PyOdps Documentation

0.3.12

The Alibaba Group Holding Ltd.

2016 02 25

Contents

1 3

2 5

PyOdps ODPSPythonSDK ODPSDataFrameODPS

CHAPTER 1

PyOdpsPython 2.6Python 3pip

```
pip install pyodps
```

PyOdps

CHAPTER 2

ODPS

```
from odps import ODPS

odps = ODPS('**your-access-id**', '**your-secret-access-key**', '**your-default-project**',
            endpoint='**your-end-point**')
```

ODPS listgetexistcreatedelete

2.1

ODPS Database

get_project

```
project = odps.get_project('my_project')  #
project = odps.get_project()              #
```

exist_project

2.2

ODPS

2.2.1

list_tables

```
for table in odps.list_tables():
    #
```

exist_table

get_table

```
>>> t = odps.get_table('dual')
>>> t.schema
odps.Schema {
  c_int_a                 bigint
```

```
  c_int_b                 bigint
  c_double_a              double
  c_double_b              double
  c_string_a              string
  c_string_b              string
  c_bool_a                boolean
  c_bool_b                boolean
  c_datetime_a            datetime
  c_datetime_b            datetime
}
>>> t.lifecycle
-1
>>> print(t.creation_time)
2014-05-15 14:58:43
>>> t.is_virtual_view
False
>>> t.size
1408
>>> t.schema.columns
[<column c_int_a, type bigint>,
 <column c_int_b, type bigint>,
 <column c_double_a, type double>,
 <column c_double_b, type double>,
 <column c_string_a, type string>,
 <column c_string_b, type string>,
 <column c_bool_a, type boolean>,
 <column c_bool_b, type boolean>,
 <column c_datetime_a, type datetime>,
 <column c_datetime_b, type datetime>]
```

2.2.2 Schema

```
>>> from odps.models import Schema, Column, Partition
>>> columns = [Column(name='num', type='bigint', comment='the column')]
>>> partitions = [Partition(name='pt', type='string', comment='the partition')]
>>> schema = Schema(columns=columns, partitions=partitions)
>>> schema.columns
[<column num, type bigint>, <partition pt, type string>]
```

Schema.from_lists

```
>>> schema = Schema.from_lists(['num'], ['bigint'], ['pt'], ['string'])
>>> schema.columns
[<column num, type bigint>, <partition pt, type string>]
```

2.2.3

Schema

```
>>> table = odps.create_table('my_new_table', schema)
>>> table = odps.create_table('my_new_table', schema, if_not_exists=True)  #
```

lifecycle

2.2.4

1 head

```
>>> t = odps.get_table('dual')
>>> for record in t.head(3):
>>>     print(record[0])  # 0
>>>     print(record['c_double_a'])  #
>>>     print(record[0: 3])  #
>>>     print(record[0, 2, 3])  #
>>>     print(record['c_int_a', 'c_double_a'])  #
```

table open_reader reader **with**

```
>>> with t.open_reader(partition='pt=test') as reader:
>>>     count = reader.count
>>>     for record in reader[5:10]  # countrecord
>>>         #
```

Tunnel APIopen_reader Tunnel API

2.2.5

open_readertable open_writer writer **with**

```
>>> with t.open_writer(partition='pt=test') as writer:
>>>     t.write(records)  # recordsrecordsblock 0
>>>
>>> with t.open_writer(partition='pt=test', blocks=[0, 1]) as writer:  # block
>>>     t.write(0, gen_records(block=0))
>>>     t.write(1, gen_records(block=1))  # block
```

Tunnel API

2.2.6

```
>>> odps.delete_table('my_table_name', if_exists=True)  #
>>> t.drop()  # Tabledrop
```

2.2.7

```
>>> for partition in table.partitions:
>>>     print(partition.name)
>>> for partition in table.iterate_partitions(spec='pt=test'):
>>>     #
```

```
>>> table.exist_partition('pt=test,sub=2015')
```

```
>>> partition = table.get_partition('pt=test')
>>> print(partition.creation_time)
2015-11-18 22:22:27
>>> partition.size
0
```

```
>>> t.create_partition('pt=test', if_not_exists=True)  #
```

```
>>> t.delete_partition('pt=test', if_exists=True)  #
>>> partition.drop()  # Partitiondrop
```

2.3 SQL

PyOdpsODPS SQL

2.3.1 SQL

```
>>> odps.execute_sql('select * from dual')  #  SQL
>>> instance = odps.run_sql('select * from dual')  #
>>> instance.wait_for_success()  #
```

2.3.2 SQL

SQLinstance `open_reader` SQL

```
>>> with odps.execute_sql('select * from dual').open_reader() as reader:
>>>     for record in reader:
>>>         # record
```

SQL `desc` `reader.raw` SQL

```
>>> with odps.execute_sql('desc dual').open_reader() as reader:
>>>     print(reader.raw)
```

2.4

ODPSUDFMapReduce

`list_resources` `exist_resource` `delete_resource`Resource `drop`

PyOdps

2.4.1

`file` `pyjararchive`

file-like

```
resource = odps.create_resource('test_file_resource', 'file', file_obj=open('/to/path/file'))  # file
resource = odps.create_resource('test_py_resource', 'py', file_obj='import this')  #
```

`open` odps `open_resource` file-like Python `open`

```
>>> with resource.open('r') as fp:  #
>>>     content = fp.read()  #
>>>     fp.seek(0)  #
>>>     lines = fp.readlines()  #
>>>     fp.write('Hello World')  #
>>>
>>> with odps.open_resource('test_file_resource', mode='r+') as fp:  #
>>>     fp.read()
>>>     fp.tell()  #
>>>     fp.seek(10)
>>>     fp.truncate()  #
>>>     fp.writelines(['Hello\n', 'World\n'])  #
>>>     fp.write('Hello World')
>>>     fp.flush()  # ODPS
```

- `r`
- `w`
- `a`
- `r+`
- `w+` `r+`
- `a+` `r+`

PyOdps `rb` `r+b`

2.4.2

```
>>> odps.create_resource('test_table_resource', 'table', table_name='my_table', partition='pt=test')
```

```
>>> table_resource = odps.get_resource('test_table_resource')
>>> table_resource.update(partition='pt=test2', project_name='my_project2')
```

2.5

ODPS ODPS SQL

2.5.1

`list_functions` `exist_function` `get_function`

2.5.2

```
>>> resource = odps.get_resource('my_udf.py')
>>> function = odps.create_function('test_function', class_type='my_udf.Test', resources=[resource,
```

2.5.3

```
>>> odps.delete_function('test_function')
>>> function.drop()  # Functiondrop
```

2.6

ODPS TunnelODPSTunnelODPS

2.6.1

```
from odps.tunnel import TableTunnel

table = odps.get_table('my_table')

tunnel = TableTunnel(odps)
upload_session = tunnel.create_upload_session(table.name, partition_spec='pt=test')

with upload_session.open_record_writer(0) as writer:
    record = table.new_record()
    record[0] = 'test1'
    record[1] = 'id1'
    writer.write(record)

    record = table.new_record(['test2', 'id2'])
    writer.write(record)

upload_session.commit([0])
```

2.6.2

```
from odps.tunnel import TableTunnel

tunnel = TableTunnel(odps)
download_session = tunnel.create_download_session('my_table', partition_spec='pt=test')

with download_session.open_record_reader(0, download_session.count) as reader:
    for record in reader:
        #
```

2.7 IPython

2.7.1

PyOdps

```
from odps.inter import setup, enter, teardown
```

```
setup('**your-access_id**', '**your-access-key**', '**your-project**', endpoint='**your-endpoint**)
```

roomdefaultroom

```
room = enter()
```

ODPS

```
o = room.odps
```

```
o.get_table('dual')
```

```
odps.Table
  name: odps_test_sqltask_finance.`dual`
  schema:
    c_int_a                 : bigint
    c_int_b                 : bigint
    c_double_a              : double
    c_double_b              : double
    c_string_a              : string
    c_string_b              : string
    c_bool_a                : boolean
    c_bool_b                : boolean
    c_datetime_a            : datetime
    c_datetime_b            : datetime
```

ODPSroom

```
room.store('', o.get_table('dual'), desc='')
```

display

```
room.display()
```

room['']room.iris

```
room['']
```

```
odps.Table
  name: odps_test_sqltask_finance.`dual`
  schema:
    c_int_a                 : bigint
    c_int_b                 : bigint
    c_double_a              : double
    c_double_b              : double
    c_string_a              : string
    c_string_b              : string
    c_bool_a                : boolean
    c_bool_b                : boolean
    c_datetime_a            : datetime
    c_datetime_b            : datetime
```

drop

```
room.drop('')
```

```
room.display()
```

roomteardownroom

```
teardown()
```

2.7.2 IPython

PyOdpsIPythonODPS

```
%load_ext odps
```

```
%enter
```

```
<odps.inter.Room at 0x11341df10>
```

```
room = _
```

room

```
o = room.odps
```

```
o.get_table('dual')
```

```
odps.Table
  name: odps_test_sqltask_finance.`dual`
  schema:
    c_int_a                 : bigint
    c_int_b                 : bigint
    c_double_a              : double
    c_double_b              : double
    c_string_a              : string
    c_string_b              : string
    c_bool_a                : boolean
    c_bool_b                : boolean
    c_datetime_a            : datetime
    c_datetime_b            : datetime
```

```
%stores
```

SQL

PyOdpsSQLODPS SQLSQL

```
%sql select * from pyodps_iris limit 5
```

SQL%%sql

```
%%sql

select * from pyodps_iris
where sepallength < 5
limit 5
```

SQL：

```
In [1]: %load_ext odps

In [2]: mytable = 'dual'

In [3]: %sql select * from :mytable
|==========================================|   1 /  1  (100.00%)         2s
Out[3]:
   c_int_a  c_int_b  c_double_a  c_double_b  c_string_a  c_string_b c_bool_a  \
0        0        0       -1203           0           0       -1203     True

  c_bool_b         c_datetime_a         c_datetime_b
0    False  2012-03-30 23:59:58  2012-03-30 23:59:59
```

pandas DataFrameODPS

PyOdpspandas DataFrameODPS:

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.arange(9).reshape(3, 3), columns=list('abc'))
```

```
%persist df pyodps_pandas_df
```

0`df`pyodps_pandas_dfODPS

2.8 PyOdps DataFrame

PyOdpsDataFrame APIpandasODPS

2.8.1

```
o = ODPS('**your-access-id**', '**your-secret-access-key**',
         project='**your-project**', endpoint='**your-end-point**'))
```

movielens 100K`pyodps_ml_100k_moviespyodps_ml_100k_userspyodps_ml_100k_ratings`

DataFrameTable

```
from odps.df import DataFrame
```

```
users = DataFrame(o.get_table('pyodps_ml_100k_users'))
```

dtypesDataFrame

```
users.dtypes
```

```
odps.Schema {
  user_id              int64
  age                  int64
  sex                  string
  occupation           string
```

```
  zip_code            string
}
```

headN

```
users.head(10)
```

```
users[['user_id', 'age']].head(5)
```

```
users.exclude('zip_code', 'age').head(5)
```

sexMTrueFalsesex_bool

```
users.select(users.exclude('zip_code', 'sex'), sex_bool=users.sex == 'M').head(5)
```

2025

```
users.age.between(20, 25).count().rename('count')
```

```
943
```

```
users.groupby(users.sex).count()
```

10

```
df = users.groupby('occupation').agg(count=users['occupation'].count())
df.sort(df['count'], ascending=False)[:10]
```

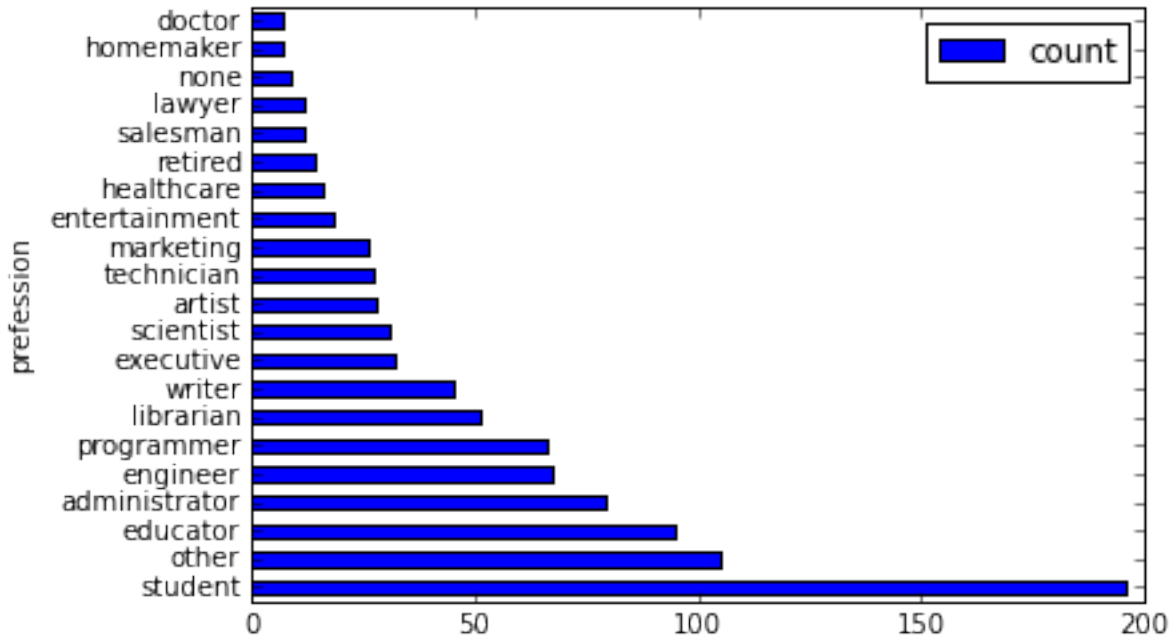
DataFrame APIvalue_counts

```
users.occupation.value_counts()[:10]
```

```
%matplotlib inline
```

```
users['occupation'].value_counts().plot(kind='barh', x='occupation', ylabel='prefession')
```

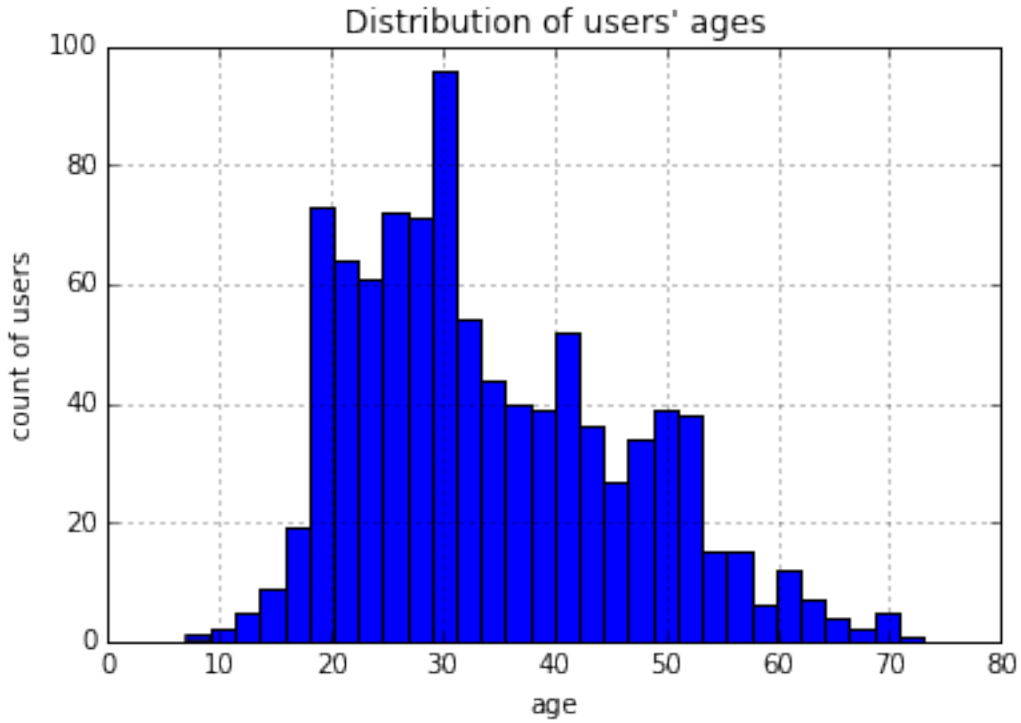
```
<matplotlib.axes._subplots.AxesSubplot at 0x10653cfd0>
```



30

```
users.age.hist(bins=30, title="Distribution of users' ages", xlabel='age', ylabel='count of users')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10667a510>
```



joinjoin

```
movies = DataFrame(o.get_table('pyodps_ml_100k_movies'))
ratings = DataFrame(o.get_table('pyodps_ml_100k_ratings'))

o.delete_table('pyodps_ml_100k_lens', if_exists=True)
lens = movies.join(ratings).join(users).persist('pyodps_ml_100k_lens')
```

```
lens.dtypes
```

```
odps.Schema {
  movie_id                                 int64
  title                                    string
  release_date                             string
  video_release_date                       string
  imdb_url                                 string
  user_id                                  int64
  rating                                   int64
  unix_timestamp                           int64
  age                                      int64
  sex                                      string
  occupation                               string
  zip_code                                 string
}
```

0808

```
labels = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79']
cut_lens = lens[lens, lens.age.cut(range(0, 81, 10), right=False, labels=labels).rename('')]
```

10

```
cut_lens['', 'age'].distinct()[:10]
```

```
cut_lens.groupby('').agg(cut_lens.rating.count().rename(''), cut_lens.rating.mean().rename(''))
```

2.8.2 DataFrame

DataFrameTable

```
from odps.df import DataFrame
```

```
iris = DataFrame(o.get_table('pyodps_iris'))
```

`dtypes`DataFrame

```
iris.dtypes
```

```
odps.Schema {
  sepallength           float64
  sepalwidth            float64
  petallength           float64
  petalwidth            float64
  name                  string
}
```

2.8.3

PyOdps DataFrame`CollectionSequenceScalar`

2.8.4

PyOdps DataFrameTableODPSDataFrameODPS SQL

PyOdps DataFrame

`int8int16int32int64float32float64booleanstringdecimaldatetime`

ODPSDataFrame

DataFrameODPSarraymap

2.8.5

DataFrame`execute``execute`

PyOdps DataFrame`repr``execute`execute

```
iris[iris.sepallength < 5][:5]
```

```
from odps import options
options.interactive = False
```

```
iris[iris.sepallength < 5][:5]
```

```
Collection: ref_0
  odps.Table
    name: odps_test_sqltask_finance.`pyodps_iris`
    schema:
      sepallength           : double
      sepalwidth            : double
      petallength           : double
      petalwidth            : double
      name                  : string

Collection: ref_1
  Filter[collection]
    collection: ref_0
    predicate:
      Less[sequence(boolean)]
        sepallength = Column[sequence(float64)] 'sepallength' from collection ref_0
        Scalar[int8]
          5

Slice[collection]
  collection: ref_1
  stop:
    Scalar[int8]
      5
```

repr

2.8.6 ODPS SQL

compile

```
print(iris[iris.sepallength < 5].exclude('sepallength')[:5].compile())
```

```
SELECT
  t1.`sepalwidth`,
  t1.`petallength`,
  t1.`petalwidth`,
  t1.`name`
FROM odps_test_sqltask_finance.`pyodps_iris` t1
WHERE t1.`sepallength` < 5
LIMIT 5
```

2.8.7

instancelogview

```
options.verbose = True
```

```
iris[iris.sepallength < 5].exclude('sepallength')[:5].execute()
```

```
Sql compiled:
SELECT t1.`sepalwidth`, t1.`petallength`, t1.`petalwidth`, t1.`name`
FROM odps_test_sqltask_finance.`pyodps_iris` t1
WHERE t1.`sepallength` < 5
```

```
LIMIT 5
logview:
http://logview
```

```
my_logs = []
def my_logger(x):
    my_logs.append(x)

options.verbose_log = my_logger
```

```
iris[iris.sepallength < 5].exclude('sepallength')[:5].execute()
```

```
print(my_logs)
```

```
['Sql compiled:', `SELECT t1.`sepalwidth`, t1.`petallength`, t1.`petalwidth`, t1.`name` nFI
```

2.8.8

DataFrameSequenceDataFrame API

```
iris.groupby('name').sepalwidth.max()
```

sepalwidthsepalwidth_maxSequenceScalarSequence

```
(iris.sepalwidth + iris.petalwidth).rename('width_sum').head(5)
```

```
from odps.df import DataFrame
```

```
iris = DataFrame(o.get_table('pyodps_iris'))
```

```
iris.dtypes
```

```
odps.Schema {
  sepallength           float64
  sepalwidth            float64
  petallength           float64
  petalwidth            float64
  name                  string
}
```

2.8.9

```
iris.sepallength.head(5)
```

```
iris['sepallength'].head(5)
```

```
iris['name', 'sepallength'].head(5)
```

```
iris.exclude('sepallength', 'petallength')[:5]
```

rename

```
iris[iris, (iris.sepalwidth + 1).rename('sepalwidthplus1')].head(5)
```

```
iris[iris.exclude('sepalwidth'), iris.sepalwidth * 2].head(5)
```

select

```
iris.select('name', sepalwidthminus1=iris.sepalwidth - 1).head(5)
```

collectionsequencecollection

```
iris['name', 'petallength'][[iris.name, ]].head(5)
```

```
df = iris['name', 'petallength']
df[[df.name, ]].head(5)
```

lambda

```
iris['name', 'petallength'][[lambda x: x.name]].head(5)
```

2.8.10

sepallength5

```
iris[iris.sepallength > 5].head(5)
```

```
iris[(iris.sepallength < 5) & (iris['petallength'] > 1.5)].head(5)
```

```
iris[(iris.sepalwidth < 2.5) | (iris.sepalwidth > 4)].head(5)
```

```
iris[~(iris.sepalwidth > 3)].head(5)
```

filter

```
iris.filter(iris.sepalwidth > 3.5, iris.sepalwidth < 4).head(5)
```

lambda

```
iris[iris.sepalwidth > 3.8]['name', lambda x: x.sepallength + 1]
```

2.8.11

```
iris[:3]
```

startsteplimit

```
iris.limit(3)
```

collectionsequence

2.8.12 DataFrame

persistDataFrame

```
iris2 = iris[iris.sepalwidth < 2.5].persist('pyodps_iris2')
```

```
iris2.head(5)
```

persistpartitionspartitions

```
iris3 = iris[iris.sepalwidth < 2.5].persist('pyodps_iris3', partitions=['name'])
```

```
iris3.data
```

```
odps.Table
  name: odps_test_sqltask_finance.`pyodps_iris3`
  schema:
    sepallength           : double
    sepalwidth            : double
    petallength           : double
    petalwidth            : double
  partitions:
    name                  : string
```

```
from odps.df import DataFrame
```

```
iris = DataFrame(o.get_table('pyodps_iris'))
lens = DataFrame(o.get_table('pyodps_ml_100k_lens'))
```

2.8.13

Sequencesin

2.8.14

DataFrameSequencemap

```
iris.sepallength.map(lambda x: x + 1).head(5)
```

mapSequencemap

```
iris.sepallength.map(lambda x: 't'+str(x), 'string').head(5)
```

mapUDFstr Function

mapPythonODPS Python UDFProjectPython UDFmapPython UDFnumpy

DataFramemap**ProjectPython UDF**

2.8.15 NULL

DataFrame APINULLisnullNULLnotnullfillnaNULL

```
iris.sepallength.isnull().head(5)
```

2.8.16

ifelseboolean01

```
(iris.sepallength > 5).ifelse('gt5', 'lte5').rename('cmp5').head(5)
```

switch

```
iris.sepallength.switch(4.9, 'eq4.9', 5.0, 'eq5.0', default='noeq').rename('equalness').head(5)
```

```
from odps.df import switch

switch(iris.sepallength == 4.9, 'eq4.9', iris.sepallength == 5.0, 'eq5.0', default='noeq').rename('eq
```

2.8.17

+-*/logsin

```
(iris.sepallength * 10).log().head(5)
```

```
fields = [iris.sepallength,
          (iris.sepallength / 2).rename('sepallength2'),
          (iris.sepallength ** 2).rename('sepallength')]
iris[fields].head(5)
```

sequencesequencescalar

```
(iris.sepallength < 5).head(5)
```

DataFrame API`3 <= iris.sepallength <= 5`between

```
(iris.sepallength.between(3, 5)).head(5)
```

betweeninclusive=False

```
(iris.sepallength.between(3, 5, inclusive=False)).head(5)
```

2.8.18 String

DataFrame APIstringSequenceScalar

```
fields = [
    iris.name.upper().rename('upper_name'),
    iris.name.extract('Iris(.*)', group=1)
]
iris[fields].head(5)
```

string

2.8.19

datetimeSequenceScalar

```
df = lens[[lens.unix_timestamp.astype('datetime').rename('dt')]]
df[df.dt,
   df.dt.year.rename('year'),
   df.dt.month.rename('month'),
   df.dt.day.rename('day'),
   df.dt.hour.rename('hour')].head(5)
```

2.8.20

isinSequencenotin

```
iris.sepallength.isin([4.9, 5.1]).rename('sepallength').head(5)
```

cutSequence

```
iris.sepallength.cut(range(6), labels=['0-1', '1-2', '2-3', '3-4', '4-5']).rename('sepallength_cut')
```

include_underinclude_over

```
labels=['0-1', '1-2', '2-3', '3-4', '4-5', '5-']
iris.sepallength.cut(range(6), labels=labels, include_over=True).rename('sepallength_cut').head(5)
```

```
from odps.df import DataFrame
```

```
iris = DataFrame(o.get_table('pyodps_iris'))
```

2.8.21

describeDataFrame

```
print(iris.describe())
```

```
   sepallength_count  sepallength_min  sepallength_max  sepallength_mean  \
0                150              4.3              7.9          5.843333

   sepallength_std  sepalwidth_count  sepalwidth_min  sepalwidth_max  \
0         0.825301               150               2             4.4

   sepalwidth_mean  sepalwidth_std  petallength_count  petallength_min  \
0            3.054        0.432147                150                1

   petallength_max  petallength_mean  petallength_std  petalwidth_count  \
0              6.9          3.758667         1.758529               150

   petalwidth_min  petalwidth_max  petalwidth_mean  petalwidth_std
0             0.1             2.5         1.198667        0.760613
```

```
iris.sepallength.max()
```

```
7.9
```

2.8.22

DataFrame APIgroupbyaggaggregate

```
iris.groupby('name').agg(iris.sepallength.max(), smin=iris.sepallength.min())
```

DataFrame APIvalue_counts

groupby

```
iris.groupby('name').agg(count=iris.name.count()).sort('count', ascending=False).head(5)
```

value_counts

```
iris['name'].value_counts().head(5)
```

```
iris.groupby('name').petallength.sum()
```

```
iris.groupby('name').agg(iris.petallength.notnull().sum())
```

Scalar

```
from odps.df import Scalar
```

```
iris.groupby(Scalar(1)).petallength.sum()
```

```
from odps.df import DataFrame
```

```
iris = DataFrame(o.get_table('pyodps_iris'))
```

2.8.23

Collectionsortsort_values

```
iris.sort('sepalwidth').head(5)
```

ascendingFalse

```
iris.sort('sepalwidth', ascending=False).head(5)
```

```
iris.sort(-iris.sepalwidth).head(5)
```

```
iris.sort(['sepalwidth', 'petallength']).head(5)
```

ascendingboolean

```
iris.sort(['sepalwidth', 'petallength'], ascending=[True, False]).head(5)
```

```
iris.sort(['sepalwidth', -iris.petallength]).head(5)
```

2.8.24

Collectiondistinct

```
iris[['name']].distinct()
```

```
iris.distinct('name')
```

```
iris.distinct('name', 'sepallength').head(3)
```

```
from odps.df import DataFrame
```

```
movies = DataFrame(o.get_table('pyodps_ml_100k_movies'))
ratings = DataFrame(o.get_table('pyodps_ml_100k_ratings'))
```

```
movies.dtypes
```

```
odps.Schema {
  movie_id                              int64
  title                                 string
  release_date                          string
  video_release_date                    string
  imdb_url                              string
}
```

```
ratings.dtypes
```

```
odps.Schema {
  user_id                       int64
  movie_id                      int64
  rating                        int64
  unix_timestamp                int64
}
```

2.8.25 Join

DataFrameCollectionjoinjoinDataFrame APIjoin

```
movies.join(ratings).head(3)
```

join

```
movies.join(ratings, on='movie_id').head(3)
```

movie_idmovie_id_xmovie_id_ysuffix('_x', '_y')

```
ratings2 = ratings[ratings.exclude('movie_id'), ratings.movie_id.rename('movie_id2')]
ratings2.dtypes
```

```
odps.Schema {
  user_id                       int64
  rating                        int64
  unix_timestamp                int64
  movie_id2                     int64
}
```

```
movies.join(ratings2, on=[('movie_id', 'movie_id2')]).head(3)
```

```
movies.join(ratings2, on=[movies.movie_id == ratings2.movie_id2]).head(3)
```

self-joinview

```
movies2 = movies.view()
movies.join(movies2, movies.movie_id == movies2.movie_id)[movies, movies2.movie_id].head(3)
```

joinDataFrameleft_joinright_joinouter_join

2.8.26 Union

unionconcat

```
mov1 = movies[movies.movie_id < 3]['movie_id', 'title']
mov2 = movies[(movies.movie_id > 3) & (movies.movie_id < 6)]['title', 'movie_id']
mov1.union(mov2)
```

```
from odps.df import DataFrame
```

```
iris = DataFrame(o.get_table('pyodps_iris'))
```

2.8.27

DataFrame API

```
grouped = iris.groupby('name')
grouped.mutate(grouped.sepallength.cumsum(), grouped.sort('sepallength').row_number()).head(10)
```

```
iris['name', 'sepallength', iris.groupby('name').sort('sepallength').sepallength.cumcount()].head(5)
```

DataFrame API

2.8.28

PyOdps DataFramepandasmatplotlib

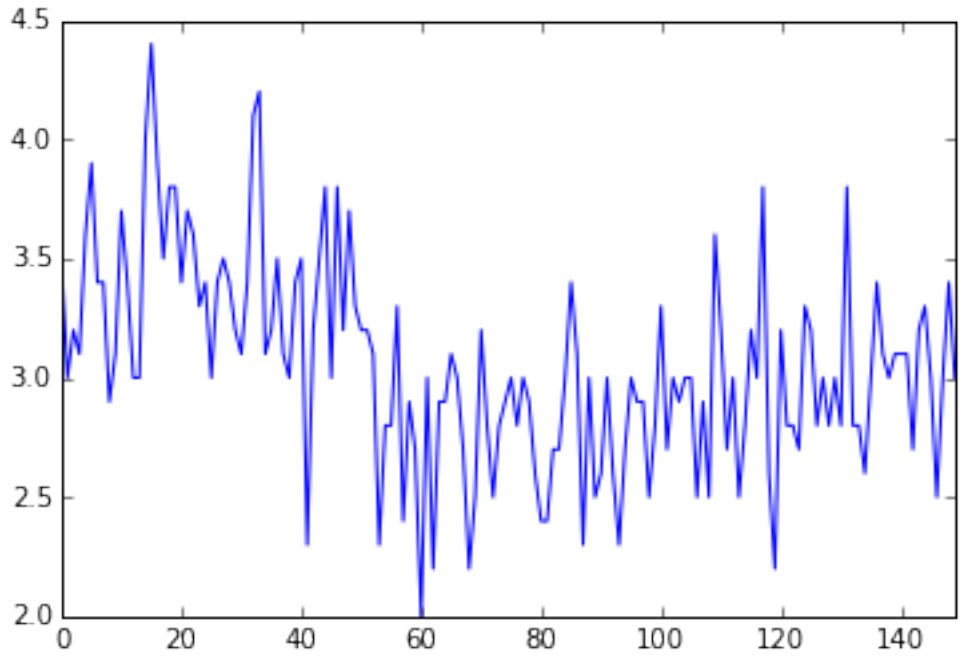
```
from odps.df import DataFrame
```

```
iris = DataFrame(o.get_table('pyodps_iris'))
```

```
%matplotlib inline
```

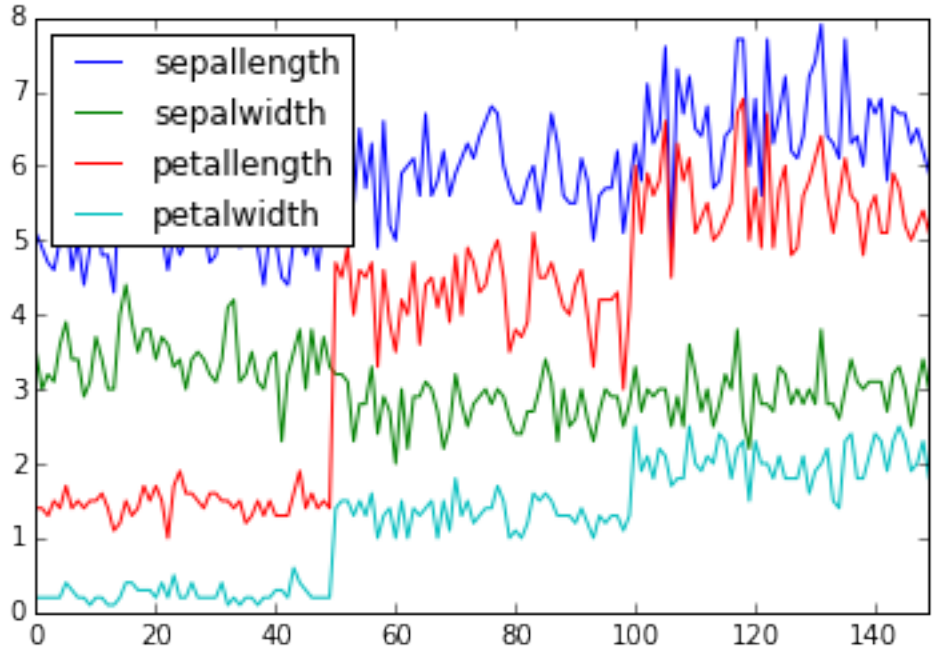
```
iris.sepalwidth.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10c2b3510>
```



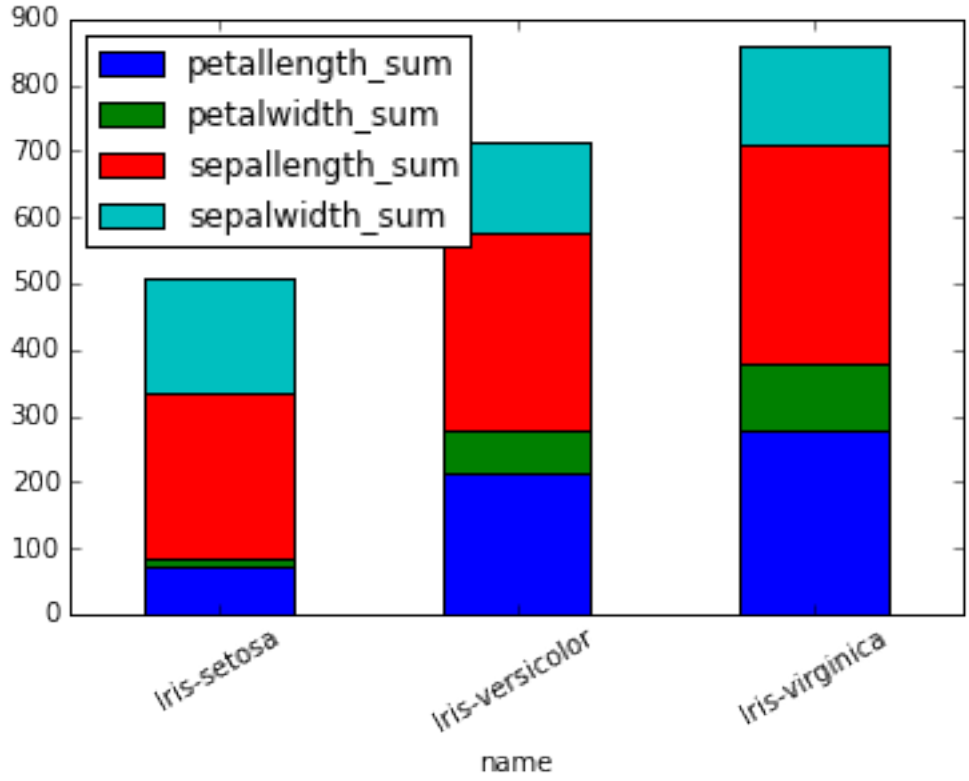

```
iris.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10db7e690>
```



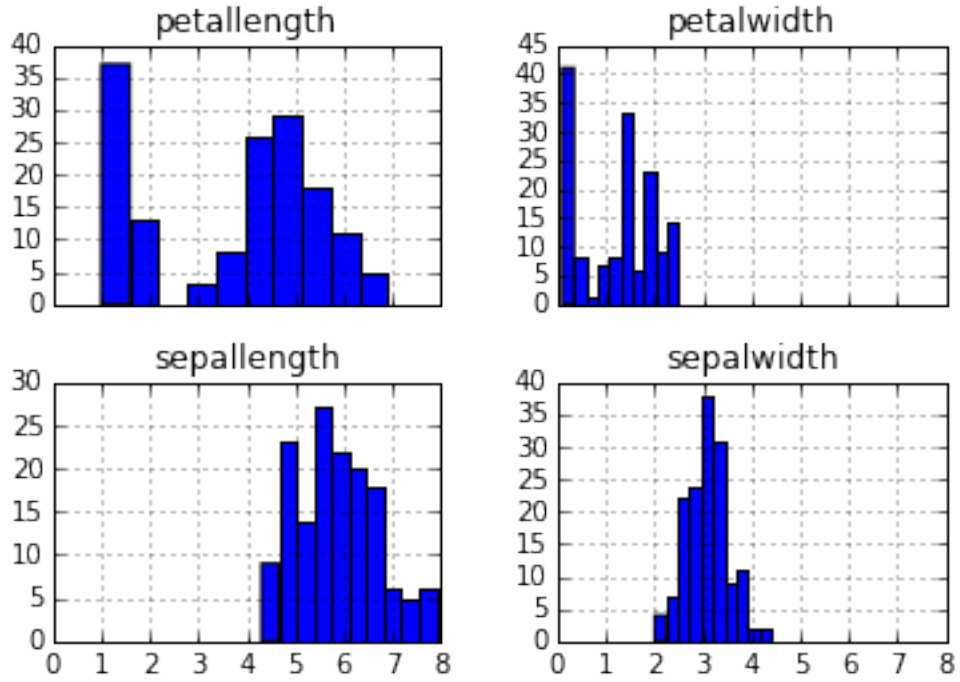

```
iris.groupby('name').sum().plot(kind='bar', x='name', stacked=True, rot=30)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x10c5f2090>
```



```
iris.hist(sharex=True)
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x10e013f90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x10e2d1c10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x10e353f10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x10e3c4410>]], dtype=object)
```



`kind`

Pandashttp://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html

2.9 API Reference

Here we give automatically generated API documentation of PyODPS.

2.9.1 Definitions

class odps.**ODPS**(*access_id*, *secret_access_key*, *project*, *endpoint='http://service.odps.aliyun.com/api'*, *tunnel_endpoint=None*)

Main entrance to ODPS.

Convenient operations on ODPS objects are provided. Please refer to ODPS docs to see the details.

Generally, basic operations such as `list`, `get`, `exist`, `create`, `delete` are provided for each ODPS object. Take the `Table` as an example.

Example

```
>>> odps = ODPS('**your access id**', '**your access key**', 'default_project')
>>>
>>> for table in odps.list_tables():
```

```
>>>     # handle each table
>>>
>>> table = odps.get_table('dual')
>>>
>>> odps.exist_table('dual') is True
>>>
>>> odps.create_table('test_table', schema)
>>>
>>> odps.delete_table('test_table')
```

create_function(*name*, *project=None*, ***kwargs*)

Create a function by given name.

- **name** – function name
- **project** – project name, if not provided, will be the default project
- **class_type** (*str*) – main class
- **resources** (*list*) – the resources that function needs to use

the created function

odps.models.Function

Example

```
>>> res = odps.get_resource('test_func.py')
>>> func = odps.create_function('test_func', class_type='test_func.Test', resources=[res, ])
```

:

odps.models.Function

create_resource(*name*, *typo*, *project=None*, ***kwargs*)

Create a resource by given name and given type.

Currently, the resource type can be `file`, `jar`, `py`, `archive`, `table`.

The `file`, `jar`, `py`, `archive` can be classified into file resource. To init the file resource, you have to provide another parameter which is a file-like object.

For the table resource, the table name, project name, and partition should be provided which the partition is optional.

- **name** – resource name
- **typo** – resource type, now support `file`, `jar`, `py`, `archive`, `table`
- **project** – project name, if not provided, will be the default project
- **kwargs** – optional arguments, I will illustrate this in the example below.

resource depends on the type, if `file` will be *odps.models.FileResource* and so on

odps.models.Resource's subclasses

Example

```
>>> from odps.models.resource import *
>>>
>>> res = odps.create_resource('test_file_resource', 'file', file_obj=open('/to/path/file'))
```

```
        >>> assert isinstance(res, FileResource)
        >>> True
        >>>
        >>> res = odps.create_resource('test_py_resource.py', 'py', file_obj=StringIO('import this'))
        >>> assert isinstance(res, PyResource)
        >>> True
        >>>
        >>> res = odps.create_resource('test_table_resource', 'table', table_name='test_table', part
        >>> assert isinstance(res, TableResource)
        >>> True
        >>>
```

:

odps.models.FileResource, *odps.models.PyResource*, *odps.models.JarResource*, *odps.models.ArchiveResource*, *odps.models.TableResource*

create_table(*name*, *schema*, *project=None*, *comment=None*, *if_not_exists=False*, *lifecycle=None*, *shard_num=None*, *hub_lifecycle=None*)

Create an table by given schema and other optional parameters.

- **name** – table name
- **schema** (*odps.models.Schema*) – table schema
- **project** – project name, if not provided, will be the default project
- **comment** – table comment
- **if_not_exists** (*bool*) – will not create if this table already exists, default False
- **lifecycle** (*int*) – table's lifecycle
- **shard_num** (*int*) – table's shard num
- **hub_lifecycle** (*int*) – hub lifecycle

the created Table

odps.models.Table

:

odps.models.Table, *odps.models.Schema*

delete_function(*name*, *project=None*)

Delete a function by given name.

- **name** – function name
- **project** – project name, if not provided, will be the default project

None

delete_offline_model(*name*, *project=None*)

Delete the offline model by given name.

- **name** – offline model's name
- **project** – project name, if not provided, will be the default project

None

delete_resource(*name*, *project=None*)
Delete resource by given name.

- **name** – resource name
- **project** – project name, if not provided, will be the default project

None

delete_table(*name*, *project=None*, *if_exists=False*)
Delete the table with given name

- **name** – table name
- **project** – project name, if not provided, will be the default project
- **if_exists** – will not delete the table if not exists, default False

None

delete_xflow(*name*, *project=None*)
Delete xflow by given name.

- **name** – xflow name
- **project** – project name, if not provided, will be the default project

None

execute_sql(*sql*, *project=None*, *priority=None*, *running_cluster=None*, ***kwargs*)
Run a given SQL statement and block until the SQL executed successfully.

- **sql** (*str*) – SQL statement
- **project** – project name, if not provided, will be the default project
- **priority** (*int*) – instance priority, 9 as default
- **running_cluster** – cluster to run this instance

instance

odps.models.Instance

Example

```
>>> instance = odps.execute_sql('select * from dual')
>>> with instance.open_reader() as reader:
>>>     for record in reader:  # iterate to handle result with schema
>>>         # handle each record
>>>
>>> instance = odps.execute_sql('desc dual')
>>> with instance.open_reader() as reader:
>>>     print(reader.raw)  # without schema, just get the raw result
```

:

odps.models.Instance

execute_xflow(*xflow_name*, *xflow_project=None*, *parameters=None*, *project=None*)
Run xflow by given name, xflow project, paremeters, block until xflow executed successfully.

- **xflow_name** (*str*) – XFlow name
- **xflow_project** (*str*) – the project XFlow deploys
- **parameters** (*dict*) – parameters
- **project** – project name, if not provided, will be the default project

instance

odps.models.Instance

:

odps.models.Instance

exist_function(*name*, *project=None*)
If the function with given name exists or not.

- **name** – function name
- **project** – project name, if not provided, will be the default project

True if the function exists or False

bool

exist_instance(*id_*, *project=None*)
If the instance with given id exists or not.

- **id** – instance id
- **project** – project name, if not provided, will be the default project

True if exists or False

bool

exist_offline_model(*name*, *project=None*)
If the offline model with given name exists or not.

- **name** – offline model's name
- **project** – project name, if not provided, will be the default project

True if offline model exists else False

bool

exist_project(*name*)
If project name which provided exists or not.

name – project name, if not provided, will be the default project

True if exists or False

bool

exist_resource(*name*, *project=None*)
If the resource with given name exists or not.

- **name** – resource name
- **project** – project name, if not provided, will be the default project

True if exists or False

bool

exist_table(*name*, *project=None*)
If the table with given name exists or not.

- **name** – table name
- **project** – project name, if not provided, will be the default project

True if table exists or False

bool

exist_xflow(*name*, *project=None*)
If the xflow with given name exists or not.

- **name** – xflow name
- **project** – project name, if not provided, will be the default project

True if exists or False

bool

get_function(*name*, *project=None*)
Get the function by given name

- **name** – function name
- **project** – project name, if not provided, will be the default project

the right function

Raise `odps.errors.NoSuchObject` if not exists

:

odps.models.Function

get_instance(*id_*, *project=None*)
Get instance by given instance id.

- **id** – instance id
- **project** – project name, if not provided, will be the default project

the right instance

odps.models.Instance

Raise `odps.errors.NoSuchObject` if not exists

:

odps.models.Instance

get_logview_address(*instanceid*, *hours*, *project=None*)
Get logview address by given instance id and hours.

- **instanceid** – instance id
- **hours** –
- **project** – project name, if not provided, will be the default project

logview address

str

get_offline_model(*name*, *project=None*)
Get offline model by given name

- **name** – offline model name
- **project** – project name, if not provided, will be the default project

offline model

odps.models.ml.OfflineModel

Raise `odps.errors.NoSuchObject` if not exists

get_project(*name=None*)
Get project by given name.

name – project name, if not provided, will be the default project

the right project

odps.models.Project

Raise `odps.errors.NoSuchObject` if not exists

:

odps.models.Project

get_resource(*name*, *project=None*)
Get a resource by given name

- **name** – resource name
- **project** – project name, if not provided, will be the default project

the right resource

odps.models.Resource

Raise `odps.errors.NoSuchObject` if not exists

:

odps.models.Resource

get_table(*name*, *project=None*)
Get table by given name.

- **name** – table name
- **project** – project name, if not provided, will be the default project

the right table

odps.models.Table

Raise odps.errors.NoSuchObject if not exists

:

odps.models.Table

get_xflow(*name*, *project=None*)
Get xflow by given name

- **name** – xflow name
- **project** – project name, if not provided, will be the default project

xflow

odps.models.XFlow

Raise odps.errors.NoSuchObject if not exists

:

odps.models.XFlow

get_xflow_results(*instance*, *project=None*)
The result given the instance of xflow

- **instance** (*odps.models.Instance*) – instance of xflow
- **project** – project name, if not provided, will be the default project

xflow result

dict

list_functions(*project=None*)
List all functions of a project.

project – project name, if not provided, will be the default project

functions

generator

list_instances(*project=None*, *from_time=None*, *end_time=None*, *status=None*, *only_owner=None*)
List instances of a project by given optional conditions including start time, end time, status and if only the owner.

- **project** – project name, if not provided, will be the default project
- **from_time** (*datetime, int or float*) – the start time of filtered instances
- **end_time** (*datetime, int or float*) – the end time of filtered instances

- **status** – including odps.models.Instance.Status.Running, odps.models.Instance.Status.Suspended, odps.models.Instance.Status.Terminate
- **only_owner** (*bool*) – True will filter the instances created by current user

instances

list

list_offline_models(*project=None*, *prefix=None*, *owner=None*)
List offline models of project by optional filter conditions including prefix and owner.

- **project** – project name, if not provided, will be the default project
- **prefix** – prefix of offline model's name
- **owner** – Aliyun account

offline models

list

list_resources(*project=None*)
List all resources of a project.

project – project name, if not provided, will be the default project

resources

generator

list_tables(*project=None*, *prefix=None*, *owner=None*)
List all tables of a project. If prefix is provided, the listed tables will all start with this prefix. If owner is provided, the listed tables will be belong to such owner.

- **project** – project name, if not provided, will be the default project
- **prefix** (*str*) – the listed tables start with this **prefix**
- **owner** (*str*) – Aliyun account, the owner which listed tables belong to

tables in this project, filtered by the optional prefix and owner.

generator

list_xflows(*project=None*, *owner=None*)
List xflows of a project which can be filtered by the xflow owner.

- **project** – project name, if not provided, will be the default project
- **owner** (*str*) – Aliyun account

xflows

list

open_resource(*name*, *project=None*, *mode='r+'*, *encoding='utf-8'*)
Open a file resource as file-like object. This is an elegant and pythonic way to handle file resource.

The argument mode stands for the open mode for this file resource. It can be binary mode if the 'b' is inside. For instance, 'rb' means opening the resource as read binary mode while 'r+b' means opening the

resource as read+write binary mode. This is most import when the file is actually binary such as tar or jpeg file, so be aware of opening this file as a correct mode.

Basically, the text mode can be 'r', 'w', 'a', 'r+', 'w+', 'a+' just like the builtin python `open` method.

- `r` means read only
- `w` means write only, the file will be truncated when opening
- `a` means append only
- `r+` means read+write without constraint
- `w+` will truncate first then opening into read+write
- `a+` can read+write, however the written content can only be appended to the end

- **name** (*odps.models.FileResource* or str) – file resource or file resource name
- **project** – project name, if not provided, will be the default project
- **mode** – the mode of opening file, described as above
- **encoding** – utf-8 as default

file-like object

Example

```
>>> with odps.open_resource('test_resource', 'r') as fp:
>>>     fp.read(1)  # read one unicode character
>>>     fp.write('test')  # wrong, cannot write under read mode
>>>
>>> with odps.open_resource('test_resource', 'wb') as fp:
>>>     fp.readlines() # wrong, cannot read under write mode
>>>     fp.write('hello world') # write bytes
>>>
>>> with odps.open_resource('test_resource') as fp: # default as read-write mode
>>>     fp.seek(5)
>>>     fp.truncate()
>>>     fp.flush()
```

read_table(*name*, *limit=None*, *start=0*, *step=None*, *project=None*, *partition=None*, ***kw*)

Read table's records.

- **name** (odps.models.table.Table or str) – table or table name
- **limit** – the records' size, if None will read all records from the table
- **start** – the record where read starts with
- **step** – default as 1
- **project** – project name, if not provided, will be the default project
- **partition** – the partition of this table to read
- **columns** (*list*) – the columns' names which are the parts of table's columns
- **compress** (*bool*) – if True, the data will be compressed during downloading

- **compress_option** (odps.tunnel.CompressOption) – the compression algorithm, level and strategy
- **endpoint** – tunnel service URL
- **reopen** – reading the table will reuse the session which opened last time, if set to True will open a new download session, default as False

records

generator

Example

```
>>> for record in odps.read_table('test_table', 100):
>>>     # deal with such 100 records
>>> for record in odps.read_table('test_table', partition='pt=test', start=100, limit=100):
>>>     # read the `pt=test` partition, skip 100 records and read 100 records
```

:

odps.models.Record

run_sql(*sql*, *project=None*, *priority=None*, *running_cluster=None*, ***kwargs*)
Run a given SQL statement asynchronously

- **sql** (*str*) – SQL statement
- **project** – project name, if not provided, will be the default project
- **priority** (*int*) – instance priority, 9 as default
- **running_cluster** – cluster to run this instance

instance

odps.models.Instance

:

odps.models.Instance

run_xflow(*xflow_name*, *xflow_project=None*, *parameters=None*, *project=None*)
Run xflow by given name, xflow project, paremeters asynchronously.

- **xflow_name** (*str*) – XFlow name
- **xflow_project** (*str*) – the project XFlow deploys
- **parameters** (*dict*) – parameters
- **project** – project name, if not provided, will be the default project

instance

odps.models.Instance

:

odps.models.Instance

stop_instance(*id_*, *project=None*)
Stop the running instance by given instance id.

- **id** – instance id
- **project** – project name, if not provided, will be the default project

None

stop_job(*id_*, *project=None*)
Stop the running instance by given instance id.

- **id** – instance id
- **project** – project name, if not provided, will be the default project

None

write_table(*name*, **block_records*, ***kw*)
Write records into given table.

- **name** (models.table.Table or str) – table or table name
- **block_records** – if given records only, the block id will be 0 as default.
- **project** – project name, if not provided, will be the default project
- **partition** – the partition of this table to write
- **compress** (*bool*) – if True, the data will be compressed during uploading
- **compress_option** (odps.tunnel.CompressOption) – the compression algorithm, level and strategy
- **endpoint** – tunnel service URL
- **reopen** – writing the table will reuse the session which opened last time, if set to True will open a new upload session, default as False

None

Example

```
>>> odps.write_table('test_table', records)  # write to block 0 as default
>>>
>>> odps.write_table('test_table', 0, records)  # write to block 0 explicitly
>>>
>>> odps.write_table('test_table', 0, records1, 1, records2)  # write to multi-blocks
>>>
>>> odps.write_table('test_table', records, partition='pt=test') # write to certain partitio
```

:

odps.models.Record

class odps.models.**Project**(***kwargs*)
Project is the counterpart of **database** in a RDBMS.

By get an object of Project, users can get the properties like name, owner, comment, creation_time, last_modified_time, and so on.

These properties will not load from remote ODPS service, unless users try to get them explicitly. If users want to check the newest status, try use reload method.

Example

```
>>> project = odps.get_project('my_project')
>>> project.last_modified_time  # this property will be fetched from the remote ODPS service
>>> project.last_modified_time  # Once loaded, the property will not bring remote call
>>> project.owner  # so do the other properties, they are fetched together
>>> project.reload()  # force to update each properties
>>> project.last_modified_time  # already updated
```

class odps.models.**Table**(***kwargs*)

Table means the same to the RDBMS table, besides, a table can consist of partitions.

Table's properties are the same to the ones of *odps.models.Project*, which will not load from remote ODPS service until users try to get them.

In order to write data into table, users should call the `open_writer` method with **with statement**. At the same time, the `open_reader` method is used to provide the ability to read records from a table or its partition.

Example

```
>>> table = odps.get_table('my_table')
>>> table.owner  # first will load from remote
>>> table.reload()  # reload to update the properties
>>>
>>> for record in table.head(5):
>>>     # check the first 5 records
>>> for record in table.head(5, partition='pt=test', columns=['my_column'])
>>>     # only check the `my_column` column from certain partition of this table
>>>
>>> with table.open_reader() as reader:
>>>     count = reader.count  # How many records of a table or its partition
>>>     for record in record[0: count]:
>>>         # read all data, actually better to split into reading for many times
>>>
>>> with table.open_writer() as writer:
>>>     writer.write(records)
>>> with table.open_writer(partition='pt=test', blocks=[0, 1]):
>>>     writer.write(0, gen_records(block=0))
>>>     writer.write(1, gen_records(block=1))  # we can do this parallel
```

drop()

drop this table.

None

head(*limit*, *partition=None*, *columns=None*)

Get the head records of a table or its partition.

- **limit** – records' size, 10000 at most
- **partition** – partition of this table
- **columns** (*list*) – the columns which is subset of the table columns

records

list

:

odps.models.Record

new_record(*values=None*)
Generate a record of the table.

values (*list*) – the values of this records

record

odps.models.Record

Example

```
>>> table = odps.create_table('test_table', schema=Schema.from_lists(['name', 'id'], ['sring
>>> record = table.new_record()
>>> record[0] = 'my_name'
>>> record[1] = 'my_id'
>>> record = table.new_record(['my_name', 'my_id'])
```

:

odps.models.Record

open_reader(**args*, ***kwds*)
Open the reader to read the entire records from this table or its partition.

- **partition** – partition of this table
- **reopen** (*bool*) – the reader will reuse last one, reopen is true means open a new reader.
- **endpoint** – the tunnel service URL
- **compress_option** (`odps.tunnel.CompressOption`) – compression algorithm, level and strategy
- **compress_algo** – compression algorithm, work when `compress_option` is not provided, can be `zlib`, `snappy`
- **compress_level** – used for `zlib`, work when `compress_option` is not provided
- **compress_strategy** – used for `zlib`, work when `compress_option` is not provided

reader, `count` means the full size, `status` means the tunnel status

Example

```
>>> with table.open_reader() as reader:
>>>     count = reader.count  # How many records of a table or its partition
>>>     for record in record[0: count]:
>>>         # read all data, actually better to split into reading for many times
```

open_writer(**args*, ***kwds*)
Open the writer to write records into this table or its partition.

- **partition** – partition of this table
- **blocks** – block ids to open
- **reopen** (*bool*) – the reader will reuse last one, reopen is true means open a new reader.
- **endpoint** – the tunnel service URL
- **compress_option** (`odps.tunnel.CompressOption`) – compression algorithm, level and strategy

- **compress_algo** – compression algorithm, work when `compress_option` is not provided, can be `zlib`, `snappy`
- **compress_level** – used for `zlib`, work when `compress_option` is not provided
- **compress_strategy** – used for `zlib`, work when `compress_option` is not provided

writer, status means the tunnel writer status

Example

```
>>> with table.open_writer() as writer:
>>>     writer.write(records)
>>> with table.open_writer(partition='pt=test', blocks=[0, 1]):
>>>     writer.write(0, gen_records(block=0))
>>>     writer.write(1, gen_records(block=1))  # we can do this parallel
```

odps.models.**Schema**
TableSchema

class odps.models.table.**TableSchema**(***kwargs*)

Schema includes the columns and partitions information of a *odps.models.Table*.

There are two ways to initialize a Schema object, first is to provide columns and partitions, the second way is to call the class method `from_lists`. See the examples below:

Example

```
>>> columns = [Column(name='num', type='bigint', comment='the column')]
>>> partitions = [Partition(name='pt', type='string', comment='the partition')]
>>> schema = Schema(columns=columns, partitions=partitions)
>>> schema.columns
[<column num, type bigint>, <partition pt, type string>]
>>>
>>> schema = Schema.from_lists(['num'], ['bigint'], ['pt'], ['string'])
>>> schema.columns
[<column num, type bigint>, <partition pt, type string>]
```

class odps.models.**Record**(*columns=None*, *schema=None*, *values=None*)

A record generally means the data of a single line in a table.

Example

```
>>> schema = Schema.from_lists(['name', 'id'], ['string', 'string'])
>>> record = Record(schema=schema, values=['test', 'test2'])
>>> record[0] = 'test'
>>> record[0]
>>> 'test'
>>> record['name']
>>> 'test'
>>> record[0:2]
>>> ('test', 'test2')
>>> record[0, 1]
>>> ('test', 'test2')
>>> record['name', 'id']
>>> for field in record:
>>>     print(field)
('name', u'test')
('id', u'test2')
>>> len(record)
2
```

```
>>> 'name' in record
True
```

class odps.models.**Instance**(***kwargs*)

Instance means that a ODPS task will sometimes run as an instance.

`status` can reflect the current situation of a instance. `is_terminated` method indicates if the instance has finished. `is_successful` method indicates if the instance runs successfully. `wait_for_success` method will block the main process until the instance has finished.

For a SQL instance, we can use open_reader to read the results.

Example

```
>>> instance = odps.execute_sql('select * from dual')  # this sql return the structured data
>>> with instance.open_reader() as reader:
>>>     # handle the record
>>>
>>> instance = odps.execute_sql('desc dual')  # this sql do not return structured data
>>> with instance.open_reader() as reader:
>>>    print(reader.raw)  # just return the raw result
```

class **Task**(***kwargs*)

Task stands for each task inside an instance.

It has a name, a task type, the start to end time, and a running status.

class **TaskProgress**(***kwargs*)

TaskProgress reprents for the progress of a task.

A single TaskProgress may consist of several stages.

Example

```
>>> progress = instance.get_task_progress('task_name')
>>> progress.get_stage_progress_formatted_string()
2015-11-19 16:39:07 M1_Stg1_job0:0/0/1[0%]  R2_1_Stg1_job0:0/0/1[0%]
```

Instance.**get_task_names**()

Get names of all tasks

task names

list

Instance.**get_task_progress**(*task_name*)

Get task's current progress

task_name – task_name

the task's progress

odps.models.Instance.Task.TaskProgress

Instance.**get_task_result**(*task_name*)

Get a single task result.

task_name – task name

task result

str

Instance.**get_task_results**()

Get all the task results.

a dict which key is task name, and value is the task result as string

dict

Instance.**get_task_statuses**()
Get all tasks' statuses

a dict which key is the task name and value is the *odps.models.Instance.Task* object

dict

Instance.**get_task_summary**(*task_name*)
Get a task's summary, mostly used for MapReduce.

task_name – task name

summary as a dict parsed from JSON

dict

Instance.**is_successful**()
If the instance runs successfully.

True if successful else False

bool

Instance.**is_terminated**()
If this instance has finished or not.

True if finished else False

bool

Instance.**stop**()
Stop this instance.

None

Instance.**wait_for_completion**(*interval=1*)
Wait for the instance to complete, and neglect the consequence.

interval – time interval to check

None

Instance.**wait_for_success**(*interval=1*)
Wait for instance to complete, and check if the instance is successful.

interval – time interval to check

None

Raise odps.errors.ODPSError if the instance failed

class odps.models.**Resource**(***kwargs*)
Resource is useful when writing UDF or MapReduce. This is an abstract class.

Basically, resource can be either a file resource or a table resource. File resource can be file, py, jar, archive in details.

:

odps.models.FileResource, *odps.models.PyResource*, *odps.models.JarResource*, *odps.models.ArchiveResource*, *odps.models.TableResource*

class odps.models.**FileResource**(**kw)
File resource represents for a file.

Use open method to open this resource as an file-like object.

close()
Close this file resource.

None

flush()
Commit the change to ODPS if any change happens. Close will do this automatically.

None

open(*mode='r'*, *encoding='utf-8'*)
The argument mode stands for the open mode for this file resource. It can be binary mode if the 'b' is inside. For instance, 'rb' means opening the resource as read binary mode while 'r+b' means opening the resource as read+write binary mode. This is most import when the file is actually binary such as tar or jpeg file, so be aware of opening this file as a correct mode.

Basically, the text mode can be 'r', 'w', 'a', 'r+', 'w+', 'a+' just like the builtin python open method.

- r means read only
- w means write only, the file will be truncated when opening
- a means append only
- r+ means read+write without constraint
- w+ will truncate first then opening into read+write
- a+ can read+write, however the written content can only be appended to the end

- **mode** – the mode of opening file, described as above
- **encoding** – utf-8 as default

file-like object

Example

```
>>> with resource.open('r') as fp:
>>>     fp.read(1)  # read one unicode character
>>>     fp.write('test')  # wrong, cannot write under read mode
>>>
>>> with resource.open('wb') as fp:
>>>     fp.readlines() # wrong, cannot read under write mode
>>>     fp.write('hello world') # write bytes
>>>
>>> with resource.open('test_resource', 'r+') as fp: # open as read-write mode
>>>     fp.seek(5)
>>>     fp.truncate()
>>>     fp.flush()
```

read(*size=-1*)
Read the file resource, read all as default.

size – unicode or byte length depends on text mode or binary mode.

unicode or bytes depends on text mode or binary mode

str or unicode(Py2), bytes or str(Py3)

readline(*size=-1*)
Read a single line.

size – If the size argument is present and non-negative, it is a maximum byte count (including the trailing newline) and an incomplete line may be returned. When size is not 0, an empty string is returned only when EOF is encountered immediately

unicode or bytes depends on text mode or binary mode

str or unicode(Py2), bytes or str(Py3)

readlines(*sizehint=-1*)
Read as lines.

sizehint – If the optional sizehint argument is present, instead of reading up to EOF, whole lines totalling approximately sizehint bytes (possibly after rounding up to an internal buffer size) are read.

lines

list

seek(*pos*, *whence=0*)
Seek to some place.

- **pos** – position to seek
- **whence** – if set to 2, will seek to the end

None

tell()
Tell the current position

current position

truncate(*size=None*)
Truncate the file resource's size.

size – If the optional size argument is present, the file is truncated to (at most) that size. The size defaults to the current position.

None

write(*content*)
Write content into the file resource

content – content to write

None

writelines(*seq*)
Write lines into the file resource.

seq – lines

None

class odps.models.**PyResource**(***kw*)
File resource representing for the .py file.

class odps.models.**JarResource**(***kw*)
File resource representing for the .jar file.

class odps.models.**ArchiveResource**(**kw)
File resource representing for the compressed file like .zip/.tgz/.tar.gz/.tar/jar

class odps.models.**TableResource**(**kw)
Take a table as a resource.

get_source_table()
Get the table object.

source table

odps.models.Table

:

odps.models.Table

get_source_table_partition()
Get the source table partition.

the source table partition

update(*project_name=None*, *table_name=None*, *partition=None*)
Update this resource.

- **project_name** – the source table's project
- **table_name** – the source table's name
- **partition** – the source table's partition

self

class odps.models.**Function**(**kwargs)
Function can be used in UDF when user writes a SQL.

drop()
Delete this Function.

None

resources
Return all the resources which this function refer to.

resources

list

:

odps.models.Resource

class odps.models.**XFlow**(**kwargs)

class odps.models.ml.**OfflineModel**(**kwargs)

2.9.2 DataFrame Reference

class odps.df.core.**DataFrame**(*data*, ***kwargs*)
Main entrance of PyOdps DataFrame.

Users can initial a DataFrame by *odps.models.Table*.

Example

```
>>> df = DataFrame(o.get_table('my_example_table'))
>>> df.dtypes
odps.Schema {
  movie_id                              int64
  title                                 string
  release_date                          string
  video_release_date                    string
  imdb_url                              string
  user_id                               int64
  rating                                int64
  unix_timestamp                        int64
  age                                   int64
  sex                                   string
  occupation                            string
  zip_code                              string
}
>>> df.count()
100000
>>>
>>> # Do the `groupby`, aggregate the `movie_id` by count, then sort the count in a reversed orc
>>> # Finally we get the top 25 results
>>> df.groupby('title').agg(count=df.movie_id.count()).sort('count', ascending=False)[:25]
>>>
>>> # We can use the `value_counts` to reach the same goal
>>> df.movie_id.value_counts()[:25]
```

class odps.df.expr.expressions.**CollectionExpr**(**args*, ***kwargs*)

Collection represents for the two-dimensions data.

Example

```
>>> # projection
>>> df = DataFrame(o.get_table('my_table')) # DataFrame is actually a CollectionExpr
>>> df['name', 'id']  # projection some columns
>>> df[[df.name, df.id]]  # projection
>>> df[df]  # means nothing, but get all the columns
>>> df[df, df.name.lower().rename('name2')]  # projection a new columns `name2` besides all the
>>> df.select(df, name2=df.name.lower())  # projection by `select`
>>> df.exclude('name')  # projection all columns but `name`
>>> df[df.exclude('name'), df.name.lower()]  # `name` will not conflict any more
>>>
>>> # filter
>>> df[(df.id < 3) & (df.name != 'test')]
>>> df.filter(df.id < 3, df.name != 'test')
>>>
>>> # slice
>>> df[: 10]
>>> df.limit(10)
>>>
>>> # Sequence
>>> df.name # an instance of :class:`odps.df.expr.expressions.SequenceExpr`
>>>
>>> # schema or dtypes
>>> df.dtypes
odps.Schema {
  name    string
  id      int64
}
>>> df.schema
```

```
odps.Schema {
  name    string
  id      int64
}
```

columns

columns

list which each element is an instance of `odps.models.Column`

concat(*left*, *right*, *distinct=False*)

Union two collections.

- **left** – left collection
- **right** – right collection
- **distinct** –

collection

Example

```
>>> df['name', 'id'].union(df2['id', 'name'])
```

distinct(*expr*, *on=None*, **ons*)

Get collection with duplicate rows removed, optionally only considering certain columns

- **expr** – collection
- **on** – sequence or sequences

dinstinct collection

Example

```
>>> df.distinct(['name', 'id'])
>>> df['name', 'id'].distinct()
```

exclude(**fields*)

Projection columns which not included in the fields

fields – field names

new collection

`odps.df.expr.expression.CollectionExpr`

filter(**predicates*)

Filter the data by predicates

predicates – the conditions to filter

new collection

odps.df.expr.expressions.CollectionExpr

groupby(*expr*, *by*, **bys*)

Group collection by a series of sequences.

- **expr** – collection
- **by** – columns to group
- **bys** – columns to group

GroupBy instance

odps.df.expr.groupby.GroupBy

head(*n=None*)

Return the first n rows. Execute at once.

n –

result frame

odps.df.backends.frame.ResultFrame

inner_join(*left*, *right*, *on=None*, *suffix=('_x', '_y')*)

Inner join two collections.

If *on* is not specified, we will find the common fields of the left and right collection. *suffix* means that if column names conflict, the suffix will be added automatically. For example, both left and right has a field named *col*, there will be col_x, and col_y in the joined collection.

- **left** – left collection
- **right** – right collection
- **on** – fields to join on
- **suffix** – when name conflict, the suffix will be added to both columns.

collection

Example

```
>>> df.dtypes.names
['name', 'id']
>>> df2.dtypes.names
['name', 'id1']
>>> df.inner_join(df2)
>>> df.inner_join(df2, on='name')
>>> df.inner_join(df2, on=('id', 'id1'))
>>> df.inner_join(df2, on=['name', ('id', 'id1')])
>>> df.inner_join(df2, on=[df.name == df2.name, df.id == df2.id1])
```

join(*left*, *right*, *on=None*, *how='inner'*, *suffix=('_x', '_y')*)

Join two collections.

If *on* is not specified, we will find the common fields of the left and right collection. *suffix* means that if column names conflict, the suffix will be added automatically. For example, both left and right has a field named *col*, there will be col_x, and col_y in the joined collection.

- **left** – left collection
- **right** – right collection
- **on** – fields to join on
- **how** – 'inner', 'left', 'right', or 'outer'

- **suffix** – when name conflict, the suffix will be added to both columns.

collection

Example

```
>>> df.dtypes.names
['name', 'id']
>>> df2.dtypes.names
['name', 'id1']
>>> df.join(df2)
>>> df.join(df2, on='name')
>>> df.join(df2, on=('id', 'id1'))
>>> df.join(df2, on=['name', ('id', 'id1')])
>>> df.join(df2, on=[df.name == df2.name, df.id == df2.id1])
```

left_join(*left*, *right*, *on=None*, *suffix=('_x', '_y')*)

Left join two collections.

If *on* is not specified, we will find the common fields of the left and right collection. *suffix* means that if column names conflict, the suffix will be added automatically. For example, both left and right has a field named *col*, there will be col_x, and col_y in the joined collection.

- **left** – left collection
- **right** – right collection
- **on** – fields to join on
- **suffix** – when name conflict, the suffix will be added to both columns.

collection

Example

```
>>> df.dtypes.names
['name', 'id']
>>> df2.dtypes.names
['name', 'id1']
>>> df.left_join(df2)
>>> df.left_join(df2, on='name')
>>> df.left_join(df2, on=('id', 'id1'))
>>> df.left_join(df2, on=['name', ('id', 'id1')])
>>> df.left_join(df2, on=[df.name == df2.name, df.id == df2.id1])
```

outer_join(*left*, *right*, *on=None*, *suffix=('_x', '_y')*)

Outer join two collections.

If *on* is not specified, we will find the common fields of the left and right collection. *suffix* means that if column names conflict, the suffix will be added automatically. For example, both left and right has a field named *col*, there will be col_x, and col_y in the joined collection.

- **left** – left collection
- **right** – right collection
- **on** – fields to join on
- **suffix** – when name conflict, the suffix will be added to both columns.

collection

Example

```
>>> df.dtypes.names
['name', 'id']
>>> df2.dtypes.names
['name', 'id1']
>>> df.outer_join(df2)
>>> df.outer_join(df2, on='name')
>>> df.outer_join(df2, on=('id', 'id1'))
>>> df.outer_join(df2, on=['name', ('id', 'id1')])
>>> df.outer_join(df2, on=[df.name == df2.name, df.id == df2.id1])
```

right_join (*left*, *right*, *on=None*, *suffix=('_x', '_y')*)

Right join two collections.

If *on* is not specified, we will find the common fields of the left and right collection. *suffix* means that if column names conflict, the suffix will be added automatically. For example, both left and right has a field named *col*, there will be col_x, and col_y in the joined collection.

- **left** – left collection
- **right** – right collection
- **on** – fields to join on
- **suffix** – when name conflict, the suffix will be added to both columns.

collection

Example

```
>>> df.dtypes.names
['name', 'id']
>>> df2.dtypes.names
['name', 'id1']
>>> df.right_join(df2)
>>> df.right_join(df2, on='name')
>>> df.right_join(df2, on=('id', 'id1'))
>>> df.right_join(df2, on=['name', ('id', 'id1')])
>>> df.right_join(df2, on=[df.name == df2.name, df.id == df2.id1])
```

select (**fields*, ***kw*)

Projection columns. Remember to avoid column names' conflict.

- **fields** – columns to project
- **kw** – columns and their names to project

new collection

odps.df.expr.expression.CollectionExpr

sort (*expr*, *by*, *ascending=True*)

Sort the collection by values. *sort* is an alias name for *sort_values*

- **expr** – collection
- **by** – the sequence or sequences to sort

- **ascending** – Sort ascending vs. descending. Sepecify list for multiple sort orders. If this is a list of bools, must match the length of the by

Sorted collection

Example

```
>>> df.sort_values(['name', 'id'])  # 1
>>> df.sort(['name', 'id'], ascending=False)  # 2
>>> df.sort(['name', 'id'], ascending=[False, True])  # 3
>>> df.sort([-df.name, df.id])  # 4, equal to #3
```

sort_values(*expr*, *by*, *ascending=True*)

Sort the collection by values. *sort* is an alias name for *sort_values*

- **expr** – collection
- **by** – the sequence or sequences to sort
- **ascending** – Sort ascending vs. descending. Sepecify list for multiple sort orders. If this is a list of bools, must match the length of the by

Sorted collection

Example

```
>>> df.sort_values(['name', 'id'])  # 1
>>> df.sort(['name', 'id'], ascending=False)  # 2
>>> df.sort(['name', 'id'], ascending=[False, True])  # 3
>>> df.sort([-df.name, df.id])  # 4, equal to #3
```

switch(*expr*, **args*, ***kw*)

Similar to the case-when in SQL. Refer to the example below

- **expr** –
- **args** –
- **kw** –

sequence or scalar

Example

```
>>> # if df.id == 3 then df.name
>>> # elif df.id == df.fid.abs() then df.name + 'test'
>>> # default: 'test'
>>> df.id.switch(3, df.name, df.fid.abs(), df.name + 'test', default='test')
```

tail(*n=None*)

Return the last n rows. Execute at once.

n –

result frame

odps.df.backends.frame.ResultFrame

to_pandas(*use_cache=None*)

Convert to pandas DataFrame. Execute at once.

use_cache – return executed result if have been executed

pandas DataFrame

union(*left*, *right*, *distinct=False*)
Union two collections.

- **left** – left collection
- **right** – right collection
- **distinct** –

collection

Example

```
>>> df['name', 'id'].union(df2['id', 'name'])
```

view()
Clone a same collection. useful for self-join.

class odps.df.expr.expressions.**SequenceExpr**(**args*, ***kwargs*)
Sequence represents for 1-dimension data.

astype(*data_type*)
Cast to a new data type.

data_type – the new data type

casted sequence

Example

```
>>> df.id.astype('float')
```

between(*expr*, *left*, *right*, *inclusive=True*)
Return a boolean sequence or scalar show whether each element is between *left* and *right*.

- **expr** – sequence or scalar
- **left** – left value
- **right** – right value
- **inclusive** – if true, will be left <= expr <= right, else will be left < expr < right

boolean sequence or scalar

concat(*left*, *right*, *distinct=False*)
Union two collections.

- **left** – left collection
- **right** – right collection
- **distinct** –

collection

Example

```
        >>> df['name', 'id'].union(df2['id', 'name'])
```

cut(*expr*, *bins*, *right=True*, *labels=None*, *include_lowest=False*, *include_under=False*, *include_over=False*)
Return indices of half-open bins to which each value of *expr* belongs.

- **expr** – sequence or scalar
- **bins** – list of scalars
- **right** – indicates whether the bins include the rightmost edge or not. If right == True(the default), then the bins [1, 2, 3, 4] indicate (1, 2], (2, 3], (3, 4]
- **labels** – Usesd as labes for the resulting bins. Must be of the same length as the resulting bins.
- **include_lowest** – Whether the first interval should be left-inclusive or not.
- **include_under** – include the bin below the leftmost edge or not
- **include_over** – include the bin above the rightmost edge or not

sequence or scalar

dtype
Return the data type. Available types: int8, int16, int32, int64, float32, float64, boolean, string, decimal, datetime

the data type

fillna(*expr*, *value*)
Fill null with value.

- **expr** – sequence or scalar
- **value** – value to fill into

sequence or scalar

head(*n=None*)
Return first n rows. Execute at once.

n –

result frame

odps.df.expr.expressions.CollectionExpr

isin(*expr*, *values*)
Return a boolean sequence or scalar showing whether each element is exactly contained in the passed *values*.

- **expr** – sequence or scalar
- **values** – *list* object or sequence

boolean sequence or scalar

isnull(*expr*)
Return a sequence or scalar according to the input indicating if the values are null.

expr – sequence or scalar

sequence or scalar

map(*func*, *rtype=None*, *func_args=None*)
Call func on each element of this sequence.

- **func** – lambda, function, *odps.models.Function*, or str which is the name of odps.models.Funtion
- **rtype** – if not provided, will be the dtype of this sequence

a new sequence

Example

```
>>> df.id.map(lambda x: x + 1)
```

notin(*expr*, *values*)
Return a boolean sequence or scalar showing whether each element is not contained in the passed *values*.

- **expr** – sequence or scalar
- **values** – *list* object or sequence

boolean sequence or scalar

notnull(*expr*)
Return a sequence or scalar according to the input indicating if the values are not null.

expr – sequence or scalar

sequence or scalar

switch(*expr*, **args*, ***kw*)
Similar to the case-when in SQL. Refer to the example below

- **expr** –
- **args** –
- **kw** –

sequence or scalar

Example

```
>>> # if df.id == 3 then df.name
>>> # elif df.id == df.fid.abs() then df.name + 'test'
>>> # default: 'test'
>>> df.id.switch(3, df.name, df.fid.abs(), df.name + 'test', default='test')
```

tail(*n=None*)
Return the last n rows. Execute at once.

n –

to_pandas(*use_cache=None*)
Convert to pandas Series. Execute at once.

use_cache – return executed result if have been executed

pandas Series

union(*left*, *right*, *distinct=False*)
Union two collections.

- **left** – left collection
- **right** – right collection
- **distinct** –

collection

Example

```
>>> df['name', 'id'].union(df2['id', 'name'])
```

value_counts(*expr*, *sort=True*)
Return object containing counts of unique values.

The resulting object will be in descending order so that the first element is the most frequently-occuring element. Exclude NA values by default

expr – sequence

collection with two columns

odps.df.expr.expressions.CollectionExpr

class odps.df.expr.expressions.**Scalar**(**args*, ***kwargs*)
Represent for the scalar type.

between(*expr*, *left*, *right*, *inclusive=True*)
Return a boolean sequence or scalar show whether each element is between *left* and *right*.

- **expr** – sequence or scalar
- **left** – left value
- **right** – right value
- **inclusive** – if true, will be left <= expr <= right, else will be left < expr < right

boolean sequence or scalar

cut(*expr*, *bins*, *right=True*, *labels=None*, *include_lowest=False*, *include_under=False*, *include_over=False*)
Return indices of half-open bins to which each value of *expr* belongs.

- **expr** – sequence or scalar
- **bins** – list of scalars
- **right** – indicates whether the bins include the rightmost edge or not. If right == True(the default), then the bins [1, 2, 3, 4] indicate (1, 2], (2, 3], (3, 4]
- **labels** – Usesd as labes for the resulting bins. Must be of the same length as the resulting bins.
- **include_lowest** – Whether the first interval should be left-inclusive or not.

- **include_under** – include the bin below the leftmost edge or not
- **include_over** – include the bin above the rightmost edge or not

sequence or scalar

fillna (*expr*, *value*)
Fill null with value.

- **expr** – sequence or scalar
- **value** – value to fill into

sequence or scalar

isin (*expr*, *values*)
Return a boolean sequence or scalar showing whether each element is exactly contained in the passed *values*.

- **expr** – sequence or scalar
- **values** – *list* object or sequence

boolean sequence or scalar

isnull (*expr*)
Return a sequence or scalar according to the input indicating if the values are null.

expr – sequence or scalar

sequence or scalar

notin (*expr*, *values*)
Return a boolean sequence or scalar showing whether each element is not contained in the passed *values*.

- **expr** – sequence or scalar
- **values** – *list* object or sequence

boolean sequence or scalar

notnull (*expr*)
Return a sequence or scalar according to the input indicating if the values are not null.

expr – sequence or scalar

sequence or scalar

switch (*expr*, **args*, ***kw*)
Similar to the case-when in SQL. Refer to the example below

- **expr** –
- **args** –
- **kw** –

sequence or scalar

Example

```
        >>> # if df.id == 3 then df.name
        >>> # elif df.id == df.fid.abs() then df.name + 'test'
        >>> # default: 'test'
        >>> df.id.switch(3, df.name, df.fid.abs(), df.name + 'test', default='test')
```

A

astype() (odps.df.expr.expressions.SequenceExpr), 53

B

between() (odps.df.expr.expressions.Scalar), 56
between() (odps.df.expr.expressions.SequenceExpr), 53

C

close() (odps.models.FileResource), 44
columns (odps.df.expr.expressions.CollectionExpr), 48
concat() (odps.df.expr.expressions.CollectionExpr), 48
concat() (odps.df.expr.expressions.SequenceExpr), 53
create_function() (odps.ODPS), 28
create_resource() (odps.ODPS), 28
create_table() (odps.ODPS), 29
cut() (odps.df.expr.expressions.Scalar), 56
cut() (odps.df.expr.expressions.SequenceExpr), 54

D

delete_function() (odps.ODPS), 29
delete_offline_model() (odps.ODPS), 29
delete_resource() (odps.ODPS), 30
delete_table() (odps.ODPS), 30
delete_xflow() (odps.ODPS), 30
distinct() (odps.df.expr.expressions.CollectionExpr), 48
drop() (odps.models.Function), 46
drop() (odps.models.Table), 39
dtype (odps.df.expr.expressions.SequenceExpr), 54

E

exclude() (odps.df.expr.expressions.CollectionExpr), 48
execute_sql() (odps.ODPS), 30
execute_xflow() (odps.ODPS), 30
exist_function() (odps.ODPS), 31
exist_instance() (odps.ODPS), 31
exist_offline_model() (odps.ODPS), 31
exist_project() (odps.ODPS), 31
exist_resource() (odps.ODPS), 31
exist_table() (odps.ODPS), 32
exist_xflow() (odps.ODPS), 32

F

fillna() (odps.df.expr.expressions.Scalar), 57
fillna() (odps.df.expr.expressions.SequenceExpr), 54
filter() (odps.df.expr.expressions.CollectionExpr), 48
flush() (odps.models.FileResource), 44

G

get_function() (odps.ODPS), 32
get_instance() (odps.ODPS), 32
get_logview_address() (odps.ODPS), 33
get_offline_model() (odps.ODPS), 33
get_project() (odps.ODPS), 33
get_resource() (odps.ODPS), 33
get_source_table() (odps.models.TableResource), 46
get_source_table_partition()
(odps.models.TableResource), 46
get_table() (odps.ODPS), 33
get_task_names() (odps.models.Instance), 42
get_task_progress() (odps.models.Instance), 42
get_task_result() (odps.models.Instance), 42
get_task_results() (odps.models.Instance), 42
get_task_statuses() (odps.models.Instance), 43
get_task_summary() (odps.models.Instance), 43
get_xflow() (odps.ODPS), 34
get_xflow_results() (odps.ODPS), 34
groupby() (odps.df.expr.expressions.CollectionExpr), 48

H

head() (odps.df.expr.expressions.CollectionExpr), 49
head() (odps.df.expr.expressions.SequenceExpr), 54
head() (odps.models.Table), 39

I

inner_join() (odps.df.expr.expressions.CollectionExpr), 49
is_successful() (odps.models.Instance), 43
is_terminated() (odps.models.Instance), 43
isin() (odps.df.expr.expressions.Scalar), 57
isin() (odps.df.expr.expressions.SequenceExpr), 54
isnull() (odps.df.expr.expressions.Scalar), 57

isnull() (odps.df.expr.expressions.SequenceExpr), 54

J

join() (odps.df.expr.expressions.CollectionExpr), 49

L

left_join() (odps.df.expr.expressions.CollectionExpr), 50
list_functions() (odps.ODPS), 34
list_instances() (odps.ODPS), 34
list_offline_models() (odps.ODPS), 35
list_resources() (odps.ODPS), 35
list_tables() (odps.ODPS), 35
list_xflows() (odps.ODPS), 35

M

map() (odps.df.expr.expressions.SequenceExpr), 55

N

new_record() (odps.models.Table), 39
notin() (odps.df.expr.expressions.Scalar), 57
notin() (odps.df.expr.expressions.SequenceExpr), 55
notnull() (odps.df.expr.expressions.Scalar), 57
notnull() (odps.df.expr.expressions.SequenceExpr), 55

O

ODPS (odps), 27
open() (odps.models.FileResource), 44
open_reader() (odps.models.Table), 40
open_resource() (odps.ODPS), 35
open_writer() (odps.models.Table), 40
outer_join() (odps.df.expr.expressions.CollectionExpr), 50

R

read() (odps.models.FileResource), 44
read_table() (odps.ODPS), 36
readline() (odps.models.FileResource), 45
readlines() (odps.models.FileResource), 45
resources (odps.models.Function), 46
right_join() (odps.df.expr.expressions.CollectionExpr), 51
run_sql() (odps.ODPS), 37
run_xflow() (odps.ODPS), 37

S

seek() (odps.models.FileResource), 45
select() (odps.df.expr.expressions.CollectionExpr), 51
sort() (odps.df.expr.expressions.CollectionExpr), 51
sort_values() (odps.df.expr.expressions.CollectionExpr), 52
stop() (odps.models.Instance), 43
stop_instance() (odps.ODPS), 37
stop_job() (odps.ODPS), 38
switch() (odps.df.expr.expressions.CollectionExpr), 52
switch() (odps.df.expr.expressions.Scalar), 57
switch() (odps.df.expr.expressions.SequenceExpr), 55

T

Table (odps.models), 39
tail() (odps.df.expr.expressions.CollectionExpr), 52
tail() (odps.df.expr.expressions.SequenceExpr), 55
tell() (odps.models.FileResource), 45
to_pandas() (odps.df.expr.expressions.CollectionExpr), 52
to_pandas() (odps.df.expr.expressions.SequenceExpr), 55
truncate() (odps.models.FileResource), 45

U

union() (odps.df.expr.expressions.CollectionExpr), 53
union() (odps.df.expr.expressions.SequenceExpr), 56
update() (odps.models.TableResource), 46

V

value_counts() (odps.df.expr.expressions.SequenceExpr), 56
view() (odps.df.expr.expressions.CollectionExpr), 53

W

wait_for_completion() (odps.models.Instance), 43
wait_for_success() (odps.models.Instance), 43
write() (odps.models.FileResource), 45
write_table() (odps.ODPS), 38
writelines() (odps.models.FileResource), 45

X

XFlow (odps.models), 46