
0                      ALGO_NAME             ALGO_XGBOOST
1          CLAS_WEIGHTS_BALANCED                      OFF
2                   ODMS_DETAILS              ODMS_ENABLE
3    ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
4                  ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
5                      PREP_AUTO                       ON
6                        booster                   gbtree
7                            eta                        1
8                      max_depth                        3
9                    ntree_limit                        0
10                     num_round                       10
11                     objective          multi:softprob

Global Statistics:
  attribute name attribute value
0       NUM_ROWS             104
1       mlogloss      0.02485...

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

ATTRIBUTE IMPORTANCE:

  PNAME ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE      GAIN     COVER  \
0  None   Petal_Length              None            None  0.743941  0.560554
```

(continues on next page)

(continued from previous page)

```
1  None    Petal_Width              None            None  0.162191  0.245400
2  None   Sepal_Length              None            None  0.003738  0.044741
3  None    Sepal_Width              None            None  0.090129  0.149306

   FREQUENCY
0   0.447761
1   0.268657
2   0.119403
3   0.164179
```

```
>>> xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Petal_Length', 'Species']])
    Sepal_Length  Sepal_Width  Petal_Length     Species  PREDICTION
 0           4.9          3.0           1.4      setosa      setosa
 1           4.9          3.1           1.5      setosa      setosa
 2           4.8          3.4           1.6      setosa      setosa
 3           5.8          4.0           1.2      setosa      setosa
...          ...          ...           ...         ...         ...
 42          6.7          3.3           5.7   virginica   virginica
 43          6.7          3.0           5.2   virginica   virginica
 44          6.5          3.0           5.2   virginica   virginica
 45          5.9          3.0           5.1   virginica   virginica
```

```
>>> xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Species']], proba = True)
    Sepal_Length  Sepal_Width     Species  PREDICTION  PROBABILITY
 0           4.9          3.0      setosa      setosa     0.993619
 1           4.9          3.1      setosa      setosa     0.993619
 2           4.8          3.4      setosa      setosa     0.993619
 3           5.8          4.0      setosa      setosa     0.988620
...          ...          ...         ...         ...          ...
 42          6.7          3.3   virginica   virginica     0.996170
 43          6.7          3.0   virginica   virginica     0.993017
 44          6.5          3.0   virginica   virginica     0.993017
 45          5.9          3.0   virginica   virginica     0.980192
```

```
>>> xgb_mod.predict_proba(test_dat.drop('Species'), supplemental_cols = test_dat[:, [
↪'Sepal_Length', 'Sepal_Width', 'Species']], topN = 1).sort_values(by = ['Sepal_
↪Length', 'Sepal_Width'])
    Sepal_Length  Sepal_Width     Species       TOP_1  TOP_1_VAL
 0           4.4          3.0      setosa      setosa   0.993619
 1           4.4          3.2      setosa      setosa   0.993619
 2           4.5          2.3      setosa      setosa   0.942128
 3           4.8          3.4      setosa      setosa   0.993619
...          ...          ...         ...         ...        ...
 42          6.7          3.3   virginica   virginica   0.996170
 43          6.9          3.1  versicolor  versicolor   0.925217
 44          6.9          3.1   virginica   virginica   0.996170
 45          7.0          3.2  versicolor  versicolor   0.990586
```

Regression Example:

Create training data and test data.

```
>>> dat = oml.sync(table = "IRIS").split()
>>> train_x = dat[0].drop('Sepal_Length')
```

(continues on next page)

(continued from previous page)

```
>>> train_y = dat[0]['Sepal_Length']
>>> test_dat = dat[1]
```

Create XGB model object.

```
>>> setting = {'xgboost_booster': 'gblinear'}
>>> xgb_mod = oml.xgb('regression', **setting)
```

Fit the XGB Model according to the training data and parameter settings.

```
>>> xgb_mod.fit(train_x, train_y)

Algorithm Name: XGBOOST

Mining Function: REGRESSION

Target: Sepal_Length

Settings:
                   setting name            setting value
0                     ALGO_NAME             ALGO_XGBOOST
1                  ODMS_DETAILS              ODMS_ENABLE
2  ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
3                 ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
4                     PREP_AUTO                       ON
5                       booster                 gblinear
6                   ntree_limit                        0
7                     num_round                       10

Computed Settings:
              setting name setting value
0  ODMS_EXPLOSION_MIN_SUPP             1

Global Statistics:
  attribute name attribute value
0       NUM_ROWS             104
1           rmse        0.364149

Attributes:
Petal_Length
Petal_Width
Sepal_Width
Species

Partition: NO

ATTRIBUTE IMPORTANCE:

  PNAME ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE    WEIGHT  CLASS
0  None   Petal_Length              None            None  0.335183      0
1  None    Petal_Width              None            None  0.368738      0
2  None    Sepal_Width              None            None  0.249208      0
3  None        Species              None      versicolor -0.197582      0
4  None        Species              None       virginica -0.170522      0
```

```
>>> xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Petal_Length', 'Species']])
```

(continues on next page)

(continued from previous page)

```
    Sepal_Length  Sepal_Width  Petal_Length     Species  PREDICTION
0            4.9          3.0           1.4      setosa    4.797075
1            4.9          3.1           1.5      setosa    4.818641
2            4.8          3.4           1.6      setosa    4.963796
3            5.8          4.0           1.2      setosa    4.979247
...          ...          ...           ...         ...         ...
42           6.7          3.3           5.7   virginica    6.990700
43           6.7          3.0           5.2   virginica    6.674599
44           6.5          3.0           5.2   virginica    6.563977
45           5.9          3.0           5.1   virginica    6.456711
```

Ranking Example:

Create XGB model object.

```
>>> setting = {'xgboost_objective': 'rank:pairwise',
...            'xgboost_max_depth': '3',
...            'xgboost_eta': '0.1',
...            'xgboost_gamma': '1.0',
...            'xgboost_num_round': '4'}
>>> xgb_mod = oml.xgb('regression', **setting)
```

Fit the XGB Model according to the training data and parameter settings.

```
>>> xgb_mod.fit(train_x, train_y)

Algorithm Name: XGBOOST

Mining Function: REGRESSION

Target: Sepal_Length

Settings:
                    setting name            setting value
0                      ALGO_NAME             ALGO_XGBOOST
1                   ODMS_DETAILS              ODMS_ENABLE
2    ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
3                  ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
4                      PREP_AUTO                       ON
5                        booster                   gbtree
6                            eta                      0.1
7                          gamma                      1.0
8                      max_depth                        3
9                    ntree_limit                        0
10                     num_round                        4
11                     objective            rank:pairwise

Computed Settings:
              setting name setting value
0   ODMS_EXPLOSION_MIN_SUPP            1

Global Statistics:
  attribute name  attribute value
0       NUM_ROWS              104
1            map                1

Attributes:
```

(continues on next page)

(continued from previous page)

```
Petal_Length
Petal_Width
Sepal_Width
Species

Partition: NO

ATTRIBUTE IMPORTANCE:

  PNAME ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE      GAIN     COVER  \
0  None   Petal_Length              None            None  0.873855  0.677624
1  None    Petal_Width              None            None  0.083504  0.184802
2  None    Sepal_Width              None            None  0.042641  0.137574

   FREQUENCY
0   0.500000
1   0.285714
2   0.214286
```

```
>>> xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:, ['Sepal_
↪Length', 'Sepal_Width', 'Petal_Length', 'Species']])
     Sepal_Length  Sepal_Width  Petal_Length    Species  PREDICTION
 0            4.9          3.0           1.4     setosa    0.243485
 1            4.9          3.1           1.5     setosa    0.243485
 2            4.8          3.4           1.6     setosa    0.243485
 3            5.8          4.0           1.2     setosa    0.310980
...           ...          ...           ...        ...         ...
 42           6.7          3.3           5.7  virginica    0.771761
 43           6.7          3.0           5.2  virginica    0.728637
 44           6.5          3.0           5.2  virginica    0.728637
 45           5.9          3.0           5.1  virginica    0.674835
```

```
>>> oml.drop('IRIS')
```

2.15 Non-Negative Matrix Factorization

class oml.**nmf**(*model_name=None*, *model_owner=None*, ***params*)

In-database Non-Negative Matrix Factorization Model

Non-Negative Matrix Factorization is a state of the art feature extraction algorithm. NMF is useful when there are many attributes and the attributes are ambiguous or have weak predictability. By combining attributes, NMF can produce meaningful patterns, topics, or themes. Each feature created by NMF is a linear combination of the original attribute set. Each feature has a set of coefficients, which are a measure of the weight of each attribute on the feature. There is a separate coefficient for each numerical attribute and for each distinct value of each categorical attribute. The coefficients are all non-negative. Non-Negative Matrix Factorization uses techniques from multivariate analysis and linear algebra. It decomposes the data as a matrix M into the product of two lower ranking matrices W and H. The sub-matrix W contains the NMF basis; the sub-matrix H contains the associated coefficients (weights).

Attributes

h : oml.DataFrame

Features extracted by the fitted model including feature id and associated coefficient. It includes the following components:

- feature_id: The ID of a feature in the model
- feature_id: The name of a feature in the model
- attribute_name: The attribute name
- attribute_subname: The attribute subname
- attribute_value: The attribute value
- coefficient: The associated feature coefficients (weights)

w : oml.DataFrame

A dataframe containing the NMF basis. It includes the following components:

- feature_id: The ID of a feature in the model
- feature_id: The name of a feature in the model
- attribute_name: The attribute name
- attribute_subname: The attribute subname
- attribute_value: The attribute value
- coefficient: The associated inverse feature coefficients (weights)

See also:

`svd`, `km`, `esa`

`__init__`(*model_name=None*, *model_owner=None*, ***params*)

Initializes an instance of nmf object.

Parameters

model_name
[string or None (default)] The name of an existing database Non-Negative Matrix Factorization model to create an oml.nmf object from. The specified database model is not dropped when the oml.nmf object is deleted.

model_owner: string or None (default)
The owner name of the existing Non-Negative Matrix Factorization model The current database user by default

params
[key-value pairs or dict] Oracle Machine Learning parameter settings. Each list element's name and value refer to the parameter setting name and value, respectively. The setting value must be numeric or string. Refer to Oracle Data Mining Model Settings for applicable parameters and valid values. Global and Automatic Data Preparation Settings in Table 4-10 apply generally to the model. Mining Function Settings for Feature Extraction and Algorithm-specific Settings are applicable to Non-Negative Matrix Factorization model.

`export_sermodel`(*table=None*, *partition=None*)

Export model.

Parameters

table
[string or None (default)] A name for the new table where the serialized model is saved. If None, the serialized model will be saved to a temporary table.

partition
[string or None (default)] Name of the partition that needs to be exported. If partition is None, all partitions are exported

Returns

oml_bytes
[an oml.Bytes object] Contains the BLOB content from the model export

feature_compare (*x*, *compare_cols=None*, *supplemental_cols=None*)
Compares features of data and generates relatedness.

Parameters

x [an OML object] The data used to measure relatedness.

compare_cols
[str, a list of str or None (default)] The column(s) used to measure data relatedness. If None, all the columns of `x` are compared to measure relatedness.

supplemental_cols
[a list of str or None (default)] A list of columns to display along with the resulting 'SIMILARITY' column.

Returns

pred
[oml.DataFrame] Contains a 'SIMILARITY' column that measures relatedness and supplementary columns if specified.

fit (*x*, *model_name=None*, *case_id=None*, *ctx_settings=None*)
Fits an NMF Model according to the training data and parameter settings.

Parameters

x [an OML object] Attribute values for building the model.

model_name
[string or None (default)] User-specified model name. The user-specified database model is not dropped when oml.nmf object is deleted. If None, a system-generated model name will be used. The system-generated model is dropped when oml.nmf object is deleted unless oml.nmf object is saved into a datastore.

case_id
[string or None (default)] The column name used as case id for building the model.

ctx_settings
[dict or None (default)] A list to specify Oracle Text attribute-specific settings. This argument is applicable to building models in Oracle Database 12.2 or later. The name of each list element refers to the text column while the list value is a scalar string specifying the attribute-specific text transformation. The valid entries in the string include TEXT, POLICY_NAME, TOKEN_TYPE, and MAX_FEATURES.

get_params (*params=None*, *deep=False*)
Fetches parameters of the model.

Parameters

params
[iterable of strings, None (default)] Names of parameters to fetch. If `params` is None, fetches all settings.

deep
[boolean, False (default)] Includes the computed and default parameters or not.

Returns

settings
[dict mapping str to str]

model_name
The given name of database mining model

model_owner
The owner name of database mining model

pivot_limit
The maximum number of classes, clusters, or features for which the predicted probabilities are presented after pivoted

predict (*x*, *supplemental_cols=None*)
Makes predictions on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

Returns

pred
[oml.DataFrame] Contains the predicted feature index on the new data and the specified `supplemental_cols`.

set_params (***params*)
Changes parameters of the model.

Parameters

params
[dict object mapping str to str] The key should be the name of the setting, and the value should be the new setting.

Returns

model
[the model itself.]

transform (*x*, *supplemental_cols=None*, *topN=None*)
Performs dimensionality reduction and returns value for each feature on new data.

Parameters

x [an OML object] Attribute values used by the model to generate scores.

supplemental_cols
[oml.DataFrame, oml.Float, oml.String, or None (default)] Data set presented with the prediction result. It must be concatenatable with `x`.

topN
[positive integer or None (default)] A positive integer that restricts the returned values to the specified number of features that have the highest topN values. If None, all features will be returned.

Returns

pred
[oml.DataFrame] Contains the values of new data after the NMF transform and the specified `supplemental_cols`.

References

- Oracle R Enterprise
- Oracle Data Mining Concepts
- Oracle Data Mining User's Guide

Examples

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

Create table containing dataset iris from sklearn.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width', 'Petal_
↪Length', 'Petal_Width'])
>>> x.insert(0, "ID", range(1, len(x) + 1))
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:'virginica'}
↪[x], iris.target)), columns = ['Species'])
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create training data and test data.

```
>>> dat = oml.sync(table = "IRIS").split()
>>> train_dat = dat[0]
>>> test_dat = dat[1]
```

Create NMF model object.

```
>>> nmf_mod = oml.nmf(ODMS_DETAILS = 'ODMS_ENABLE')
```

Fit the NMF Model according to the training data and parameter settings.

```
>>> nmf_mod = nmf_mod.fit(train_dat)
```

Show model details.

```
>>> nmf_mod

Algorithm Name: Non-Negative Matrix Factorizationx

Mining Function: FEATURE_EXTRACTION

Settings:
                   setting name                    setting value
0                     ALGO_NAME  ALGO_NONNEGATIVE_MATRIX_FACTOR
1           NMFS_CONV_TOLERANCE                             .05
2      NMFS_NONNEGATIVE_SCORING       NMFS_NONNEG_SCORING_ENABLE
3           NMFS_NUM_ITERATIONS                              50
4              NMFS_RANDOM_SEED                              -1
5                  ODMS_DETAILS                     ODMS_ENABLE
6   ODMS_MISSING_VALUE_TREATMENT          ODMS_MISSING_VALUE_AUTO
7                 ODMS_SAMPLING             ODMS_SAMPLING_DISABLE
8                     PREP_AUTO                              ON

Computed Settings:
              setting name setting value
0         FEAT_NUM_FEATURES             2
1       NMFS_NUM_ITERATIONS             2
2   ODMS_EXPLOSION_MIN_SUPP             1

Global Statistics:
  attribute name attribute value
0      CONVERGED             YES
1     CONV_ERROR       0.0444448
2     ITERATIONS               2
3       NUM_ROWS             111
4    SAMPLE_SIZE             111

Attributes:
ID
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

H:

     FEATURE_ID  FEATURE_NAME ATTRIBUTE_NAME ATTRIBUTE_VALUE  COEFFICIENT
 0            1             1             ID            None     0.581551
 1            1             1   Petal_Length            None     0.355323
 2            1             1    Petal_Width            None     0.158492
 3            1             1   Sepal_Length            None     0.656558
...         ...           ...            ...             ...          ...
 12           2             2    Sepal_Width            None     0.248640
 13           2             2        Species          setosa     0.249221
 14           2             2        Species      versicolor     0.042316
 15           2             2        Species       virginica     0.093861

W:
```

(continues on next page)

(continued from previous page)

```
    FEATURE_ID  FEATURE_NAME ATTRIBUTE_NAME ATTRIBUTE_VALUE  COEFFICIENT
0            1             1             ID            None     0.288559
1            1             1   Petal_Length            None    -0.062579
2            1             1    Petal_Width            None    -0.370128
3            1             1   Sepal_Length            None     0.502382
...        ...           ...            ...             ...          ...
12           2             2    Sepal_Width            None     0.082511
13           2             2        Species          setosa     0.353453
14           2             2        Species      versicolor    -0.359264
15           2             2        Species       virginica    -0.275074
```

Use the model to make predictions on test data.

```
>>> nmf_mod.predict(test_dat, supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_
↪Width', 'Petal_Length', 'Species']])
    Sepal_Length  Sepal_Width  Petal_Length     Species  FEATURE_ID
0            5.0          3.6           1.4      setosa           2
1            5.0          3.4           1.5      setosa           2
2            4.4          2.9           1.4      setosa           2
3            4.9          3.1           1.5      setosa           2
...          ...          ...           ...         ...         ...
35           6.9          3.1           5.4   virginica           2
36           5.8          2.7           5.1   virginica           2
37           6.2          3.4           5.4   virginica           2
38           5.9          3.0           5.1   virginica           2
```

```
>>> nmf_mod.transform(test_dat, supplemental_cols = test_dat[:, ['Sepal_Length']],
↪topN = 2).sort_values(by = ['Sepal_Length', 'TOP_1', 'TOP_1_VAL'])
    Sepal_Length  TOP_1  TOP_1_VAL  TOP_2  TOP_2_VAL
0            4.4      2   0.464041      1   0.000000
1            4.4      2   0.482051      1   0.045518
2            4.8      2   0.475169      1   0.083874
3            4.8      2   0.510372      1   0.101880
...          ...    ...        ...    ...        ...
35           7.2      1   0.915012      2   0.850330
36           7.2      1   0.938112      2   0.745207
37           7.6      2   0.980757      1   0.864508
38           7.9      1   1.048287      2   0.947744
```

```
>>> nmf_mod.feature_compare(test_dat, compare_cols = "Sepal_Length", supplemental_
↪cols = ["Species"])
     Species_A  Species_B  SIMILARITY
0       setosa     setosa    1.000000
1       setosa     setosa    0.946651
2       setosa     setosa    0.983637
3       setosa     setosa    0.967275
4       setosa     setosa    0.934549
...        ...        ...         ...
737  virginica  virginica    0.836373
738  virginica  virginica    0.934549
739  virginica  virginica    0.983637
740  virginica  virginica    0.950912

[741 rows x 3 columns]
```

```
>>> nmf_mod.feature_compare(test_dat, compare_cols = ["Sepal_Length", "Petal_Length"],
↪ supplemental_cols = ["Species"])
      Species_A  Species_B  SIMILARITY
0        setosa     setosa    0.990134
1        setosa     setosa    0.929516
2        setosa     setosa    0.976885
3        setosa     setosa    0.953770
...         ...        ...         ...
737   virginica  virginica    0.849758
738   virginica  virginica    0.944063
739   virginica  virginica    0.983637
740   virginica  virginica    0.958018

[741 rows x 3 columns]
```

Set parameters and refit model to produce U matrix output.

```
>>> new_setting = {'nmfs_conv_tolerance':0.05}
>>> nmf_mod2 = nmf_mod.set_params(**new_setting).fit(train_dat, case_id = "ID")
>>> nmf_mod2

Algorithm Name: Non-Negative Matrix Factorizationx

Mining Function: FEATURE_EXTRACTION

Settings:
                   setting name                  setting value
0                     ALGO_NAME  ALGO_NONNEGATIVE_MATRIX_FACTOR
1           NMFS_CONV_TOLERANCE                            0.05
2      NMFS_NONNEGATIVE_SCORING       NMFS_NONNEG_SCORING_ENABLE
3           NMFS_NUM_ITERATIONS                              50
4              NMFS_RANDOM_SEED                              -1
5                  ODMS_DETAILS                     ODMS_ENABLE
6  ODMS_MISSING_VALUE_TREATMENT         ODMS_MISSING_VALUE_AUTO
7                 ODMS_SAMPLING           ODMS_SAMPLING_DISABLE
8                     PREP_AUTO                              ON

Computed Settings:
              setting name setting value
0        FEAT_NUM_FEATURES             2
1      NMFS_NUM_ITERATIONS             8
2  ODMS_EXPLOSION_MIN_SUPP             1

Global Statistics:
  attribute name attribute value
0      CONVERGED             YES
1     CONV_ERROR       0.0277253
2     ITERATIONS               8
3       NUM_ROWS             111
4    SAMPLE_SIZE             111

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species
```

(continues on next page)

(continued from previous page)

```
Partition: NO

H:

    FEATURE_ID  FEATURE_NAME ATTRIBUTE_NAME ATTRIBUTE_VALUE   COEFFICIENT
 0           1             1   Petal_Length            None  9.889792e-02
 1           1             1    Petal_Width            None  1.060984e-01
 2           1             1   Sepal_Length            None  1.947197e-01
 3           1             1    Sepal_Width            None  5.099539e-01
...        ...           ...            ...             ...           ...
 10          2             2    Sepal_Width            None  2.420062e-01
 11          2             2        Species          setosa  1.643131e-08
 12          2             2        Species      versicolor  5.158020e-01
 13          2             2        Species       virginica  4.948837e-01

W:

    FEATURE_ID  FEATURE_NAME ATTRIBUTE_NAME ATTRIBUTE_VALUE  COEFFICIENT
 0           1             1   Petal_Length            None    -0.071259
 1           1             1    Petal_Width            None    -0.059774
 2           1             1   Sepal_Length            None     0.077608
 3           1             1    Sepal_Width            None     0.571981
...        ...           ...            ...             ...          ...
 10          2             2    Sepal_Width            None     0.003195
 11          2             2        Species          setosa    -0.221185
 12          2             2        Species      versicolor     0.325338
 13          2             2        Species       virginica     0.289804
```

```
>>> oml.drop('IRIS')
```

CHAPTER THREE

AUTOMATIC MACHINE LEARNING

3.1 Algorithm Selection

class oml.automl.**AlgorithmSelection**(*mining_function='classification'*, *score_metric=None*, *parallel=None*)

Algorithm Selection automl submodule automatically ranks the best algorithm from a set of supported ML/DL algorithms by only using the characteristics of the dataset and task. Algorithm Selection uses advanced machine learning and meta-learning techniques.

Support Matrix:

OML/OAA algorithms: dt, glm, glm_ridge, nn, rf, svm_gaussian, svm_linear, nb

Mining functions: classification, regression

Scoring metrics per target type: refer to `score_metric` of *oml.automl.ModelTuning.tune()*

__init__(*mining_function='classification'*, *score_metric=None*, *parallel=None*)

Creates a algorithm selection automl object.

Parameters

mining_function
[str] Supported mining functions (same as oml) is 'classification' and 'regression'

score_metric
[str] Accepts supported scoring metric (mining_function dependent)

parallel
[int] Controls degree of parallelism within the submodule, cannot exceed the degree of parallelism limit controlled by service level in ADW. Ignored in Windows. Higher parallelism requires more memory allocation in the database.

select(*X*, *y*, *case_id=None*, *k=3*, *cv='auto'*, *X_valid=None*, *y_valid=None*, *time_budget=None*, *oml_text_policy=None*)

Algorithm Selection automl submodule automatically ranks the best algorithm from a set of supported ML/DL algorithms by only using the characteristics of the dataset and task. Does not tune the Algorithms.

Parameters

X
[oml.DataFrame] Training dataset features

y [oml.DataFrame, str] Target values. If y is a single column OML object, target values specified by y must be combinable with X. If y is a string, y is the name of the column in X that specifies the target values.

case_id
[str] Column name that forms is the case_id column (default: None). Aids in doing any sampling and dataset split

k
[int or None, 3 (default)] returns ranking for only top k best algorithms (default: 3). When k=None returns ranking for all algorithms.

cv
['auto', int, or None] Determines the cross-validation splitting strategy. (Default: 'auto') Valid inputs for cv are: None, to use X_valid and y_valid for validation 'auto' : cv folds determined based on size of the dataset, disabling cv-folds for large datasets integer, to specify the number of folds in a (Stratified)KFold, For integer/None inputs, if the estimator is a classifier and y is either binary or multiclass, StratifiedKFold is used. In all other cases, KFold is used.int or None

X_valid
[oml.DataFrame] Validation dataset features, used for model scoring when cv is disabled (default: None)

y_valid
[oml.DataFrame, str, optional] Target values for model validation when cv is disabled. If y_valid is a single column OML object, target values specified by y_valid must be combinable with X_valid. If y_valid is a string, y_valid is the name of the column in X_valid that specifies the target values. If supplied, `y_valid` must be of the same type as `y` (Default: None)

time_budget
[int or None] Time budget in seconds where None means no time budget constraint (best effort). With insufficient time budget, returned ranking scores would be incomplete or all nan.

oml_text_policy: str
Default value is None. This is the name of the text policy for textual columns such as CLOB.

oml_text_policy
[str] This OML-specific parameter determines the text policy to use when extracting features from text(CLOB) columns. It is assumed that this text policy is already created on the Database. By default, None.

Returns

algo_rankings: list of tuples
sorted list of top k algorithms and their predicted rank scores (best to worst) [(algo_name, pred_score), . . .]

Examples

```
>>> import oml
>>> from oml import automl
>>> import pandas as pd
```

(continues on next page)

(continued from previous page)

```
>>> import numpy as np
>>> from sklearn import datasets
```

Load the breast cancer dataset into the database

```
>>> bc = datasets.load_breast_cancer()
>>> bc_data = bc.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns = bc.feature_names)
>>> y = pd.DataFrame(bc.target, columns = ['TARGET'])
>>> row_id = pd.DataFrame(np.arange(bc_data.shape[0]), columns = ['CASE_ID'])
>>> df = oml.create(pd.concat([row_id, X, y], axis=1), table = 'BreastCancer')
```

Split dataset into train and test

```
>>> train, test = df.split(ratio=(0.8, 0.2), seed = 1234, hash_cols=['CASE_ID'])
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
```

Create an automatic algorithm selection object with f1_macro score_metric

```
>>> asel = automl.AlgorithmSelection(mining_function='classification', score_metric=
↪'f1_macro', parallel=4)
```

Run algorithm selection to get the top k predicted algorithms and their ranking without tuning

```
>>> algo_ranking = asel.select(X, y, case_id='CASE_ID', k=3)
```

Show the selected and tuned model

```
>>> [(m, "{:.2f}".format(s)) for m,s in algo_ranking]
[('svm_gaussian', '0.97'), ('svm_linear', '0.97'), ('glm_ridge', '0.97')]
```

```
>>> oml.drop('BreastCancer')
```

3.2 Feature Selection

class oml.automl.**FeatureSelection**(*mining_function='classification'*, *score_metric=None*, *parallel=None*)

Automated Feature Selection object, uses meta-learning to quickly identify most relevant feature subsets given a training dataset and an OML algorithm.

Support Matrix:

OML/OAA algorithms: dt, glm, glm_ridge, nn, rf, svm_gaussian, svm_linear, nb

Mining functions: classification, regression

Scoring metrics per target type: refer to `score_metric` of *oml.automl.ModelTuning.tune()*

__init__(*mining_function='classification'*, *score_metric=None*, *parallel=None*)

Creates a feature selection automl object.

Parameters

mining_function
[str] Supported mining functions (same as oml) is 'classification' and 'regression'

score_metric
[str] Accepts supported scoring metric (mining_function dependent)

parallel
[int] Controls degree of parallelism within the submodule, cannot exceed the degree of parallelism limit controlled by service level in ADW. Ignored in Windows. Higher parallelism requires more memory allocation in the database.

reduce(*algo_name*, *X*, *y*, *case_id=None*, *cv='auto'*, *adaptive_sampling=True*, *X_valid=None*, *y_valid=None*, *time_budget=None*, *oml_text_policy=None*)
Automatically selects a subset of relevant features from the given dataset for the given algorithm.

Parameters

algo_name
[str] Name of the algorithm (one of the supported algorithms)

X
[oml.DataFrame] Training dataset features

y [oml.DataFrame, str] Target values. If y is a single column OML object, target values specified by y must be combinable with X. If y is a string, y is the name of the column in X that specifies the target values.

case_id
[str] Column name that forms is the case_id column (default: None). Aids in doing any sampling and dataset split

cv
['auto', int, or None] Determines the cross-validation splitting strategy. (Default: 'auto') Valid inputs for cv are: None, to use X_valid and y_valid for validation 'auto' : cv folds determined based on size of the dataset, disabling cv-folds for large datasets integer, to specify the number of folds in a (Stratified)KFold, For integer/None inputs, if the estimator is a classifier and y is either binary or multiclass, StratifiedKFold is used. In all other cases, KFold is used.int or None

adaptive_sampling
[bool] Controls adaptive sampling of dataset to reduce overall runtime. Set to False to disable adaptive sampling. Disabling this might significantly increase runtime. (Default: True)

X_valid
[oml.DataFrame] Validation dataset features, used for model scoring when cv is disabled (default: None)

y_valid
[oml.DataFrame, str, optional] Target values for model validation when cv is disabled. If y_valid is a single column OML object, target values specified by y_valid must be combinable with X_valid. If y_valid is a string, y_valid is the name of the column in X_valid that specifies the target values. If supplied, `y_valid` must be of the same type as `y` (Default: None)

time_budget
[int or None] Time budget in seconds. When None - no time budget constraint (best effort). With insufficient time budget, the feature reduction may be premature with partially optimized feature subset.

oml_text_policy
[str] This OML-specific parameter determines the text policy to use when extracting features from text(CLOB) columns. It is assumed that this text policy is already created on the Database. By default, None.

Returns

selected_features
[list of column names that can be directly]

used on a Dataset dataframe.

Examples

```
>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> from sklearn import datasets
```

Load the digits dataset into the database

```
>>> digits = datasets.load_digits()
>>> X = pd.DataFrame(digits.data, columns = [ 'pixel{}'.format(i) for i in␣
↪range(digits.data.shape[1]) ])
>>> y = pd.DataFrame(digits.target, columns = ['digit'])
>>> df = oml.create(pd.concat([X, y], axis=1), table = 'DIGITS')
```

Split dataset into train and test

```
>>> train, test = df.split(ratio=(0.8, 0.2), seed = 1234, strata_cols='digit')
>>> X_train, y_train = train.drop('digit'), train['digit']
>>> X_test, y_test = test.drop('digit'), test['digit']
```

Default model performance before feature selection

```
>>> mod = oml.svm(mining_function='classification').fit(X_train, y_train)
>>> "{:.2}".format(mod.score(X_test, y_test))
'0.92'
```

Create an automatic feature selection object with accuracy as score_metric

```
>>> fs = automl.FeatureSelection(mining_function='classification', score_metric=
↪'accuracy', parallel=4)
```

Get the reduced feature subset on the train dataset

```
>>> subset = fs.reduce('svm_linear', X_train, y_train)
>>> "{} features reduced to {}".format(len(X_train.columns), len(subset))
'64 features reduced to 25'
```

Use the subset to select the features and create model on the new reduced dataset

```
>>> X_new  = X_train[subset]
>>> X_test_new = X_test[subset]
>>> mod = oml.svm(mining_function='classification').fit(X_new, y_train)
```

(continues on next page)

(continued from previous page)

```
>>> "{:.2} with {:.1f}x feature reduction".format(mod.score(X_test_new, y_test),␣
↪len(X_train.columns)/len(X_new.columns))
'0.92 with 2.6x feature reduction'
```

```
>>> oml.drop('DIGITS')
```

For reproducible results, add a case_id column with unique row identifiers

```
>>> row_id = pd.DataFrame(np.arange(digits.data.shape[0]), columns = ['CASE_ID'])
>>> df_cid = oml.create(pd.concat([row_id, X, y], axis=1), table = 'DIGITS_CID')
```

Split the dataset as before but with case_id as a hash column

```
>>> train, test = df_cid.split(ratio=(0.8, 0.2), seed = 1234, hash_cols='CASE_ID',␣
↪strata_cols='digit')
>>> X_train, y_train = train.drop('digit'), train['digit']
>>> X_test, y_test = test.drop('digit'), test['digit']
```

Provide the case_id column name to the feature selection reduce function

```
>>> subset = fs.reduce('svm_linear', X_train, y_train, case_id='CASE_ID')
>>> "{} features reduced to {} with case_id".format(len(X_train.columns)-1,␣
↪len(subset))
'64 features reduced to 41 with case_id'
```

```
>>> oml.drop('DIGITS_CID')
```

3.3 Model Selection

class oml.automl.**ModelSelection**(*mining_function='classification'*, *score_metric=None*, *parallel=None*)

Model Selection automl submodule automatically selects the best algorithm (using Algorithm Selection) from a set of supported ML/DL algorithms and tunes it.

Support Matrix:

OML/OAA algorithms: dt, glm, glm_ridge, nn, rf, svm_gaussian, svm_linear, nb

Mining functions: classification, regression

Scoring metrics per target type: refer to `score_metric` of `oml.automl.ModelTuning.tune()`

__init__(*mining_function='classification'*, *score_metric=None*, *parallel=None*)

Creates a model selection automl object.

Parameters

mining_function
[str] Supported mining functions (same as oml) is 'classification' and 'regression'

score_metric
[str] Accepts supported scoring metric (mining_function dependent)

parallel
[int] Controls degree of parallelism within the submodule, cannot exceed the degree of parallelism limit controlled by service level in ADW. Ignored in Windows. Higher parallelism requires more memory allocation in the database.

select (*X*, *y*, *case_id=None*, *k=3*, *solver='fast'*, *cv='auto'*, *adaptive_sampling=True*, *X_valid=None*, *y_valid=None*, *time_budget=None*, *oml_text_policy=None*)

ModelSelection.select automatically ranks algorithms, tunes the top k algorithsm and reports the best tuned oml model object for the given training data.

Parameters

X
[oml.DataFrame] Training dataset features

y [oml.DataFrame, str] Target values. If y is a single column OML object, target values specified by y must be combinable with X. If y is a string, y is the name of the column in X that specifies the target values.

case_id
[str] Column name that forms is the case_id column (default: None). Aids in doing any sampling and dataset split

k
[int or None] The number of top-ranked algorithms tuned to find the best model. When k=None, all algorithms are tuned.

solver
[str] Solver used internally by Model Selection. Can be exhaustive or fast (default: fast). fast - Model Selection uses predictive techniques to identify the best models. Much faster than exhaustive. exhaustive - Model Selection exhaustively searches the model space to identify the best model. Slower but can be more accurate.

cv
['auto', int, or None] Determines the cross-validation splitting strategy. (Default: 'auto') Valid inputs for cv are: None, to use X_valid and y_valid for validation 'auto' : cv folds determined based on size of the dataset, disabling cv-folds for large datasets integer, to specify the number of folds in a (Stratified)KFold, For integer/None inputs, if the estimator is a classifier and y is either binary or multiclass, StratifiedKFold is used. In all other cases, KFold is used.int or None

adaptive_sampling
[bool] Controls adaptive sampling of dataset to reduce overall runtime. Set to False to disable adaptive sampling. Disabling this might significantly increase runtime. (Default: True)

X_valid
[oml.DataFrame] Validation dataset features, used for model scoring when cv is disabled (default: None)

y_valid
[oml.DataFrame, str, optional] Target values for model validation when cv is disabled. If y_valid is a single column OML object, target values specified by y_valid must be combinable with X_valid. If y_valid is a string, y_valid is the name of the column in X_valid that specifies the target values. If supplied, `y_valid` must be of the same type as `y` (Default: None)

time_budget
[int or None] Time budget in seconds where None means no time budget constraint (best

effort). With insufficient time budget, returned model may be a result of default or partially tuned hyperparameters.

oml_text_policy: str
Default value is None. This is the name of the text policy for textual columns such as CLOB.

Returns

best_model, algo_name: tuple
where the best model out of top k (oml proxy model), and algo_name is the algorithm name of the best_model (str)

Examples

```
>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> from sklearn import datasets
```

Load the breast cancer dataset into the database

```
>>> bc = datasets.load_breast_cancer()
>>> bc_data = bc.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns = bc.feature_names)
>>> y = pd.DataFrame(bc.target, columns = ['TARGET'])
>>> row_id = pd.DataFrame(np.arange(bc_data.shape[0]), columns = ['CASE_ID'])
>>> df = oml.create(pd.concat([row_id, X, y], axis=1), table = 'BreastCancer')
```

Split dataset into train and test

```
>>> train, test = df.split(ratio=(0.8, 0.2), seed = 1234, hash_cols=['CASE_ID'])
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
```

Create an automatic model selection object with f1_macro score_metric

```
>>> ms = automl.ModelSelection(mining_function='classification', score_metric='f1_
↪macro', parallel=4)
```

Run model selection to get the top (k=1) predicted algorithm (defaults to tuned model)

```
>>> select_model, algo_name = ms.select(X, y, case_id='CASE_ID', k=1)
```

Show the selected and tuned model

```
>>> select_model

Algorithm Name: Neural Network

Mining Function: CLASSIFICATION

Target: TARGET

Settings:
                    setting name                  setting value
```

(continues on next page)

(continued from previous page)

```
0                    ALGO_NAME         ALGO_NEURAL_NETWORK
1         CLAS_WEIGHTS_BALANCED                         ON
2              NNET_ACTIVATIONS  'NNET_ACTIVATIONS_LOG_SIG'
3       NNET_HELDASIDE_MAX_FAIL                          6
4           NNET_HELDASIDE_RATIO                       .25
5             NNET_HIDDEN_LAYERS                         1
6               NNET_ITERATIONS                        100
7           NNET_NODES_PER_LAYER                        17
8              NNET_REGULARIZER  NNET_REGULARIZER_HELDASIDE
9               NNET_TOLERANCE                     .000001
10                ODMS_DETAILS                 ODMS_ENABLE
11  ODMS_MISSING_VALUE_TREATMENT     ODMS_MISSING_VALUE_AUTO
12             ODMS_RANDOM_SEED                          0
13                ODMS_SAMPLING       ODMS_SAMPLING_DISABLE
14                    PREP_AUTO                         ON

Computed Settings:
               setting name               setting value
0  LBFGS_GRADIENT_TOLERANCE  .000000001000000000000001
1       LBFGS_HISTORY_DEPTH                          20
2       LBFGS_SCALE_HESSIAN  LBFGS_SCALE_HESSIAN_ENABLE
3              NNET_SOLVER           NNET_SOLVER_LBFGS

Global Statistics:
  attribute name attribute value
0      CONVERGED             YES
1     ITERATIONS              14
2     LOSS_VALUE        0.125772
3       NUM_ROWS             455

Attributes:
area error
compactness error
concave points error
concavity error
fractal dimension error
mean area
mean compactness
mean concave points
mean concavity
mean fractal dimension
mean perimeter
mean radius
mean smoothness
mean symmetry
mean texture
perimeter error
radius error
smoothness error
symmetry error
texture error
worst area
worst compactness
worst concave points
worst concavity
worst fractal dimension
worst perimeter
```

(continues on next page)

(continued from previous page)

```
worst radius
worst smoothness
worst symmetry
worst texture

Partition: NO

Topology:

   HIDDEN_LAYER_ID  NUM_NODE       ACTIVATION_FUNCTION
0                0        17  NNET_ACTIVATIONS_LOG_SIG

Weights:

     LAYER  IDX_FROM  IDX_TO ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE  \
0        0       0.0       0     area error              None            None
1        0       0.0       1     area error              None            None
2        0       0.0       2     area error              None            None
3        0       0.0       3     area error              None            None
4        0       0.0       4     area error              None            None
..     ...       ...     ...            ...               ...             ...
540      1      13.0       0           None              None            None
541      1      14.0       0           None              None            None
542      1      15.0       0           None              None            None
543      1      16.0       0           None              None            None
544      1       NaN       0           None              None            None

     TARGET_VALUE    WEIGHT
0             NaN -0.237091
1             NaN  0.438276
2             NaN  0.009406
3             NaN -0.133564
4             NaN -0.512087
..            ...       ...
540           1.0  0.989600
541           1.0 -1.219637
542           1.0  0.579070
543           1.0  0.104128
544           1.0  0.019274

[545 rows x 8 columns]
```

Score on the selected and tuned model

```
>>> "{:.2}".format(select_model.score(X_test, y_test))
'0.98'
```

Alternative to run model selection to get the top (k=1) predicted algorithm (defaults to tuned model)

```
>>> select_model = ms.select(X, y, case_id='CASE_ID', k=1)
```

Show the algorithm of the selected model

```
>>> select_model[1]
'nn'
```

Show the selected and tuned model

```
>>> select_model[0]

Algorithm Name: Neural Network

Mining Function: CLASSIFICATION

Target: TARGET

Settings:
                    setting name                setting value
0                      ALGO_NAME          ALGO_NEURAL_NETWORK
1          CLAS_WEIGHTS_BALANCED                           ON
2               NNET_ACTIVATIONS  'NNET_ACTIVATIONS_LOG_SIG'
3        NNET_HELDASIDE_MAX_FAIL                            6
4           NNET_HELDASIDE_RATIO                          .25
5             NNET_HIDDEN_LAYERS                            1
6                NNET_ITERATIONS                          100
7           NNET_NODES_PER_LAYER                           17
8               NNET_REGULARIZER  NNET_REGULARIZER_HELDASIDE
9                 NNET_TOLERANCE                      .000001
10                  ODMS_DETAILS                  ODMS_ENABLE
11  ODMS_MISSING_VALUE_TREATMENT      ODMS_MISSING_VALUE_AUTO
12              ODMS_RANDOM_SEED                            0
13                 ODMS_SAMPLING        ODMS_SAMPLING_DISABLE
14                     PREP_AUTO                           ON

Computed Settings:
               setting name               setting value
0  LBFGS_GRADIENT_TOLERANCE  .000000001000000000000001
1       LBFGS_HISTORY_DEPTH                          20
2       LBFGS_SCALE_HESSIAN  LBFGS_SCALE_HESSIAN_ENABLE
3               NNET_SOLVER           NNET_SOLVER_LBFGS

Global Statistics:
  attribute name attribute value
0      CONVERGED             YES
1     ITERATIONS              14
2     LOSS_VALUE        0.125772
3       NUM_ROWS             455

Attributes:
area error
compactness error
concave points error
concavity error
fractal dimension error
mean area
mean compactness
mean concave points
mean concavity
mean fractal dimension
mean perimeter
mean radius
mean smoothness
mean symmetry
mean texture
```

(continues on next page)

(continued from previous page)

```
perimeter error
radius error
smoothness error
symmetry error
texture error
worst area
worst compactness
worst concave points
worst concavity
worst fractal dimension
worst perimeter
worst radius
worst smoothness
worst symmetry
worst texture

Partition: NO

Topology:

   HIDDEN_LAYER_ID  NUM_NODE       ACTIVATION_FUNCTION
0                0        17  NNET_ACTIVATIONS_LOG_SIG

Weights:

     LAYER  IDX_FROM  IDX_TO ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE  \
0        0       0.0       0     area error              None            None
1        0       0.0       1     area error              None            None
2        0       0.0       2     area error              None            None
3        0       0.0       3     area error              None            None
4        0       0.0       4     area error              None            None
..     ...       ...     ...            ...               ...             ...
540      1      13.0       0           None              None            None
541      1      14.0       0           None              None            None
542      1      15.0       0           None              None            None
543      1      16.0       0           None              None            None
544      1       NaN       0           None              None            None

     TARGET_VALUE    WEIGHT
0             NaN -0.237091
1             NaN  0.438276
2             NaN  0.009406
3             NaN -0.133564
4             NaN -0.512087
..            ...       ...
540           1.0  0.989600
541           1.0 -1.219637
542           1.0  0.579070
543           1.0  0.104128
544           1.0  0.019274

[545 rows x 8 columns]
```

Score on the selected and tuned model

```
>>> "{:.2}".format(select_model[0].score(X_test, y_test))
'0.98'
```

```
>>> oml.drop('BreastCancer')
```

3.4 Model Tuning

class oml.automl.**ModelTuning**(*mining_function='classification'*, *score_metric=None*, *parallel=None*)

Model Tuning automl submodule uses a highly parallel, asynchronous gradient-based hyperparameter optimization algorithm to tune the hyperparameters for a given algorithm and training data

Support Matrix:

OML/OAA algorithms: dt, glm, glm_ridge, nn, rf, svm_gaussian, svm_linear, nb

Mining functions: classification, regression

Default param_space used per mining_function, per supported algorithm:

classification :

dt :

`{'TREE_TERM_MINPCT_NODE': {'type': 'continuous', 'range': [0.01, 10.0]}, 'CLAS_MAX_SUP_BINS': {'type': 'discrete', 'range': [2, 250]}, 'TREE_TERM_MINPCT_SPLIT': {'type': 'continuous', 'range': [0.01, 20.0]}, 'TREE_IMPURITY_METRIC': {'type': 'categorical', 'range': ['TREE_IMPURITY_ENTROPY', 'TREE_IMPURITY_GINI']}, 'TREE_TERM_MAX_DEPTH': {'type': 'discrete', 'range': [2, 20]}, 'CLAS_WEIGHTS_BALANCED': {'type': 'categorical', 'range': ['ON', 'OFF']}}`

glm :

`{'GLMS_NUM_ITERATIONS': {'type': 'discrete', 'range': [10, 500]}, 'GLMS_SOLVER': {'type': 'categorical', 'range': ['GLMS_SOLVER_SGD', 'GLMS_SOLVER_CHOL']}, 'CLAS_WEIGHTS_BALANCED': {'type': 'categorical', 'range': ['ON', 'OFF']}}`

glm_ridge :

`{'GLMS_RIDGE_VALUE': {'type': 'continuous', 'range': [0.01, 500]}, 'GLMS_NUM_ITERATIONS': {'type': 'discrete', 'range': [10, 500]}, 'GLMS_SOLVER': {'type': 'categorical', 'range': ['GLMS_SOLVER_SGD', 'GLMS_SOLVER_CHOL']}, 'CLAS_WEIGHTS_BALANCED': {'type': 'categorical', 'range': ['ON', 'OFF']}}`

nb :

`{'NABS_SINGLETON_THRESHOLD': {'type': 'continuous', 'range': [0, 0.1]}, 'NABS_PAIRWISE_THRESHOLD': {'type': 'continuous', 'range': [0, 0.1]}, 'CLAS_WEIGHTS_BALANCED': {'type': 'categorical', 'range': ['ON', 'OFF']}}`

nn :

```
{'NNET_HIDDEN_LAYERS': {'type': 'discrete',
'range': [1, 2]}, 'NNET_ALL_ACTIVATIONS': {'type':
'categorical', 'range': ['NNET_ACTIVATIONS_LOG_SIG',
'NNET_ACTIVATIONS_LINEAR', 'NNET_ACTIVATIONS_TANH',
'NNET_ACTIVATIONS_ARCTAN', 'NNET_ACTIVATIONS_BIPOLAR_SIG']},
'NNET_NODES': {'type': 'discrete', 'range': [1, 50]},
'NNET_REGULARIZER': {'type': 'categorical', 'range':
['NNET_REGULARIZER_HELDASIDE', 'NNET_REGULARIZER_L2',
'NNET_REGULARIZER_NONE']}, 'CLAS_WEIGHTS_BALANCED': {'type':
'categorical', 'range': ['ON', 'OFF']}}
```

rf :

```
{'RFOR_SAMPLING_RATIO': {'type': 'continuous',
'range': [0.01, 1]}, 'TREE_TERM_MINPCT_NODE':
{'type': 'continuous', 'range': [0.01, 10.0]},
'CLAS_MAX_SUP_BINS': {'type': 'discrete', 'range': [2,
250]}, 'TREE_TERM_MAX_DEPTH': {'type': 'discrete',
'range': [2, 20]}, 'RFOR_NUM_TREES': {'type': 'discrete',
'range': [5, 300]}, 'TREE_IMPURITY_METRIC': {'type':
'categorical', 'range': ['TREE_IMPURITY_ENTROPY',
'TREE_IMPURITY_GINI']}, 'RFOR_MTRY': {'type': 'discrete',
'range': [1, '$nr_features']}, 'TREE_TERM_MINPCT_SPLIT':
{'type': 'continuous', 'range': [0.01, 20.0]},
'CLAS_WEIGHTS_BALANCED': {'type': 'categorical', 'range':
['ON', 'OFF']}}
```

svm_gaussian :

```
{'SVMS_COMPLEXITY_FACTOR': {'type': 'continuous', 'range':
[0.1, 100]}, 'SVMS_NUM_PIVOTS': {'type': 'discrete',
'range': [1, 600]}, 'SVMS_STD_DEV': {'type': 'continuous',
'range': [0.1, 16]}, 'CLAS_WEIGHTS_BALANCED': {'type':
'categorical', 'range': ['ON', 'OFF']}}
```

svm_linear :

```
{'SVMS_COMPLEXITY_FACTOR': {'type': 'continuous', 'range':
[0.1, 100]}, 'SVMS_SOLVER': {'type': 'categorical',
'range': ['SVMS_SOLVER_SGD', 'SVMS_SOLVER_IPM']},
'CLAS_WEIGHTS_BALANCED': {'type': 'categorical', 'range':
['ON', 'OFF']}}
```

regression :

glm :

```
{'GLMS_NUM_ITERATIONS': {'type': 'discrete', 'range': [10,
500]}, 'GLMS_SOLVER': {'type': 'categorical', 'range':
['GLMS_SOLVER_SGD', 'GLMS_SOLVER_CHOL']}}
```

glm_ridge :

```
{'GLMS_RIDGE_VALUE': {'type': 'continuous', 'range': [0.01,
500]}, 'GLMS_NUM_ITERATIONS': {'type': 'discrete', 'range':
[10, 500]}, 'GLMS_SOLVER': {'type': 'categorical', 'range':
['GLMS_SOLVER_SGD', 'GLMS_SOLVER_CHOL', 'GLMS_SOLVER_QR',
'GLMS_SOLVER_LBFGS_ADMM']}}
```

nn :

`{'NNET_HIDDEN_LAYERS': {'type': 'discrete', 'range': [1, 2]}, 'NNET_ALL_ACTIVATIONS': {'type': 'categorical', 'range': ['NNET_ACTIVATIONS_LOG_SIG', 'NNET_ACTIVATIONS_LINEAR', 'NNET_ACTIVATIONS_TANH', 'NNET_ACTIVATIONS_ARCTAN', 'NNET_ACTIVATIONS_BIPOLAR_SIG']}, 'NNET_NODES': {'type': 'discrete', 'range': [1, 50]}, 'NNET_REGULARIZER': {'type': 'categorical', 'range': ['NNET_REGULARIZER_HELDASIDE', 'NNET_REGULARIZER_L2', 'NNET_REGULARIZER_NONE']}}`

svm_gaussian :

`{'SVMS_STD_DEV': {'type': 'continuous', 'range': [0.1, 16]}, 'SVMS_COMPLEXITY_FACTOR': {'type': 'continuous', 'range': [0.1, 100]}, 'SVMS_NUM_PIVOTS': {'type': 'discrete', 'range': [1, 600]}, 'SVMS_EPSILON': {'type': 'continuous', 'range': [0.001, 0.5]}}`

svm_linear :

`{'SVMS_COMPLEXITY_FACTOR': {'type': 'continuous', 'range': [0.1, 100]}}`

`__init__` (*mining_function='classification'*, *score_metric=None*, *parallel=None*)

Creates a model tuning automl object.

Parameters

mining_function
[str] Supported mining functions (same as oml) is 'classification' and 'regression'

score_metric
[str or None] Accepts a score metric from the list of supported scoring metrics for model tuning (default: None, equivalent to mining function specific metrics. 'accuracy' for classification, and 'r2' for regression).

parallel
[int] Controls degree of parallelism within the submodule, cannot exceed the degree of parallelism limit controlled by service level in ADW. Ignored in Windows. Higher parallelism requires more memory allocation in the database.

Supported score_metric per target type

Defaults for: binary_classification is *balanced_accuracy*, multi_classification is *balanced_accuracy*, regression is *neg_mean_squared_error*

binary_classification – balanced_accuracy, accuracy, f1, precision, recall, roc_auc, f1_micro, f1_macro, f1_weighted, f1_binary, recall_micro, recall_macro, recall_weighted, recall_binary, precision_micro, precision_macro, precision_weighted, precision_binary

multi_classification – balanced_accuracy, accuracy, f1_micro, f1_macro, f1_weighted, recall_micro, recall_macro, recall_weighted, precision_micro, precision_macro, precision_weighted

regression – neg_mean_squared_error, r2, neg_mean_absolute_error, neg_median_absolute_error

More information on scoring metrics can be found here: Classification metrics, Regression metrics.

Note: Scoring variations like `recall_macro` is equivalent to `sklearn.metrics.recall_score(..., average='macro')` score_metric='f1' requires positive target to be encoded as 1 by default.

tune(*algo_name*, *X*, *y*, *case_id=None*, *param_space='default'*, *K=4*, *cv='auto'*, *adaptive_sampling=True*, *X_valid=None*, *y_valid=None*, *time_budget=None*)

ModelTuning.tune identifies and reports the best tuned oml model object for the given training data.

Parameters

algo_name
[str] Name of the algorithm (one of the supported algorithms)

X
[oml.DataFrame] Training dataset features

y [oml.DataFrame, str] Target values. If y is a single column OML object, target values specified by y must be combinable with X. If y is a string, y is the name of the column in X that specifies the target values.

case_id
[str] Column name that forms is the case_id column (default: None). Aids in doing any sampling and dataset split efficiently

param_space
[dict] The search ranges for each hyperparameter (default: none, uses predefined model specific hyperparameter search space). Example param_space input for rf model tuning: ``{{ 'RFOR_NUM_TREES': {{ 'range': [5, 500], 'type': 'discrete'}},

'RFOR_SAMPLING_RATIO': {{ 'range': [0.01, 0.5], 'type': 'continuous'}}, 'TREE_IMPURITY_METRIC': {{ 'range': ['TREE_IMPURITY_ENTROPY', 'TREE_IMPURITY_GINI'], 'type': 'categorical'}},

}}``

K
[int] Denotes the number of hyperparameter choices (in pairs) the gradient based tuning algorithm tries per hyperparameter per tuning iteration (default: 4). Takes only multiples of 2 within the range [4,16].

cv
['auto', int, or None] Determines the cross-validation splitting strategy. (Default: 'auto') Valid inputs for cv are: None, to use X_valid and y_valid for validation 'auto' : cv folds determined based on size of the dataset, disabling cv-folds for large datasets integer, to specify the number of folds in a (Stratified)KFold, For integer/None inputs, if the estimator is a classifier and y is either binary or multiclass, StratifiedKFold is used. In all other cases, KFold is used.int or None

adaptive_sampling
[bool] Controls adaptive sampling of dataset to reduce overall runtime. Set to False to disable adaptive sampling. Disabling this might significantly increase runtime. (Default: True)

X_valid
[oml.DataFrame] Validation dataset features, used for model scoring when cv is disabled (default: None)

y_valid
[oml.DataFrame, str, optional] Target values for model validation when cv is disabled. If y_valid is a single column OML object, target values specified by y_valid must be

combinable with X_valid. If y_valid is a string, y_valid is the name of the column in X_valid that specifies the target values. If supplied, `y_valid` must be of the same type as `y` (Default: None)

time_budget
[int or None] Time budget in seconds where None is no time budget constraint (best effort). With insufficient time budget, returned model may be a result of default or partially tuned hyperparameters.

Returns

results
[dict]

- 'best_model': Best tuned model proxy ready for inference,
- 'all_evals': List of all hyperparameter choices tried and their corresponding score

Examples

```
>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets
```

Load the breast cancer dataset into the database with a unique case id column for reproducibility

```
>>> bc = datasets.load_breast_cancer()
>>> bc_data = bc.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns = bc.feature_names)
>>> y = pd.DataFrame(bc.target, columns = ['TARGET'])
>>> row_id = pd.DataFrame(np.arange(bc.data.shape[0]), columns = ['case_id'])
>>> df = oml.create(pd.concat([row_id, X, y], axis=1), table = 'BreastCancer')
```

Split dataset into train and test

```
>>> train, test = df.split(ratio=(0.8, 0.2), seed = 1234)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
```

Start an automatic model tuning run with decision tree algorithm

```
>>> at = automl.ModelTuning(mining_function='classification', score_metric='accuracy',
↪ parallel=4)
>>> results = at.tune('dt', X, y, case_id='case_id')
```

Show tuned model details

```
>>> tuned_model = results['best_model']
>>> tuned_model

Algorithm Name: Decision Tree

Mining Function: CLASSIFICATION
```

(continues on next page)

(continued from previous page)

```
Target: TARGET

Settings:
                    setting name            setting value
0                      ALGO_NAME       ALGO_DECISION_TREE
1              CLAS_MAX_SUP_BINS                       32
2          CLAS_WEIGHTS_BALANCED                      OFF
3                   ODMS_DETAILS              ODMS_ENABLE
4    ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
5                  ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
6                      PREP_AUTO                       ON
7           TREE_IMPURITY_METRIC       TREE_IMPURITY_GINI
8            TREE_TERM_MAX_DEPTH                        8
9          TREE_TERM_MINPCT_NODE                     3.34
10        TREE_TERM_MINPCT_SPLIT                      0.1
11         TREE_TERM_MINREC_NODE                       10
12        TREE_TERM_MINREC_SPLIT                       20

Attributes:
mean concave points
mean concavity
mean texture
worst area
worst concave points
worst radius
worst texture

Partition: NO
```

Show the best tuned model train score and the corresponding hyperparameters

```
>>> score, params = results['all_evals'][0]
>>> "{:.2}".format(score)
'0.94'
```

Also get the corresponding hyperparameter values, they may differ as many hyperparameters could have similar scores

```
>>> sorted(params.items())
[('CLAS_MAX_SUP_BINS', 32), ('TREE_IMPURITY_METRIC', 'TREE_IMPURITY_GINI'), ('TREE_
↪TERM_MAX_DEPTH', 7), ('TREE_TERM_MINPCT_NODE', 0.05), ('TREE_TERM_MINPCT_SPLIT', 0.
↪1)]
```

An example invocation of model tuning with user-defined search ranges for selected hyperparameters on a new tuning metric (f1_macro)

```
>>> search_space={'RFOR_SAMPLING_RATIO': {'type': 'continuous', 'range': [0.05, 0.5]},
↪ 'RFOR_NUM_TREES': {'type': 'discrete', 'range': [50, 55]}, 'TREE_IMPURITY_METRIC':
↪{'type': 'categorical', 'range': ['TREE_IMPURITY_ENTROPY', 'TREE_IMPURITY_GINI']},}
>>> at = automl.ModelTuning(mining_function='classification', score_metric='f1_macro',
↪ parallel=4)
>>> results = at.tune('rf', X, y, case_id='case_id', param_space=search_space)
>>> score, params = results['all_evals'][0]
>>> "{:.2}".format(score)
'0.93'
```

Print the corresponding hyperparameter values of the tuned model >>> ["{}:{}".format(k, round(params[k],

3) if isinstance(params[k], float) else params[k]) for k in sorted(params)] # doctest: +NORMALIZE_WHITESPACE, +SKIP ['RFOR_NUM_TREES:53', 'RFOR_SAMPLING_RATIO:0.2', 'TREE_IMPURITY_METRIC:TREE_IMPURITY_GINI']

Some hyperparameter search ranges may need to be defined based on the training dataset sizes (e.g. number of samples, features). The dataset specific placeholders like $nr_features, $nr_samples can be used for this purpose as shown below.

```
>>> search_space={'RFOR_MTRY': {'type': 'discrete', 'range': [1, '$nr_features/2']}}
>>> results = at.tune('rf', X, y, case_id='case_id', param_space=search_space)
>>> score, params = results['all_evals'][0]
>>> "{:.2}".format(score)
'0.93'
```

```
>>> ["{}:{}".format(k, params[k]) for k in sorted(params)]
['RFOR_MTRY:1']
```

```
>>> oml.drop('BreastCancer')
```

CHAPTER

FOUR

MACHINE LEARNING EXPLAINABILITY

The Machine Learning Explainability module provides model-agnostic functionality to identify the important features that impact a trained model's predictions.

4.1 Global Feature Permutation Importance Explanations

class oml.mlx.**GlobalFeatureImportance**(*mining_function='classification'*, *score_metric=None*, *random_state=None*, *parallel=None*)

Interface to the GPIExplainer.

Parameters

mining_function
[str, optional] Type of the model task: 'classification' or 'regression', by default 'classification'.

score_metric
[str, optional] Scoring metric to be used to compute the feature importance explanation, by default None.

random_state
[int, optional] Random seed value, by default None.

parallel
[int, optional] Degree of parallelization, by default None.

__init__(*mining_function='classification'*, *score_metric=None*, *random_state=None*, *parallel=None*)

Initialize self. See help(type(self)) for accurate signature.

explain(*model*, *X*, *y*, *case_id=None*, *n_iter=10*, *selected_features=None*, *sampling='auto'*)

Computes the global feature permutation importance explanation.

Parameters

model
[oml.algo.model.odmModel] Input model.

X
[oml.DataFrame] Input dataset values.

y [str, oml.DataFrame, oml.Float] Name of the target column in the dataset, X, or a separate target DataFrame from the same table as X.

case_id
[str, oml.DataFrame, oml.Float, optional] Name of the case_id column in the dataset, X, or a separate case_id DataFrame (single column) from the same table as X, by default None.

n_iter
[int, optional] Number of iterations to perform for the feature permutation importance algorithm, by default 10.

selected_features
[list of str, optional] List of feature names to include in the feature permutation importance algorithm, by default None (permute all features).

sampling
[dict, 'auto' (default), optional] Dictionary object describing the type of down-sampling to apply and any arguments to the down-sampler. Must contain a 'technique' key, which sets the sampling method (currently supports 'random') and the corresponding sampling arguments. For random sampling, also requires a 'sampling_fraction' key, which contains a float between 0 and 1 specifying the fraction of dataset to randomly sample. Can also be 'auto', which will automatically configure the sampling parameters. By default 'auto'.

Returns

mlx.oml.explanations.perm_importance.GPIExplanation
Explanation object describing the feature importance on the provided model and dataset.

4.1.1 Binary Classification

Examples

```
>>> import oml
>>> from oml.mlx import GlobalFeatureImportance
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets
```

Load the Breast Cancer binary classification dataset into the database, adding a unique case id column.

```
>>> bc_ds = datasets.load_breast_cancer()
>>> bc_data = bc_ds.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns=bc_ds.feature_names)
>>> y = pd.DataFrame(bc_ds.target, columns=['TARGET'])
>>> row_id = pd.DataFrame(np.arange(bc_data.shape[0]), columns=['CASE_ID'])
>>> df = oml.create(pd.concat([X, y, row_id], axis=1), table='BreastCancer')
```

Split the dataset into train and test.

```
>>> train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID', seed=32)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
```

Train a Random Forest model.

```
>>> model = oml.algo.rf(ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
>>> "RF accuracy score = {:.2f}".format(model.score(X_test, y_test))
'RF accuracy score = 0.95'
```

Create the MLX Global Feature Importance explainer, using the binary f1 metric.

```
>>> gfi = GlobalFeatureImportance(mining_function='classification', score_metric='f1',
↪ random_state=32, parallel=4)
```

Run the explainer to generate the global feature importance. The explanation can be constructed using any dataset (e.g., Train, test, validation). Here, we use the train dataset.

```
>>> explanation = gfi.explain(model, X, y, case_id='CASE_ID', n_iter=10)
```

Display the explanation.

```
>>> explanation
Global Feature Importance:
        [0] worst concave points: Value: 0.0263, Error: 0.0069
        [1] worst perimeter: Value: 0.0077, Error: 0.0027
        [2] worst radius: Value: 0.0076, Error: 0.0031
        [3] worst area: Value: 0.0045, Error: 0.0037
        [4] mean concave points: Value: 0.0034, Error: 0.0033
        [5] worst texture: Value: 0.0017, Error: 0.0015
        [6] area error: Value: 0.0012, Error: 0.0014
        [7] worst concavity: Value: 0.0008, Error: 0.0008
        [8] worst symmetry: Value: 0.0004, Error: 0.0007
        [9] mean texture: Value: 0.0003, Error: 0.0007
        [10] mean perimeter: Value: 0.0003, Error: 0.0015
        [11] mean radius: Value: 0.0000, Error: 0.0000
        [12] mean smoothness: Value: 0.0000, Error: 0.0000
        [13] mean compactness: Value: 0.0000, Error: 0.0000
        [14] mean concavity: Value: 0.0000, Error: 0.0000
        [15] mean symmetry: Value: 0.0000, Error: 0.0000
        [16] mean fractal dimension: Value: 0.0000, Error: 0.0000
        [17] radius error: Value: 0.0000, Error: 0.0000
        [18] texture error: Value: 0.0000, Error: 0.0000
        [19] smoothness error: Value: 0.0000, Error: 0.0000
        [20] compactness error: Value: 0.0000, Error: 0.0000
        [21] concavity error: Value: 0.0000, Error: 0.0000
        [22] concave points error: Value: 0.0000, Error: 0.0000
        [23] symmetry error: Value: 0.0000, Error: 0.0000
        [24] fractal dimension error: Value: 0.0000, Error: 0.0000
        [25] worst compactness: Value: 0.0000, Error: 0.0000
        [26] worst fractal dimension: Value: 0.0000, Error: 0.0000
        [27] mean area: Value: -0.0001, Error: 0.0011
        [28] worst smoothness: Value: -0.0003, Error: 0.0013
        [29] perimeter error: Value: -0.0014, Error: 0.0020
```

```
>>> oml.drop('BreastCancer')
```

4.1.2 Multi-Class Classification

Examples

```
>>> import oml
>>> from oml.mlx import GlobalFeatureImportance
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets
```

Load the Iris multi-class classification dataset into the database, adding a unique case id column.

```
>>> iris_ds = datasets.load_iris()
>>> iris_data = iris_ds.data.astype(float)
>>> X = pd.DataFrame(iris_data, columns=iris_ds.feature_names)
>>> y = pd.DataFrame(iris_ds.target, columns=['TARGET'])
>>> row_id = pd.DataFrame(np.arange(iris_data.shape[0]), columns=['CASE_ID'])
>>> df = oml.create(pd.concat([X, y, row_id], axis=1), table='Iris')
```

Split the dataset into train and test.

```
>>> train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID', seed=32)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
```

Train an SVM model.

```
>>> model = oml.algo.svm(ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
>>> "SVM accuracy score = {:.2f}".format(model.score(X_test, y_test))
'SVM accuracy score = 0.94'
```

Create the MLX Global Feature Importance explainer, using the f1_weighted metric.

```
>>> gfi = GlobalFeatureImportance(mining_function='classification', score_metric='f1_
↪weighted', random_state=32, parallel=4)
```

Run the explainer to generate the global feature importance. The explanation can be constructed using any dataset (e.g., Train, test, validation). Here, we use the test dataset.

```
>>> explanation = gfi.explain(model, X_test, y_test, case_id='CASE_ID', n_iter=10)
```

Display the explanation.

```
>>> explanation
Global Feature Importance:
        [0] petal length (cm): Value: 0.3462, Error: 0.0824
        [1] petal width (cm): Value: 0.2417, Error: 0.0687
        [2] sepal width (cm): Value: 0.0926, Error: 0.0452
        [3] sepal length (cm): Value: 0.0253, Error: 0.0152
```

```
>>> oml.drop('Iris')
```

4.1.3 Regression

Examples

```
>>> import oml
>>> from oml.mlx import GlobalFeatureImportance
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets
```

Load the Diabetes regression dataset into the database, adding a unique case id column.

```
>>> diabetes_ds = datasets.load_diabetes()
>>> diabetes_data = diabetes_ds.data
>>> X = pd.DataFrame(diabetes_data, columns=diabetes_ds.feature_names)
>>> y = pd.DataFrame(diabetes_ds.target, columns=['TARGET'])
>>> row_id = pd.DataFrame(np.arange(diabetes_data.shape[0]), columns=['CASE_ID'])
>>> df = oml.create(pd.concat([X, y, row_id], axis=1), table='Diabetes')
```

Split the dataset into train and test.

```
>>> train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID', seed=32)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
```

Train a Generalized Linear regression model.

```
>>> model = oml.algo.glm(mining_function='regression', ODMS_RANDOM_SEED=32).fit(X, y,␣
↪case_id='CASE_ID')
>>> "GLM R^2 score = {:.2f}".format(model.score(X_test, y_test))
'GLM R^2 score = 0.44'
```

Create the MLX Global Feature Importance explainer, using the r2 metric.

```
>>> gfi = GlobalFeatureImportance(mining_function='regression', score_metric='r2',␣
↪random_state=32, parallel=4)
```

Run the explainer to generate the global feature importance. The explanation can be constructed using any dataset (e.g., Train, test, validation). Here, we use the entire dataset.

```
>>> explanation = gfi.explain(model, df, 'TARGET', case_id='CASE_ID', n_iter=10)
```

Display the explanation.

```
>>> explanation
Global Feature Importance:
  [0] ...
  [1] ...
  [2] ...
  [3] ...
  [4] ...
  [5] ...
  [6] ...
  [7] ...
  [8] ...
  [9] ...
```

```
>>> oml.drop('Diabetes')
```

CHAPTER

FIVE

DATASTORE

Oracle Machine Learning for Python Datastore

The Datastore module contains functions that persist and manage Python objects in Oracle Database.

oml.ds.save(objs, name[, description, . . .])	Saves Python objects to a datastore in the user's Oracle Database schema.
oml.ds.dir([name, regex_match, dstype])	Lists existing datastores available to the current session user.
oml.ds.describe(name[, owner])	Describes the contents of the named datastore available to the current session user.
oml.ds.load(name[, objs, owner, to_globals])	Loads Python objects from a datastore in the user's Oracle Database schema.
oml.ds.delete(name[, objs, regex_match])	Deletes one or more datastores from the user's Oracle Database schema or specific Python objects to delete from within a datastore.

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

Saves Python objects to a datastore in the user's Oracle Database schema.

Parameters

objs
[dict, list of str] The dict key specifies the *object_name* under which the dict value is saved in a datastore. list of str specifies the names of objects in the current workspace to be saved in a datastore. The same name is used as *object_name*.

name
[str] The name of the datastore in which to save the Python objects.

description
[str, “” (default)] The comments for the datastore.

grantable
[bool, False (default)] Indicates whether to create a new datastore the read privilege for which can be granted to other users. Ignored when used with arguments `overwrite` or `append`.

overwrite
[bool, False (default)] Indicates whether to overwrite the datastore, if it already exists.

append
[bool, False (default)] Indicates whether to append the Python objects to the datastore, if it already exists.

compression
[bool, False (default)] Whether to compress the serialized Python objects

Raises

ValueError

- If `name` is not a str or is an empty str.
- If both `overwrite` and `append` are True.
- If the named datastore already exists while neither `overwrite` or `append` is True.

Notes

By default, Python objects are saved to a new datastore with the specified `name`. `overwrite` and `append` can be used to save Python objects to an existing datastore, but only one of them can be True.

Examples

See *Datastore Pipeline*

`oml.ds.`**`dir`**(*name=None*, *regex_match=False*, *dstype='user'*)

Lists existing datastores available to the current session user.

This datastore listing may be optionally filtered by `regex_match` and/or `dstype` arguments.

Parameters

name
[str or None (default)] The datastore name or a regular expression to match the names. Returns information of all the datastores when `name` is None.

regex_match
[bool, False (default)] Indicates whether to treat `name` as the regular expression str specifying the matching datastore names.

dstype
['user' (default), 'all', 'grant', 'granted', 'grantable' or 'private']

- user : Lists datastores created by current session user.
- grant : Lists these datastore the read privilege for which has been granted by the current session user to other users.
- granted : Lists these datastore the read privilege for which has been granted by other users to the current session user.
- grantable : Lists these datastores the read privilege for which can be granted by the current session user to other users.
- all : Lists all these datastores to which the current session user has read access.

- private : Lists these datastores the read privileges for which cannot be granted by the current session user to other users.

Returns

ds_list
[`pandas.DataFrame`] A data frame object is returned with columns 'datastore_name', 'object_count', 'size', 'date' and 'description' when argument `type` is 'user', 'private' or 'grantable'; these columns specify the datastore name, the number of objects in the named datastore, the size of the datastore in bytes, the datastore creation date, and the datastore comments respectively. The output data frame has one extra column 'owner' specifying the owner of the datastores when argument `type` is 'all' or 'granted'. When the argument `type` is 'grant', the output data frame contains columns 'datastore_name' and 'grantee' specifying the datastore name and the name of the user to which the read privilege of the named datastore has been granted by the current session user.

Examples

See *Datastore Pipeline*

`oml.ds.`**`describe`**(*name*, *owner=None*)

Describes the contents of the named datastore available to the current session user.

Provides information about the objects saved in the datastore specified by argument `name` and `owner`.

Parameters

name
[str] The datastore name.

owner
[str or None (default)] The owner of the datastore to summarize. The `owner` argument is case-sensitive for db user name. If the `owner` argument is not specified, then value is the current user.

Returns

ds_info
[`pandas.DataFrame`] Each row represents an object in the datastore. The columns represent the following object attributes:

- 'object_name': name
- 'class': class
- 'size': size in bytes
- 'length': length
- 'row_count': number of rows
- 'col_count': number of columns

Raises

ValueError

- If `name` is not a str or is an empty str.
- If the owner of the datastore is not the current user and the read privilege for the datastore are not granted to the current user.
- If the datastore does not exists.

Examples

See *Datastore Pipeline*

`oml.ds.load`(*name*, *objs=None*, *owner=None*, *to_globals=True*)

Loads Python objects from a datastore in the user's Oracle Database schema.

Parameters

name
[str] The name of the datastore.

objs
[list of str or None (default)] The names of the Python objects in the datastore to load. If this argument is not specified, then all of the Python objects in the datastore will be loaded.

owner
[str or None (default)] The owner of the datastore to load. If the `owner` argument is not specified, then value is set to current user. The `owner` argument is case-sensitive for db user name.

to_globals
[bool, True (default)] Indicates whether the objects are loaded to global workspace or a dict that is returned. If *True*, objects are loaded into global workspace, otherwise a dict containing object name and value pairs is returned to caller.

Returns

loaded_obj
[list of str or dict] The names of Python objects loaded from the datastore or a dict of object name and value pairs.

Raises

ValueError

- If `name` is not a str or is an empty str.
- If the owner of the datastore is not the current user and the read privilege for the datastore are not granted to the current user.

Examples

See *Datastore Pipeline*

`oml.ds.delete`(*name*, *objs=None*, *regex_match=False*)

Deletes one or more datastores from the user's Oracle Database schema or specific Python objects to delete from within a datastore.

When `regex_match` is False, if argument `objs` is None, and argument `name` is not None, then function `delete` deletes the datastore specified in the `name` argument from the user's Oracle Database schema; If arguments `name` and `objs` are used, then the function deletes the specified Python objects from the named datastore supplied in the `name` argument.

When argument `regex_match` is True, if argument `objs` is None, then function `delete` deletes the datastores in the user's Oracle Database schema whose names match the regular expression specified in the `name` argument; If argument `name` is a str and argument `objs` is used, the function deletes the objects whose names match the regular expression specified in the `objs` argument from the named datastore; If argument `name` is a list of str, argument `objs` must be a list of str of the same length as `name`, the function deletes the objects whose names match the regular expression specified in the `objs` respectively from the datastore named in argument `name`.

Parameters

name
[str or list of str] The name of the datastore to delete or modify; a regular expression to match the datastores to delete; list of names of the datastores to delete objects from.

objs
[str, list of str or None, or None (default)] The names of objects to delete or regular expressions to match the objects to delete. When argument `objs` is not used, the entire datastore is deleted. If a list, then the objects whose names match the regular expression at each position will be deleted from the datastore whose name is at the corresponding position in `name`. If instead of a string, there is a None at any given position, the entire datastore is deleted.

regex_match
[bool, False (default)] If True, `objs` is treated as regular expressions if `objs` is not None. If True and `objs` is None, `name` is treated as regular expressions.

Returns

deleted
[str, set of str, or dict mapping str to set]

- If a single datastore is deleted by name, return name of datastore.
- If datastores matching a regular expression are deleted, or multiple datastores are deleted, return names of deleted datastores as a set.
- If named objects are deleted from a single datastore, return names of deleted objects as a set.
- If objects whose names match a regular expression are deleted, or if both datastores and objects are deleted, return dictionary mapping datastore names to the set of names of objects from the datastore that were deleted.

Raises

ValueError

- If the datastore specified by argument `name` does not exist.
- If argument `regex_match` is False and argument `name` is a list of str larger than 1 and argument `objs` is not None.
- If argument `regex_match` is True and argument `name` and `objs` do not have the same list length.

Examples

See *Datastore Pipeline*

An example showing the complete OML Datastore pipeline

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

Load three datasets.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['SEPAL_LENGTH','SEPAL_WIDTH','PETAL_LENGTH
↪','PETAL_WIDTH'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:'virginica'}
↪[x], iris.target)), columns = ['SPECIES'])
>>> oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>> oml_iris.columns
['SEPAL_LENGTH', 'SEPAL_WIDTH', 'PETAL_LENGTH', 'PETAL_WIDTH', 'SPECIES']
>>> diabetes = datasets.load_diabetes()
>>> x = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
>>> y = pd.DataFrame(diabetes.target, columns=['disease_progression'])
>>> oml_diabetes = oml.create(pd.concat([x, y], axis=1), table = "DIABETES")
>>> oml_diabetes.columns
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6', 'disease_progression']
>>> wine = datasets.load_wine()
>>> x = pd.DataFrame(wine.data, columns = wine.feature_names)
>>> y = pd.DataFrame(wine.target, columns = ['Class'])
>>> oml_wine = oml.create(pd.concat([x, y], axis=1), table = "WINE")
>>> oml_wine.columns
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols',
↪'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue',
↪'od280/od315_of_diluted_wines', 'proline', 'Class']
```

Save two data sets to a datastore.

```
>>> oml.ds.save(objs={'oml_iris':oml_iris, 'oml_wine':oml_wine},
...             name="ds_pydata", description = "python datasets")
```

Save a dataset to an existing datastore.

```
>>> oml.ds.save(objs={'oml_diabetes':oml_diabetes},
...             name="ds_pydata", append=True)
>>> regr1 = linear_model.LinearRegression()
>>> regr1.fit(diabetes.data, diabetes.target)
LinearRegression()
>>> regr2 = oml.glm("regression")
>>> X = oml_diabetes.drop('disease_progression')
>>> y = oml_diabetes['disease_progression']
>>> regr2 = regr2.fit(X, y)
>>> oml.ds.save(objs={'regr1':regr1, 'regr2':regr2},
...             name="ds_pymodel", grantable=True)
```

Grant the read privilege of datastore to every user.

```
>>> oml.grant(name="ds_pymodel", typ="datastore", user=None)
```

Show all saved datastore.

```
>>> oml.ds.dir(dstype="all")[['owner', 'datastore_name', 'object_count']]
     owner datastore_name  object_count
0  PYQUSER       ds_pydata             3
1  PYQUSER      ds_pymodel             2
```

Show datastore for which other users have been granted the read privilege.

```
>>> oml.ds.dir(dstype="grant")
  datastore_name grantee
0     ds_pymodel  PUBLIC
```

Show datastore whose names match pattern.

```
>>> oml.ds.dir(name='pydata', regex_match=True)[['datastore_name', 'object_count']]
  datastore_name  object_count
0      ds_pydata             3
```

Revoke the granted privilege.

```
>>> oml.revoke(name="ds_pymodel", typ="datastore", user=None)
>>> oml.ds.dir(dstype="grant")
Empty DataFrame
Columns: [datastore_name, grantee]
Index: []
```

Describe one datastore.

```
>>> oml.ds.describe(name="ds_pydata")[['object_name', 'class']]
    object_name          class
0  oml_diabetes  oml.DataFrame
1      oml_iris  oml.DataFrame
2      oml_wine  oml.DataFrame
```

Load all Python objects from datastore to current workspace.

```
>>> sorted(oml.ds.load(name="ds_pydata"))
['oml_diabetes', 'oml_iris', 'oml_wine']
```

Load the named python object from datastore.

```
>>> oml.ds.load(name="ds_pymodel", objs=["regr1"])
['regr1']
```

```
>>> del regr1, oml_iris, oml_wine, oml_diabetes
```

Delete python objects from named datastore.

```
>>> sorted(oml.ds.delete(name="ds_pydata",
...                      objs=["oml_iris", "oml_wine"]))
['oml_iris', 'oml_wine']
```

Delete named datastore.

```
>>> oml.ds.delete(name="ds_pydata")
'ds_pydata'
>>> oml.ds.dir()[['datastore_name', 'object_count']]
  datastore_name  object_count
0     ds_pymodel             2
```

Delete all datastores whose names match pattern.

```
>>> oml.ds.delete(name="_pymodel", regex_match=True)
{'ds_pymodel'}
>>> oml.ds.dir()
Empty DataFrame
Columns: [datastore_name, object_count, size, date, description]
Index: []
```

```
>>> oml.drop("IRIS")
>>> oml.drop("DIABETES")
>>> oml.drop("WINE")
```

CHAPTER

SIX

EMBEDDED EXECUTION

Oracle Machine Learning for Python Embedded Execution

The embedded execution contains functions to execute python script in Oracle database server.

6.1 Embedded Python Execution

Embedded Python execution functions described in this section can invoke Python functions that are stored as scripts in the OML Python script repository.

`oml.do_eval`(func[, func_value, func_owner, ...])	Runs the user-defined Python function using a Python engine spawned and controlled by the database environment.
`oml.table_apply`(data, func[, func_value, ...])	Runs the user-defined Python function with data pulled from a database table or view using a Python engine spawned and controlled by the database environment.
`oml.row_apply`(data, func[, func_value, ...])	Partitions an in-database data set into row chunks and executes a python function on the data pulled from those chunks within Python processes running inside Oracle Database server.
`oml.group_apply`(data, index, func[, ...])	Partitions an in-database data set by the column(s) specified in `index` and executes a python function on those partitions within Python processes running inside Oracle Database server.
`oml.index_apply`(times, func[, func_value, ...])	Executes a python function multiple times inside Oracle Database server.

`oml.`**`do_eval`** (*func*, *func_value=None*, *func_owner=None*, *graphics=False*, ***kwargs*)

Runs the user-defined Python function using a Python engine spawned and controlled by the database environment.

Parameters

func

[function, str or `oml.script.Callable`] `func` can be one of the following:

- A Python function.
- Name of a registered script that defines a Python function.
- A string that if evaluated, defines a Python function.
- A callable object returned from `script_load` function.

func_value
[pandas.DataFrame or oml.DataFrame or None (default)] Ignored when running against Autonomous Database A DataFrame to use as a template for the return value.

func_owner
[str or None (default)] An optional value specifying the owner of the registered script when argument `func` is set to a registered script name.

graphics
[bool, False (default)] If True, images rendered from extant `matplotlib.figure.Figure` objects are included in the result. Ignored if `func_value` is not None.

****kwargs**
Special Control Arguments and/or additional arguments to `func`.

Returns

result
[oml.DataFrame or oml.embed.object.Object] If `func_value` is supplied, the return value is an oml.DataFrame with the same column names and types as the template. Otherwise, the return value is a transparency layer representation of a serialized python object stored in the database. See *More on Output*.

result (Autonomous database)
[Python object or oml.embed.data_image._DataImage] If no image is rendered in the script, returns whatever Python object returned by the function. Otherwise, returns an oml.embed.data_image._DataImage object. See *More on Output*.

Examples

```
>>> import oml
>>> import pandas as pd
```

Invoke given Python code “num” times with input “scale”.

```
>>> def return_df(num, scale):
...   import pandas as pd
...   id = list(range(0, int(num)))
...   res = [i/scale for i in id]
...   return pd.DataFrame({"ID":id, "RES":res})
...
>>> res = oml.do_eval(func=return_df, func_value=pd.DataFrame({"ID":[0],
...                   "RES":[0]}), scale = 100, num = 10)
>>> type(res)
<class 'oml.core.frame.DataFrame'>
>>> res
   ID   RES
0   0  0.00
1   1  0.01
2   2  0.02
```

(continues on next page)

(continued from previous page)

```
3   3  0.03
4   4  0.04
5   5  0.05
6   6  0.06
7   7  0.07
8   8  0.08
9   9  0.09
```

`oml.`**`table_apply`**(*data*, *func*, *func_value=None*, *func_owner=None*, *graphics=False*, ***kwargs*)

Runs the user-defined Python function with data pulled from a database table or view using a Python engine spawned and controlled by the database environment.

Parameters

data
[oml.DataFrame] The oml.DataFrame that represents the data `func` is applied on.

func
[function, str or *`oml.script.Callable`*] `func` can be one of the following:

- A Python function.
- Name of a registered script that defines a Python function.
- A string that if evaluated, defines a Python function.
- A callable object returned from `script_load` function.

func_value
[pandas.DataFrame or oml.DataFrame or None (default)] Ignored when running against Autonomous Database. A DataFrame to use as a template for the return value.

func_owner
[str or None (default)] An optional value specifying the owner of the registered script when argument `func` is set to a registered script name.

graphics
[bool, False (default)] If True, images rendered from extant `matplotlib.figure.Figure` objects are included in the result. Ignored if `func_value` is not None.

****kwargs**
Special Control Arguments and/or additional arguments to `func`.

Returns

result
[oml.DataFrame or oml.embed.object.Object] If `func_value` is supplied, the return value is an oml.DataFrame with the same column names and types as the template. Otherwise, the return value is a transparency layer representation of a serialized python object stored in the database. See *More on Output*.

result (Autonomous database)
[Python object or oml.embed.data_image._DataImage] If no image is rendered in the script, returns whatever Python object returned by the function. Otherwise, returns an oml.embed.data_image._DataImage object. See *More on Output*.

Examples

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

Push the iris data set to a table with numeric and categorical columns in the database.

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

Display the type of data.

```
>>> type(oml_iris)
<class 'oml.core.frame.DataFrame'>
```

Build a regression model using in-memory data.

```
>>> iris = oml_iris.pull()
>>> regr = linear_model.LinearRegression()
>>> regr.fit(iris[['SEPAL_WIDTH', 'PETAL_LENGTH', 'PETAL_WIDTH']], iris[['SEPAL_LENGTH
↪']])
LinearRegression()
>>> regr.coef_
array([[ 0.65083716,  0.70913196, -0.55648266]])
```

Use table_apply to predict using the model on the first 10 rows of IRIS table.

```
>>> predict = '''
... def predict(dat, regr):
...   import pandas as pd
...   pred = regr.predict(dat[['SEPAL_WIDTH', 'PETAL_LENGTH', 'PETAL_WIDTH']])
...   return pd.concat([dat,pd.DataFrame(pred)], axis=1)
... '''
>>> res = oml.table_apply(data=oml_iris.head(n=10), func=predict, regr=regr)
>>> res
   SEPAL_LENGTH  SEPAL_WIDTH  PETAL_LENGTH  PETAL_WIDTH SPECIES         0
0           5.1          3.5           1.4          0.2  setosa  5.015416
1           4.9          3.0           1.4          0.2  setosa  4.689997
2           4.7          3.2           1.3          0.2  setosa  4.749251
3           4.6          3.1           1.5          0.2  setosa  4.825994
4           5.0          3.6           1.4          0.2  setosa  5.080499
5           5.4          3.9           1.7          0.4  setosa  5.377194
6           4.6          3.4           1.4          0.3  setosa  4.894684
7           5.0          3.4           1.5          0.2  setosa  5.021245
8           4.4          2.9           1.4          0.2  setosa  4.624913
9           4.9          3.1           1.5          0.1  setosa  4.881642
```

oml.**row_apply**(*data*, *func*, *func_value=None*, *func_owner=None*, *rows=1*, *parallel=None*, *graphics=False*, ***kwargs*)

Partitions an in-database data set into row chunks and executes a python function on the data pulled from those chunks within Python processes running inside Oracle Database server.

Parameters

data
[oml.DataFrame] The OML DataFrame that represents the in-database data that `func` is applied on.

func
[function, str or *oml.script.Callable*] `func` can be one of the following:

- A Python function.
- Name of a registered script that defines a Python function.
- A string that if evaluated, defines a Python function.
- A callable object returned from `script_load` function.

func_value
[pandas.DataFrame or oml.DataFrame or None (default)] Ignored when running against Autonomous Database. A DataFrame to use as a template for the return value.

func_owner
[str or None (default)] An optional value specifying the owner of the registered script when argument `func` is set to a registered script name.

rows
[int, 1 (default)] The maximum number of rows in each chunk.

parallel
[bool or int or None (default)] A preferred degree of parallelism to use in the embedded Python job; either a positive integer greater than or equal to 1 for a specific degree of parallelism, a value of 'False' or '0' for no parallelism, a value of 'True' for the `data` default parallelism, or 'None' for database default for the operation.

graphics
[bool, False (default)] If True, images rendered from extant `matplotlib.figure.Figure` objects are included in the result. Ignored if `func_value` is not None.

****kwargs**
Special Control Arguments and/or additional arguments to `func`.

Returns

result
[oml.DataFrame or oml.embed.objectList.ObjectList] If `func_value` is supplied, the return value is an oml.DataFrame with the same column names and types as the template. Otherwise, the return value is a transparency layer representation of a list of serialized Python objects. See *More on Output*.

result (Autonomous database)
[pandas.DataFrame or a list of oml.embed.data_image._DataImage] If no image is rendered in the script, returns a `pandas.DataFrame`. Otherwise, returns a list of oml.embed.data_image._DataImage objects. See *More on Output*.

Examples

```
>>> from sklearn.datasets import load_iris
>>> from sklearn import linear_model
>>> import pandas as pd
```

Push the iris data set to a table with numeric and categorical columns in the database.

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

Display the type of data.

```
>>> type(oml_iris)
<class 'oml.core.frame.DataFrame'>
```

Build a regression model to predict PETAL_WIDTH using in-memory data.

```
>>> iris = oml_iris.pull()
>>> regr = linear_model.LinearRegression()
>>> regr.fit(iris[['SEPAL_LENGTH', 'SEPAL_WIDTH', 'PETAL_LENGTH']], iris[['PETAL_WIDTH
↪']])
LinearRegression()
>>> regr.coef_
array([[-0.20726607,  0.22282854,  0.52408311]])
```

Apply function *make_pred()* to each chunk (4 rows) of *input_data*. For each chunk (4 rows), the fitted regression model is applied to make PETAL_WIDTH predictions. The result is the concatenation of the SPECIES column, ground truth PETAL_WIDTH column and the predicted PETAL_WIDTH.

```
>>> def make_pred(dat, regr):
...     import pandas as pd
...     import numpy as np
...     pred = regr.predict(dat[['SEPAL_LENGTH', 'SEPAL_WIDTH', 'PETAL_LENGTH']])
...     return pd.concat([dat[['SPECIES', 'PETAL_WIDTH']], pd.DataFrame(pred,␣
↪columns=['PRED_PETAL_WIDTH'])], axis=1)
...
>>> input_data = oml_iris.split(ratio=(0.9, 0.1), strata_cols='SPECIES')[1]
>>> input_data.crosstab(index = 'SPECIES').sort_values('SPECIES')
      SPECIES  count
0      setosa      5
1  versicolor      6
2   virginica      2
>>> res = oml.row_apply(input_data, rows=4, func=make_pred, regr=regr, func_value=pd.
↪DataFrame([('a', 1, 1)], columns=['SPECIES', 'PETAL_WIDTH', 'PRED_PETAL_WIDTH']),␣
↪parallel=2)
>>> res
       SPECIES  PETAL_WIDTH  PRED_PETAL_WIDTH
0       setosa          0.4          0.344846
1       setosa          0.3          0.335509
2       setosa          0.2          0.294117
3       setosa          0.2          0.220982
4       setosa          0.2          0.080937
5   versicolor          1.5          1.504615
6   versicolor          1.3          1.560570
```

(continues on next page)

(continued from previous page)

```
7   versicolor          1.0          1.008352
8   versicolor          1.3          1.131905
9   versicolor          1.3          1.215622
10  versicolor          1.3          1.272388
11   virginica          1.8          1.623561
12   virginica          1.8          1.878132
```

oml.**group_apply**(*data*, *index*, *func*, *func_value=None*, *func_owner=None*, *parallel=None*, *orderby=None*, *graphics=False*, ***kwargs*)

Partitions an in-database data set by the column(s) specified in `index` and executes a python function on those partitions within Python processes running inside Oracle Database server.

Parameters

data
[oml.DataFrame] The OML DataFrame that represents the in-database data that `func` is applied on.

index
[OML data object] The columns to partition the `data` before sending it to `func`.

func
[function, str or *oml.script.Callable*] `func` can be one of the following:

- A Python function.
- Name of a registered script that defines a Python function.
- A string that if evaluated, defines a Python function.
- A callable object returned from `script_load` function.

func_value
[pandas.DataFrame or oml.DataFrame or None (default)] Ignored when running against Autonomous Database. A DataFrame to use as a template for the return value.

func_owner
[str or None (default)] An optional value specifying the owner of the registered script when argument `func` is set to a registered script name.

parallel
[bool or int or None (default)] A preferred degree of parallelism to use in the embedded Python job; either a positive integer greater than or equal to 1 for a specific degree of parallelism, a value of 'False' or '0' for no parallelism, a value of 'True' for the `data` default parallelism, or 'None' for database default for the operation.

orderby
[oml.DataFrame, oml.Float, or oml.String] An optional argument used to specify the ordering of group partitions.

graphics
[bool, False (default)] If True, images rendered from extant `matplotlib.figure.Figure` objects are included in the result. Ignored if `func_value` is not None.

****kwargs**
Special Control Arguments and/or additional arguments to `func`.

Returns

result
[oml.DataFrame or oml.embed.objectList.ObjectList] If `func_value` is supplied, the return value is an oml.DataFrame with the same column names and types as the template. Otherwise, the return value is a transparency layer representation of a dict mapping partition keys to serialized Python objects. See *More on Output*.

result (Autonomous database)
[dict] If no image is rendered in the script, returns a dict of Python objects returned by the function. Otherwise, returns a dict of oml.embed.data_image._DataImage objects. See *More on Output*.

Examples

```
>>> from sklearn import datasets
>>> import pandas as pd
```

Create table IRIS in the database from the iris data set.

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

Display the type of data.

```
>>> type(oml_iris)
<class 'oml.core.frame.DataFrame'>
```

Repeat for each unique value of SPECIES.

```
>>> def group_count(dat):
...   import pandas as pd
...   return pd.DataFrame([(dat["SPECIES"][0], dat.shape[0])],\
...                       columns = ["SPECIES", "CNT"])
...
>>>
>>> index = oml.DataFrame(oml_iris['SPECIES'])
>>> res = oml.group_apply(oml_iris, index, func=group_count,
...                       oml_input_type="pandas.DataFrame",
...                       func_value=pd.DataFrame([('a', 1)],
...                                   columns=["SPECIES", "CNT"]))
>>> res.sort_values(by="SPECIES")
      SPECIES  CNT
0      setosa   50
1  versicolor   50
2   virginica   50
```

Build one regression model per species.

```
>>> def build_lm(dat):
...   from sklearn import linear_model
...   lm = linear_model.LinearRegression()
...   X = dat[["PETAL_WIDTH"]]
...   y = dat[["PETAL_LENGTH"]]
```

(continues on next page)

(continued from previous page)

```
...     lm.fit(X, y)
...     return lm
...
>>>
>>> mod = oml.group_apply(oml_iris[:,["PETAL_LENGTH", "PETAL_WIDTH",
...                                   "SPECIES"]], index, func=build_lm)
>>> sorted(mod.pull().items())
[('setosa', LinearRegression()), ('versicolor', LinearRegression()), ('virginica',␣
↪LinearRegression())]
>>> oml.drop("IRIS")
```

oml.**index_apply**(*times*, *func*, *func_value=None*, *func_owner=None*, *parallel=None*, *graphics=False*, ***kwargs*)

Executes a python function multiple times inside Oracle Database server.

Parameters

times
[int] The number of times to execute the function.

func
[function, str or *oml.script.Callable*] `func` can be one of the following:

- A Python function.
- Name of a registered script that defines a Python function.
- A string that if evaluated, defines a Python function.
- A callable object returned from `script_load` function.

func_value
[pandas.DataFrame or oml.DataFrame or None (default)] Ignored when running against Autonomous Database A DataFrame to use as a template for the return value.

func_owner
[str or None (default)] An optional value specifying the owner of the registered script when argument `func` is set to a registered script name.

parallel
[bool or int or None (default)] A preferred degree of parallelism to use in the embedded Python job; either a positive integer greater than or equal to 1 for a specific degree of parallelism, a value of 'False' or '0' for no parallelism, a value of 'True' for the `data` default parallelism, or 'None' for database default for the operation.

graphics
[bool, False (default)] If True, images rendered from extant `matplotlib.figure.Figure` objects are included in the result. Ignored if `func_value` is not None.

****kwargs**
Special Control Arguments and/or additional arguments to `func`.

Returns

result
[oml.DataFrame or oml.embed.objectList.ObjectList] If `func_value` is supplied, the return value is an oml.DataFrame with the same column names and types as the template. Otherwise, the return value is a transparency layer representation of a list of serialized python object stored in the database. See *More on Output*.

result (Autonomous database)
[list] If no image is rendered in the script, returns a list of Python objects returned by the function. Otherwise, returns a list of oml.embed.data_image._DataImage objects. See *More on Output*.

Examples

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

Create table IRIS_DATA in the database from the iris data set.

```
>>> iris = load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['SEPAL_LENGTH','SEPAL_WIDTH',
...                                         'PETAL_LENGTH','PETAL_WIDTH'])
>>> iris_data = oml.create(x, table = 'IRIS_DATA')
```

Build regression model 'times' times with 'times'-fold data.

```
>>> def build_lm(idx):
...     import oml
...     import random
...     from sklearn import linear_model
...     regr = linear_model.LinearRegression()
...     # Sync the IRIS_DATA table from Oracle Database, use KFold()
...     # to split the synchronized proxy object into 2 consecutive folds,
...     # then use 'idx' to select each fold as train_dat
...     train_dat = oml.sync(table = 'IRIS_DATA').KFold(n_splits=2\
...     )[idx-1][0].pull()
...     X = train_dat[["PETAL_LENGTH"]]
...     y = train_dat[["PETAL_WIDTH"]]
...     regr.fit(X, y)
...     return regr
...
>>>
>>> res = oml.index_apply(times=2, func=build_lm, oml_connect=True)
>>> res
[LinearRegression(), LinearRegression()]
>>>
>>> oml.drop("IRIS_DATA")
```

6.1.1 More on Output

When a Python script runs in the Oracle database server, by default, the server serializes the resulting Python object into a byte string. Also, the server captures all `matplotlib.figure.Figure` objects created by the script and converts them into PNG format.

By default, the embedded Python execution functions return `oml.embed.object.Object` and `oml.embed.objectList.ObjectList` objects, which are the transparency classes representing the table containing the serialized Python object(s) and PNG images. Calling the method `__repr__()` displays the PNG images and prints out the string representation of the Python object. The method `pull(graphics)` deserializes the data in the table. By default, `graphics` is False, and `pull()` returns the deserialized version of the Python object that the Python script returned. `pull(True)` returns a list containing PNG image data and image title for each figure.

Alternatively, if the user provides a template, the server converts the object returned by the Python script to a Oracle database table. Currently, conversion is supported on the following types of objects:

- `numpy.ndarray` of dimensionality at most 2. 1-D arrays are converted to a 1-column table. 0-D arrays are converted to a 1-row, 1-column table.
- 1-Dimensional `numpy.recarray`.
- `pandas.DataFrame`. Indexes are ignored.
- A tuple is converted into a 1-row table. A list of tuples of equal length is converted into a table with as many rows as the length of the list and as many columns as the length of each tuple. An empty list is converted into a 0-row table.

If the user chooses to have the server convert the result to a Oracle Database table, any `matplotlib.figure.Figure` objects created by the script are ignored.

Examples

```
>>> from sklearn.datasets import load_iris
>>> import pandas as pd
>>> iris = load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['SEPAL_LENGTH','SEPAL_WIDTH','PETAL_LENGTH
↪','PETAL_WIDTH'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:'virginica'}
↪[x], iris.target)), columns = ['SPECIES'])
>>> oml_iris = oml.push(pd.concat([x, y], axis=1))
>>> oml_iris_data = oml_iris.drop('SPECIES')
>>> def generate_data_and_images(dat):
...   import numpy as np
...   import matplotlib.pyplot as plt
...   x = dat[:,0]
...   y = dat[:,1]
...   fig = plt.figure(1);
...   fig2 = plt.figure(2);
...   ax = fig.add_subplot(111);
...   ax2 = fig2.add_subplot(111);
...   ax.hist(x)
...   ax.set_title("subplot 1")
...   ax2.hist(y)
...   ax2.set_title("subplot 2")
...   return np.corrcoef(x, y)
...
```

Apply function `generate_data_and_images` to iris data.

```
>>> res = oml.table_apply(oml_iris_data, generate_data_and_images, graphics=True)
>>> type(res)
<class 'oml.embed.object.Object'>
```

Calling `res.__repr__()` will display both figures and the representation of the serialized Python object result.

```
>>> res
array([[ 1.        , -0.11756978],
       [-0.11756978,  1.        ]])
```

Deserialize the Python object result.

```
>>> res.pull()
array([[ 1.        , -0.11756978],
       [-0.11756978,  1.        ]])
>>> type(res.pull())
<class 'numpy.ndarray'>
```

Get a list of figures created in the python script in PNG format.

```
>>> titles, images = res.pull(graphics=True)
>>> [img[:30] for img in images]
[b
↪'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02\x80\x00\x00\x01\xe0\x08\x06\x00\x00\x005
↪', b
↪'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02\x80\x00\x00\x01\xe0\x08\x06\x00\x00\x005
↪']
```

Get a list of titles of the figures created in the function.

```
>>> titles
['subplot 1', 'subplot 2']
```

Apply function `generate_data_and_images` to each chunk (50 rows) of iris data.

```
>>> res = oml.row_apply(oml_iris_data, generate_data_and_images, rows=50,␣
↪graphics=True)
>>> type(res)
<class 'oml.embed.objectList.ObjectList'>
```

Calling `res.__repr__()` will display both figures and the representation of the serialized Python object result.

```
>>> res
[array([[1.        , 0.74254669],
       [0.74254669, 1.        ]]), array([[1.        , 0.52591072],
       [0.52591072, 1.        ]]), array([[1.        , 0.45722782],
       [0.45722782, 1.        ]])]
```

Deserialize the Python object result.

```
>>> res.pull()
[array([[1.        , 0.74254669],
       [0.74254669, 1.        ]]), array([[1.        , 0.52591072],
       [0.52591072, 1.        ]]), array([[1.        , 0.45722782],
       [0.45722782, 1.        ]])]
```

Get a list of figures created in the python script in PNG format.

```
>>> title, data = res.pull(graphics=True)
>>> [img[:30] for chunk_data in data for img in chunk_data]
[b
↪'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02\x80\x00\x00\x01\xe0\x08\x06\x00\x00\x005
↪', b
↪'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02\x80\x00\x00\x01\xe0\x08\x06\x00\x00\x005
↪', b
↪'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02\x80\x00\x00\x01\xe0\x08\x06\x00\x00\x005
↪', b
↪'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02\x80\x00\x00\x01\xe0\x08\x06\x00\x00\x005
↪', b
↪'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02\x80\x00\x00\x01\xe0\x08\x06\x00\x00\x005
↪', b
↪'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02\x80\x00\x00\x01\xe0\x08\x06\x00\x00\x005
↪']
```

(continues on next page)

(continued from previous page)

Get a list of titles of the figures created in the function.

```
>>> [t for chunk_titles in title for t in chunk_titles]
['subplot 1', 'subplot 2', 'subplot 1', 'subplot 2', 'subplot 1', 'subplot 2']
```

Apply function `group_count_and_images` to each group (Species) of iris data.

```
>>> def group_count_and_images (dat):
...   import numpy as np
...   import matplotlib.pyplot as plt
...   np.random.seed(22)
...   fig = plt.figure(1);
...   fig2 = plt.figure(2);
...   ax = fig.add_subplot(111);
...   ax2 = fig2.add_subplot(111);
...   ax.plot(range(100), np.random.normal(size=100), marker = "o",
...           color = "red", markersize = 2)
...   ax.set_title("Random Red Dots")
...   ax2.plot(range(100,0,-1), marker = "o", color = "blue", markersize = 2)
...   ax2.set_title("Random Blue Dots")
...   return dat.shape[0]
...
>>> index = oml.DataFrame(oml_iris['SPECIES'])
>>> res = oml.group_apply(oml_iris, index, func=group_count_and_images, graphics=True)
>>> type(res)
<class 'oml.embed.objectList.ObjectList'>
```

Calling `res.__repr__()` will display both figures and the representation of the serialized Python object result.

```
>>> res
{'setosa': 50, 'versicolor': 50, 'virginica': 50}
```

Deserialize the Python object result.

```
>>> res.pull()
{'setosa': 50, 'versicolor': 50, 'virginica': 50}
```

Get a list of figures created in the python script in PNG format.

```
>>> data = res.pull(graphics=True)
>>> [img[:10] for grp in sorted(data.keys()) for img in data[grp][1]]
[b'\x89PNG\r\n\x1a\n\x00\x00', b'\x89PNG\r\n\x1a\n\x00\x00', b
↪'\x89PNG\r\n\x1a\n\x00\x00', b'\x89PNG\r\n\x1a\n\x00\x00', b
↪'\x89PNG\r\n\x1a\n\x00\x00', b'\x89PNG\r\n\x1a\n\x00\x00']
```

Get a list of titles of the figures created in the function.

```
>>> [t for grp in sorted(data.keys()) for t in data[grp][0]]
['Random Red Dots', 'Random Blue Dots', 'Random Red Dots', 'Random Blue Dots',
↪'Random Red Dots', 'Random Blue Dots']
```

Apply function `index_images` for 2 times.

```
>>> def index_images (time):
...   import numpy as np
```

(continues on next page)

(continued from previous page)

```
...     import matplotlib.pyplot as plt
...     np.random.seed(time)
...     fig = plt.figure(1);
...     fig2 = plt.figure(2);
...     ax = fig.add_subplot(111);
...     ax2 = fig2.add_subplot(111);
...     ax.plot(range(100), np.random.normal(size=100), marker = "o",
...             color = "red", markersize = 2)
...     ax.set_title("Random Red Dots")
...     ax2.plot(range(100,0,-1), marker = "o", color = "blue", markersize = 2)
...     ax2.set_title("Random Blue Dots")
...     return time
...
>>> res = oml.index_apply(times=2, func=index_images, graphics=True)
>>> type(res)
<class 'oml.embed.objectList.ObjectList'>
```

Calling `res.__repr__()` will display both figures and the representation of the serialized Python object result.

```
>>> res
[1, 2]
```

Deserialize the Python object result.

```
>>> res.pull()
[1, 2]
```

Get a list of figures created in the python script in PNG format.

```
>>> title, data = res.pull(graphics=True)
>>> [img[:30] for index_data in data for img in index_data]
[b
↪'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02\x80\x00\x00\x01\xe0\x08\x06\x00\x00\x005
↪', b
↪'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02\x80\x00\x00\x01\xe0\x08\x06\x00\x00\x005
↪', b
↪'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02\x80\x00\x00\x01\xe0\x08\x06\x00\x00\x005
↪', b
↪'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02\x80\x00\x00\x01\xe0\x08\x06\x00\x00\x005
↪']
```

Get a list of titles of the figures created in the function.

```
>>> [t for index_titles in title for t in index_titles]
['Random Red Dots', 'Random Blue Dots', 'Random Red Dots', 'Random Blue Dots']
```

6.1.2 Special Control Arguments

Special control arguments, which start with `oml_`, control what happens before or after the execution of the closure. They are not passed to the function specified by `func` argument.

Arguments

oml_connect : bool, False (default)

If True, automatically connect to OML4P inside the closure. This is equivalent to calling *oml.connect()* with the same credentials as the client session.

oml_input_type : 'pandas.DataFrame', 'numpy.recarray', or 'default' (default)

Type of object to construct from data in the Oracle database. By default, a 2-dimensional `numpy.ndarray` of type `numpy.float64` is constructed if all columns are numeric. Otherwise, a `pandas.DataFrame` is constructed.

oml_na_omit : bool, False (default)

If True, omit from the table to be evaluated all rows with any missing values.

Examples

```
>>> from sklearn import datasets
>>> import pandas as pd
```

Create table IRIS in the database from the iris data set but with some missing values.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['SEPAL_LENGTH','SEPAL_WIDTH',
...                  'PETAL_LENGTH','PETAL_WIDTH'])
>>> y = pd.DataFrame(iris.target, columns = ['SPECIES'])
>>> x.iloc[47, 1] = None
>>> x.iloc[30, 1] = None
>>> x.iloc[14, 1] = None
>>> oml_iris = oml.create(pd.concat([x, y], axis=1),
...                       oranumber = False, table = 'IRIS')
>>> oml_iris = oml_iris['SEPAL_LENGTH'].concat(
...               oml.Float(oml_iris['SEPAL_WIDTH'], 'NUMBER')).concat(
...               oml_iris[:, 2:])
```

Get the number of rows of with no missing entries for each species.

```
>>> def get_shape(repetition):
...   import oml
...   iris = oml.sync(table='IRIS')
...   subtable = iris[iris['SPECIES'] == (repetition - 1)].dropna()
...   return subtable.shape
...
>>> sorted(oml.index_apply(3, get_shape, oml_connect=True).pull())
[(47, 5), (50, 5), (50, 5)]
```

Do the same thing, but in a different way.

```
>>> def get_shape(dat):
...   return (dat.shape[0], len(dat.dtype))
...
>>> result = oml.group_apply(oml_iris,
...                  oml.DataFrame(oml_iris[['SPECIES']]),
...                  get_shape, oml_input_type='numpy.recarray',
...                  oml_na_omit=True)
>>> [result.pull()[k] for k in sorted(result.pull().keys())]
[(47, 5), (50, 5), (50, 5)]
>>> oml.drop("IRIS")
```

6.2 Script Management

The OML functions described in this section are used to create and manage scripts in the OML Python script repository.

oml.script.create(name, func[, is_global, . . .])	Creates a Python script, which contains a single function definition, in Oracle Database Python script repository.
oml.script.dir([name, regex_match, sctype])	Lists the scripts present in Oracle Database Python script repository.
oml.script.load(name[, owner])	Loads the named script from Oracle Database Python script repository as a callable object.
oml.script.drop(name[, is_global, silent])	Drops the named script from Oracle Database Python script repository.

oml.script.**create**(*name*, *func*, *is_global=False*, *overwrite=False*, *description=''*)

Creates a Python script, which contains a single function definition, in Oracle Database Python script repository.

Parameters

name
[str] Specifies the name for the created script in the Python script repository. Must be a single word with no spaces.

func
[function, *oml.script.Callable*, or str] A Python function definition or callable object to be used with functions *oml.do_eval()*, *oml.table_apply()*, *oml.group_apply()*, *oml.row_apply()*, or *oml.index_apply()*.

description
[str, "" (default)] The comments for the script.

is_global
[bool, False (default)] Indicates whether to create a global Python script. The default value is False, which indicates the Python script to create is a private script available only to the current session user. When True, a global script is created and every user has read access to it.

overwrite
[bool, False (default)] Specifies whether to overwrite the named Python script if already exists.

Notes

The function definition of `func` will only be able to run in the Oracle Database server if

- it does not refer to any variable defined outside its scope
- if the first line of the function definition has no indentation
- if the function definition does not share a line with the definition of other objects.
- `func` is not a coroutine function.

Examples of invalid function definitions are below:

```
a_list, invalid_lambda = ([1,2,3], lambda x: x + 3)
if True:
  invalid_lambda2 = lambda y: y + 4
invalid_lambda3  = lambda x: x in a_list
invalid_lambda4 = lambda x: x**2 ; invalid_lambda5 = lambda x: x**3
```

If the object passed in as the parameter `func` is a function, this method normally retrieves the source of func using `inspect.getsource()`.

If `func` was defined in interactive mode via standard input, this method searches backwards through the interactive python history to find the function definition. The function may not be found if:

- The function definition is not of a format that can run in the Oracle Database server.
- `func` was defined by unpacking a tuple.
- `func` was added to the global symbol table by accessing and manipulating `globals()`
- The original variable name that the function was assigned to is deleted or reassigned, even if it is reassigned to the same object. An example scenario is below.

```
add_one = lambda x: x+1
add_one_2 = add_one
#calling the line below hides the function definition
add_one = add_one_2
```

If a non-standard interpreter that does not support `inspect.getsource()` is used, then the user should pass `func` as a string.

Examples

See *oml.script.create examples*

`oml.script.dir`(*name=None*, *regex_match=False*, *sctype='user'*)

Lists the scripts present in Oracle Database Python script repository.

Parameters

name
[str or None (default)] The script name or a regular expression to match the names of the scripts. When `name` is None, returns all the scripts given argument `sctype`.

regex_match
[bool, False (default)] Indicates whether argument `name` is a regular expression to match.

sctype
['user' (default), 'all', 'grant', 'granted', or 'global'.]

- user : Lists Python scripts created by current session user.
- grant : Lists Python scripts the read privilege for which has been granted by the current session user to other users.
- granted : Lists Python scripts the read privilege for which has been granted by other users to the current session user.
- global : Lists all user-created global Python scripts.
- system : Lists all system predefined Python scripts.

- all : Lists all Python scripts to which the current session user has read access.

Returns

script_df
[`pandas.DataFrame`] Contains the columns 'name', 'script', 'description', 'date' and optionally the columns 'owner' and 'grantee'.

Examples

See *oml.script.dir examples*

`oml.script.load(name, owner=None)`
Loads the named script from Oracle Database Python script repository as a callable object.

Parameters

name
[str] The name of the script to load from the script repository.

owner
[str or None (default)] An optional string specifying the owner of the named script. Without the `owner` argument, this method tries to match `name` first to scripts created by the current session user and then to global Python scripts. The `owner` argument is case-sensitive for db user name.

Returns

callable
[oml.script.script.Callable] A callable object that acts identically to the function defined in the script. Stores a copy of its source code so it can be used with `oml.script.create()` and Embedded Python Execution functions.

Examples

See *oml.script.load examples*

`oml.script.drop(name, is_global=False, silent=False)`
Drops the named script from Oracle Database Python script repository.

Parameters

name
[str] Specifies the name of script to drop from the Python script repository.

is_global
[bool, False (default)] Indicates whether the script to drop is a global or private Python script. The default value is False, which indicates a private script.

silent
[bool, False (default)] Indicates whether to display an error message when `oml.script.drop` encounters an error in dropping the named Python script.

Examples

See *oml.script.drop examples*

An example showing the complete OML Script Management pipeline

```
>>> from sklearn import datasets
>>> import pandas as pd
```

Create table IRIS in the database from the iris data set.

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

Create a Python script for the current user.

```
>>> def build_lm1(dat):
...     from sklearn import linear_model
...     regr = linear_model.LinearRegression()
...     import pandas as pd
...     dat = pd.get_dummies(dat, drop_first=True)
...     X = dat[["SEPAL_WIDTH", "PETAL_LENGTH", "PETAL_WIDTH", "SPECIES_versicolor",
↪"SPECIES_virginica"]]
...     y = dat[["SEPAL_LENGTH"]]
...     regr.fit(X, y)
...     return regr
...
>>>
>>> oml.script.create("MYLM", func=build_lm1, description="my lm model build")
>>> res = oml.table_apply(oml_iris, func="MYLM", oml_input_type="pandas.DataFrame")
>>> res
LinearRegression()
>>> res.pull().coef_
array([[ 0.49588894,  0.82924391, -0.31515517, -0.72356196, -1.02349781]])
```

Create a global Python script available to any user.

```
>>> build_lm2 = '''def build_lm2(dat):
...     from sklearn import linear_model
...     regr = linear_model.LinearRegression()
...     X = dat[["PETAL_WIDTH"]]
...     y = dat[["PETAL_LENGTH"]]
...     regr.fit(X, y)
...     return regr
... '''
>>> oml.script.create("GLBLM", func=build_lm2, is_global=True)
>>> res = oml.table_apply(oml_iris, func="GLBLM", oml_input_type="pandas.DataFrame")
>>> res
LinearRegression()
```

List Python scripts.

```
>>> oml.script.dir()[['name', 'script', 'description']]
   name                                              script         description
0  MYLM  def build_lm1(dat):\n  from sklearn import lin...  my lm model build
>>> oml.script.dir(name="LM", regex_match=True, sctype="all")[['owner', 'name',
↪'script', 'description']]
     owner    name                                              script  \
0   PYQSYS  GLBLM  def build_lm2(dat):\n  from sklearn import lin...
1  PYQUSER   MYLM  def build_lm1(dat):\n  from sklearn import lin...

         description
0               None
1  my lm model build
```

Load a Python script into a Python function.

```
>>> MYLM = oml.script.load(name="MYLM")
>>> GMYLM = oml.script.load(name="GLBLM")
>>> MYLM(oml_iris.pull()).coef_
array([[ 0.49588894,  0.82924391, -0.31515517, -0.72356196, -1.02349781]])
>>> GMYLM(oml_iris.pull())
LinearRegression()
```

Drop Python scripts.

```
>>> oml.script.drop("MYLM")
>>> oml.script.drop("GLBLM", is_global=True)
>>> oml.script.dir(sctype="all")
Empty DataFrame
Columns: [owner, name, script, description, date]
Index: []
>>> oml.drop("IRIS")
```

CHAPTER

SEVEN

EMBEDDING MODEL UTILITY

The utils module provides embeddings model conversion functionality

7.1 Embedding Model Configuration

class oml.utils.embeddings.**EmbeddingModelConfig**(*preconfigured_model: str = None*)

The EmbeddingModelConfig class contains the properties required for the tool to perform downloading, exporting, augmenting, validation, and storing of the ONNX model.

__init__(*preconfigured_model: str = None*)

Initializes this EmbeddingModelConfig with an empty set of properties, or the properties of a preconfigured model if specified.

Args: preconfigured_model (str, optional): The name of a preconfigured model. If the model name is not None and not in the preconfigured.json, an error will be thrown. Defaults to None.

static from_template(*name: str, **kwargs*)

Instantiates a configuration object from a named template.

Args:

name (str): The name of the template. kwargs (dict): key=value pair of overrides.

Returns:

EmbeddingModelConfig: An instance of the EmbeddingModelConfig class. The attributes of this class will be created from properties in the template and overrides.

static show_preconfigured(*include_properties: bool = False, model_name: str = None*) → list

Shows a list of preconfigured model names, or properties. By default, this function returns a list of names only. If the properties are required, pass the *include_properties* parameter as True. The returned list will contain a single dict where each key of the dict is the name of a preconfigured model and the value is the property set for that model. Finally, if only a single set of properties for a specific model is required, pass the name of the model in the *model_name* parameter (the *include_properties* parameter should also be True). This will return a list of a single dict with the properties for the specified model.

Args:

include_properties (bool,optional): A flag indicating whether properties should be included in the results. Defaults to False so only names will be included by default. model_name (str,optional): A model name to filter by when including properties. This argument will be ignored if include_properties is False. Otherwise only the properties of this model will be included in the results.

Returns:

list: A list of preconfigured model names or properties.

static show_templates() → list
Shows a list of existing templates.

Returns:
list: A list of template names.

Examples

```
>>> from oml.utils import EmbeddingModelConfig
```

7.2 Embedding Model

class oml.utils.embeddings.**EmbeddingModel**(*model_name: str*, *config: oml.utils.embeddings.EmbeddingModelConfig = None*, *settings: dict = None*)

The EmbeddingModel class is used to convert embedding models to the ONNX format, then export this intermediary model to an augmented ONNX model with pre-processing and post-processing steps embedded into the final model. The final model is then persisted in the database.

__init__(*model_name: str*, *config: oml.utils.embeddings.EmbeddingModelConfig = None*, *settings: dict = None*)
Initializes this EmbeddingModel class

Args:
model_name (str): The full name of a huggingface model config (EmbeddingModelConfig, optional): An instance of an EmbeddingModelConfig that contains the properties for the model. When this argument

is not specified, it is assumed that the model is a preconfigured model. Defaults to None.

settings (dict, optional): A set of settings (global properties). Defaults to None.

export2db(*export_name: str*)
Exports the model to the database.

Args:
export_name (str): The name that will be used for the mining model object. This name must be compliant with existing rules for object names in the database.

export2file(*export_name: str*, *output_dir=None*)
Exports the model to the database.

Args:
export_name (str): The name that will be used for the mining model object. This name must be compliant with existing rules for object names in the database. output_dir (str): The optional output directory to store the generated .onnx files.

Examples

```
>>> from oml.utils import EmbeddingModel
```

INDEX