(continued from previous page)

```
2 NaN -1.0   7.0
3 NaN -1.0  13.0
4 NaN  0.0  20.0
5 NaN  2.0  28.0
```

Difference with 3rd previous row

```
>>> df.diff(periods=3)
     a    b     c
0  NaN  NaN   NaN
1  NaN  NaN   NaN
2  NaN  NaN   NaN
3  3.0  2.0  15.0
4  3.0  4.0  21.0
5  3.0  6.0  27.0
```

Difference with following row

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

pandas.core.groupby.DataFrameGroupBy.ffill

DataFrameGroupBy.**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.DataFrameGroupBy.fillna

property DataFrameGroupBy.**fillna**

Fill NA/NaN values using the specified method.

Parameters

value [scalar, dict, Series, or DataFrame] Value to use to fill holes (e.g. 0), alternately a dict/Series/DataFrame of values specifying which value to use for each index (for a Series) or column (for a DataFrame). Values not in the dict/Series/DataFrame will not be filled. This value cannot be a list.

method [{'backfill', 'bfill', 'pad', 'ffill', None}, default None] Method to use for filling holes in reindexed Series pad / ffill: propagate last valid observation forward to next valid backfill / bfill: use next valid observation to fill gap.

axis [{0 or 'index', 1 or 'columns'}] Axis along which to fill missing values.

inplace [bool, default False] If True, fill in-place. Note: this will modify any other views on this object (e.g., a no-copy slice for a column in a DataFrame).

limit [int, default None] If method is specified, this is the maximum number of consecutive NaN values to forward/backward fill. In other words, if there is a gap with more than this number of consecutive NaNs, it will only be partially filled. If method is not specified, this is the maximum number of entries along the entire axis where NaNs will be filled. Must be greater than 0 if not None.

downcast [dict, default is None] A dict of item->dtype of what to downcast if possible, or the string 'infer' which will try to downcast to an appropriate equal type (e.g. float64 to int64 if possible).

Returns

DataFrame or None Object with missing values filled or None if `inplace=True`.

See also:

`interpolate` Fill NaN values using interpolation.
`reindex` Conform object to new index.
`asfreq` Convert TimeSeries to specified frequency.

Examples

```
>>> df = pd.DataFrame([[np.nan, 2, np.nan, 0],
...                    [3, 4, np.nan, 1],
...                    [np.nan, np.nan, np.nan, 5],
...                    [np.nan, 3, np.nan, 4]],
...                   columns=list('ABCD'))
>>> df
     A    B   C  D
0  NaN  2.0 NaN  0
1  3.0  4.0 NaN  1
2  NaN  NaN NaN  5
3  NaN  3.0 NaN  4
```

Replace all NaN elements with 0s.

```
>>> df.fillna(0)
    A   B   C   D
0   0.0 2.0 0.0 0
1   3.0 4.0 0.0 1
2   0.0 0.0 0.0 5
3   0.0 3.0 0.0 4
```

We can also propagate non-null values forward or backward.

```
>>> df.fillna(method='ffill')
    A   B   C   D
0   NaN 2.0 NaN 0
1   3.0 4.0 NaN 1
```

(continues on next page)

(continued from previous page)

```
2   3.0 4.0 NaN 5
3   3.0 3.0 NaN 4
```

Replace all NaN elements in column 'A', 'B', 'C', and 'D', with 0, 1, 2, and 3 respectively.

```
>>> values = {'A': 0, 'B': 1, 'C': 2, 'D': 3}
>>> df.fillna(value=values)
    A   B   C   D
0   0.0 2.0 2.0 0
1   3.0 4.0 2.0 1
2   0.0 1.0 2.0 5
3   0.0 3.0 2.0 4
```

Only replace the first NaN element.

```
>>> df.fillna(value=values, limit=1)
    A   B   C   D
0   0.0 2.0 2.0 0
1   3.0 4.0 NaN 1
2   NaN 1.0 NaN 5
3   NaN 3.0 NaN 4
```

pandas.core.groupby.DataFrameGroupBy.filter

DataFrameGroupBy.**filter**(*self*, *func*, *dropna=True*, **args*, ***kwargs*)

Return a copy of a DataFrame excluding elements from groups that do not satisfy the boolean criterion specified by func.

Parameters

f [function] Function to apply to each subframe. Should return True or False.

dropna [Drop groups that do not pass the filter. True by default;] If False, groups that evaluate False are filled with NaNs.

Returns

filtered [DataFrame]

Notes

Each subframe is endowed the attribute 'name' in case you need to know which group you are working on.

Examples

```
>>> df = pd.DataFrame({'A' : ['foo', 'bar', 'foo', 'bar',
...                           'foo', 'bar'],
...                    'B' : [1, 2, 3, 4, 5, 6],
...                    'C' : [2.0, 5., 8., 1., 2., 9.]})
>>> grouped = df.groupby('A')
>>> grouped.filter(lambda x: x['B'].mean() > 3.)
     A  B    C
1  bar  2  5.0
3  bar  4  1.0
5  bar  6  9.0
```

pandas.core.groupby.DataFrameGroupBy.hist

property DataFrameGroupBy.**hist**

Make a histogram of the DataFrame's.

A histogram is a representation of the distribution of data. This function calls `matplotlib.pyplot.hist()`, on each series in the DataFrame, resulting in one histogram per column.

Parameters

data [DataFrame] The pandas object holding the data.

column [str or sequence] If passed, will be used to limit data to a subset of columns.

by [object, optional] If passed, then used to form histograms for separate groups.

grid [bool, default True] Whether to show axis grid lines.

xlabelsize [int, default None] If specified changes the x-axis label size.

xrot [float, default None] Rotation of x axis labels. For example, a value of 90 displays the x labels rotated 90 degrees clockwise.

ylabelsize [int, default None] If specified changes the y-axis label size.

yrot [float, default None] Rotation of y axis labels. For example, a value of 90 displays the y labels rotated 90 degrees clockwise.

ax [Matplotlib axes object, default None] The axes to plot the histogram on.

sharex [bool, default True if ax is None else False] In case subplots=True, share x axis and set some x axis labels to invisible; defaults to True if ax is None otherwise False if an ax is passed in. Note that passing in both an ax and sharex=True will alter all x axis labels for all subplots in a figure.

sharey [bool, default False] In case subplots=True, share y axis and set some y axis labels to invisible.

figsize [tuple] The size in inches of the figure to create. Uses the value in *matplotlib.rcParams* by default.

layout [tuple, optional] Tuple of (rows, columns) for the layout of the histograms.

bins [int or sequence, default 10] Number of histogram bins to be used. If an integer is given, bins + 1 bin edges are calculated and returned. If bins is a sequence, gives bin edges, including left edge of first bin and right edge of last bin. In this case, bins is returned unmodified.

backend [str, default None] Backend to use instead of the backend specified in the option `plotting.backend`. For instance, 'matplotlib'. Alternatively, to specify the `plotting.backend` for the whole session, set `pd.options.plotting.backend`.

New in version 1.0.0.

****kwargs** All other plotting keyword arguments to be passed to `matplotlib.pyplot.hist()`.

Returns

matplotlib.AxesSubplot or numpy.ndarray of them

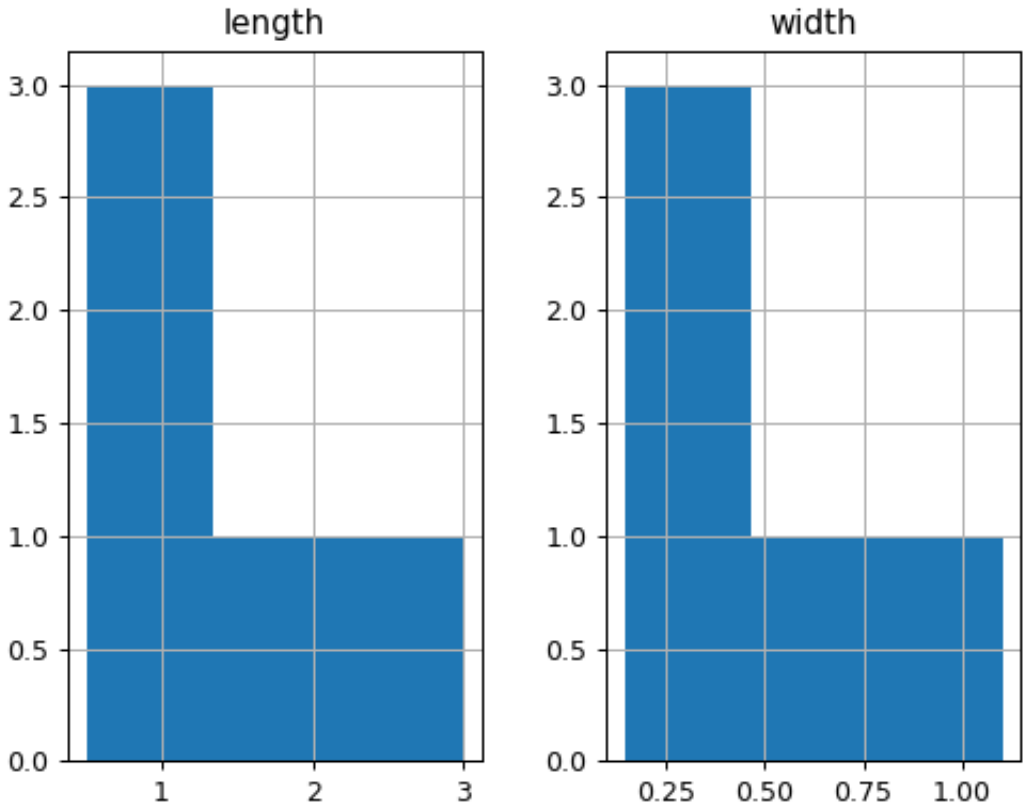
See also:

`matplotlib.pyplot.hist` Plot a histogram using matplotlib.

Examples

This example draws a histogram based on the length and width of some animals, displayed in three bins

```
>>> df = pd.DataFrame({
...     'length': [1.5, 0.5, 1.2, 0.9, 3],
...     'width': [0.7, 0.2, 0.15, 0.2, 1.1]
...     }, index=['pig', 'rabbit', 'duck', 'chicken', 'horse'])
>>> hist = df.hist(bins=3)
```



pandas.core.groupby.DataFrameGroupBy.idxmax

property DataFrameGroupBy.**idxmax**

Return index of first occurrence of maximum over requested axis.

NA/null values are excluded.

Parameters

axis [{0 or 'index', 1 or 'columns'}, default 0] The axis to use. 0 or 'index' for row-wise, 1 or 'columns' for column-wise.

skipna [bool, default True] Exclude NA/null values. If an entire row/column is NA, the result will be NA.

Returns

Series Indexes of maxima along the specified axis.

Raises

ValueError

- If the row/column is empty

See also:

Series.idxmax

Notes

This method is the DataFrame version of `ndarray.argmax`.

pandas.core.groupby.DataFrameGroupBy.idxmin

property DataFrameGroupBy.**idxmin**

Return index of first occurrence of minimum over requested axis.

NA/null values are excluded.

Parameters

axis [{0 or 'index', 1 or 'columns'}, default 0] The axis to use. 0 or 'index' for row-wise, 1 or 'columns' for column-wise.

skipna [bool, default True] Exclude NA/null values. If an entire row/column is NA, the result will be NA.

Returns

Series Indexes of minima along the specified axis.

Raises

ValueError

- If the row/column is empty

See also:

Series.idxmin

Notes

This method is the DataFrame version of `ndarray.argmin`.

pandas.core.groupby.DataFrameGroupBy.mad

property DataFrameGroupBy.**mad**

Return the mean absolute deviation of the values for the requested axis.

Parameters

axis [{index (0), columns (1)}] Axis for the function to be applied on.

skipna [bool, default True] Exclude NA/null values when computing the result.

level [int or level name, default None] If the axis is a MultiIndex (hierarchical), count along a particular level, collapsing into a Series.

numeric_only [bool, default None] Include only float, int, boolean columns. If None, will attempt to use everything, then use only numeric data. Not implemented for Series.

****kwargs** Additional keyword arguments to be passed to the function.

Returns

Series or DataFrame (if level specified)

pandas.core.groupby.DataFrameGroupBy.nunique

DataFrameGroupBy.**nunique**(*self*, *dropna: bool = True*)

Return DataFrame with number of distinct observations per group for each column.

Parameters

dropna [bool, default True] Don't include NaN in the counts.

Returns

nunique: DataFrame

Examples

```
>>> df = pd.DataFrame({'id': ['spam', 'egg', 'egg', 'spam',
...                           'ham', 'ham'],
...                    'value1': [1, 5, 5, 2, 5, 5],
...                    'value2': list('abbaxy')})
>>> df
     id  value1 value2
0  spam       1      a
1   egg       5      b
2   egg       5      b
3  spam       2      a
4   ham       5      x
5   ham       5      y
```

```
>>> df.groupby('id').nunique()
    id  value1  value2
id
egg    1       1       1
ham    1       1       2
spam   1       2       1
```

Check for rows with the same id but conflicting values:

```
>>> df.groupby('id').filter(lambda g: (g.nunique() > 1).any())
     id  value1 value2
0  spam       1      a
3  spam       2      a
4   ham       5      x
5   ham       5      y
```

pandas.core.groupby.DataFrameGroupBy.pct_change

DataFrameGroupBy.**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.DataFrameGroupBy.plot

property DataFrameGroupBy.**plot**

Class implementing the .plot attribute for groupby objects.

pandas.core.groupby.DataFrameGroupBy.quantile

DataFrameGroupBy.**quantile**(*self*, *q=0.5*, *interpolation: str = 'linear'*)

Return group values at the given quantile, a la numpy.percentile.

Parameters

q [float or array-like, default 0.5 (50% quantile)] Value(s) between 0 and 1 providing the quantile(s) to compute.

interpolation [{'linear', 'lower', 'higher', 'midpoint', 'nearest'}] Method to use when the desired quantile falls between two points.

Returns

Series or DataFrame Return type determined by caller of GroupBy object.

See also:

Series.quantile Similar method for Series.
DataFrame.quantile Similar method for DataFrame.
numpy.percentile NumPy method to compute qth percentile.

Examples

```
>>> df = pd.DataFrame([
...     ['a', 1], ['a', 2], ['a', 3],
...     ['b', 1], ['b', 3], ['b', 5]
... ], columns=['key', 'val'])
>>> df.groupby('key').quantile()
    val
key
a    2.0
b    3.0
```

pandas.core.groupby.DataFrameGroupBy.rank

DataFrameGroupBy.**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.DataFrameGroupBy.resample

DataFrameGroupBy.**resample**(*self*, *rule*, **args*, ***kwargs*)

Provide resampling when using a TimeGrouper.

Given a grouper, the function resamples it according to a string "string" -> "frequency".

See the *frequency aliases* documentation for more details.

Parameters

rule [str or DateOffset] The offset string or object representing target grouper conversion.

***args, **kwargs** Possible arguments are *how*, *fill_method*, *limit*, *kind* and *on*, and other arguments of *TimeGrouper*.

Returns

Grouper Return a new grouper with our resampler appended.

See also:

Grouper Specify a frequency to resample with when grouping by a key.
DatetimeIndex.resample Frequency conversion and resampling of time series.

Examples

```
>>> idx = pd.date_range('1/1/2000', periods=4, freq='T')
>>> df = pd.DataFrame(data=4 * [range(2)],
...                   index=idx,
...                   columns=['a', 'b'])
>>> df.iloc[2, 0] = 5
>>> df
                     a  b
2000-01-01 00:00:00  0  1
2000-01-01 00:01:00  0  1
2000-01-01 00:02:00  5  1
2000-01-01 00:03:00  0  1
```

Downsample the DataFrame into 3 minute bins and sum the values of the timestamps falling into a bin.

```
>>> df.groupby('a').resample('3T').sum()
                          a  b
a
0   2000-01-01 00:00:00  0  2
    2000-01-01 00:03:00  0  1
5   2000-01-01 00:00:00  5  1
```

Upsample the series into 30 second bins.

```
>>> df.groupby('a').resample('30S').sum()
                     a  b
a
0   2000-01-01 00:00:00  0  1
    2000-01-01 00:00:30  0  0
    2000-01-01 00:01:00  0  1
    2000-01-01 00:01:30  0  0
    2000-01-01 00:02:00  0  0
    2000-01-01 00:02:30  0  0
    2000-01-01 00:03:00  0  1
5   2000-01-01 00:02:00  5  1
```

Resample by month. Values are assigned to the month of the period.

```
>>> df.groupby('a').resample('M').sum()
            a  b
a
0   2000-01-31  0  3
5   2000-01-31  5  1
```

Downsample the series into 3 minute bins as above, but close the right side of the bin interval.

```
>>> df.groupby('a').resample('3T', closed='right').sum()
                          a  b
a
0   1999-12-31 23:57:00  0  1
    2000-01-01 00:00:00  0  2
5   2000-01-01 00:00:00  5  1
```

Downsample the series into 3 minute bins and close the right side of the bin interval, but label each bin using the right edge instead of the left.

```
>>> df.groupby('a').resample('3T', closed='right', label='right').sum()
                         a  b
a
0   2000-01-01 00:00:00  0  1
    2000-01-01 00:03:00  0  2
5   2000-01-01 00:03:00  5  1
```

Add an offset of twenty seconds.

```
>>> df.groupby('a').resample('3T', loffset='20s').sum()
                        a  b
a
0   2000-01-01 00:00:20  0  2
    2000-01-01 00:03:20  0  1
5   2000-01-01 00:00:20  5  1
```

pandas.core.groupby.DataFrameGroupBy.shift

DataFrameGroupBy.**shift**(*self*, *periods=1*, *freq=None*, *axis=0*, *fill_value=None*)

Shift each group by periods observations.

Parameters

periods [int, default 1] Number of periods to shift.

freq [frequency string]

axis [axis to shift, default 0]

fill_value [optional] New in version 0.24.0.

Returns

Series or DataFrame Object shifted within each group.

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.DataFrameGroupBy.size

DataFrameGroupBy.**size**(*self*)

Compute group sizes.

Returns

Series Number of rows in each group.

See also:

Series.groupby
DataFrame.groupby

pandas.core.groupby.DataFrameGroupBy.skew

property DataFrameGroupBy.**skew**

Return unbiased skew over requested axis.

Normalized by N-1.

Parameters

axis [{index (0), columns (1)}] Axis for the function to be applied on.

skipna [bool, default True] Exclude NA/null values when computing the result.

level [int or level name, default None] If the axis is a MultiIndex (hierarchical), count along a particular level, collapsing into a Series.

numeric_only [bool, default None] Include only float, int, boolean columns. If None, will attempt to use everything, then use only numeric data. Not implemented for Series.

****kwargs** Additional keyword arguments to be passed to the function.

Returns

Series or DataFrame (if level specified)

pandas.core.groupby.DataFrameGroupBy.take

property DataFrameGroupBy.**take**

Return the elements in the given *positional* indices along an axis.

This means that we are not indexing according to actual values in the index attribute of the object. We are indexing according to the actual position of the element in the object.

Parameters

indices [array-like] An array of ints indicating which positions to take.

axis [{0 or 'index', 1 or 'columns', None}, default 0] The axis on which to select elements. `0` means that we are selecting rows, `1` means that we are selecting columns.

is_copy [bool] Before pandas 1.0, `is_copy=False` can be specified to ensure that the return value is an actual copy. Starting with pandas 1.0, `take` always returns a copy, and the keyword is therefore deprecated.

Deprecated since version 1.0.0.

****kwargs** For compatibility with `numpy.take()`. Has no effect on the output.

Returns

taken [same type as caller] An array-like containing the elements taken from the object.

See also:

DataFrame.loc Select a subset of a DataFrame by labels.
DataFrame.iloc Select a subset of a DataFrame by positions.
numpy.take Take elements from an array along an axis.

Examples

```
>>> df = pd.DataFrame([('falcon', 'bird', 389.0),
...                    ('parrot', 'bird', 24.0),
...                    ('lion', 'mammal', 80.5),
...                    ('monkey', 'mammal', np.nan)],
...                   columns=['name', 'class', 'max_speed'],
...                   index=[0, 2, 3, 1])
>>> df
     name   class  max_speed
0  falcon    bird      389.0
2  parrot    bird       24.0
3    lion  mammal       80.5
1  monkey  mammal        NaN
```

Take elements at positions 0 and 3 along the axis 0 (default).

Note how the actual indices selected (0 and 1) do not correspond to our selected indices 0 and 3. That's because we are selecting the 0th and 3rd rows, not rows whose indices equal 0 and 3.

```
>>> df.take([0, 3])
     name   class  max_speed
0  falcon    bird      389.0
1  monkey  mammal        NaN
```

Take elements at indices 1 and 2 along the axis 1 (column selection).

```
>>> df.take([1, 2], axis=1)
    class  max_speed
0    bird      389.0
2    bird       24.0
3  mammal       80.5
1  mammal        NaN
```

We may take elements using negative integers for positive indices, starting from the end of the object, just like with Python lists.

```
>>> df.take([-1, -2])
     name   class  max_speed
1  monkey  mammal        NaN
3    lion  mammal       80.5
```

pandas.core.groupby.DataFrameGroupBy.tshift

property DataFrameGroupBy.**tshift**

Shift the time index, using the index's frequency if available.

Parameters

periods [int] Number of periods to move, can be positive or negative.

freq [DateOffset, timedelta, or str, default None] Increment to use from the tseries module or time rule expressed as a string (e.g. 'EOM').

axis [{0 or 'index', 1 or 'columns', None}, default 0] Corresponds to the axis that contains the Index.

Returns

shifted [Series/DataFrame]

Notes

If freq is not specified then tries to use the freq or inferred_freq attributes of the index. If neither of those attributes exist, a ValueError is thrown

The following methods are available only for `SeriesGroupBy` objects.

SeriesGroupBy.nlargest	Return the largest *n* elements.
SeriesGroupBy.nsmallest	Return the smallest *n* elements.
SeriesGroupBy.nunique(self, dropna)	Return number of unique elements in the group.
SeriesGroupBy.unique	Return unique values of Series object.
SeriesGroupBy.value_counts(self[, ...])	
SeriesGroupBy.is_monotonic_increasing	Return boolean if values in the object are monotonic_increasing.
SeriesGroupBy.is_monotonic_decreasing	Return boolean if values in the object are monotonic_decreasing.

pandas.core.groupby.SeriesGroupBy.nlargest

property SeriesGroupBy.**nlargest**

Return the largest *n* elements.

Parameters

n [int, default 5] Return this many descending sorted values.

keep [{'first', 'last', 'all'}, default 'first'] When there are duplicate values that cannot all fit in a Series of *n* elements:

- **first** [return the first *n* occurrences in order] of appearance.
- **last** [return the last *n* occurrences in reverse] order of appearance.
- **all** [keep all occurrences. This can result in a Series of] size larger than *n*.

Returns

Series The *n* largest values in the Series, sorted in decreasing order.

See also:

Series.nsmallest Get the *n* smallest elements.
Series.sort_values Sort Series by values.
Series.head Return the first *n* rows.

Notes

Faster than `.sort_values(ascending=False).head(n)` for small *n* relative to the size of the `Series` object.

Examples

```
>>> countries_population = {"Italy": 59000000, "France": 65000000,
...                         "Malta": 434000, "Maldives": 434000,
...                         "Brunei": 434000, "Iceland": 337000,
...                         "Nauru": 11300, "Tuvalu": 11300,
...                         "Anguilla": 11300, "Monserat": 5200}
>>> s = pd.Series(countries_population)
>>> s
Italy       59000000
France      65000000
Malta         434000
Maldives      434000
Brunei        434000
Iceland       337000
Nauru          11300
Tuvalu         11300
Anguilla       11300
Monserat        5200
dtype: int64
```

The *n* largest elements where `n=5` by default.

```
>>> s.nlargest()
France      65000000
Italy       59000000
Malta         434000
Maldives      434000
Brunei        434000
dtype: int64
```

The *n* largest elements where `n=3`. Default *keep* value is 'first' so Malta will be kept.

```
>>> s.nlargest(3)
France    65000000
Italy     59000000
Malta       434000
dtype: int64
```

The *n* largest elements where `n=3` and keeping the last duplicates. Brunei will be kept since it is the last with value 434000 based on the index order.

```
>>> s.nlargest(3, keep='last')
France      65000000
Italy       59000000
Brunei        434000
dtype: int64
```

The *n* largest elements where `n=3` with all duplicates kept. Note that the returned Series has five elements due to the three duplicates.

```
>>> s.nlargest(3, keep='all')
France      65000000
Italy       59000000
Malta         434000
Maldives      434000
Brunei        434000
dtype: int64
```

pandas.core.groupby.SeriesGroupBy.nsmallest

property SeriesGroupBy.**nsmallest**

Return the smallest *n* elements.

Parameters

n [int, default 5] Return this many ascending sorted values.

keep [{'first', 'last', 'all'}, default 'first'] When there are duplicate values that cannot all fit in a Series of *n* elements:

- **first** [return the first *n* occurrences in order] of appearance.
- **last** [return the last *n* occurrences in reverse] order of appearance.
- **all** [keep all occurrences. This can result in a Series of] size larger than *n*.

Returns

Series The *n* smallest values in the Series, sorted in increasing order.

See also:

Series.nlargest Get the *n* largest elements.
Series.sort_values Sort Series by values.
Series.head Return the first *n* rows.

Notes

Faster than `.sort_values().head(n)` for small *n* relative to the size of the `Series` object.

Examples

```
>>> countries_population = {"Italy": 59000000, "France": 65000000,
...                         "Brunei": 434000, "Malta": 434000,
...                         "Maldives": 434000, "Iceland": 337000,
...                         "Nauru": 11300, "Tuvalu": 11300,
...                         "Anguilla": 11300, "Monserat": 5200}
>>> s = pd.Series(countries_population)
>>> s
Italy       59000000
France      65000000
Brunei        434000
Malta         434000
Maldives      434000
Iceland       337000
Nauru          11300
Tuvalu         11300
Anguilla       11300
Monserat        5200
dtype: int64
```

The *n* smallest elements where `n=5` by default.

```
>>> s.nsmallest()
Monserat      5200
Nauru        11300
Tuvalu       11300
```

(continues on next page)

(continued from previous page)

```
Anguilla     11300
Iceland     337000
dtype: int64
```

The *n* smallest elements where `n=3`. Default *keep* value is 'first' so Nauru and Tuvalu will be kept.

```
>>> s.nsmallest(3)
Monserat     5200
Nauru       11300
Tuvalu      11300
dtype: int64
```

The *n* smallest elements where `n=3` and keeping the last duplicates. Anguilla and Tuvalu will be kept since they are the last with value 11300 based on the index order.

```
>>> s.nsmallest(3, keep='last')
Monserat     5200
Anguilla    11300
Tuvalu      11300
dtype: int64
```

The *n* smallest elements where `n=3` with all duplicates kept. Note that the returned Series has four elements due to the three duplicates.

```
>>> s.nsmallest(3, keep='all')
Monserat     5200
Nauru       11300
Tuvalu      11300
Anguilla    11300
dtype: int64
```

pandas.core.groupby.SeriesGroupBy.nunique

`SeriesGroupBy.`**`nunique`**(*self*, *dropna: bool = True*) → pandas.core.series.Series

Return number of unique elements in the group.

Returns

Series Number of unique values within each group.

pandas.core.groupby.SeriesGroupBy.unique

property `SeriesGroupBy.`**`unique`**

Return unique values of Series object.

Uniques are returned in order of appearance. Hash table-based unique, therefore does NOT sort.

Returns

ndarray or ExtensionArray The unique values returned as a NumPy array. See Notes.

See also:

`unique` Top-level unique method for any 1-d array-like object.
`Index.unique` Return Index with unique values from an Index object.

Notes

Returns the unique values as a NumPy array. In case of an extension-array backed Series, a new `ExtensionArray` of that type with just the unique values is returned. This includes

- Categorical
- Period
- Datetime with Timezone
- Interval
- Sparse
- IntegerNA

See Examples section.

Examples

```
>>> pd.Series([2, 1, 3, 3], name='A').unique()
array([2, 1, 3])
```

```
>>> pd.Series([pd.Timestamp('2016-01-01') for _ in range(3)]).unique()
array(['2016-01-01T00:00:00.000000000'], dtype='datetime64[ns]')
```

```
>>> pd.Series([pd.Timestamp('2016-01-01', tz='US/Eastern')
...            for _ in range(3)]).unique()
<DatetimeArray>
['2016-01-01 00:00:00-05:00']
Length: 1, dtype: datetime64[ns, US/Eastern]
```

An unordered Categorical will return categories in the order of appearance.

```
>>> pd.Series(pd.Categorical(list('baabc'))).unique()
[b, a, c]
Categories (3, object): [b, a, c]
```

An ordered Categorical preserves the category ordering.

```
>>> pd.Series(pd.Categorical(list('baabc'), categories=list('abc'),
...                          ordered=True)).unique()
[b, a, c]
Categories (3, object): [a < b < c]
```

pandas.core.groupby.SeriesGroupBy.value_counts

`SeriesGroupBy.`**`value_counts`**(*self*, *normalize=False*, *sort=True*, *ascending=False*, *bins=None*, *dropna=True*)

pandas.core.groupby.SeriesGroupBy.is_monotonic_increasing

property SeriesGroupBy.**is_monotonic_increasing**

Return boolean if values in the object are monotonic_increasing.

Returns

bool

pandas.core.groupby.SeriesGroupBy.is_monotonic_decreasing

property SeriesGroupBy.**is_monotonic_decreasing**

Return boolean if values in the object are monotonic_decreasing.

Returns

bool

The following methods are available only for DataFrameGroupBy objects.

DataFrameGroupBy.corrwith	Compute pairwise correlation.
DataFrameGroupBy.boxplot(grouped[, ...])	Make box plots from DataFrameGroupBy data.

pandas.core.groupby.DataFrameGroupBy.corrwith

property DataFrameGroupBy.**corrwith**

Compute pairwise correlation.

Pairwise correlation is computed between rows or columns of DataFrame with rows or columns of Series or DataFrame. DataFrames are first aligned along both axes before computing the correlations.

Parameters

other [DataFrame, Series] Object with which to compute correlations.

axis [{0 or 'index', 1 or 'columns'}, default 0] The axis to use. 0 or 'index' to compute column-wise, 1 or 'columns' for row-wise.

drop [bool, default False] Drop missing indices from result.

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.

New in version 0.24.0.

Returns

Series Pairwise correlations.

See also:

DataFrame.corr

pandas.core.groupby.DataFrameGroupBy.boxplot

DataFrameGroupBy.**boxplot**(*grouped*, *subplots=True*, *column=None*, *fontsize=None*, *rot=0*, *grid=True*, *ax=None*, *figsize=None*, *layout=None*, *sharex=False*, *sharey=True*, *backend=None*, ***kwargs*)

Make box plots from DataFrameGroupBy data.

Parameters

grouped [Grouped DataFrame]

subplots [bool]

- `False` - no subplots will be used
- `True` - create a subplot for each group.

column [column name or list of names, or vector] Can be any valid input to groupby.

fontsize [int or str]

rot [label rotation angle]

grid [Setting this to True will show the grid]

ax [Matplotlib axis object, default None]

figsize [A tuple (width, height) in inches]

layout [tuple (optional)] The layout of the plot: (rows, columns).

sharex [bool, default False] Whether x-axes will be shared among subplots.

New in version 0.23.1.

sharey [bool, default True] Whether y-axes will be shared among subplots.

New in version 0.23.1.

backend [str, default None] Backend to use instead of the backend specified in the option `plotting.backend`. For instance, 'matplotlib'. Alternatively, to specify the `plotting.backend` for the whole session, set `pd.options.plotting.backend`.

New in version 1.0.0.

****kwargs** All other plotting keyword arguments to be passed to matplotlib's boxplot function.

Returns

dict of key/value = group key/DataFrame.boxplot return value

or DataFrame.boxplot return value in case subplots=figures=False

Examples

```
>>> import itertools
>>> tuples = [t for t in itertools.product(range(1000), range(4))]
>>> index = pd.MultiIndex.from_tuples(tuples, names=['lvl0', 'lvl1'])
>>> data = np.random.randn(len(index),4)
>>> df = pd.DataFrame(data, columns=list('ABCD'), index=index)
>>>
>>> grouped = df.groupby(level='lvl1')
>>> boxplot_frame_groupby(grouped)
>>>
>>> grouped = df.unstack(level='lvl1').groupby(level=0, axis=1)
>>> boxplot_frame_groupby(grouped, subplots=False)
```

3.12 Resampling

Resampler objects are returned by resample calls: *pandas.DataFrame.resample()*, *pandas.Series.resample()*.

3.12.1 Indexing, iteration

Resampler.__iter__(self)	Resampler iterator.
Resampler.groups	Dict {group name -> group labels}.
Resampler.indices	Dict {group name -> group indices}.
Resampler.get_group(self, name[, obj])	Construct DataFrame from group with provided name.

pandas.core.resample.Resampler.__iter__

Resampler.**__iter__**(*self*)

Resampler iterator.

Returns

Generator yielding sequence of (name, subsetted object)

for each group.

See also:

GroupBy.__iter__

pandas.core.resample.Resampler.groups

property Resampler.**groups**

Dict {group name -> group labels}.

pandas.core.resample.Resampler.indices

property Resampler.**indices**
Dict {group name -> group indices}.

pandas.core.resample.Resampler.get_group

Resampler.**get_group**(*self*, *name*, *obj=None*)
Construct DataFrame from group with provided name.

Parameters

name [object] The name of the group to get as a DataFrame.

obj [DataFrame, default None] The DataFrame to take the DataFrame out of. If it is None, the object groupby was called on will be used.

Returns

group [same type as obj]

3.12.2 Function application

Resampler.apply(self, func, *args, **kwargs)	Aggregate using one or more operations over the specified axis.
Resampler.aggregate(self, func, *args, **kwargs)	Aggregate using one or more operations over the specified axis.
Resampler.transform(self, arg, *args, **kwargs)	Call function producing a like-indexed Series on each group and return a Series with the transformed values.
Resampler.pipe(self, func, *args, **kwargs)	Apply a function *func* with arguments to this Resampler object and return the function's result.

pandas.core.resample.Resampler.apply

Resampler.**apply**(*self*, *func*, **args*, ***kwargs*)
Aggregate using one or more operations over the specified axis.

Parameters

func [function, str, list or dict] Function to use for aggregating the data. If a function, must either work when passed a DataFrame or when passed to DataFrame.apply.

Accepted combinations are:

- function
- string function name
- list of functions and/or function names, e.g. `[np.sum, 'mean']`
- dict of axis labels -> functions, function names or list of such.

***args** Positional arguments to pass to *func*.

****kwargs** Keyword arguments to pass to *func*.

Returns

scalar, Series or DataFrame The return can be:

- scalar : when Series.agg is called with single function
- Series : when DataFrame.agg is called with a single function
- DataFrame : when DataFrame.agg is called with several functions

Return scalar, Series or DataFrame.

See also:

DataFrame.groupby.aggregate
DataFrame.resample.transform
DataFrame.aggregate

Notes

agg is an alias for *aggregate*. Use the alias.

A passed user-defined-function will be passed a Series for evaluation.

Examples

```
>>> s = pd.Series([1,2,3,4,5],
                  index=pd.date_range('20130101', periods=5,freq='s'))
2013-01-01 00:00:00    1
2013-01-01 00:00:01    2
2013-01-01 00:00:02    3
2013-01-01 00:00:03    4
2013-01-01 00:00:04    5
Freq: S, dtype: int64
```

```
>>> r = s.resample('2s')
DatetimeIndexResampler [freq=<2 * Seconds>, axis=0, closed=left,
                        label=left, convention=start, base=0]
```

```
>>> r.agg(np.sum)
2013-01-01 00:00:00    3
2013-01-01 00:00:02    7
2013-01-01 00:00:04    5
Freq: 2S, dtype: int64
```

```
>>> r.agg(['sum','mean','max'])
                     sum  mean  max
2013-01-01 00:00:00    3   1.5    2
2013-01-01 00:00:02    7   3.5    4
2013-01-01 00:00:04    5   5.0    5
```

```
>>> r.agg({'result' : lambda x: x.mean() / x.std(),
           'total' : np.sum})
                     total    result
2013-01-01 00:00:00      3  2.121320
2013-01-01 00:00:02      7  4.949747
2013-01-01 00:00:04      5       NaN
```

pandas.core.resample.Resampler.aggregate

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

Aggregate using one or more operations over the specified axis.

Parameters

func [function, str, list or dict] Function to use for aggregating the data. If a function, must either work when passed a DataFrame or when passed to DataFrame.apply.

Accepted combinations are:

- function
- string function name
- list of functions and/or function names, e.g. `[np.sum, 'mean']`
- dict of axis labels -> functions, function names or list of such.

***args** Positional arguments to pass to *func*.

****kwargs** Keyword arguments to pass to *func*.

Returns

scalar, Series or DataFrame The return can be:

- scalar : when Series.agg is called with single function
- Series : when DataFrame.agg is called with a single function
- DataFrame : when DataFrame.agg is called with several functions

Return scalar, Series or DataFrame.

See also:

DataFrame.groupby.aggregate
DataFrame.resample.transform
DataFrame.aggregate

Notes

agg is an alias for *aggregate*. Use the alias.

A passed user-defined-function will be passed a Series for evaluation.

Examples

```
>>> s = pd.Series([1,2,3,4,5],
                  index=pd.date_range('20130101', periods=5,freq='s'))
2013-01-01 00:00:00    1
2013-01-01 00:00:01    2
2013-01-01 00:00:02    3
2013-01-01 00:00:03    4
2013-01-01 00:00:04    5
Freq: S, dtype: int64
```

```
>>> r = s.resample('2s')
DatetimeIndexResampler [freq=<2 * Seconds>, axis=0, closed=left,
                        label=left, convention=start, base=0]
```

```
>>> r.agg(np.sum)
2013-01-01 00:00:00    3
2013-01-01 00:00:02    7
2013-01-01 00:00:04    5
Freq: 2S, dtype: int64
```

```
>>> r.agg(['sum','mean','max'])
                     sum  mean  max
2013-01-01 00:00:00    3   1.5    2
2013-01-01 00:00:02    7   3.5    4
2013-01-01 00:00:04    5   5.0    5
```

```
>>> r.agg({'result' : lambda x: x.mean() / x.std(),
           'total' : np.sum})
                     total    result
2013-01-01 00:00:00      3  2.121320
2013-01-01 00:00:02      7  4.949747
2013-01-01 00:00:04      5       NaN
```

pandas.core.resample.Resampler.transform

Resampler.**transform**(*self*, *arg*, **args*, ***kwargs*)

Call function producing a like-indexed Series on each group and return a Series with the transformed values.

Parameters

arg [function] To apply to each group. Should return a Series with the same index.

Returns

transformed [Series]

Examples

```
>>> resampled.transform(lambda x: (x - x.mean()) / x.std())
```

pandas.core.resample.Resampler.pipe

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

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

New in version 0.23.0.

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))
```