expandas Documentation

Release 0.1.1

sinhrks

March 13, 2015

Contents

Contents:

CHAPTER 1

Data Handling

1.1 Data Preparation

This section describes how to prepare basic data format named `ModelFrame`. `ModelFrame` defines a metadata to specify target (response variable) and data (explanatory variable / features). Using these metadata, `ModelFrame` can call other statistics/ML functions in more simple way.

You can create `ModelFrame` as the same manner as `pandas.DataFrame`. The below example shows how to create basic `ModelFrame`, which DOESN'T have target values.

```
>>> import expandas as expd

>>> df = expd.ModelFrame({'A': [1, 2, 3], 'B': [2, 3, 4],
...                       'C': [3, 4, 5]}, index=['a', 'b', 'c'])
>>> df
   A  B  C
a  1  2  3
b  2  3  4
c  3  4  5

>>> type(df)
<class 'expandas.core.frame.ModelFrame'>
```

You can check whether the created `ModelFrame` has target values using `ModelFrame.has_target()` function.

```
>>> df.has_target()
False
```

Target values can be specifyied via `target` keyword. You can simply pass a column name to be handled as target. Target column name can be confirmed via `target_name` property.

```
>>> df2 = expd.ModelFrame({'A': [1, 2, 3], 'B': [2, 3, 4],
...                        'C': [3, 4, 5]}, target='A')
>>> df2
   A  B  C
0  1  2  3
1  2  3  4
2  3  4  5

>>> df2.has_target()
True

>>> df2.target_name
'A'
```

Also, you can pass any list-likes to be handled as a target. In this case, target column will be named as ".target".

```
>>> df3 = expd.ModelFrame({'A': [1, 2, 3], 'B': [2, 3, 4],
...                        'C': [3, 4, 5]}, target=[4, 5, 6])
>>> df3
   .target  A  B  C
0        4  1  2  3
1        5  2  3  4
2        6  3  4  5

>>> df3.has_target()
True

>>> df3.target_name
'.target'
```

Also, you can pass `pandas.DataFrame` and `pandas.Series` as data and target.

```
>>> import pandas as pd
df4 = expd.ModelFrame({'A': [1, 2, 3], 'B': [2, 3, 4],
...                    'C': [3, 4, 5]}, target=pd.Series([4, 5, 6]))
>>> df4
   .target  A  B  C
0        4  1  2  3
1        5  2  3  4
2        6  3  4  5

>>> df4.has_target()
True

>>> df4.target_name
'.target'
```

Note: Target values are mandatory to perform operations which require response variable, such as regression and supervised learning.

1.2 Data Manipulation

You can access to each property as the same as `pandas.DataFrame`. Sliced results will be `ModelSeries` (simple wrapper for `pandas.Series` to support some data manipulation) or `ModelFrame`

```
>>> df
   A  B  C
a  1  2  3
b  2  3  4
c  3  4  5

>>> sliced = df['A']
>>> sliced
a    1
b    2
c    3
Name: A, dtype: int64

>>> type(sliced)
<class 'expandas.core.series.ModelSeries'>
```

```
>>> subset = df[['A', 'B']]
>>> subset
   A  B
a  1  2
b  2  3
c  3  4

>>> type(subset)
<class 'expandas.core.frame.ModelFrame'>
```

`ModelFrame` has a special properties `data` to access data (features) and `target` to access target.

```
>>> df2
   A  B  C
0  1  2  3
1  2  3  4
2  3  4  5

>>> df2.target_name
'A'

>>> df2.data
   B  C
0  2  3
1  3  4
2  4  5

>>> df2.target
0    1
1    2
2    3
Name: A, dtype: int64
```

You can update data and target via properties, in addition to standard `pandas.DataFrame` ways.

```
>>> df2.target = [9, 9, 9]
>>> df2
   A  B  C
0  9  2  3
1  9  3  4
2  9  4  5

>>> df2.data = pd.DataFrame({'X': [1, 2, 3], 'Y': [4, 5, 6]})
>>> df2
   A  X  Y
0  9  1  4
1  9  2  5
2  9  3  6

>>> df2['X'] = [0, 0, 0]
>>> df2
   A  X  Y
0  9  0  4
1  9  0  5
2  9  0  6
```

You can change target column specifying `target_name` property. Specifying a column which doesn't exist in `ModelFrame` results in target column to be data column.

```
>>> df2.target_name
'A'

>>> df2.target_name = 'X'
>>> df2.target_name
'X'

>>> df2.target_name = 'XXXX'
>>> df2.has_target()
False

>>> df2.data
   A  X  Y
0  9  0  4
1  9  0  5
2  9  0  6
```

CHAPTER 2

Use scikit-learn

This section describes how to use `scikit-learn` functionalities via `expandas`.

2.1 Basics

You can create `ModelFrame` instance from `scikit-learn` datasets directly.

```
>>> import expandas as expd
>>> import sklearn.datasets as datasets

>>> df = expd.ModelFrame(datasets.load_iris())
>>> df.head()
   .target  sepal length (cm)  sepal width (cm)  petal length (cm)  \
0        0                5.1               3.5                1.4
1        0                4.9               3.0                1.4
2        0                4.7               3.2                1.3
3        0                4.6               3.1                1.5
4        0                5.0               3.6                1.4

   petal width (cm)
0               0.2
1               0.2
2               0.2
3               0.2
4               0.2

# make columns be readable
>>> df.columns = ['.target', 'sepal length', 'sepal width', 'petal length', 'petal width']
```

`ModelFrame` has accessor methods which makes easier access to `scikit-learn` namespace.

```
>>> df.cluster.KMeans
<class 'sklearn.cluster.k_means_.KMeans'>
```

Following table shows `scikit-learn` module and corresponding `ModelFrame` module. Some accessors has its abbreviated versions.

Note: Currently, `ModelFrame` can handle target which consists from a single column. Modules which uses multiple target columns cannot be handled automatically, and marked with *(WIP)*.

---

scikit-learn	ModelFrame accessor
sklearn.cluster	ModelFrame.cluster
sklearn.covariance	ModelFrame.covariance
sklearn.cross_decomposition	ModelFrame.cross_decomposition (WIP)
sklearn.cross_validation	ModelFrame.cross_validation, crv
sklearn.datasets	(not accesible from accessor)
sklearn.decomposition	ModelFrame.decomposition
sklearn.dummy	ModelFrame.dummy
sklearn.ensemble	ModelFrame.ensemble
sklearn.feature_extraction	ModelFrame.feature_extraction
sklearn.feature_selection	ModelFrame.feature_selection
sklearn.gaussian_process	ModelFrame.gaussian_process (WIP)
sklearn.grid_search	ModelFrame.grid_search
sklearn.isotonic	ModelFrame.isotonic
sklearn.kernel_approximation	ModelFrame.kernel_approximation
sklearn.lda	ModelFrame.lda
sklearn.learning_curve	ModelFrame.learning_curve
sklearn.linear_model	ModelFrame.linear_model, lm
sklearn.manifold	ModelFrame.manifold
sklearn.metrics	ModelFrame.metrics
sklearn.mixture	ModelFrame.mixture
sklearn.multiclass	ModelFrame.multiclass
sklearn.naive_bayes	ModelFrame.naive_bayes
sklearn.neighbors	ModelFrame.neighbors
sklearn.neural_network	ModelFrame.neural_network
sklearn.pipeline	ModelFrame.pipeline
sklearn.preprocessing	ModelFrame.preprocessing, pp
sklearn.qda	ModelFrame.qda
sklearn.semi_supervised	ModelFrame.semi_supervised
sklearn.svm	ModelFrame.svm
sklearn.tree	ModelFrame.tree
sklearn.utils	(not accesible from accessor)

Thus, you can instanciate each estimator via ModelFrame accessors. Once create an estimator, you can pass it to ModelFrame.fit then predict. ModelFrame automatically uses its data and target properties for each operations.

```
>>> estimator = df.cluster.KMeans(n_clusters=3)
>>> df.fit(estimator)

>>> predicted = df.predict(estimator)
>>> predicted
0    1
1    1
2    1
...
147    2
148    2
149    0
Length: 150, dtype: int32
```

ModelFrame preserves the most recently used estimator in estimator atribute, and predicted results in predicted attibute.

```
>>> df.estimator
KMeans(copy_x=True, init='k-means++', max_iter=300, n_clusters=3, n_init=10,
    n_jobs=1, precompute_distances=True, random_state=None, tol=0.0001,
    verbose=0)

>>> df.predicted
0    1
1    1
2    1
...
147    2
148    2
149    0
Length: 150, dtype: int32
```

`ModelFrame` has following methods corresponding to various `scikit-learn` estimators. The last results are saved as corresponding `ModelFrame` properties.

`ModelFrame` method	`ModelFrame` property
`ModelFrame.fit`	(None)
`ModelFrame.transform`	(None)
`ModelFrame.fit_transform`	(None)
`ModelFrame.inverse_transform`	(None)
`ModelFrame.predict`	`ModelFrame.predicted`
`ModelFrame.fit_predict`	`ModelFrame.predicted`
`ModelFrame.score`	(None)
`ModelFrame.predict_proba`	`ModelFrame.proba`
`ModelFrame.predict_log_proba`	`ModelFrame.log_proba`
`ModelFrame.decision_function`	`ModelFrame.decision`

Note: If you access to a property before calling `ModelFrame` methods, `ModelFrame` automatically calls corresponding method of the latest estimator and return the result.

Following example shows to perform PCA, then revert principal components back to original space.

```
>>> estimator = df.decomposition.PCA()
>>> df.fit(estimator)

>>> transformed = df.transform(estimator)
>>> transformed.head()
   .target         0         1         2         3
0        0 -2.684207 -0.326607  0.021512  0.001006
1        0 -2.715391  0.169557  0.203521  0.099602
2        0 -2.889820  0.137346 -0.024709  0.019305
3        0 -2.746437  0.311124 -0.037672 -0.075955
4        0 -2.728593 -0.333925 -0.096230 -0.063129

>>> type(transformed)
<class 'expandas.core.frame.ModelFrame'>

>>> transformed.inverse_transform(estimator)
     .target    0    1    2    3
0          0  5.1  3.5  1.4  0.2
1          0  4.9  3.0  1.4  0.2
2          0  4.7  3.2  1.3  0.2
3          0  4.6  3.1  1.5  0.2
4          0  5.0  3.6  1.4  0.2
..       ...  ...  ...  ...  ...
```

```
145         2  6.7  3.0  5.2  2.3
146         2  6.3  2.5  5.0  1.9
147         2  6.5  3.0  5.2  2.0
148         2  6.2  3.4  5.4  2.3
149         2  5.9  3.0  5.1  1.8

[150 rows x 5 columns]
```

Note: `columns` information will be lost once transformed to principal components.

If `ModelFrame` both has `target` and `predicted` values, the model evaluation can be performed using functions available in `ModelFrame.metrics`.

```
>>> estimator = df.svm.SVC()
>>> df.fit(estimator)

>>> df.predict(estimator)
0    0
1    0
2    0
...
147    2
148    2
149    2
Length: 150, dtype: int64

>>> df.predicted
0    0
1    0
2    0
...
147    2
148    2
149    2
Length: 150, dtype: int64

>>> df.metrics.confusion_matrix()
Predicted   0   1   2
Target
0          50   0   0
1           0  48   2
2           0   0  50
```

2.2 Use Module Level Functions

Some `scikit-learn` modules define functions which handle data without instanciating estimators. You can call these functions from accessor methods directly, and `ModelFrame` will pass corresponding data on background. Following example shows to use `sklearn.cluster.k_means` function to perform K-means.

Important: When you use module level function, `ModelFrame.predicted` WILL NOT be updated. Thus, using estimator is recommended.

```
# no need to pass data explicitly
# sklearn.cluster.kmeans returns centroids, cluster labels and inertia
```

```
>>> c, l, i = df.cluster.k_means(n_clusters=3)
>>> l
0     1
1     1
2     1
...
147    2
148    2
149    0
Length: 150, dtype: int32
```

2.3 Pipeline

`ModelFrame` can handle pipeline as the same as normal estimators.

```
>>> estimators = [('reduce_dim', df.decomposition.PCA()),
...               ('svm', df.svm.SVC())]
>>> pipe = df.pipeline.Pipeline(estimators)
>>> df.fit(pipe)

>>> df.predict(pipe)
0    0
1    0
2    0
...
147    2
148    2
149    2
Length: 150, dtype: int64
```

Above expression is the same as below:

```
>>> df2 = df.copy()
>>> df2 = df2.fit_transform(df2.decomposition.PCA())
>>> svm = df2.svm.SVC()
>>> df2.fit(svm)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
>>> df2.predict(svm)
0     0
1     0
2     0
...
147    2
148    2
149    2
Length: 150, dtype: int64
```

2.4 Cross Validation

`scikit-learn` has some classes for cross validation. `cross_validation.train_test_split` splits data to training and test set. You can access to the function via `cross_validation` accessor.

```
>>> train_df, test_df = df.cross_validation.train_test_split()
>>> train_df
     .target  sepal length  sepal width  petal length  petal width
0          0           4.8          3.4           1.9          0.2
1          1           6.3          3.3           4.7          1.6
2          0           4.8          3.4           1.6          0.2
3          2           7.7          2.6           6.9          2.3
4          0           5.4          3.4           1.7          0.2
..       ...           ...          ...           ...          ...
107        0           5.1          3.7           1.5          0.4
108        1           6.7          3.1           4.7          1.5
109        0           4.7          3.2           1.3          0.2
110        0           5.8          4.0           1.2          0.2
111        0           5.1          3.5           1.4          0.2

[112 rows x 5 columns]

>>> test_df
    .target  sepal length  sepal width  petal length  petal width
0         2           6.3          2.7           4.9          1.8
1         0           4.5          2.3           1.3          0.3
2         2           5.8          2.8           5.1          2.4
3         0           4.3          3.0           1.1          0.1
4         0           5.0          3.0           1.6          0.2
..      ...           ...          ...           ...          ...
33        1           6.7          3.1           4.4          1.4
34        0           4.6          3.6           1.0          0.2
35        1           5.7          3.0           4.2          1.2
36        1           5.9          3.0           4.2          1.5
37        2           6.4          2.8           5.6          2.1

[38 rows x 5 columns]
```

Also, there are some iterative classes which returns indexes for training sets and test sets. You can slice `ModelFrame` using these indexes.

```
>>> kf = df.cross_validation.KFold(n=150, n_folds=3)
>>> for train_index, test_index in kf:
...     print('training set shape: ', df.iloc[train_index, :].shape,
...           'test set shape: ', df.iloc[test_index, :].shape)
('training set shape: ', (100, 5), 'test set shape: ', (50, 5))
('training set shape: ', (100, 5), 'test set shape: ', (50, 5))
('training set shape: ', (100, 5), 'test set shape: ', (50, 5))
```

For further simplification, `ModelFrame.cross_validation.iterate` can accept such iterators and returns `ModelFrame` corresponding to training and test data.

```
>>> kf = df.cross_validation.KFold(n=150, n_folds=3)
>>> for train_df, test_df in df.cross_validation.iterate(kf):
...     print('training set shape: ', train_df.shape,
...           'test set shape: ', test_df.shape)
('training set shape: ', (100, 5), 'test set shape: ', (50, 5))
('training set shape: ', (100, 5), 'test set shape: ', (50, 5))
('training set shape: ', (100, 5), 'test set shape: ', (50, 5))
```

2.5 Grid Search

You can perform grid search using `ModelFrame.fit`.

```
>>> tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
...                     'C': [1, 10, 100]},
...                    {'kernel': ['linear'], 'C': [1, 10, 100]}]

>>> df = expd.ModelFrame(datasets.load_digits())
>>> cv = df.grid_search.GridSearchCV(df.svm.SVC(C=1), tuned_parameters,
...                                  cv=5, scoring='precision')

>>> df.fit(cv)

>>> cv.best_estimator_
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.001,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In addition, `ModelFrame.grid_search` has a `describe` function to organize each grid search result as `ModelFrame` accepting estimator.

```
>>> df.grid_search.describe(cv)
       mean       std    C   gamma  kernel
0  0.974108  0.013139    1  0.0010     rbf
1  0.951416  0.020010    1  0.0001     rbf
2  0.975372  0.011280   10  0.0010     rbf
3  0.962534  0.020218   10  0.0001     rbf
4  0.975372  0.011280  100  0.0010     rbf
5  0.964695  0.016686  100  0.0001     rbf
6  0.951811  0.018410    1     NaN  linear
7  0.951811  0.018410   10     NaN  linear
8  0.951811  0.018410  100     NaN  linear
```

CHAPTER 3

Use patsy

This section describes data transformation using `patsy`. `ModelFrame.transform` can accept `patsy` style formula.

```
>>> import expandas as expd

# create modelframe which doesn't have target
>>> df = expd.ModelFrame({'X': [1, 2, 3], 'Y': [2, 3, 4],
...                       'Z': [3, 4, 5]}, index=['a', 'b', 'c'])

>>> df
   X  Y  Z
a  1  2  3
b  2  3  4
c  3  4  5

# transform with patsy formula
>>> transformed = df.transform('Z ~ Y + X')
>>> transformed
   Z  Intercept  Y  X
a  3          1  2  1
b  4          1  3  2
c  5          1  4  3

# transformed data should have target specified by formula
>>> transformed.target
a    3
b    4
c    5
Name: Z, dtype: float64

>>> transformed.data
   Intercept  Y  X
a          1  2  1
b          1  3  2
c          1  4  3
```

If you do not want intercept, specify with `0`.

```
>>> df.transform('Z ~ Y + 0')
   Z  Y
a  3  2
b  4  3
c  5  4
```

Also, you can use formula which doesn't have left side.

```
# create modelframe which has target
>>> df2 = expd.ModelFrame({'X': [1, 2, 3], 'Y': [2, 3, 4],'Z': [3, 4, 5]},
...                       target =[7, 8, 9], index=['a', 'b', 'c'])

>>> df2
   .target  X  Y  Z
a        7  1  2  3
b        8  2  3  4
c        9  3  4  5

# overwrite data with transformed data
>>> df2.data = df2.transform('Y + Z')
>>> df2
   .target  Intercept  Y  Z
a        7          1  2  3
b        8          1  3  4
c        9          1  4  5

# data has been updated based on formula
>>> df2.data
   Intercept  Y  Z
a          1  2  3
b          1  3  4
c          1  4  5

# target is not changed
>>> df2.target
a    7
b    8
c    9
Name: .target, dtype: int64
```

Below example is performing deviation coding via patsy formula.

```
>>> df3 = expd.ModelFrame({'X': [1, 2, 3, 4, 5], 'Y': [1, 3, 2, 2, 1],
...                        'Z': [1, 1, 1, 2, 2]}, target='Z',
...                        index=['a', 'b', 'c', 'd', 'e'])

>>> df3
   X  Y  Z
a  1  1  1
b  2  3  1
c  3  2  1
d  4  2  2
e  5  1  2

>>> df3.transform('C(X, Sum)')
   Intercept  C(X, Sum)[S.1]  C(X, Sum)[S.2]  C(X, Sum)[S.3]  C(X, Sum)[S.4]
a          1               1               0               0               0
b          1               0               1               0               0
c          1               0               0               1               0
d          1               0               0               0               1
e          1              -1              -1              -1              -1

>>> df3.transform('C(Y, Sum)')
   Intercept  C(Y, Sum)[S.1]  C(Y, Sum)[S.2]
a          1               1               0
```

```
b          1              -1              -1
c          1               0               1
d          1               0               1
e          1               1               0
```

API:

CHAPTER 4

expandas.core package

4.1 Submodules

class expandas.core.accessor.**AccessorMethods**(*df*, *module_name=None*, *attrs=None*)
Bases: object

Accessor to related functionalities.

class expandas.core.frame.**ModelFrame**(*data*, *target=None*, **args*, ***kwargs*)
Bases: pandas.core.frame.DataFrame

Data structure subclassing pandas.DataFrame to define a metadata to specify target (response variable) and data (explanatory variable / features).

Parameters **data** : same as pandas.DataFrame

target : str or array-like

Column name or values to be used as target

args : arguments passed to pandas.DataFrame

kwargs : keyword arguments passed to pandas.DataFrame

Attributes

T	Transpose index and columns
at	
axes	
blocks	Internal property, property synonym for as_blocks()
cluster	Property to access sklearn.cluster.
covariance	Property to access sklearn.covariance.
cross_decomposition	Property to access sklearn.cross_decomposition
cross_validation	Property to access sklearn.cross_validation.
crv	Property to access sklearn.cross_validation.
data	Return data (explanatory variable / features)
decision	Return current estimator's decision function
decomposition	Property to access sklearn.decomposition
dtypes	Return the dtypes in this object
dummy	Property to access sklearn.dummy
empty	True if NDFrame is entirely empty [no items]

Continued on next page

Table 4.1 – continued from previous page

ensemble	Property to access sklearn.ensemble.
estimator	Return most recently used estimator
feature_extraction	Property to access sklearn.feature_extraction.
feature_selection	Property to access sklearn.feature_selection.
ftypes	Return the ftypes (indication of sparse/dense and dtype) in this object.
gaussian_process	Property to access sklearn.gaussian_process.
grid_search	Property to access sklearn.grid_search.
iat	
iloc	
isotonic	Property to access sklearn.isotonic.
ix	
kernel_approximation	Property to access sklearn.kernel_approximation
lda	Property to access sklearn.lda
learning_curve	Property to access sklearn.learning_curve.
linear_model	Property to access sklearn.linear_model.
lm	Property to access sklearn.linear_model.
loc	
log_proba	Return current estimator's log probabilities
manifold	Property to access sklearn.manifold.
metrics	Property to access sklearn.metrics.
mixture	Property to access sklearn.mixture
multiclass	Property to access sklearn.multiclass.
naive_bayes	Property to access sklearn.naive_bayes
ndim	Number of axes / array dimensions
neighbors	Property to access sklearn.neighbors.
neural_network	Property to access sklearn.neural_network
pipeline	Property to access sklearn.pipeline.
pp	Property to access sklearn.preprocessing.
predicted	Return current estimator's predicted results
preprocessing	Property to access sklearn.preprocessing.
proba	Return current estimator's probabilities
qda	Property to access sklearn.qda
random_projection	Property to access sklearn.random_projection.
semi_supervised	Property to access sklearn.semi_supervised.
shape	
size	number of elements in the NDFrame
svm	Property to access sklearn.svm.
target	Return target (response variable)
target_name	Return target column name
tree	Property to access sklearn.tree
values	Numpy representation of NDFrame

is_copy	

Methods

abs()	Return an object with absolute value taken.
add(other[, axis, level, fill_value])	Binary operator add with support to substitute a fill_value for missing data in
add_prefix(prefix)	Concatenate prefix string with panel items names.
add_suffix(suffix)	Concatenate suffix string with panel items names

Table 4.2 – conti

align(other[, join, axis, level, copy, ...])	Align two object on their axes with the
all([axis, bool_only, skipna, level])	Return whether all elements are True over requested axis
any([axis, bool_only, skipna, level])	Return whether any element is True over requested axis
append(other[, ignore_index, verify_integrity])	Append columns of other to end of this frame's columns and index, returning
apply(func[, axis, broadcast, raw, reduce, args])	Applies function along input axis of DataFrame.
applymap(func)	Apply a function to a DataFrame that is intended to operate elementwise, i.e.
as_blocks()	Convert the frame to a dict of dtype -> Constructor Types that each has a hom
as_matrix([columns])	Convert the frame to its Numpy-array representation.
asfreq(freq[, method, how, normalize])	Convert all TimeSeries inside to specified frequency using DateOffset objects
astype(dtype[, copy, raise_on_error])	Cast object to input numpy.dtype
at_time(time[, asof])	Select values at particular time of day (e.g.
between_time(start_time, end_time[, ...])	Select values between particular times of the day (e.g., 9:00-9:30 AM)
bfill([axis, inplace, limit, downcast])	Synonym for NDFrame.fillna(method='bfill')
bool()	Return the bool of a single element PandasObject
boxplot([column, by, ax, fontsize, rot, ...])	Make a box plot from DataFrame column optionally grouped by some colum
clip([lower, upper, out])	Trim values at input threshold(s)
clip_lower(threshold)	Return copy of the input with values below given value truncated
clip_upper(threshold)	Return copy of input with values above given value truncated
combine(other, func[, fill_value, overwrite])	Add two DataFrame objects and do not propagate NaN values, so if for a
combineAdd(other)	Add two DataFrame objects and do not propagate
combineMult(other)	Multiply two DataFrame objects and do not propagate NaN values, so if
combine_first(other)	Combine two DataFrame objects and default to non-null values in frame calli
compound([axis, skipna, level])	Return the compound percentage of the values for the requested axis
consolidate([inplace])	Compute NDFrame with "consolidated" internals (data of each dtype groupe
convert_objects([convert_dates, ...])	Attempt to infer better dtype for object columns
copy([deep])	Make a copy of this object
corr([method, min_periods])	Compute pairwise correlation of columns, excluding NA/null values
corrwith(other[, axis, drop])	Compute pairwise correlation between rows or columns of two DataFrame ob
count([axis, level, numeric_only])	Return Series with number of non-NA/null observations over requested axis.
cov([min_periods])	Compute pairwise covariance of columns, excluding NA/null values
cummax([axis, dtype, out, skipna])	Return cumulative max over requested axis.
cummin([axis, dtype, out, skipna])	Return cumulative min over requested axis.
cumprod([axis, dtype, out, skipna])	Return cumulative prod over requested axis.
cumsum([axis, dtype, out, skipna])	Return cumulative sum over requested axis.
decision_function(estimator, *args, **kwargs)	Call estimator's decision_function method.
describe([percentile_width, percentiles, ...])	Generate various summary statistics, excluding NaN values.
diff([periods])	1st discrete difference of object
div(other[, axis, level, fill_value])	Binary operator truediv with support to substitute a fill_value for missing data
divide(other[, axis, level, fill_value])	Binary operator truediv with support to substitute a fill_value for missing data
dot(other)	Matrix multiplication with DataFrame or Series objects
drop(labels[, axis, level, inplace])	Return new object with labels in requested axis removed
drop_duplicates(*args, **kwargs)	Return DataFrame with duplicate rows removed, optionally only
dropna([axis, how, thresh, subset, inplace])	Return object with labels on given axis omitted where alternately any
duplicated(*args, **kwargs)	Return boolean Series denoting duplicate rows, optionally only
eq(other[, axis, level])	Wrapper for flexible comparison methods eq
equals(other)	Determines if two NDFrame objects contain the same elements.
eval(expr, **kwargs)	Evaluate an expression in the context of the calling DataFrame instance.
ffill([axis, inplace, limit, downcast])	Synonym for NDFrame.fillna(method='ffill')
fillna([value, method, axis, inplace, ...])	Fill NA/NaN values using the specified method
filter([items, like, regex, axis])	Restrict the info axis to set of items or wildcard
first(offset)	Convenience method for subsetting initial periods of time series data
first_valid_index()	Return label for first non-NA/null value

Table 4.2 – conti

fit(estimator, *args, **kwargs)	Call estimator's fit method.
fit_predict(estimator, *args, **kwargs)	Call estimator's fit_predict method.
fit_transform(estimator, *args, **kwargs)	Call estimator's fit_transform method.
floordiv(other[, axis, level, fill_value])	Binary operator floordiv with support to substitute a fill_value for missing da
from_csv(path[, header, sep, index_col, ...])	Read delimited file into DataFrame
from_dict(data[, orient, dtype])	Construct DataFrame from dict of array-like or dicts
from_items(items[, columns, orient])	Convert (key, value) pairs to DataFrame.
from_records(data[, index, exclude, ...])	Convert structured or record ndarray to DataFrame
ge(other[, axis, level])	Wrapper for flexible comparison methods ge
get(key[, default])	Get item from object for given key (DataFrame column, Panel slice, etc.).
get_dtype_counts()	Return the counts of dtypes in this object
get_ftype_counts()	Return the counts of ftypes in this object
get_value(index, col[, takeable])	Quickly retrieve single value at passed column and index
get_values()	same as values (but handles sparseness conversions)
groupby([by, axis, level, as_index, sort, ...])	Group series using mapper (dict or key function, apply given function
gt(other[, axis, level])	Wrapper for flexible comparison methods gt
has_data()	Return whether ModelFrame has data
has_target()	Return whether ModelFrame has target
head([n])	Returns first n rows
hist(data[, column, by, grid, xlabelsize, ...])	Draw histogram of the DataFrame's series using matplotlib / pylab.
icol(i)	
idxmax([axis, skipna])	Return index of first occurrence of maximum over requested axis.
idxmin([axis, skipna])	Return index of first occurrence of minimum over requested axis.
iget_value(i, j)	
info([verbose, buf, max_cols, memory_usage, ...])	Concise summary of a DataFrame.
insert(loc, column, value[, allow_duplicates])	Insert column into DataFrame at specified location.
interpolate([method, axis, limit, inplace, ...])	Interpolate values according to different methods.
inverse_transform(estimator, *args, **kwargs)	Call estimator's inverse_transform method.
irow(i[, copy])	
isin(values)	Return boolean DataFrame showing whether each element in the DataFrame
isnull()	Return a boolean same-sized object indicating if the values are null
iteritems()	Iterator over (column, series) pairs
iterkv(*args, **kwargs)	iteritems alias used to get around 2to3. Deprecated
iterrows()	Iterate over rows of DataFrame as (index, Series) pairs.
itertuples([index])	Iterate over rows of DataFrame as tuples, with index value
join(other[, on, how, lsuffix, rsuffix, sort])	Join columns with other DataFrame either on index or on a key column.
keys()	Get the 'info axis' (see Indexing for more)
kurt([axis, skipna, level, numeric_only])	Return unbiased kurtosis over requested axis
kurtosis([axis, skipna, level, numeric_only])	Return unbiased kurtosis over requested axis
last(offset)	Convenience method for subsetting final periods of time series data
last_valid_index()	Return label for last non-NA/null value
le(other[, axis, level])	Wrapper for flexible comparison methods le
load(path)	Deprecated.
lookup(row_labels, col_labels)	Label-based "fancy indexing" function for DataFrame.
lt(other[, axis, level])	Wrapper for flexible comparison methods lt
mad([axis, skipna, level])	Return the mean absolute deviation of the values for the requested axis
mask(cond)	Returns copy whose values are replaced with nan if the
max([axis, skipna, level, numeric_only])	This method returns the maximum of the values in the object.
mean([axis, skipna, level, numeric_only])	Return the mean of the values for the requested axis
median([axis, skipna, level, numeric_only])	Return the median of the values for the requested axis
memory_usage([index])	Memory usage of DataFrame columns.
merge(right[, how, on, left_on, right_on, ...])	Merge DataFrame objects by performing a database-style join operation by c

Table 4.2 – conti

min([axis, skipna, level, numeric_only])	This method returns the minimum of the values in the object.
mod(other[, axis, level, fill_value])	Binary operator mod with support to substitute a fill_value for missing data in
mode([axis, numeric_only])	Gets the mode of each element along the axis selected.
mul(other[, axis, level, fill_value])	Binary operator mul with support to substitute a fill_value for missing data in
multiply(other[, axis, level, fill_value])	Binary operator mul with support to substitute a fill_value for missing data in
ne(other[, axis, level])	Wrapper for flexible comparison methods ne
notnull()	Return a boolean same-sized object indicating if the values are
pct_change([periods, fill_method, limit, freq])	Percent change over given number of periods.
pivot([index, columns, values])	Reshape data (produce a "pivot" table) based on column values.
pivot_table(*args, **kwargs)	Create a spreadsheet-style pivot table as a DataFrame.
plot(data[, x, y, kind, ax, subplots, ...])	Make plots of DataFrame using matplotlib / pylab.
pop(item)	Return item and drop from frame.
pow(other[, axis, level, fill_value])	Binary operator pow with support to substitute a fill_value for missing data in
predict(estimator, *args, **kwargs)	Call estimator's predict method.
predict_log_proba(estimator, *args, **kwargs)	Call estimator's predict_log_proba method.
predict_proba(estimator, *args, **kwargs)	Call estimator's predict_proba method.
prod([axis, skipna, level, numeric_only])	Return the product of the values for the requested axis
product([axis, skipna, level, numeric_only])	Return the product of the values for the requested axis
quantile([q, axis, numeric_only])	Return values at the given quantile over requested axis, a la numpy.percentile
query(expr, **kwargs)	Query the columns of a frame with a boolean expression.
radd(other[, axis, level, fill_value])	Binary operator radd with support to substitute a fill_value for missing data in
rank([axis, numeric_only, method, ...])	Compute numerical data ranks (1 through n) along axis.
rdiv(other[, axis, level, fill_value])	Binary operator rtruediv with support to substitute a fill_value for missing dat
reindex([index, columns])	Conform DataFrame to new index with optional filling logic, placing NA/Nal
reindex_axis(labels[, axis, method, level, ...])	Conform input object to new index with optional filling logic, placing NA/Na
reindex_like(other[, method, copy, limit])	return an object with matching indicies to myself
rename([index, columns])	Alter axes input function or functions.
rename_axis(mapper[, axis, copy, inplace])	Alter index and / or columns using input function or functions.
reorder_levels(order[, axis])	Rearrange index levels using input order.
replace([to_replace, value, inplace, limit, ...])	Replace values given in 'to_replace' with 'value'.
resample(rule[, how, axis, fill_method, ...])	Convenience method for frequency conversion and resampling of regular time
reset_index([level, drop, inplace, ...])	For DataFrame with multi-level index, return new DataFrame with labeling in
rfloordiv(other[, axis, level, fill_value])	Binary operator rfloordiv with support to substitute a fill_value for missing da
rmod(other[, axis, level, fill_value])	Binary operator rmod with support to substitute a fill_value for missing data i
rmul(other[, axis, level, fill_value])	Binary operator rmul with support to substitute a fill_value for missing data in
rpow(other[, axis, level, fill_value])	Binary operator rpow with support to substitute a fill_value for missing data i
rsub(other[, axis, level, fill_value])	Binary operator rsub with support to substitute a fill_value for missing data in
rtruediv(other[, axis, level, fill_value])	Binary operator rtruediv with support to substitute a fill_value for missing dat
save(path)	Deprecated.
score(estimator, *args, **kwargs)	Call estimator's score method.
select(crit[, axis])	Return data corresponding to axis labels matching criteria
select_dtypes([include, exclude])	Return a subset of a DataFrame including/excluding columns based on their d
sem([axis, skipna, level, ddof])	Return unbiased standard error of the mean over requested axis.
set_axis(axis, labels)	public verson of axis assignment
set_index(keys[, drop, append, inplace, ...])	Set the DataFrame index (row labels) using one or more existing columns.
set_value(index, col, value[, takeable])	Put single value at passed column and index
shift([periods, freq, axis])	Shift index by desired number of periods with an optional time freq
skew([axis, skipna, level, numeric_only])	Return unbiased skew over requested axis
slice_shift([periods, axis])	Equivalent to *shift* without copying data.
sort([columns, axis, ascending, inplace, ...])	Sort DataFrame either by labels (along either axis) or by the values in
sort_index([axis, by, ascending, inplace, ...])	Sort DataFrame either by labels (along either axis) or by the values in
sortlevel([level, axis, ascending, inplace, ...])	Sort multilevel index by chosen axis and primary level.

Table 4.2 – conti

squeeze()	squeeze length 1 dimensions
stack([level, dropna])	Pivot a level of the (possibly hierarchical) column labels, returning a DataFra
std([axis, skipna, level, ddof])	Return unbiased standard deviation over requested axis.
sub(other[, axis, level, fill_value])	Binary operator sub with support to substitute a fill_value for missing data in
subtract(other[, axis, level, fill_value])	Binary operator sub with support to substitute a fill_value for missing data in
sum([axis, skipna, level, numeric_only])	Return the sum of the values for the requested axis
swapaxes(axis1, axis2[, copy])	Interchange axes and swap values axes appropriately
swaplevel(i, j[, axis])	Swap levels i and j in a MultiIndex on a particular axis
tail([n])	Returns last n rows
take(indices[, axis, convert, is_copy])	Analogous to ndarray.take
to_clipboard([excel, sep])	Attempt to write text representation of object to the system clipboard This ca
to_csv(*args, **kwargs)	Write DataFrame to a comma-separated values (csv) file
to_dense()	Return dense representation of NDFrame (as opposed to sparse)
to_dict(*args, **kwargs)	Convert DataFrame to dictionary.
to_excel(*args, **kwargs)	Write DataFrame to a excel sheet
to_gbq(destination_table[, project_id, ...])	Write a DataFrame to a Google BigQuery table.
to_hdf(path_or_buf, key, **kwargs)	activate the HDFStore
to_html([buf, columns, col_space, colSpace, ...])	Render a DataFrame as an HTML table.
to_json([path_or_buf, orient, date_format, ...])	Convert the object to a JSON string.
to_latex([buf, columns, col_space, ...])	Render a DataFrame to a tabular environment table.
to_msgpack([path_or_buf])	msgpack (serialize) object to input file path
to_panel()	Transform long (stacked) format (DataFrame) into wide (3D, Panel) format.
to_period([freq, axis, copy])	Convert DataFrame from DatetimeIndex to PeriodIndex with desired
to_pickle(path)	Pickle (serialize) object to input file path
to_records([index, convert_datetime64])	Convert DataFrame to record array.
to_sparse([fill_value, kind])	Convert to SparseDataFrame
to_sql(name, con[, flavor, schema, ...])	Write records stored in a DataFrame to a SQL database.
to_stata(fname[, convert_dates, ...])	A class for writing Stata binary dta files from array-like objects
to_string([buf, columns, col_space, ...])	Render a DataFrame to a console-friendly tabular output.
to_timestamp([freq, how, axis, copy])	Cast to DatetimeIndex of timestamps, at *beginning* of period
to_wide(*args, **kwargs)	
transform(estimator, *args, **kwargs)	Call estimator's transform method.
transpose()	Transpose index and columns
truediv(other[, axis, level, fill_value])	Binary operator truediv with support to substitute a fill_value for missing data
truncate([before, after, axis, copy])	Truncates a sorted NDFrame before and/or after some particular dates.
tshift([periods, freq, axis])	Shift the time index, using the index's frequency if available
tz_convert(tz[, axis, level, copy])	Convert the axis to target time zone.
tz_localize(*args, **kwargs)	Localize tz-naive TimeSeries to target time zone
unstack([level])	Pivot a level of the (necessarily hierarchical) index labels, returning a DataFra
update(other[, join, overwrite, ...])	Modify DataFrame in place using non-NA values from passed DataFrame.
var([axis, skipna, level, ddof])	Return unbiased variance over requested axis.
where(cond[, other, inplace, axis, level, ...])	Return an object of same shape as self and whose corresponding entries are fr
xs(key[, axis, level, copy, drop_level])	Returns a cross-section (row(s) or column(s)) from the Series/DataFrame.

cluster
Property to access `sklearn.cluster`. See `expandas.skaccessors.cluster`

covariance
Property to access `sklearn.covariance`. See `expandas.skaccessors.covariance`

cross_decomposition
Property to access `sklearn.cross_decomposition`

cross_validation

Property to access `sklearn.cross_validation`. See `expandas.skaccessors.cross_validation`

crv
Property to access `sklearn.cross_validation`. See `expandas.skaccessors.cross_validation`

data
Return data (explanatory variable / features)

Returns **data** : `ModelFrame`

decision
Return current estimator's decision function

Returns **decisions** : `ModelFrame`

decision_function(*estimator*, **args*, ***kwargs*)
Call estimator's decision_function method.

Parameters **args** : arguments passed to decision_function method

kwargs : keyword arguments passed to decision_function method

Returns **returned** : decisions

decomposition
Property to access `sklearn.decomposition`

dummy
Property to access `sklearn.dummy`

ensemble
Property to access `sklearn.ensemble`. See `expandas.skaccessors.ensemble`

estimator
Return most recently used estimator

Returns **estimator** : estimator

feature_extraction
Property to access `sklearn.feature_extraction`. See `expandas.skaccessors.feature_extraction`

feature_selection
Property to access `sklearn.feature_selection`. See `expandas.skaccessors.feature_selection`

fit(*estimator*, **args*, ***kwargs*)
Call estimator's fit method.

Parameters **args** : arguments passed to fit method

kwargs : keyword arguments passed to fit method

Returns **returned** : None or fitted estimator

fit_predict(*estimator*, **args*, ***kwargs*)
Call estimator's fit_predict method.

Parameters **args** : arguments passed to fit_predict method

kwargs : keyword arguments passed to fit_predict method

Returns **returned** : predicted result

fit_transform(*estimator*, **args*, ***kwargs*)
Call estimator's fit_transform method.

Parameters **args** : arguments passed to fit_transform method

kwargs : keyword arguments passed to fit_transform method

Returns **returned** : transformed result

gaussian_process
Property to access `sklearn.gaussian_process`. See `expandas.skaccessors.gaussian_process`

grid_search
Property to access `sklearn.grid_search`. See `expandas.skaccessors.grid_search`

has_data()
Return whether `ModelFrame` has data

Returns **has_data** : bool

has_target()
Return whether `ModelFrame` has target

Returns **has_target** : bool

inverse_transform(*estimator*, **args*, ***kwargs*)
Call estimator's inverse_transform method.

Parameters **args** : arguments passed to inverse_transform method

kwargs : keyword arguments passed to inverse_transform method

Returns **returned** : transformed result

isotonic
Property to access `sklearn.isotonic`. See `expandas.skaccessors.isotonic`

kernel_approximation
Property to access `sklearn.kernel_approximation`

lda
Property to access `sklearn.lda`

learning_curve
Property to access `sklearn.learning_curve`. See `expandas.skaccessors.learning_curve`

linear_model
Property to access `sklearn.linear_model`. See `expandas.skaccessors.linear_model`

lm
Property to access `sklearn.linear_model`. See `expandas.skaccessors.linear_model`

log_proba
Return current estimator's log probabilities

Returns **probabilities** : `ModelFrame`

manifold
Property to access `sklearn.manifold`. See `expandas.skaccessors.manifold`

metrics
Property to access `sklearn.metrics`. See `expandas.skaccessors.metrics`

mixture
Property to access `sklearn.mixture`

multiclass
Property to access `sklearn.multiclass`. See `expandas.skaccessors.multiclass`

naive_bayes
Property to access `sklearn.naive_bayes`

neighbors
Property to access `sklearn.neighbors`. See `expandas.skaccessors.neighbors`

neural_network
Property to access `sklearn.neural_network`

pipeline
Property to access `sklearn.pipeline`. See `expandas.skaccessors.pipeline`

pp
Property to access `sklearn.preprocessing`. See `expandas.skaccessors.preprocessing`

predict(*estimator*, **args*, ***kwargs*)
Call estimator's predict method.

Parameters **args** : arguments passed to predict method

kwargs : keyword arguments passed to predict method

Returns **returned** : predicted result

predict_log_proba(*estimator*, **args*, ***kwargs*)
Call estimator's predict_log_proba method.

Parameters **args** : arguments passed to predict_log_proba method

kwargs : keyword arguments passed to predict_log_proba method

Returns **returned** : probabilities

predict_proba(*estimator*, **args*, ***kwargs*)
Call estimator's predict_proba method.

Parameters **args** : arguments passed to predict_proba method

kwargs : keyword arguments passed to predict_proba method

Returns **returned** : probabilities

predicted
Return current estimator's predicted results

Returns **predicted** : `ModelSeries`

preprocessing
Property to access `sklearn.preprocessing`. See `expandas.skaccessors.preprocessing`

proba
Return current estimator's probabilities

Returns **probabilities** : `ModelFrame`

qda
Property to access `sklearn.qda`

random_projection
Property to access `sklearn.random_projection`. See `expandas.skaccessors.random_projection`

score(*estimator*, **args*, ***kwargs*)
Call estimator's score method.

Parameters **args** : arguments passed to score method

kwargs : keyword arguments passed to score method

Returns **returned** : score

semi_supervised
Property to access `sklearn.semi_supervised`. See `expandas.skaccessors.semi_supervised`

svm
Property to access `sklearn.svm`. See `expandas.skaccessors.svm`

target
Return target (response variable)

Returns **target** : `ModelSeries`

target_name
Return target column name

Returns **target** : object

transform(*estimator*, **args*, ***kwargs*)
Call estimator's transform method.

Parameters **args** : arguments passed to transform method

kwargs : keyword arguments passed to transform method

Returns **returned** : transformed result

tree
Property to access `sklearn.tree`

class `expandas.core.series.`**ModelSeries**(*data=None*, *index=None*, *dtype=None*, *name=None*, *copy=False*, *fastpath=False*)
Bases: `pandas.core.series.Series`

Wrapper for `pandas.Series` to support `sklearn.preprocessing`

Attributes

`T`	return the transpose, which is by definition self
`at`	
`axes`	
`base`	return the base object if the memory of the underlying data is shared
`blocks`	Internal property, property synonym for as_blocks()
`data`	return the data pointer of the underlying data
`dtype`	return the dtype object of the underlying data
`dtypes`	return the dtype object of the underlying data
`empty`	True if NDFrame is entirely empty [no items]
`flags`	return the ndarray.flags for the underlying data
`ftype`	return if the data is sparseldense
`ftypes`	return if the data is sparseldense
`iat`	
`iloc`	
`imag`	
`is_time_series`	
`itemsize`	return the size of the dtype of the item of the underlying data
`ix`	
`loc`	
`nbytes`	return the number of bytes in the underlying data

Continued on next page

Table 4.3 – continued from previous page

ndim	return the number of dimensions of the underlying data, by definition 1
pp	Property to access sklearn.preprocessing.
preprocessing	Property to access sklearn.preprocessing.
real	
shape	return a tuple of the shape of the underlying data
size	return the number of elements in the underlying data
strides	return the strides of the underlying data
values	Return Series as ndarray

cat	
dt	
is_copy	
str	

Methods

abs()	Return an object with absolute value taken.
add(other[, level, fill_value, axis])	Binary operator add with support to substitute a fill_value for missing data
add_prefix(prefix)	Concatenate prefix string with panel items names.
add_suffix(suffix)	Concatenate suffix string with panel items names
align(other[, join, axis, level, copy, ...])	Align two object on their axes with the
all([axis, bool_only, skipna, level])	Return whether all elements are True over requested axis
any([axis, bool_only, skipna, level])	Return whether any element is True over requested axis
append(to_append[, verify_integrity])	Concatenate two or more Series.
apply(func[, convert_dtype, args])	Invoke function on values of Series.
argmax([axis, out, skipna])	Index of first occurrence of maximum of values.
argmin([axis, out, skipna])	Index of first occurrence of minimum of values.
argsort([axis, kind, order])	Overrides ndarray.argsort.
as_blocks()	Convert the frame to a dict of dtype -> Constructor Types that each has a homog
as_matrix([columns])	Convert the frame to its Numpy-array representation.
asfreq(freq[, method, how, normalize])	Convert all TimeSeries inside to specified frequency using DateOffset objects.
asof(where)	Return last good (non-NaN) value in TimeSeries if value is NaN for requested da
astype(dtype[, copy, raise_on_error])	Cast object to input numpy.dtype
at_time(time[, asof])	Select values at particular time of day (e.g.
autocorr()	Lag-1 autocorrelation
between(left, right[, inclusive])	Return boolean Series equivalent to left <= series <= right.
between_time(start_time, end_time[, ...])	Select values between particular times of the day (e.g., 9:00-9:30 AM)
bfill([axis, inplace, limit, downcast])	Synonym for NDFrame.fillna(method='bfill')
bool()	Return the bool of a single element PandasObject
clip([lower, upper, out])	Trim values at input threshold(s)
clip_lower(threshold)	Return copy of the input with values below given value truncated
clip_upper(threshold)	Return copy of input with values above given value truncated
combine(other, func[, fill_value])	Perform elementwise binary operation on two Series using given function
combine_first(other)	Combine Series values, choosing the calling Series's values first.
compound([axis, skipna, level])	Return the compound percentage of the values for the requested axis
compress(condition[, axis, out])	Return selected slices of an array along given axis as a Series
consolidate([inplace])	Compute NDFrame with "consolidated" internals (data of each dtype grouped to
convert_objects([convert_dates, ...])	Attempt to infer better dtype for object columns
copy([deep])	Make a copy of this object
corr(other[, method, min_periods])	Compute correlation with *other* Series, excluding missing values

Table 4.4 – continued from previous page

count([level])	Return number of non-NA/null observations in the Series
cov(other[, min_periods])	Compute covariance with Series, excluding missing values
cummax([axis, dtype, out, skipna])	Return cumulative max over requested axis.
cummin([axis, dtype, out, skipna])	Return cumulative min over requested axis.
cumprod([axis, dtype, out, skipna])	Return cumulative prod over requested axis.
cumsum([axis, dtype, out, skipna])	Return cumulative sum over requested axis.
describe([percentile_width, percentiles, ...])	Generate various summary statistics, excluding NaN values.
diff([periods])	1st discrete difference of object
div(other[, level, fill_value, axis])	Binary operator truediv with support to substitute a fill_value for missing data
divide(other[, level, fill_value, axis])	Binary operator truediv with support to substitute a fill_value for missing data
dot(other)	Matrix multiplication with DataFrame or inner-product with Series
drop(labels[, axis, level, inplace])	Return new object with labels in requested axis removed
drop_duplicates([take_last, inplace])	Return Series with duplicate values removed
dropna([axis, inplace])	Return Series without null values
duplicated([take_last])	Return boolean Series denoting duplicate values
eq(other)	
equals(other)	Determines if two NDFrame objects contain the same elements.
factorize([sort, na_sentinel])	Encode the object as an enumerated type or categorical variable
ffill([axis, inplace, limit, downcast])	Synonym for NDFrame.fillna(method='ffill')
fillna([value, method, axis, inplace, ...])	Fill NA/NaN values using the specified method
filter([items, like, regex, axis])	Restrict the info axis to set of items or wildcard
first(offset)	Convenience method for subsetting initial periods of time series data
first_valid_index()	Return label for first non-NA/null value
floordiv(other[, level, fill_value, axis])	Binary operator floordiv with support to substitute a fill_value for missing data
from_array(arr[, index, name, dtype, copy, ...])	
from_csv(path[, sep, parse_dates, header, ...])	Read delimited file into Series
ge(other)	
get(key[, default])	Get item from object for given key (DataFrame column, Panel slice, etc.).
get_dtype_counts()	Return the counts of dtypes in this object
get_ftype_counts()	Return the counts of ftypes in this object
get_value(label[, takeable])	Quickly retrieve single value at passed index label
get_values()	same as values (but handles sparseness conversions); is a view
groupby([by, axis, level, as_index, sort, ...])	Group series using mapper (dict or key function, apply given function
gt(other)	
hasnans()	return if I have any nans; enables various perf speedups
head([n])	Returns first n rows
hist([by, ax, grid, xlabelsize, xrot, ...])	Draw histogram of the input series using matplotlib
idxmax([axis, out, skipna])	Index of first occurrence of maximum of values.
idxmin([axis, out, skipna])	Index of first occurrence of minimum of values.
iget(i[, axis])	Return the i-th value or values in the Series by location
iget_value(i[, axis])	Return the i-th value or values in the Series by location
interpolate([method, axis, limit, inplace, ...])	Interpolate values according to different methods.
irow(i[, axis])	Return the i-th value or values in the Series by location
isin(values)	Return a boolean Series showing whether each element in the Series is exa
isnull()	Return a boolean same-sized object indicating if the values are null
item()	return the first element of the underlying data as a python scalar
iteritems()	Lazily iterate over (index, value) tuples
iterkv(*args, **kwargs)	iteritems alias used to get around 2to3. Deprecated
keys()	Alias for index
kurt([axis, skipna, level, numeric_only])	Return unbiased kurtosis over requested axis
kurtosis([axis, skipna, level, numeric_only])	Return unbiased kurtosis over requested axis
last(offset)	Convenience method for subsetting final periods of time series data

Table 4.4 – continued from previous page

last_valid_index()	Return label for last non-NA/null value
le(other)	
load(path)	Deprecated.
lt(other)	
mad([axis, skipna, level])	Return the mean absolute deviation of the values for the requested axis
map(arg[, na_action])	Map values of Series using input correspondence (which can be
mask(cond)	Returns copy whose values are replaced with nan if the
max([axis, skipna, level, numeric_only])	This method returns the maximum of the values in the object.
mean([axis, skipna, level, numeric_only])	Return the mean of the values for the requested axis
median([axis, skipna, level, numeric_only])	Return the median of the values for the requested axis
min([axis, skipna, level, numeric_only])	This method returns the minimum of the values in the object.
mod(other[, level, fill_value, axis])	Binary operator mod with support to substitute a fill_value for missing data
mode()	Returns the mode(s) of the dataset.
mul(other[, level, fill_value, axis])	Binary operator mul with support to substitute a fill_value for missing data
multiply(other[, level, fill_value, axis])	Binary operator mul with support to substitute a fill_value for missing data
ne(other)	
nlargest([n, take_last])	Return the largest *n* elements.
nonzero()	Return the indices of the elements that are non-zero
notnull()	Return a boolean same-sized object indicating if the values are
nsmallest([n, take_last])	Return the smallest *n* elements.
nunique([dropna])	Return number of unique elements in the object.
order([na_last, ascending, kind, ...])	Sorts Series object, by value, maintaining index-value link.
pct_change([periods, fill_method, limit, freq])	Percent change over given number of periods.
plot(data[, kind, ax, figsize, use_index, ...])	Make plots of Series using matplotlib / pylab.
pop(item)	Return item and drop from frame.
pow(other[, level, fill_value, axis])	Binary operator pow with support to substitute a fill_value for missing data
prod([axis, skipna, level, numeric_only])	Return the product of the values for the requested axis
product([axis, skipna, level, numeric_only])	Return the product of the values for the requested axis
ptp([axis, out])	
put(*args, **kwargs)	return a ndarray with the values put
quantile([q])	Return value at the given quantile, a la numpy.percentile.
radd(other[, level, fill_value, axis])	Binary operator radd with support to substitute a fill_value for missing data
rank([method, na_option, ascending, pct])	Compute data ranks (1 through n).
ravel([order])	Return the flattened underlying data as an ndarray
rdiv(other[, level, fill_value, axis])	Binary operator rtruediv with support to substitute a fill_value for missing data
reindex([index])	Conform Series to new index with optional filling logic, placing NA/NaN in loca
reindex_axis(labels[, axis])	for compatibility with higher dims
reindex_like(other[, method, copy, limit])	return an object with matching indicies to myself
rename([index])	Alter axes input function or functions.
rename_axis(mapper[, axis, copy, inplace])	Alter index and / or columns using input function or functions.
reorder_levels(order)	Rearrange index levels using input order.
repeat(reps)	return a new Series with the values repeated reps times
replace([to_replace, value, inplace, limit, ...])	Replace values given in 'to_replace' with 'value'.
resample(rule[, how, axis, fill_method, ...])	Convenience method for frequency conversion and resampling of regular time-se
reset_index([level, drop, name, inplace])	Analogous to the pandas.DataFrame.reset_index() function, see doc
reshape(*args, **kwargs)	return an ndarray with the values shape
rfloordiv(other[, level, fill_value, axis])	Binary operator rfloordiv with support to substitute a fill_value for missing data
rmod(other[, level, fill_value, axis])	Binary operator rmod with support to substitute a fill_value for missing data
rmul(other[, level, fill_value, axis])	Binary operator rmul with support to substitute a fill_value for missing data
round([decimals, out])	Return *a* with each element rounded to the given number of decimals.
rpow(other[, level, fill_value, axis])	Binary operator rpow with support to substitute a fill_value for missing data
rsub(other[, level, fill_value, axis])	Binary operator rsub with support to substitute a fill_value for missing data

Table 4.4 – continued from previous page

rtruediv(other[, level, fill_value, axis])	Binary operator rtruediv with support to substitute a fill_value for missing data
save(path)	Deprecated.
searchsorted(v[, side, sorter])	Find indices where elements should be inserted to maintain order.
select(crit[, axis])	Return data corresponding to axis labels matching criteria
sem([axis, skipna, level, ddof])	Return unbiased standard error of the mean over requested axis.
set_axis(axis, labels)	public verson of axis assignment
set_value(label, value[, takeable])	Quickly set single value at passed label.
shift([periods, freq, axis])	Shift index by desired number of periods with an optional time freq
skew([axis, skipna, level, numeric_only])	Return unbiased skew over requested axis
slice_shift([periods, axis])	Equivalent to *shift* without copying data.
sort([axis, ascending, kind, na_position, ...])	Sort values and index labels by value.
sort_index([ascending])	Sort object by labels (along an axis)
sortlevel([level, ascending, sort_remaining])	Sort Series with MultiIndex by chosen level.
squeeze()	squeeze length 1 dimensions
std([axis, skipna, level, ddof])	Return unbiased standard deviation over requested axis.
sub(other[, level, fill_value, axis])	Binary operator sub with support to substitute a fill_value for missing data
subtract(other[, level, fill_value, axis])	Binary operator sub with support to substitute a fill_value for missing data
sum([axis, skipna, level, numeric_only])	Return the sum of the values for the requested axis
swapaxes(axis1, axis2[, copy])	Interchange axes and swap values axes appropriately
swaplevel(i, j[, copy])	Swap levels i and j in a MultiIndex
tail([n])	Returns last n rows
take(indices[, axis, convert, is_copy])	return Series corresponding to requested indices
to_clipboard([excel, sep])	Attempt to write text representation of object to the system clipboard This can be
to_csv(path[, index, sep, na_rep, ...])	Write Series to a comma-separated values (csv) file
to_dense()	Return dense representation of NDFrame (as opposed to sparse)
to_dict()	Convert Series to {label -> value} dict
to_frame([name])	Convert Series to DataFrame
to_hdf(path_or_buf, key, **kwargs)	activate the HDFStore
to_json([path_or_buf, orient, date_format, ...])	Convert the object to a JSON string.
to_msgpack([path_or_buf])	msgpack (serialize) object to input file path
to_period([freq, copy])	Convert TimeSeries from DatetimeIndex to PeriodIndex with desired
to_pickle(path)	Pickle (serialize) object to input file path
to_sparse([kind, fill_value])	Convert Series to SparseSeries
to_sql(name, con[, flavor, schema, ...])	Write records stored in a DataFrame to a SQL database.
to_string([buf, na_rep, float_format, ...])	Render a string representation of the Series
to_timestamp([freq, how, copy])	Cast to datetimeindex of timestamps, at *beginning* of period
tolist()	Convert Series to a nested list
transpose()	return the transpose, which is by definition self
truediv(other[, level, fill_value, axis])	Binary operator truediv with support to substitute a fill_value for missing data
truncate([before, after, axis, copy])	Truncates a sorted NDFrame before and/or after some particular dates.
tshift([periods, freq, axis])	Shift the time index, using the index's frequency if available
tz_convert(tz[, axis, level, copy])	Convert the axis to target time zone.
tz_localize(*args, **kwargs)	Localize tz-naive TimeSeries to target time zone
unique()	Return array of unique values in the object.
unstack([level])	Unstack, a.k.a.
update(other)	Modify Series in place using non-NA values from passed Series.
valid([inplace])	
value_counts([normalize, sort, ascending, ...])	Returns object containing counts of unique values.
var([axis, skipna, level, ddof])	Return unbiased variance over requested axis.
view([dtype])	
where(cond[, other, inplace, axis, level, ...])	Return an object of same shape as self and whose corresponding entries are from
xs(key[, axis, level, copy, drop_level])	Returns a cross-section (row(s) or column(s)) from the Series/DataFrame.

pp
Property to access `sklearn.preprocessing`. See `expandas.skaccessors.preprocessing`

preprocessing
Property to access `sklearn.preprocessing`. See `expandas.skaccessors.preprocessing`

to_frame(*name=None*)
Convert Series to DataFrame

Parameters **name** : object, default None

The passed name should substitute for the series name (if it has one).

Returns **data_frame** : DataFrame

4.2 Module contents

CHAPTER 5

expandas.skaccessors package

5.1 Subpackages

5.1.1 expandas.skaccessors.test package

Submodules

Module contents

5.2 Submodules

class expandas.skaccessors.cluster.**ClusterMethods**(*df*, *module_name=None*, *attrs=None*)

Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.cluster.

Attributes

bicluster	Property to access sklearn.cluster.bicluster

Methods

affinity_propagation(*args, **kwargs)	Call sklearn.cluster.affinity_propagation using automatic mappin
dbscan(*args, **kwargs)	Call sklearn.cluster.dbscan using automatic mapping.
k_means(n_clusters, *args, **kwargs)	Call sklearn.cluster.k_means using automatic mapping.
mean_shift(*args, **kwargs)	Call sklearn.cluster.mean_shift using automatic mapping.
spectral_clustering(*args, **kwargs)	Call sklearn.cluster.spectral_clustering using automatic mapping

affinity_propagation(**args*, ***kwargs*)

Call sklearn.cluster.affinity_propagation using automatic mapping.

- S: ModelFrame.data

bicluster

Property to access sklearn.cluster.bicluster

dbscan(**args*, ***kwargs*)
Call `sklearn.cluster.dbscan` using automatic mapping.

- X: `ModelFrame.data`

k_means(*n_clusters*, **args*, ***kwargs*)
Call `sklearn.cluster.k_means` using automatic mapping.

- X: `ModelFrame.data`

mean_shift(**args*, ***kwargs*)
Call `sklearn.cluster.mean_shift` using automatic mapping.

- X: `ModelFrame.data`

spectral_clustering(**args*, ***kwargs*)
Call `sklearn.cluster.spectral_clustering` using automatic mapping.

- `affinity`: `ModelFrame.data`

class `expandas.skaccessors.covariance.`**CovarianceMethods**(*df*, *module_name=None*, *attrs=None*)

Bases: `expandas.core.accessor.AccessorMethods`

Accessor to `sklearn.covariance`.

Methods

`empirical_covariance`(*args, **kwargs)	Call `sklearn.covariance.empirical_covariance` using automatic ma
`ledoit_wolf`(*args, **kwargs)	Call `sklearn.covariance.ledoit_wolf` using automatic mapping.
`oas`(*args, **kwargs)	Call `sklearn.covariance.oas` using automatic mapping.

empirical_covariance(**args*, ***kwargs*)
Call `sklearn.covariance.empirical_covariance` using automatic mapping.

- X: `ModelFrame.data`

ledoit_wolf(**args*, ***kwargs*)
Call `sklearn.covariance.ledoit_wolf` using automatic mapping.

- X: `ModelFrame.data`

oas(**args*, ***kwargs*)
Call `sklearn.covariance.oas` using automatic mapping.

- X: `ModelFrame.data`

class `expandas.skaccessors.cross_validation.`**CrossValidationMethods**(*df*, *module_name=None*, *attrs=None*)

Bases: `expandas.core.accessor.AccessorMethods`

Accessor to `sklearn.cross_validation`.

Methods

`StratifiedShuffleSplit`(*args, **kwargs)	Instanciate `sklearn.cross_validation.StratifiedShuffleSp`
`check_cv`(cv, *args, **kwargs)	Call `sklearn.cross_validation.check_cv` using automatic mappi

Table 5.4 – continued from previous page

cross_val_score(estimator, *args, **kwargs)	Call sklearn.cross_validation.cross_val_score using autom
iterate(cv)	Generate ModelFrame using iterators for cross validation
permutation_test_score(estimator, *args, ...)	Call sklearn.cross_validation.permutation_test_score u
train_test_split(*args, **kwargs)	Call sklearn.cross_validation.train_test_split using auto

StratifiedShuffleSplit(**args*, ***kwargs*)

Instanciate sklearn.cross_validation.StratifiedShuffleSplit using automatic mapping.

- y: ModelFrame.target

check_cv(*cv*, **args*, ***kwargs*)

Call sklearn.cross_validation.check_cv using automatic mapping.

- X: ModelFrame.data
- y: ModelFrame.target

cross_val_score(*estimator*, **args*, ***kwargs*)

Call sklearn.cross_validation.cross_val_score using automatic mapping.

- X: ModelFrame.data
- y: ModelFrame.target

iterate(*cv*)

Generate ModelFrame using iterators for cross validation

Parameters **cv** : cross validation iterator

Returns **generated** : generator of ModelFrame

permutation_test_score(*estimator*, **args*, ***kwargs*)

Call sklearn.cross_validation.permutation_test_score using automatic mapping.

- X: ModelFrame.data
- y: ModelFrame.target

train_test_split(**args*, ***kwargs*)

Call sklearn.cross_validation.train_test_split using automatic mapping.

class expandas.skaccessors.decomposition.**DecompositionMethods**(*df*, *module_name=None*, *attrs=None*)

Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.decomposition.

Methods

dict_learning(n_components, alpha, *args, ...)	Call sklearn.decomposition.dict_learning using automatic map
dict_learning_online(*args, **kwargs)	Call sklearn.decomposition.dict_learning_online using auto
fastica(*args, **kwargs)	Call sklearn.decomposition.fastica using automatic mapping.
sparse_encode(dictionary, *args, **kwargs)	Call sklearn.decomposition.sparce_encode using automatic map

dict_learning(*n_components*, *alpha*, **args*, ***kwargs*)

Call sklearn.decomposition.dict_learning using automatic mapping.

- X: ModelFrame.data

dict_learning_online(**args*, ***kwargs*)
Call sklearn.decomposition.dict_learning_online using automatic mapping.

- X: ModelFrame.data

fastica(**args*, ***kwargs*)
Call sklearn.decomposition.fastica using automatic mapping.

- X: ModelFrame.data

sparse_encode(*dictionary*, **args*, ***kwargs*)
Call sklearn.decomposition.sparce_encode using automatic mapping.

- X: ModelFrame.data

class expandas.skaccessors.ensemble.**EnsembleMethods**(*df*, *module_name=None*, *attrs=None*)
Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.ensemble.

Attributes

partial_dependence	Property to access sklearn.ensemble.partial_dependence

partial_dependence
Property to access sklearn.ensemble.partial_dependence

class expandas.skaccessors.ensemble.**PartialDependenceMethods**(*df*, *module_name=None*, *attrs=None*)
Bases: expandas.core.accessor.AccessorMethods

Methods

partial_dependence(gbrt, target_variables, ...)	Call sklearn.ensemble.partial_dependence using automa
plot_partial_dependence(gbrt, features, **kwargs)	Call sklearn.ensemble.plot_partial_dependence using

partial_dependence(*gbrt*, *target_variables*, ***kwargs*)
Call sklearn.ensemble.partial_dependence using automatic mapping.

- X: ModelFrame.data

plot_partial_dependence(*gbrt*, *features*, ***kwargs*)
Call sklearn.ensemble.plot_partial_dependence using automatic mapping.

- X: ModelFrame.data

class expandas.skaccessors.feature_extraction.**FeatureExtractionMethods**(*df*, *module_name=None*, *attrs=None*)
Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.feature_extraction.

Attributes

image	Property to access sklearn.feature_extraction.image
text	Property to access sklearn.feature_extraction.text

image
Property to access sklearn.feature_extraction.image

text
Property to access sklearn.feature_extraction.text

class expandas.skaccessors.feature_selection.**FeatureSelectionMethods**(*df*, *module_name=None*, *attrs=None*)

Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.feature_selection.

class expandas.skaccessors.gaussian_process.**GaussianProcessMethods**(*df*, *module_name=None*, *attrs=None*)

Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.gaussian_process.

Attributes

correlation_models	Property to access sklearn.gaussian_process.correlation_models
regression_models	Property to access sklearn.gaussian_process.regression_models

correlation_models
Property to access sklearn.gaussian_process.correlation_models

regression_models
Property to access sklearn.gaussian_process.regression_models

class expandas.skaccessors.gaussian_process.**RegressionModelsMethods**(*df*, *module_name=None*, *attrs=None*)

Bases: expandas.core.accessor.AccessorMethods

class expandas.skaccessors.grid_search.**GridSearchMethods**(*df*, *module_name=None*, *attrs=None*)

Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.grid_search.

Methods

describe(estimator)	Describe grid search results

describe (*estimator*)
Describe grid search results

Parameters **estimator** : fitted grid search estimator

Returns **described** : ModelFrame

class expandas.skaccessors.isotonic.**IsotonicMethods** (*df*, *module_name=None*, *attrs=None*)
Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.isotonic.

Attributes

IsotonicRegression	sklearn.isotonic.IsotonicRegression

Methods

check_increasing(*args, **kwargs)	Call sklearn.isotonic.check_increasing using automatic mapping.
isotonic_regression(*args, **kwargs)	Call sklearn.isotonic.isotonic_regression using automatic mapping

IsotonicRegression
sklearn.isotonic.IsotonicRegression

check_increasing (**args*, ***kwargs*)
Call sklearn.isotonic.check_increasing using automatic mapping.

- x: ModelFrame.index
- y: ModelFrame.target

isotonic_regression (**args*, ***kwargs*)
Call sklearn.isotonic.isotonic_regression using automatic mapping.

- y: ModelFrame.target

class expandas.skaccessors.learning_curve.**LearningCurveMethods** (*df*, *module_name=None*, *attrs=None*)
Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.learning_curve.

Methods

learning_curve(estimator, *args, **kwargs)	Call sklearn.lerning_curve.learning_curve using automatic ma
validation_curve(estimator, param_name, ...)	Call sklearn.learning_curve.validation_curve using automati

learning_curve(*estimator*, **args*, ***kwargs*)
Call `sklearn.lerning_curve.learning_curve` using automatic mapping.

- X: `ModelFrame.data`
- y: `ModelFrame.target`

validation_curve(*estimator*, *param_name*, *param_range*, **args*, ***kwargs*)
Call `sklearn.learning_curve.validation_curve` using automatic mapping.

- X: `ModelFrame.data`
- y: `ModelFrame.target`

class expandas.skaccessors.linear_model.**LinearModelMethods**(*df*, *module_name=None*, *attrs=None*)
Bases: expandas.core.accessor.AccessorMethods

Accessor to `sklearn.linear_model`.

Methods

lars_path(*args, **kwargs)	Call `sklearn.linear_model.lars_path` using automatic mapping.
lasso_path(*args, **kwargs)	Call `sklearn.linear_model.lasso_path` using automatic mapping.
lasso_stability_path(*args, **kwargs)	Call `sklearn.linear_model.lasso_stability_path` using automatic
orthogonal_mp_gram(*args, **kwargs)	Call `sklearn.linear_model.orthogonal_mp_gram` using automatic ma

lars_path(**args*, ***kwargs*)
Call `sklearn.linear_model.lars_path` using automatic mapping.

- X: `ModelFrame.data`
- y: `ModelFrame.target`

lasso_path(**args*, ***kwargs*)
Call `sklearn.linear_model.lasso_path` using automatic mapping.

- X: `ModelFrame.data`
- y: `ModelFrame.target`

lasso_stability_path(**args*, ***kwargs*)
Call `sklearn.linear_model.lasso_stability_path` using automatic mapping.

- X: `ModelFrame.data`
- y: `ModelFrame.target`

orthogonal_mp_gram(**args*, ***kwargs*)
Call `sklearn.linear_model.orthogonal_mp_gram` using automatic mapping.

- Gram: `ModelFrame.data.T.dot(ModelFrame.data)`
- Xy: `ModelFrame.data.T.dot(ModelFrame.target)`

class expandas.skaccessors.manifold.**ManifoldMethods**(*df*, *module_name=None*, *attrs=None*)
Bases: expandas.core.accessor.AccessorMethods

Accessor to `sklearn.manifold`.

Methods

locally_linear_embedding(n_neighbors, ...)	Call sklearn.manifold.locally_linear_embedding using auto
spectral_embedding(*args, **kwargs)	Call sklearn.manifold.spectral_embedding using automatic ma

locally_linear_embedding(*n_neighbors*, *n_components*, **args*, ***kwargs*)
Call sklearn.manifold.locally_linear_embedding using automatic mapping.

- X: ModelFrame.data

spectral_embedding(**args*, ***kwargs*)
Call sklearn.manifold.spectral_embedding using automatic mapping.

- adjacency: ModelFrame.data

class expandas.skaccessors.metrics.**MetricsMethods**(*df*, *module_name=None*, *attrs=None*)
Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.metrics.

Attributes

pairwise	Not implemented

Methods

auc([kind, reorder])	Calcurate AUC of ROC curve or precision recall curve
average_precision_score(*args, **kwargs)	Call sklearn.metrics.average_precision_score usi
confusion_matrix(*args, **kwargs)	Call sklearn.metrics.confusion_matrix using automa
consensus_score(*args, **kwargs)	Not implemented
f1_score(*args, **kwargs)	Call sklearn.metrics.f1_score using automatic mapping
fbeta_score(beta, *args, **kwargs)	Call sklearn.metrics.fbeta_score using automatic map
hinge_loss(*args, **kwargs)	Call sklearn.metrics.hinge_loss using automatic mapp
log_loss(*args, **kwargs)	Call sklearn.metrics.log_loss using automatic mapping
precision_recall_curve(*args, **kwargs)	Call sklearn.metrics.precision_recall_curve usin
precision_recall_fscore_support(*args, **kwargs)	Call sklearn.metrics.precision_recall_fscore_s
precision_score(*args, **kwargs)	Call sklearn.metrics.precision_score using automati
recall_score(*args, **kwargs)	Call sklearn.metrics.recall_score using automatic ma
roc_auc_score(*args, **kwargs)	Call sklearn.metrics.roc_auc_score using automatic n
roc_curve(*args, **kwargs)	Call sklearn.metrics.roc_curve using automatic mappin
silhouette_samples(*args, **kwargs)	Call sklearn.metrics.silhouette_samples using auto
silhouette_score(*args, **kwargs)	Call sklearn.metrics.silhouette_score using automa

auc(*kind='roc'*, *reorder=False*, ***kwargs*)
Calcurate AUC of ROC curve or precision recall curve

Parameters **kind** : {'roc', 'precision_recall_curve'}

Returns **float** : AUC

average_precision_score(**args*, ***kwargs*)
Call sklearn.metrics.average_precision_score using automatic mapping.

- y_true: ModelFrame.target
- y_score: ModelFrame.decision

confusion_matrix(**args*, ***kwargs*)
Call sklearn.metrics.confusion_matrix using automatic mapping.

- y_true: ModelFrame.target
- y_pred: ModelFrame.predicted

consensus_score(**args*, ***kwargs*)
Not implemented

f1_score(**args*, ***kwargs*)
Call sklearn.metrics.f1_score using automatic mapping.

- y_true: ModelFrame.target
- y_pred: ModelFrame.predicted

fbeta_score(*beta*, **args*, ***kwargs*)
Call sklearn.metrics.fbeta_score using automatic mapping.

- y_true: ModelFrame.target
- y_pred: ModelFrame.predicted

hinge_loss(**args*, ***kwargs*)
Call sklearn.metrics.hinge_loss using automatic mapping.

- y_true: ModelFrame.target
- y_pred_decision: ModelFrame.decision

log_loss(**args*, ***kwargs*)
Call sklearn.metrics.log_loss using automatic mapping.

- y_true: ModelFrame.target
- y_pred: ModelFrame.proba

pairwise
Not implemented

precision_recall_curve(**args*, ***kwargs*)
Call sklearn.metrics.precision_recall_curve using automatic mapping.

- y_true: ModelFrame.target
- y_probas_pred: ModelFrame.decision

precision_recall_fscore_support(**args*, ***kwargs*)
Call sklearn.metrics.precision_recall_fscore_support using automatic mapping.

- y_true: ModelFrame.target
- y_pred: ‘‘ModelFrame.predicted‘

precision_score(**args*, ***kwargs*)
Call sklearn.metrics.precision_score using automatic mapping.

- y_true: ModelFrame.target
- y_pred: ModelFrame.predicted

recall_score(**args*, ***kwargs*)
Call sklearn.metrics.recall_score using automatic mapping.

- y_true: ModelFrame.target
- y_true: ModelFrame.predicted

roc_auc_score(**args*, ***kwargs*)
Call sklearn.metrics.roc_auc_score using automatic mapping.

- y_true: ModelFrame.target
- y_score: ModelFrame.decision

roc_curve(**args*, ***kwargs*)
Call sklearn.metrics.roc_curve using automatic mapping.

- y_true: ModelFrame.target
- y_score: ModelFrame.decision

silhouette_samples(**args*, ***kwargs*)
Call sklearn.metrics.silhouette_samples using automatic mapping.

- X: ModelFrame.data
- labels: ModelFrame.predicted

silhouette_score(**args*, ***kwargs*)
Call sklearn.metrics.silhouette_score using automatic mapping.

- X: ModelFrame.data
- labels: ModelFrame.predicted

class expandas.skaccessors.multiclass.**MultiClassMethods**(*df*, *module_name=None*, *attrs=None*)

Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.multiclass.

Attributes

OneVsOneClassifier	sklearn.multiclass.OneVsOneClassifier
OneVsRestClassifier	sklearn.multiclass.OneVsRestClassifier
OutputCodeClassifier	sklearn.multiclass.OutputCodeClassifier

Methods

fit_ecoc(*args, **kwargs)	Deprecated
fit_ovo(*args, **kwargs)	Deprecated
fit_ovr(*args, **kwargs)	Deprecated
predict_ecoc(*args, **kwargs)	Deprecated
predict_ovo(*args, **kwargs)	Deprecated
predict_ovr(*args, **kwargs)	Deprecated

OneVsOneClassifier
sklearn.multiclass.OneVsOneClassifier

OneVsRestClassifier
sklearn.multiclass.OneVsRestClassifier

OutputCodeClassifier
sklearn.multiclass.OutputCodeClassifier

fit_ecoc(**args*, ***kwargs*)
Deprecated

fit_ovo(**args*, ***kwargs*)
Deprecated

fit_ovr(**args*, ***kwargs*)
Deprecated

predict_ecoc(**args*, ***kwargs*)
Deprecated

predict_ovo(**args*, ***kwargs*)
Deprecated

predict_ovr(**args*, ***kwargs*)
Deprecated

class expandas.skaccessors.neighbors.**NeighborsMethods**(*df*, *module_name=None*, *attrs=None*)

Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.neighbors.

class expandas.skaccessors.pipeline.**PipelineMethods**(*df*, *module_name=None*, *attrs=None*)

Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.pipeline.

Attributes

make_pipeline	sklearn.pipeline.make_pipeline
make_union	sklearn.pipeline.make_union

make_pipeline
sklearn.pipeline.make_pipeline

make_union
sklearn.pipeline.make_union

class expandas.skaccessors.preprocessing.**PreprocessingMethods**(*df*, *module_name=None*, *attrs=None*)

Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.preprocessing.

Methods

add_dummy_feature([value])	Call sklearn.preprocessing.add_dummy_feature using automatic mapping.

add_dummy_feature(*value=1.0*)
Call sklearn.preprocessing.add_dummy_feature using automatic mapping.

- X: ModelFrame.data

class expandas.skaccessors.svm.**SVMMethods**(*df*, *module_name=None*, *attrs=None*)
Bases: expandas.core.accessor.AccessorMethods

Accessor to sklearn.svm.

Attributes

liblinear	Not implemented
libsvm	Not implemented
libsvm_sparse	Not implemented

Methods

l1_min_c(*args, **kwargs)	Call sklearn.svm.l1_min_c using automatic mapping.

l1_min_c(**args*, ***kwargs*)
Call sklearn.svm.l1_min_c using automatic mapping.

- X: ModelFrame.data
- y: ModelFrame.target

liblinear
Not implemented

libsvm
Not implemented

libsvm_sparse
Not implemented

5.3 Module contents

e

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

V