pandas.Grouper

class pandas.**Grouper**(**args*, ***kwargs*)

A Grouper allows the user to specify a groupby instruction for an object.

This specification will select a column via the key parameter, or if the level and/or axis parameters are given, a level of the index of the target object.

If *axis* and/or *level* are passed as keywords to both *Grouper* and *groupby*, the values passed to *Grouper* take precedence.

Parameters

key [str, defaults to None] Groupby key, which selects the grouping column of the target.

level [name/number, defaults to None] The level for the target index.

freq [str / frequency object, defaults to None] This will groupby the specified frequency if the target selection (via key or level) is a datetime-like object. For full specification of available frequencies, please see here.

axis [str, int, defaults to 0] Number/name of the axis.

sort [bool, default to False] Whether to sort the resulting labels.

closed [{'left' or 'right'}] Closed end of interval. Only when *freq* parameter is passed.

label [{'left' or 'right'}] Interval boundary to use for labeling. Only when *freq* parameter is passed.

convention [{'start', 'end', 'e', 's'}] If grouper is PeriodIndex and *freq* parameter is passed.

base [int, default 0] Only when *freq* parameter is passed.

loffset [str, DateOffset, timedelta object] Only when *freq* parameter is passed.

Returns

A specification for a groupby instruction

Examples

Syntactic sugar for `df.groupby('A')`

```
>>> df.groupby(Grouper(key='A'))
```

Specify a resample operation on the column 'date'

```
>>> df.groupby(Grouper(key='date', freq='60s'))
```

Specify a resample operation on the level 'date' on the columns axis with a frequency of 60s

```
>>> df.groupby(Grouper(level='date', freq='60s', axis=1))
```

Attributes

ax	
groups	

3.11.2 Function application

GroupBy.apply(self, func, *args, **kwargs)	Apply function *func* group-wise and combine the results together.
GroupBy.agg(self, func, *args, **kwargs)	
GroupBy.aggregate(self, func, *args, **kwargs)	
GroupBy.transform(self, func, *args, **kwargs)	
GroupBy.pipe(self, func, *args, **kwargs)	Apply a function *func* with arguments to this GroupBy object and return the function's result.

pandas.core.groupby.GroupBy.apply

GroupBy.**apply**(*self*, *func*, **args*, ***kwargs*)

Apply function *func* group-wise and combine the results together.

The function passed to *apply* must take a dataframe as its first argument and return a DataFrame, Series or scalar. *apply* will then take care of combining the results back together into a single dataframe or series. *apply* is therefore a highly flexible grouping method.

While *apply* is a very flexible method, its downside is that using it can be quite a bit slower than using more specific methods like *agg* or *transform*. Pandas offers a wide range of method that will be much faster than using *apply* for their specific purposes, so try to use them before reaching for *apply*.

Parameters

func [callable] A callable that takes a dataframe as its first argument, and returns a dataframe, a series or a scalar. In addition the callable may take positional and keyword arguments.

args, kwargs [tuple and dict] Optional positional and keyword arguments to pass to *func*.

Returns

applied [Series or DataFrame]

See also:

pipe Apply function to the full GroupBy object instead of to each group.
aggregate Apply aggregate function to the GroupBy object.
transform Apply function column-by-column to the GroupBy object.
Series.apply Apply a function to a Series.
DataFrame.apply Apply a function to each row or column of a DataFrame.

pandas.core.groupby.GroupBy.agg

GroupBy.**agg**(*self*, *func*, **args*, ***kwargs*)

pandas.core.groupby.GroupBy.aggregate

GroupBy.**aggregate**(*self*, *func*, **args*, ***kwargs*)

pandas.core.groupby.GroupBy.transform

GroupBy.**transform**(*self*, *func*, **args*, ***kwargs*)

pandas.core.groupby.GroupBy.pipe

GroupBy.**pipe**(*self*, *func*, **args*, ***kwargs*)

Apply a function *func* with arguments to this GroupBy object and return the function's result.

New in version 0.21.0.

Use *.pipe* when you want to improve readability by chaining together functions that expect Series, DataFrames, GroupBy or Resampler objects. Instead of writing

```
>>> h(g(f(df.groupby('group')), arg1=a), arg2=b, arg3=c)
```

You can write

```
>>> (df.groupby('group')
...    .pipe(f)
...    .pipe(g, arg1=a)
...    .pipe(h, arg2=b, arg3=c))
```

which is much more readable.

Parameters

func [callable or tuple of (callable, string)] Function to apply to this GroupBy object or, alternatively, a *(callable, data_keyword)* tuple where *data_keyword* is a string indicating the keyword of *callable* that expects the GroupBy object.

args [iterable, optional] Positional arguments passed into *func*.

kwargs [dict, optional] A dictionary of keyword arguments passed into *func*.

Returns

object [the return type of *func*.]

See also:

Series.pipe Apply a function with arguments to a series.
DataFrame.pipe Apply a function with arguments to a dataframe.
apply Apply function to each group instead of to the full GroupBy object.

Notes

See more here

Examples

```
>>> df = pd.DataFrame({'A': 'a b a b'.split(), 'B': [1, 2, 3, 4]})
>>> df
   A  B
0  a  1
1  b  2
2  a  3
3  b  4
```

To get the difference between each groups maximum and minimum value in one pass, you can do

```
>>> df.groupby('A').pipe(lambda x: x.max() - x.min())
   B
A
a  2
b  2
```

3.11.3 Computations / descriptive stats

GroupBy.all(self, skipna)	Return True if all values in the group are truthful, else False.
GroupBy.any(self, skipna)	Return True if any value in the group is truthful, else False.
GroupBy.bfill(self[, limit])	Backward fill the values.
GroupBy.count(self)	Compute count of group, excluding missing values.
GroupBy.cumcount(self, ascending)	Number each item in each group from 0 to the length of that group - 1.
GroupBy.cummax(self[, axis])	Cumulative max for each group.
GroupBy.cummin(self[, axis])	Cumulative min for each group.
GroupBy.cumprod(self[, axis])	Cumulative product for each group.
GroupBy.cumsum(self[, axis])	Cumulative sum for each group.
GroupBy.ffill(self[, limit])	Forward fill the values.
GroupBy.first(self, **kwargs)	Compute first of group values.
GroupBy.head(self[, n])	Return first n rows of each group.
GroupBy.last(self, **kwargs)	Compute last of group values.
GroupBy.max(self, **kwargs)	Compute max of group values.
GroupBy.mean(self, *args, **kwargs)	Compute mean of groups, excluding missing values.
GroupBy.median(self, **kwargs)	Compute median of groups, excluding missing values.
GroupBy.min(self, **kwargs)	Compute min of group values.
GroupBy.ngroup(self, ascending)	Number each group from 0 to the number of groups - 1.
GroupBy.nth(self, n, List[int]], dropna, . . .)	Take the nth row from each group if n is an int, or a subset of rows if n is a list of ints.
GroupBy.ohlc(self)	Compute sum of values, excluding missing values.
GroupBy.prod(self, **kwargs)	Compute prod of group values.
GroupBy.rank(self, method, ascending, . . .)	Provide the rank of values within each group.

continues on next page

Table 402 – continued from previous page

GroupBy.pct_change(self[, periods, ...])	Calculate pct_change of each value to previous entry in group.
GroupBy.size(self)	Compute group sizes.
GroupBy.sem(self, ddof)	Compute standard error of the mean of groups, excluding missing values.
GroupBy.std(self, ddof, *args, **kwargs)	Compute standard deviation of groups, excluding missing values.
GroupBy.sum(self, **kwargs)	Compute sum of group values.
GroupBy.var(self, ddof, *args, **kwargs)	Compute variance of groups, excluding missing values.
GroupBy.tail(self[, n])	Return last n rows of each group.

pandas.core.groupby.GroupBy.all

GroupBy.**all**(*self*, *skipna: bool = True*)

Return True if all values in the group are truthful, else False.

Parameters

skipna [bool, default True] Flag to ignore nan values during truth testing.

Returns

bool

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.any

GroupBy.**any**(*self*, *skipna: bool = True*)

Return True if any value in the group is truthful, else False.

Parameters

skipna [bool, default True] Flag to ignore nan values during truth testing.

Returns

bool

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.bfill

GroupBy.**bfill**(*self*, *limit=None*)

Backward fill the values.

Parameters

limit [int, optional] Limit of how many values to fill.

Returns

Series or DataFrame Object with missing values filled.

See also:

Series.backfill
DataFrame.backfill
Series.fillna
DataFrame.fillna

pandas.core.groupby.GroupBy.count

GroupBy.**count**(*self*)

Compute count of group, excluding missing values.

Returns

Series or DataFrame Count of values within each group.

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.cumcount

GroupBy.**cumcount**(*self*, *ascending: bool = True*)

Number each item in each group from 0 to the length of that group - 1.

Essentially this is equivalent to

```
>>> self.apply(lambda x: pd.Series(np.arange(len(x)), x.index))
```

Parameters

ascending [bool, default True] If False, number in reverse, from length of group - 1 to 0.

Returns

Series Sequence number of each element within each group.

See also:

ngroup Number the groups themselves.

Examples

```
>>> df = pd.DataFrame([['a'], ['a'], ['a'], ['b'], ['b'], ['a']],
...                   columns=['A'])
>>> df
   A
0  a
1  a
2  a
3  b
4  b
5  a
>>> df.groupby('A').cumcount()
0    0
1    1
2    2
3    0
4    1
```

(continues on next page)

(continued from previous page)

```
5    3
dtype: int64
>>> df.groupby('A').cumcount(ascending=False)
0    3
1    2
2    1
3    1
4    0
5    0
dtype: int64
```

pandas.core.groupby.GroupBy.cummax

GroupBy.**cummax**(*self*, *axis=0*, ***kwargs*)

Cumulative max for each group.

Returns

Series or DataFrame

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.cummin

GroupBy.**cummin**(*self*, *axis=0*, ***kwargs*)

Cumulative min for each group.

Returns

Series or DataFrame

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.cumprod

GroupBy.**cumprod**(*self*, *axis=0*, **args*, ***kwargs*)

Cumulative product for each group.

Returns

Series or DataFrame

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.cumsum

GroupBy.**cumsum**(*self*, *axis=0*, **args*, ***kwargs*)

Cumulative sum for each group.

Returns

Series or DataFrame

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.ffill

GroupBy.**ffill**(*self*, *limit=None*)

Forward fill the values.

Parameters

limit [int, optional] Limit of how many values to fill.

Returns

Series or DataFrame Object with missing values filled.

See also:

Series.pad
DataFrame.pad
Series.fillna
DataFrame.fillna

pandas.core.groupby.GroupBy.first

GroupBy.**first**(*self*, ***kwargs*)

Compute first of group values.

Returns

Series or DataFrame Computed first of values within each group.

pandas.core.groupby.GroupBy.head

GroupBy.**head**(*self*, *n=5*)

Return first n rows of each group.

Similar to `.apply(lambda x: x.head(n))`, but it returns a subset of rows from the original DataFrame with original index and order preserved (`as_index` flag is ignored).

Does not work for negative values of *n*.

Returns

Series or DataFrame

See also:

Series.groupby
DataFrame.groupby

Examples

```
>>> df = pd.DataFrame([[1, 2], [1, 4], [5, 6]],
...                   columns=['A', 'B'])
>>> df.groupby('A').head(1)
   A  B
0  1  2
2  5  6
>>> df.groupby('A').head(-1)
Empty DataFrame
Columns: [A, B]
Index: []
```

pandas.core.groupby.GroupBy.last

GroupBy.**last**(*self*, ***kwargs*)

Compute last of group values.

Returns

Series or DataFrame Computed last of values within each group.

pandas.core.groupby.GroupBy.max

GroupBy.**max**(*self*, ***kwargs*)

Compute max of group values.

Returns

Series or DataFrame Computed max of values within each group.

pandas.core.groupby.GroupBy.mean

GroupBy.**mean**(*self*, **args*, ***kwargs*)

Compute mean of groups, excluding missing values.

Returns

pandas.Series or pandas.DataFrame

See also:

Series.groupby
DataFrame.groupby

Examples

```
>>> df = pd.DataFrame({'A': [1, 1, 2, 1, 2],
...                    'B': [np.nan, 2, 3, 4, 5],
...                    'C': [1, 2, 1, 1, 2]}, columns=['A', 'B', 'C'])
```

Groupby one column and return the mean of the remaining columns in each group.

```
>>> df.groupby('A').mean()
     B         C
A
```

(continues on next page)

(continued from previous page)

```
1  3.0  1.333333
2  4.0  1.500000
```

Groupby two columns and return the mean of the remaining column.

```
>>> df.groupby(['A', 'B']).mean()
       C
A B
1 2.0  2
  4.0  1
2 3.0  1
  5.0  2
```

Groupby one column and return the mean of only particular column in the group.

```
>>> df.groupby('A')['B'].mean()
A
1    3.0
2    4.0
Name: B, dtype: float64
```

pandas.core.groupby.GroupBy.median

GroupBy.**median**(*self*, ***kwargs*)

Compute median of groups, excluding missing values.

For multiple groupings, the result index will be a MultiIndex

Returns

Series or DataFrame Median of values within each group.

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.min

GroupBy.**min**(*self*, ***kwargs*)

Compute min of group values.

Returns

Series or DataFrame Computed min of values within each group.

pandas.core.groupby.GroupBy.ngroup

GroupBy.**ngroup**(*self*, *ascending: bool = True*)

Number each group from 0 to the number of groups - 1.

This is the enumerative complement of cumcount. Note that the numbers given to the groups match the order in which the groups would be seen when iterating over the groupby object, not the order they are first observed.

Parameters

ascending [bool, default True] If False, number in reverse, from number of group - 1 to 0.

Returns

Series Unique numbers for each group.

See also:

cumcount Number the rows in each group.

Examples

```
>>> df = pd.DataFrame({"A": list("aaabba")})
>>> df
   A
0  a
1  a
2  a
3  b
4  b
5  a
>>> df.groupby('A').ngroup()
0    0
1    0
2    0
3    1
4    1
5    0
dtype: int64
>>> df.groupby('A').ngroup(ascending=False)
0    1
1    1
2    1
3    0
4    0
5    1
dtype: int64
>>> df.groupby(["A", [1,1,2,3,2,1]]).ngroup()
0    0
1    0
2    1
3    3
4    2
5    0
dtype: int64
```

pandas.core.groupby.GroupBy.nth

GroupBy.**nth**(*self*, *n: Union[int, List[int]]*, *dropna: Union[str, NoneType] = None*) → pandas.core.frame.DataFrame

Take the nth row from each group if n is an int, or a subset of rows if n is a list of ints.

If dropna, will take the nth non-null row, dropna is either 'all' or 'any'; this is equivalent to calling dropna(how=dropna) before the groupby.

Parameters

n [int or list of ints] A single nth value for the row or a list of nth values.

dropna [None or str, optional] Apply the specified dropna operation before counting which row is the nth row. Needs to be None, 'any' or 'all'.

Returns

Series or DataFrame N-th value within each group.

See also:

Series.groupby
DataFrame.groupby

Examples

```
>>> df = pd.DataFrame({'A': [1, 1, 2, 1, 2],
...                    'B': [np.nan, 2, 3, 4, 5]}, columns=['A', 'B'])
>>> g = df.groupby('A')
>>> g.nth(0)
     B
A
1  NaN
2  3.0
>>> g.nth(1)
     B
A
1  2.0
2  5.0
>>> g.nth(-1)
     B
A
1  4.0
2  5.0
>>> g.nth([0, 1])
     B
A
1  NaN
1  2.0
2  3.0
2  5.0
```

Specifying *dropna* allows count ignoring `NaN`

```
>>> g.nth(0, dropna='any')
     B
A
1  2.0
2  3.0
```

NaNs denote group exhausted when using dropna

```
>>> g.nth(3, dropna='any')
    B
A
1 NaN
2 NaN
```

Specifying *as_index=False* in *groupby* keeps the original index.

```
>>> df.groupby('A', as_index=False).nth(1)
   A    B
1  1  2.0
4  2  5.0
```

pandas.core.groupby.GroupBy.ohlc

GroupBy.**ohlc**(*self*) → pandas.core.frame.DataFrame

Compute sum of values, excluding missing values.

For multiple groupings, the result index will be a MultiIndex

Returns

DataFrame Open, high, low and close values within each group.

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.prod

GroupBy.**prod**(*self*, ***kwargs*)

Compute prod of group values.

Returns

Series or DataFrame Computed prod of values within each group.

pandas.core.groupby.GroupBy.rank

GroupBy.**rank**(*self*, *method: str = 'average'*, *ascending: bool = True*, *na_option: str = 'keep'*, *pct: bool = False*, *axis: int = 0*)

Provide the rank of values within each group.

Parameters

method [{'average', 'min', 'max', 'first', 'dense'}, default 'average']

- average: average rank of group.
- min: lowest rank in group.
- max: highest rank in group.
- first: ranks assigned in order they appear in the array.
- dense: like 'min', but rank always increases by 1 between groups.

ascending [bool, default True] False for ranks by high (1) to low (N).

na_option [{'keep', 'top', 'bottom'}, default 'keep']

- keep: leave NA values where they are.
- top: smallest rank if ascending.
- bottom: smallest rank if descending.

pct [bool, default False] Compute percentage rank of data within each group.

axis [int, default 0] The axis of the object over which to compute the rank.

Returns

DataFrame with ranking of values within each group

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.pct_change

GroupBy.**pct_change**(*self*, *periods=1*, *fill_method='pad'*, *limit=None*, *freq=None*, *axis=0*)

Calculate pct_change of each value to previous entry in group.

Returns

Series or DataFrame Percentage changes within each group.

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.size

GroupBy.**size**(*self*)

Compute group sizes.

Returns

Series Number of rows in each group.

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.sem

GroupBy.**sem**(*self*, *ddof: int = 1*)

Compute standard error of the mean of groups, excluding missing values.

For multiple groupings, the result index will be a MultiIndex.

Parameters

ddof [int, default 1] Degrees of freedom.

Returns

Series or DataFrame Standard error of the mean of values within each group.

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.std

GroupBy.**std**(*self*, *ddof: int = 1*, **args*, ***kwargs*)

Compute standard deviation of groups, excluding missing values.

For multiple groupings, the result index will be a MultiIndex.

Parameters

ddof [int, default 1] Degrees of freedom.

Returns

Series or DataFrame Standard deviation of values within each group.

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.sum

GroupBy.**sum**(*self*, ***kwargs*)

Compute sum of group values.

Returns

Series or DataFrame Computed sum of values within each group.

pandas.core.groupby.GroupBy.var

GroupBy.**var**(*self*, *ddof: int = 1*, **args*, ***kwargs*)

Compute variance of groups, excluding missing values.

For multiple groupings, the result index will be a MultiIndex.

Parameters

ddof [int, default 1] Degrees of freedom.

Returns

Series or DataFrame Variance of values within each group.

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.GroupBy.tail

GroupBy.**tail**(*self*, *n=5*)

Return last n rows of each group.

Similar to `.apply(lambda x: x.tail(n))`, but it returns a subset of rows from the original DataFrame with original index and order preserved (`as_index` flag is ignored).

Does not work for negative values of *n*.

Returns

Series or DataFrame

See also:

Series.groupby
DataFrame.groupby

Examples

```
>>> df = pd.DataFrame([['a', 1], ['a', 2], ['b', 1], ['b', 2]],
...                   columns=['A', 'B'])
>>> df.groupby('A').tail(1)
   A  B
1  a  2
3  b  2
>>> df.groupby('A').tail(-1)
Empty DataFrame
Columns: [A, B]
Index: []
```

The following methods are available in both `SeriesGroupBy` and `DataFrameGroupBy` objects, but may differ slightly, usually in that the `DataFrameGroupBy` version usually permits the specification of an axis argument, and often an argument indicating whether to restrict application to columns of a specific data type.

DataFrameGroupBy.all(self, skipna)	Return True if all values in the group are truthful, else False.
DataFrameGroupBy.any(self, skipna)	Return True if any value in the group is truthful, else False.
DataFrameGroupBy.bfill(self[, limit])	Backward fill the values.
DataFrameGroupBy.corr	Compute pairwise correlation of columns, excluding NA/null values.
DataFrameGroupBy.count(self)	Compute count of group, excluding missing values.
DataFrameGroupBy.cov	Compute pairwise covariance of columns, excluding NA/null values.
DataFrameGroupBy.cummax(self[, axis])	Cumulative max for each group.
DataFrameGroupBy.cummin(self[, axis])	Cumulative min for each group.
DataFrameGroupBy.cumprod(self[, axis])	Cumulative product for each group.
DataFrameGroupBy.cumsum(self[, axis])	Cumulative sum for each group.
DataFrameGroupBy.describe(self, **kwargs)	Generate descriptive statistics.
DataFrameGroupBy.diff	First discrete difference of element.
DataFrameGroupBy.ffill(self[, limit])	Forward fill the values.
DataFrameGroupBy.fillna	Fill NA/NaN values using the specified method.
DataFrameGroupBy.filter(self, func[, dropna])	Return a copy of a DataFrame excluding elements from groups that do not satisfy the boolean criterion specified by func.
DataFrameGroupBy.hist	Make a histogram of the DataFrame's.
DataFrameGroupBy.idxmax	Return index of first occurrence of maximum over requested axis.
DataFrameGroupBy.idxmin	Return index of first occurrence of minimum over requested axis.
DataFrameGroupBy.mad	Return the mean absolute deviation of the values for the requested axis.
DataFrameGroupBy.nunique(self, dropna)	Return DataFrame with number of distinct observations per group for each column.
DataFrameGroupBy.pct_change(self[, periods, ...])	Calculate pct_change of each value to previous entry in group.
DataFrameGroupBy.plot	Class implementing the .plot attribute for groupby objects.
DataFrameGroupBy.quantile(self[, q])	Return group values at the given quantile, a la numpy.percentile.
DataFrameGroupBy.rank(self, method, ...)	Provide the rank of values within each group.
DataFrameGroupBy.resample(self, rule, *args, ...)	Provide resampling when using a TimeGrouper.
DataFrameGroupBy.shift(self[, periods, ...])	Shift each group by periods observations.
DataFrameGroupBy.size(self)	Compute group sizes.
DataFrameGroupBy.skew	Return unbiased skew over requested axis.
DataFrameGroupBy.take	Return the elements in the given *positional* indices along an axis.
DataFrameGroupBy.tshift	Shift the time index, using the index's frequency if available.

pandas.core.groupby.DataFrameGroupBy.all

DataFrameGroupBy.**all**(*self*, *skipna: bool = True*)

Return True if all values in the group are truthful, else False.

Parameters

skipna [bool, default True] Flag to ignore nan values during truth testing.

Returns

bool

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.DataFrameGroupBy.any

DataFrameGroupBy.**any**(*self*, *skipna: bool = True*)

Return True if any value in the group is truthful, else False.

Parameters

skipna [bool, default True] Flag to ignore nan values during truth testing.

Returns

bool

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.DataFrameGroupBy.bfill

DataFrameGroupBy.**bfill**(*self*, *limit=None*)

Backward fill the values.

Parameters

limit [int, optional] Limit of how many values to fill.

Returns

Series or DataFrame Object with missing values filled.

See also:

Series.backfill
DataFrame.backfill
Series.fillna
DataFrame.fillna

pandas.core.groupby.DataFrameGroupBy.corr

property DataFrameGroupBy.**corr**

Compute pairwise correlation of columns, excluding NA/null values.

Parameters

method [{'pearson', 'kendall', 'spearman'} or callable] Method of correlation:

- pearson : standard correlation coefficient
- kendall : Kendall Tau correlation coefficient
- spearman : Spearman rank correlation
- **callable: callable with input two 1d ndarrays** and returning a float. Note that the returned matrix from corr will have 1 along the diagonals and will be symmetric regardless of the callable's behavior.

  New in version 0.24.0.

min_periods [int, optional] Minimum number of observations required per pair of columns to have a valid result. Currently only available for Pearson and Spearman correlation.

Returns

DataFrame Correlation matrix.

See also:

DataFrame.corrwith
Series.corr

Examples

```
>>> def histogram_intersection(a, b):
...     v = np.minimum(a, b).sum().round(decimals=1)
...     return v
>>> df = pd.DataFrame([(.2, .3), (.0, .6), (.6, .0), (.2, .1)],
...                   columns=['dogs', 'cats'])
>>> df.corr(method=histogram_intersection)
      dogs  cats
dogs   1.0   0.3
cats   0.3   1.0
```

pandas.core.groupby.DataFrameGroupBy.count

DataFrameGroupBy.**count**(*self*)

Compute count of group, excluding missing values.

Returns

DataFrame Count of values within each group.

pandas.core.groupby.DataFrameGroupBy.cov

property DataFrameGroupBy.**cov**

Compute pairwise covariance of columns, excluding NA/null values.

Compute the pairwise covariance among the series of a DataFrame. The returned data frame is the covariance matrix of the columns of the DataFrame.

Both NA and null values are automatically excluded from the calculation. (See the note below about bias from missing values.) A threshold can be set for the minimum number of observations for each value created. Comparisons with observations below this threshold will be returned as `NaN`.

This method is generally used for the analysis of time series data to understand the relationship between different measures across time.

Parameters

min_periods [int, optional] Minimum number of observations required per pair of columns to have a valid result.

Returns

DataFrame The covariance matrix of the series of the DataFrame.

See also:

Series.cov Compute covariance with another Series.
core.window.EWM.cov Exponential weighted sample covariance.
core.window.Expanding.cov Expanding sample covariance.
core.window.Rolling.cov Rolling sample covariance.

Notes

Returns the covariance matrix of the DataFrame's time series. The covariance is normalized by N-1.

For DataFrames that have Series that are missing data (assuming that data is missing at random) the returned covariance matrix will be an unbiased estimate of the variance and covariance between the member Series.

However, for many applications this estimate may not be acceptable because the estimate covariance matrix is not guaranteed to be positive semi-definite. This could lead to estimate correlations having absolute values which are greater than one, and/or a non-invertible covariance matrix. See Estimation of covariance matrices for more details.

Examples

```
>>> df = pd.DataFrame([(1, 2), (0, 3), (2, 0), (1, 1)],
...                   columns=['dogs', 'cats'])
>>> df.cov()
          dogs      cats
dogs  0.666667 -1.000000
cats -1.000000  1.666667
```

```
>>> np.random.seed(42)
>>> df = pd.DataFrame(np.random.randn(1000, 5),
...                   columns=['a', 'b', 'c', 'd', 'e'])
>>> df.cov()
          a         b         c         d         e
a  0.998438 -0.020161  0.059277 -0.008943  0.014144
b -0.020161  1.059352 -0.008543 -0.024738  0.009826
```

(continues on next page)

(continued from previous page)

```
c  0.059277 -0.008543  1.010670 -0.001486 -0.000271
d -0.008943 -0.024738 -0.001486  0.921297 -0.013692
e  0.014144  0.009826 -0.000271 -0.013692  0.977795
```

Minimum number of periods

This method also supports an optional `min_periods` keyword that specifies the required minimum number of non-NA observations for each column pair in order to have a valid result:

```
>>> np.random.seed(42)
>>> df = pd.DataFrame(np.random.randn(20, 3),
...                   columns=['a', 'b', 'c'])
>>> df.loc[df.index[:5], 'a'] = np.nan
>>> df.loc[df.index[5:10], 'b'] = np.nan
>>> df.cov(min_periods=12)
          a         b         c
a  0.316741       NaN -0.150812
b       NaN  1.248003  0.191417
c -0.150812  0.191417  0.895202
```

pandas.core.groupby.DataFrameGroupBy.cummax

DataFrameGroupBy.**cummax**(*self*, *axis=0*, ***kwargs*)

Cumulative max for each group.

Returns

Series or DataFrame

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.DataFrameGroupBy.cummin

DataFrameGroupBy.**cummin**(*self*, *axis=0*, ***kwargs*)

Cumulative min for each group.

Returns

Series or DataFrame

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.DataFrameGroupBy.cumprod

DataFrameGroupBy.**cumprod**(*self*, *axis=0*, **args*, ***kwargs*)

Cumulative product for each group.

Returns

Series or DataFrame

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.DataFrameGroupBy.cumsum

DataFrameGroupBy.**cumsum**(*self*, *axis=0*, **args*, ***kwargs*)

Cumulative sum for each group.

Returns

Series or DataFrame

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.DataFrameGroupBy.describe

DataFrameGroupBy.**describe**(*self*, ***kwargs*)

Generate descriptive statistics.

Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding `NaN` values.

Analyzes both numeric and object series, as well as `DataFrame` column sets of mixed data types. The output will vary depending on what is provided. Refer to the notes below for more detail.

Parameters

percentiles [list-like of numbers, optional] The percentiles to include in the output. All should fall between 0 and 1. The default is `[.25, .5, .75]`, which returns the 25th, 50th, and 75th percentiles.

include ['all', list-like of dtypes or None (default), optional] A white list of data types to include in the result. Ignored for `Series`. Here are the options:

- 'all' : All columns of the input will be included in the output.
- A list-like of dtypes : Limits the results to the provided data types. To limit the result to numeric types submit `numpy.number`. To limit it instead to object columns submit the `numpy.object` data type. Strings can also be used in the style of `select_dtypes` (e.g. `df.describe(include=['O'])`). To select pandas categorical columns, use `'category'`
- None (default) : The result will include all numeric columns.

exclude [list-like of dtypes or None (default), optional,] A black list of data types to omit from the result. Ignored for `Series`. Here are the options:

- A list-like of dtypes : Excludes the provided data types from the result. To exclude numeric types submit `numpy.number`. To exclude object columns submit the data type `numpy.object`. Strings can also be used in the style of `select_dtypes` (e.g. `df.describe(include=['O'])`). To exclude pandas categorical columns, use `'category'`
- None (default) : The result will exclude nothing.

Returns

Series or DataFrame Summary statistics of the Series or Dataframe provided.

See also:

DataFrame.count Count number of non-NA/null observations.
DataFrame.max Maximum of the values in the object.
DataFrame.min Minimum of the values in the object.
DataFrame.mean Mean of the values.

DataFrame.std Standard deviation of the observations.
DataFrame.select_dtypes Subset of a DataFrame including/excluding columns based on their dtype.

Notes

For numeric data, the result's index will include `count`, `mean`, `std`, `min`, `max` as well as lower, `50` and upper percentiles. By default the lower percentile is `25` and the upper percentile is `75`. The `50` percentile is the same as the median.

For object data (e.g. strings or timestamps), the result's index will include `count`, `unique`, `top`, and `freq`. The `top` is the most common value. The `freq` is the most common value's frequency. Timestamps also include the `first` and `last` items.

If multiple object values have the highest count, then the `count` and `top` results will be arbitrarily chosen from among those with the highest count.

For mixed data types provided via a `DataFrame`, the default is to return only an analysis of numeric columns. If the dataframe consists only of object and categorical data without any numeric columns, the default is to return an analysis of both the object and categorical columns. If `include='all'` is provided as an option, the result will include a union of attributes of each type.

The *include* and *exclude* parameters can be used to limit which columns in a `DataFrame` are analyzed for the output. The parameters are ignored when analyzing a `Series`.

Examples

Describing a numeric `Series`.

```
>>> s = pd.Series([1, 2, 3])
>>> s.describe()
count    3.0
mean     2.0
std      1.0
min      1.0
25%      1.5
50%      2.0
75%      2.5
max      3.0
dtype: float64
```

Describing a categorical `Series`.

```
>>> s = pd.Series(['a', 'a', 'b', 'c'])
>>> s.describe()
count     4
unique    3
top       a
freq      2
dtype: object
```

Describing a timestamp `Series`.

```
>>> s = pd.Series([
...   np.datetime64("2000-01-01"),
...   np.datetime64("2010-01-01"),
...   np.datetime64("2010-01-01")
```

(continues on next page)

(continued from previous page)

```
... ])
>>> s.describe()
count                         3
unique                        2
top       2010-01-01 00:00:00
freq                          2
first     2000-01-01 00:00:00
last      2010-01-01 00:00:00
dtype: object
```

Describing a `DataFrame`. By default only numeric fields are returned.

```
>>> df = pd.DataFrame({'categorical': pd.Categorical(['d','e','f']),
...                    'numeric': [1, 2, 3],
...                    'object': ['a', 'b', 'c']
...                   })
>>> df.describe()
       numeric
count      3.0
mean       2.0
std        1.0
min        1.0
25%        1.5
50%        2.0
75%        2.5
max        3.0
```

Describing all columns of a `DataFrame` regardless of data type.

```
>>> df.describe(include='all')
        categorical  numeric object
count             3      3.0      3
unique            3      NaN      3
top               f      NaN      c
freq              1      NaN      1
mean            NaN      2.0    NaN
std             NaN      1.0    NaN
min             NaN      1.0    NaN
25%             NaN      1.5    NaN
50%             NaN      2.0    NaN
75%             NaN      2.5    NaN
max             NaN      3.0    NaN
```

Describing a column from a `DataFrame` by accessing it as an attribute.

```
>>> df.numeric.describe()
count    3.0
mean     2.0
std      1.0
min      1.0
25%      1.5
50%      2.0
75%      2.5
max      3.0
Name: numeric, dtype: float64
```

Including only numeric columns in a `DataFrame` description.

```
>>> df.describe(include=[np.number])
       numeric
count      3.0
mean       2.0
std        1.0
min        1.0
25%        1.5
50%        2.0
75%        2.5
max        3.0
```

Including only string columns in a `DataFrame` description.

```
>>> df.describe(include=[np.object])
       object
count       3
unique      3
top         c
freq        1
```

Including only categorical columns from a `DataFrame` description.

```
>>> df.describe(include=['category'])
       categorical
count            3
unique           3
top              f
freq             1
```

Excluding numeric columns from a `DataFrame` description.

```
>>> df.describe(exclude=[np.number])
       categorical object
count            3      3
unique           3      3
top              f      c
freq             1      1
```

Excluding object columns from a `DataFrame` description.

```
>>> df.describe(exclude=[np.object])
       categorical  numeric
count            3      3.0
unique           3      NaN
top              f      NaN
freq             1      NaN
mean           NaN      2.0
std            NaN      1.0
min            NaN      1.0
25%            NaN      1.5
50%            NaN      2.0
75%            NaN      2.5
max            NaN      3.0
```

pandas.core.groupby.DataFrameGroupBy.diff

property DataFrameGroupBy.**diff**

First discrete difference of element.

Calculates the difference of a DataFrame element compared with another element in the DataFrame (default is the element in the same column of the previous row).

Parameters

periods [int, default 1] Periods to shift for calculating difference, accepts negative values.

axis [{0 or 'index', 1 or 'columns'}, default 0] Take difference over rows (0) or columns (1).

Returns

DataFrame

See also:

Series.diff First discrete difference for a Series.
DataFrame.pct_change Percent change over given number of periods.
DataFrame.shift Shift index by desired number of periods with an optional time freq.

Notes

For boolean dtypes, this uses `operator.xor()` rather than `operator.sub()`.

Examples

Difference with previous row

```
>>> df = pd.DataFrame({'a': [1, 2, 3, 4, 5, 6],
...                    'b': [1, 1, 2, 3, 5, 8],
...                    'c': [1, 4, 9, 16, 25, 36]})
>>> df
   a  b   c
0  1  1   1
1  2  1   4
2  3  2   9
3  4  3  16
4  5  5  25
5  6  8  36
```

```
>>> df.diff()
     a    b     c
0  NaN  NaN   NaN
1  1.0  0.0   3.0
2  1.0  1.0   5.0
3  1.0  1.0   7.0
4  1.0  2.0   9.0
5  1.0  3.0  11.0
```

Difference with previous column

```
>>> df.diff(axis=1)
    a    b     c
0 NaN  0.0   0.0
1 NaN -1.0   3.0
```

(continues on next page)