pandas.tseries.offsets.BMonthEnd.rollforward

BMonthEnd.**rollforward**(*self*, *dt*)

Roll provided date forward to next offset only if not on offset.

Returns

TimeStamp Rolled timestamp if not on offset, otherwise unchanged timestamp.

pandas.tseries.offsets.BMonthEnd.__call__

BMonthEnd.**__call__**(*self*, *other*)

Call self as a function.

3.8.42 BMonthBegin

BMonthBegin	alias of *pandas.tseries.offsets.BusinessMonthBegin*

pandas.tseries.offsets.BMonthBegin

pandas.tseries.offsets.**BMonthBegin**

alias of *pandas.tseries.offsets.BusinessMonthBegin*

Properties

BMonthBegin.base	Returns a copy of the calling offset object with n=1 and all other attributes equal.
BMonthBegin.freqstr	
BMonthBegin.kwds	
BMonthBegin.name	
BMonthBegin.nanos	
BMonthBegin.normalize	
BMonthBegin.rule_code	

pandas.tseries.offsets.BMonthBegin.base

property BMonthBegin.**base**

Returns a copy of the calling offset object with n=1 and all other attributes equal.

pandas.tseries.offsets.BMonthBegin.freqstr

BMonthBegin.**freqstr**

pandas.tseries.offsets.BMonthBegin.kwds

property BMonthBegin.**kwds**

pandas.tseries.offsets.BMonthBegin.name

property BMonthBegin.**name**

pandas.tseries.offsets.BMonthBegin.nanos

property BMonthBegin.**nanos**

pandas.tseries.offsets.BMonthBegin.normalize

BMonthBegin.**normalize = False**

pandas.tseries.offsets.BMonthBegin.rule_code

property BMonthBegin.**rule_code**

Methods

BMonthBegin.apply(self, other)	
BMonthBegin.apply_index(self, other)	Vectorized apply of DateOffset to DatetimeIndex, raises NotImplentedError for offsets without a vectorized implementation.
BMonthBegin.copy(self)	
BMonthBegin.isAnchored(self)	
BMonthBegin.onOffset(self, dt)	
BMonthBegin.is_anchored(self)	
BMonthBegin.is_on_offset(self, dt)	
BMonthBegin.rollback(self, dt)	Roll provided date backward to next offset only if not on offset.
BMonthBegin.rollforward(self, dt)	Roll provided date forward to next offset only if not on offset.
BMonthBegin.__call__(self, other)	Call self as a function.

pandas.tseries.offsets.BMonthBegin.apply

BMonthBegin.**apply**(*self*, *other*)

pandas.tseries.offsets.BMonthBegin.apply_index

BMonthBegin.**apply_index**(*self*, *other*)

Vectorized apply of DateOffset to DatetimeIndex, raises NotImplentedError for offsets without a vectorized implementation.

Parameters

i [DatetimeIndex]

Returns

y [DatetimeIndex]

pandas.tseries.offsets.BMonthBegin.copy

BMonthBegin.**copy**(*self*)

pandas.tseries.offsets.BMonthBegin.isAnchored

BMonthBegin.**isAnchored**(*self*)

pandas.tseries.offsets.BMonthBegin.onOffset

BMonthBegin.**onOffset**(*self*, *dt*)

pandas.tseries.offsets.BMonthBegin.is_anchored

BMonthBegin.**is_anchored**(*self*)

pandas.tseries.offsets.BMonthBegin.is_on_offset

BMonthBegin.**is_on_offset**(*self*, *dt*)

pandas.tseries.offsets.BMonthBegin.rollback

BMonthBegin.**rollback**(*self*, *dt*)

Roll provided date backward to next offset only if not on offset.

Returns

TimeStamp Rolled timestamp if not on offset, otherwise unchanged timestamp.

pandas.tseries.offsets.BMonthBegin.rollforward

BMonthBegin.**rollforward**(*self*, *dt*)

Roll provided date forward to next offset only if not on offset.

Returns

TimeStamp Rolled timestamp if not on offset, otherwise unchanged timestamp.

pandas.tseries.offsets.BMonthBegin.__call__

BMonthBegin.**__call__**(*self*, *other*)

Call self as a function.

3.8.43 CBMonthEnd

CBMonthEnd	alias of *pandas.tseries.offsets.CustomBusinessMonthEnd*

pandas.tseries.offsets.CBMonthEnd

pandas.tseries.offsets.**CBMonthEnd**

alias of *pandas.tseries.offsets.CustomBusinessMonthEnd*

Properties

CBMonthEnd.base	Returns a copy of the calling offset object with n=1 and all other attributes equal.
CBMonthEnd.cbday_roll	Define default roll function to be called in apply method.
CBMonthEnd.freqstr	
CBMonthEnd.kwds	
CBMonthEnd.m_offset	
CBMonthEnd.month_roll	Define default roll function to be called in apply method.
CBMonthEnd.name	
CBMonthEnd.nanos	
CBMonthEnd.normalize	
CBMonthEnd.offset	Alias for self._offset.
CBMonthEnd.rule_code	

pandas.tseries.offsets.CBMonthEnd.base

property CBMonthEnd.**base**
Returns a copy of the calling offset object with n=1 and all other attributes equal.

pandas.tseries.offsets.CBMonthEnd.cbday_roll

CBMonthEnd.**cbday_roll**
Define default roll function to be called in apply method.

pandas.tseries.offsets.CBMonthEnd.freqstr

CBMonthEnd.**freqstr**

pandas.tseries.offsets.CBMonthEnd.kwds

property CBMonthEnd.**kwds**

pandas.tseries.offsets.CBMonthEnd.m_offset

CBMonthEnd.**m_offset**

pandas.tseries.offsets.CBMonthEnd.month_roll

CBMonthEnd.**month_roll**
Define default roll function to be called in apply method.

pandas.tseries.offsets.CBMonthEnd.name

property CBMonthEnd.**name**

pandas.tseries.offsets.CBMonthEnd.nanos

property CBMonthEnd.**nanos**

pandas.tseries.offsets.CBMonthEnd.normalize

CBMonthEnd.**normalize = False**

pandas.tseries.offsets.CBMonthEnd.offset

property CBMonthEnd.**offset**
Alias for self._offset.

pandas.tseries.offsets.CBMonthEnd.rule_code

property CBMonthEnd.**rule_code**

Methods

CBMonthEnd.apply(self, other)	
CBMonthEnd.apply_index(self, other)	Vectorized apply of DateOffset to DatetimeIndex, raises NotImplentedError for offsets without a vectorized implementation.
CBMonthEnd.copy(self)	
CBMonthEnd.isAnchored(self)	
CBMonthEnd.onOffset(self, dt)	
CBMonthEnd.is_anchored(self)	
CBMonthEnd.is_on_offset(self, dt)	
CBMonthEnd.rollback(self, dt)	Roll provided date backward to next offset only if not on offset.
CBMonthEnd.rollforward(self, dt)	Roll provided date forward to next offset only if not on offset.
CBMonthEnd.__call__(self, other)	Call self as a function.

pandas.tseries.offsets.CBMonthEnd.apply

CBMonthEnd.**apply**(*self*, *other*)

pandas.tseries.offsets.CBMonthEnd.apply_index

CBMonthEnd.**apply_index**(*self*, *other*)
Vectorized apply of DateOffset to DatetimeIndex, raises NotImplentedError for offsets without a vectorized implementation.

Parameters

i [DatetimeIndex]

Returns

y [DatetimeIndex]

pandas.tseries.offsets.CBMonthEnd.copy

CBMonthEnd.**copy**(*self*)

pandas.tseries.offsets.CBMonthEnd.isAnchored

CBMonthEnd.**isAnchored**(*self*)

pandas.tseries.offsets.CBMonthEnd.onOffset

CBMonthEnd.**onOffset**(*self*, *dt*)

pandas.tseries.offsets.CBMonthEnd.is_anchored

CBMonthEnd.**is_anchored**(*self*)

pandas.tseries.offsets.CBMonthEnd.is_on_offset

CBMonthEnd.**is_on_offset**(*self*, *dt*)

pandas.tseries.offsets.CBMonthEnd.rollback

CBMonthEnd.**rollback**(*self*, *dt*)

Roll provided date backward to next offset only if not on offset.

Returns

TimeStamp Rolled timestamp if not on offset, otherwise unchanged timestamp.

pandas.tseries.offsets.CBMonthEnd.rollforward

CBMonthEnd.**rollforward**(*self*, *dt*)

Roll provided date forward to next offset only if not on offset.

Returns

TimeStamp Rolled timestamp if not on offset, otherwise unchanged timestamp.

pandas.tseries.offsets.CBMonthEnd.__call__

CBMonthEnd.**__call__**(*self*, *other*)

Call self as a function.

3.8.44 CBMonthBegin

CBMonthBegin	alias of *pandas.tseries.offsets.CustomBusinessMonthBegin*

pandas.tseries.offsets.CBMonthBegin

pandas.tseries.offsets.**CBMonthBegin**
alias of *pandas.tseries.offsets.CustomBusinessMonthBegin*

Properties

CBMonthBegin.base	Returns a copy of the calling offset object with n=1 and all other attributes equal.
CBMonthBegin.cbday_roll	Define default roll function to be called in apply method.
CBMonthBegin.freqstr	
CBMonthBegin.kwds	
CBMonthBegin.m_offset	
CBMonthBegin.month_roll	Define default roll function to be called in apply method.
CBMonthBegin.name	
CBMonthBegin.nanos	
CBMonthBegin.normalize	
CBMonthBegin.offset	Alias for self._offset.
CBMonthBegin.rule_code	

pandas.tseries.offsets.CBMonthBegin.base

property CBMonthBegin.**base**
Returns a copy of the calling offset object with n=1 and all other attributes equal.

pandas.tseries.offsets.CBMonthBegin.cbday_roll

CBMonthBegin.**cbday_roll**
Define default roll function to be called in apply method.

pandas.tseries.offsets.CBMonthBegin.freqstr

CBMonthBegin.**freqstr**

pandas.tseries.offsets.CBMonthBegin.kwds

property CBMonthBegin.**kwds**

pandas.tseries.offsets.CBMonthBegin.m_offset

CBMonthBegin.**m_offset**

pandas.tseries.offsets.CBMonthBegin.month_roll

CBMonthBegin.**month_roll**
Define default roll function to be called in apply method.

pandas.tseries.offsets.CBMonthBegin.name

property CBMonthBegin.**name**

pandas.tseries.offsets.CBMonthBegin.nanos

property CBMonthBegin.**nanos**

pandas.tseries.offsets.CBMonthBegin.normalize

CBMonthBegin.**normalize = False**

pandas.tseries.offsets.CBMonthBegin.offset

property CBMonthBegin.**offset**
Alias for self._offset.

pandas.tseries.offsets.CBMonthBegin.rule_code

property CBMonthBegin.**rule_code**

Methods

CBMonthBegin.apply(self, other)	
CBMonthBegin.apply_index(self, other)	Vectorized apply of DateOffset to DatetimeIndex, raises NotImplentedError for offsets without a vectorized implementation.
CBMonthBegin.copy(self)	
CBMonthBegin.isAnchored(self)	
CBMonthBegin.onOffset(self, dt)	
CBMonthBegin.is_anchored(self)	
CBMonthBegin.is_on_offset(self, dt)	
CBMonthBegin.rollback(self, dt)	Roll provided date backward to next offset only if not on offset.
CBMonthBegin.rollforward(self, dt)	Roll provided date forward to next offset only if not on offset.
CBMonthBegin.__call__(self, other)	Call self as a function.

pandas.tseries.offsets.CBMonthBegin.apply

CBMonthBegin.**apply**(*self*, *other*)

pandas.tseries.offsets.CBMonthBegin.apply_index

CBMonthBegin.**apply_index**(*self*, *other*)

Vectorized apply of DateOffset to DatetimeIndex, raises NotImplentedError for offsets without a vectorized implementation.

Parameters

i [DatetimeIndex]

Returns

y [DatetimeIndex]

pandas.tseries.offsets.CBMonthBegin.copy

CBMonthBegin.**copy**(*self*)

pandas.tseries.offsets.CBMonthBegin.isAnchored

CBMonthBegin.**isAnchored**(*self*)

pandas.tseries.offsets.CBMonthBegin.onOffset

CBMonthBegin.**onOffset**(*self*, *dt*)

pandas.tseries.offsets.CBMonthBegin.is_anchored

CBMonthBegin.**is_anchored**(*self*)

pandas.tseries.offsets.CBMonthBegin.is_on_offset

CBMonthBegin.**is_on_offset**(*self*, *dt*)

pandas.tseries.offsets.CBMonthBegin.rollback

CBMonthBegin.**rollback**(*self*, *dt*)

Roll provided date backward to next offset only if not on offset.

Returns

TimeStamp Rolled timestamp if not on offset, otherwise unchanged timestamp.

pandas.tseries.offsets.CBMonthBegin.rollforward

CBMonthBegin.**rollforward**(*self*, *dt*)

Roll provided date forward to next offset only if not on offset.

Returns

TimeStamp Rolled timestamp if not on offset, otherwise unchanged timestamp.

pandas.tseries.offsets.CBMonthBegin.__call__

CBMonthBegin.**__call__**(*self*, *other*)

Call self as a function.

3.8.45 CDay

CDay	alias of *pandas.tseries.offsets.CustomBusinessDay*

pandas.tseries.offsets.CDay

pandas.tseries.offsets.**CDay**
alias of *pandas.tseries.offsets.CustomBusinessDay*

Properties

CDay.base	Returns a copy of the calling offset object with n=1 and all other attributes equal.
CDay.freqstr	
CDay.kwds	
CDay.name	
CDay.nanos	
CDay.normalize	
CDay.offset	Alias for self._offset.
CDay.rule_code	

pandas.tseries.offsets.CDay.base

property CDay.**base**
Returns a copy of the calling offset object with n=1 and all other attributes equal.

pandas.tseries.offsets.CDay.freqstr

CDay.**freqstr**

pandas.tseries.offsets.CDay.kwds

property CDay.**kwds**

pandas.tseries.offsets.CDay.name

property CDay.**name**

pandas.tseries.offsets.CDay.nanos

property CDay.**nanos**

pandas.tseries.offsets.CDay.normalize

CDay.**normalize = False**

pandas.tseries.offsets.CDay.offset

property CDay.**offset**
Alias for self._offset.

pandas.tseries.offsets.CDay.rule_code

property CDay.**rule_code**

Methods

CDay.apply(self, other)	
CDay.apply_index(self, i)	Vectorized apply of DateOffset to DatetimeIndex, raises NotImplentedError for offsets without a vectorized implementation.
CDay.copy(self)	
CDay.isAnchored(self)	
CDay.onOffset(self, dt)	
CDay.is_anchored(self)	
CDay.is_on_offset(self, dt)	
CDay.rollback(self, dt)	Roll provided date backward to next offset only if not on offset.
CDay.rollforward(self, dt)	Roll provided date forward to next offset only if not on offset.
CDay.__call__(self, other)	Call self as a function.

pandas.tseries.offsets.CDay.apply

CDay.**apply**(*self*, *other*)

pandas.tseries.offsets.CDay.apply_index

CDay.**apply_index**(*self*, *i*)
Vectorized apply of DateOffset to DatetimeIndex, raises NotImplentedError for offsets without a vectorized implementation.

Parameters

i [DatetimeIndex]

Returns

y [DatetimeIndex]

pandas.tseries.offsets.CDay.copy

CDay.**copy**(*self*)

pandas.tseries.offsets.CDay.isAnchored

CDay.**isAnchored**(*self*)

pandas.tseries.offsets.CDay.onOffset

CDay.**onOffset**(*self*, *dt*)

pandas.tseries.offsets.CDay.is_anchored

CDay.**is_anchored**(*self*)

pandas.tseries.offsets.CDay.is_on_offset

CDay.**is_on_offset**(*self*, *dt*)

pandas.tseries.offsets.CDay.rollback

CDay.**rollback**(*self*, *dt*)
Roll provided date backward to next offset only if not on offset.

Returns

TimeStamp Rolled timestamp if not on offset, otherwise unchanged timestamp.

pandas.tseries.offsets.CDay.rollforward

CDay.**rollforward**(*self*, *dt*)
Roll provided date forward to next offset only if not on offset.

Returns

TimeStamp Rolled timestamp if not on offset, otherwise unchanged timestamp.

pandas.tseries.offsets.CDay.__call__

CDay.**__call__**(*self*, *other*)
Call self as a function.

3.9 Frequencies

to_offset(freq)	Return DateOffset object from string or tuple representation or datetime.timedelta object.

3.9.1 pandas.tseries.frequencies.to_offset

pandas.tseries.frequencies.**to_offset**(*freq*) → Union[*pandas.tseries.offsets.DateOffset*, NoneType]

Return DateOffset object from string or tuple representation or datetime.timedelta object.

Parameters

freq [str, tuple, datetime.timedelta, DateOffset or None]

Returns

DateOffset None if freq is None.

Raises

ValueError If freq is an invalid frequency

See also:

DateOffset

Examples

```
>>> to_offset('5min')
<5 * Minutes>
```

```
>>> to_offset('1D1H')
<25 * Hours>
```

```
>>> to_offset(('W', 2))
<2 * Weeks: weekday=6>
```

```
>>> to_offset((2, 'B'))
<2 * BusinessDays>
```

```
>>> to_offset(datetime.timedelta(days=1))
<Day>
```

```
>>> to_offset(Hour())
<Hour>
```

3.10 Window

Rolling objects are returned by `.rolling` calls: *pandas.DataFrame.rolling()*, *pandas.Series.rolling()*, etc. Expanding objects are returned by `.expanding` calls: *pandas.DataFrame.expanding()*, *pandas.Series.expanding()*, etc. EWM objects are returned by `.ewm` calls: *pandas.DataFrame.ewm()*, *pandas.Series.ewm()*, etc.

3.10.1 Standard moving window functions

Rolling.count(self)	The rolling count of any non-NaN observations inside the window.
Rolling.sum(self, *args, **kwargs)	Calculate rolling sum of given DataFrame or Series.
Rolling.mean(self, *args, **kwargs)	Calculate the rolling mean of the values.
Rolling.median(self, **kwargs)	Calculate the rolling median.
Rolling.var(self[, ddof])	Calculate unbiased rolling variance.
Rolling.std(self[, ddof])	Calculate rolling standard deviation.
Rolling.min(self, *args, **kwargs)	Calculate the rolling minimum.
Rolling.max(self, *args, **kwargs)	Calculate the rolling maximum.
Rolling.corr(self[, other, pairwise])	Calculate rolling correlation.
Rolling.cov(self[, other, pairwise, ddof])	Calculate the rolling sample covariance.
Rolling.skew(self, **kwargs)	Unbiased rolling skewness.
Rolling.kurt(self, **kwargs)	Calculate unbiased rolling kurtosis.
Rolling.apply(self, func[, raw, engine, ...])	The rolling function's apply function.
Rolling.aggregate(self, func, *args, **kwargs)	Aggregate using one or more operations over the specified axis.
Rolling.quantile(self, quantile[, interpolation])	Calculate the rolling quantile.
Window.mean(self, *args, **kwargs)	Calculate the window mean of the values.
Window.sum(self, *args, **kwargs)	Calculate window sum of given DataFrame or Series.
Window.var(self[, ddof])	Calculate unbiased window variance.
Window.std(self[, ddof])	Calculate window standard deviation.

pandas.core.window.rolling.Rolling.count

Rolling.**count**(*self*)

The rolling count of any non-NaN observations inside the window.

Returns

Series or DataFrame Returned object type is determined by the caller of the rolling calculation.

See also:

Series.rolling Calling object with Series data.
DataFrame.rolling Calling object with DataFrames.
DataFrame.count Count of the full DataFrame.

Examples

```
>>> s = pd.Series([2, 3, np.nan, 10])
>>> s.rolling(2).count()
0    1.0
1    2.0
2    1.0
3    1.0
dtype: float64
>>> s.rolling(3).count()
0    1.0
1    2.0
2    2.0
3    2.0
dtype: float64
>>> s.rolling(4).count()
0    1.0
1    2.0
2    2.0
3    3.0
dtype: float64
```

pandas.core.window.rolling.Rolling.sum

Rolling.**sum**(*self*, **args*, ***kwargs*)

Calculate rolling sum of given DataFrame or Series.

Parameters

***args, **kwargs** For compatibility with other rolling methods. Has no effect on the computed value.

Returns

Series or DataFrame Same type as the input, with the same index, containing the rolling sum.

See also:

Series.sum Reducing sum for Series.
DataFrame.sum Reducing sum for DataFrame.

Examples

```
>>> s = pd.Series([1, 2, 3, 4, 5])
>>> s
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

```
>>> s.rolling(3).sum()
0     NaN
1     NaN
2     6.0
```

(continues on next page)

(continued from previous page)

```
3     9.0
4    12.0
dtype: float64
```

```
>>> s.expanding(3).sum()
0     NaN
1     NaN
2     6.0
3    10.0
4    15.0
dtype: float64
```

```
>>> s.rolling(3, center=True).sum()
0     NaN
1     6.0
2     9.0
3    12.0
4     NaN
dtype: float64
```

For DataFrame, each rolling sum is computed column-wise.

```
>>> df = pd.DataFrame({"A": s, "B": s ** 2})
>>> df
   A   B
0  1   1
1  2   4
2  3   9
3  4  16
4  5  25
```

```
>>> df.rolling(3).sum()
      A     B
0   NaN   NaN
1   NaN   NaN
2   6.0  14.0
3   9.0  29.0
4  12.0  50.0
```

pandas.core.window.rolling.Rolling.mean

`Rolling.`**`mean`**(*self*, **args*, ***kwargs*)

Calculate the rolling mean of the values.

Parameters

***args** Under Review.

****kwargs** Under Review.

Returns

Series or DataFrame Returned object type is determined by the caller of the rolling calculation.

See also:

`Series.rolling` Calling object with Series data.

DataFrame.rolling Calling object with DataFrames.
Series.mean Equivalent method for Series.
DataFrame.mean Equivalent method for DataFrame.

Examples

The below examples will show rolling mean calculations with window sizes of two and three, respectively.

```
>>> s = pd.Series([1, 2, 3, 4])
>>> s.rolling(2).mean()
0    NaN
1    1.5
2    2.5
3    3.5
dtype: float64
```

```
>>> s.rolling(3).mean()
0    NaN
1    NaN
2    2.0
3    3.0
dtype: float64
```

pandas.core.window.rolling.Rolling.median

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

Calculate the rolling median.

Parameters

****kwargs** For compatibility with other rolling methods. Has no effect on the computed median.

Returns

Series or DataFrame Returned type is the same as the original object.

See also:

Series.rolling Calling object with Series data.
DataFrame.rolling Calling object with DataFrames.
Series.median Equivalent method for Series.
DataFrame.median Equivalent method for DataFrame.

Examples

Compute the rolling median of a series with a window size of 3.

```
>>> s = pd.Series([0, 1, 2, 3, 4])
>>> s.rolling(3).median()
0    NaN
1    NaN
2    1.0
3    2.0
4    3.0
dtype: float64
```

pandas.core.window.rolling.Rolling.var

Rolling.**var**(*self*, *ddof=1*, **args*, ***kwargs*)

Calculate unbiased rolling variance.

Normalized by N-1 by default. This can be changed using the *ddof* argument.

Parameters

ddof [int, default 1] Delta Degrees of Freedom. The divisor used in calculations is `N - ddof`, where `N` represents the number of elements.

***args, **kwargs** For NumPy compatibility. No additional arguments are used.

Returns

Series or DataFrame Returns the same object type as the caller of the rolling calculation.

See also:

`Series.rolling` Calling object with Series data.
`DataFrame.rolling` Calling object with DataFrames.
`Series.var` Equivalent method for Series.
`DataFrame.var` Equivalent method for DataFrame.
`numpy.var` Equivalent method for Numpy array.

Notes

The default *ddof* of 1 used in `Series.var()` is different than the default *ddof* of 0 in `numpy.var()`.

A minimum of 1 period is required for the rolling calculation.

Examples

```
>>> s = pd.Series([5, 5, 6, 7, 5, 5, 5])
>>> s.rolling(3).var()
0         NaN
1         NaN
2    0.333333
3    1.000000
4    1.000000
5    1.333333
6    0.000000
dtype: float64
```

```
>>> s.expanding(3).var()
0         NaN
1         NaN
2    0.333333
3    0.916667
4    0.800000
5    0.700000
6    0.619048
dtype: float64
```

pandas.core.window.rolling.Rolling.std

Rolling.**std**(*self*, *ddof=1*, **args*, ***kwargs*)

Calculate rolling standard deviation.

Normalized by N-1 by default. This can be changed using the *ddof* argument.

Parameters

ddof [int, default 1] Delta Degrees of Freedom. The divisor used in calculations is `N - ddof`, where `N` represents the number of elements.

***args, **kwargs** For NumPy compatibility. No additional arguments are used.

Returns

Series or DataFrame Returns the same object type as the caller of the rolling calculation.

See also:

`Series.rolling` Calling object with Series data.
`DataFrame.rolling` Calling object with DataFrames.
`Series.std` Equivalent method for Series.
`DataFrame.std` Equivalent method for DataFrame.
`numpy.std` Equivalent method for Numpy array.

Notes

The default *ddof* of 1 used in Series.std is different than the default *ddof* of 0 in numpy.std.

A minimum of one period is required for the rolling calculation.

Examples

```
>>> s = pd.Series([5, 5, 6, 7, 5, 5, 5])
>>> s.rolling(3).std()
0         NaN
1         NaN
2    0.577350
3    1.000000
4    1.000000
5    1.154701
6    0.000000
dtype: float64
```

```
>>> s.expanding(3).std()
0         NaN
1         NaN
2    0.577350
3    0.957427
4    0.894427
5    0.836660
6    0.786796
dtype: float64
```

pandas.core.window.rolling.Rolling.min

Rolling.**min**(*self*, **args*, ***kwargs*)

Calculate the rolling minimum.

Parameters

****kwargs** Under Review.

Returns

Series or DataFrame Returned object type is determined by the caller of the rolling calculation.

See also:

Series.rolling Calling object with a Series.
DataFrame.rolling Calling object with a DataFrame.
Series.min Similar method for Series.
DataFrame.min Similar method for DataFrame.

Examples

Performing a rolling minimum with a window size of 3.

```
>>> s = pd.Series([4, 3, 5, 2, 6])
>>> s.rolling(3).min()
0    NaN
1    NaN
2    3.0
3    2.0
4    2.0
dtype: float64
```

pandas.core.window.rolling.Rolling.max

Rolling.**max**(*self*, **args*, ***kwargs*)

Calculate the rolling maximum.

Parameters

***args, **kwargs** Arguments and keyword arguments to be passed into func.

Returns

Series or DataFrame Return type is determined by the caller.

See also:

Series.rolling Series rolling.
DataFrame.rolling DataFrame rolling.

pandas.core.window.rolling.Rolling.corr

`Rolling.`**`corr`**(*self*, *other=None*, *pairwise=None*, ***kwargs*)

Calculate rolling correlation.

Parameters

other [Series, DataFrame, or ndarray, optional] If not supplied then will default to self.

pairwise [bool, default None] Calculate pairwise combinations of columns within a DataFrame. If *other* is not specified, defaults to *True*, otherwise defaults to *False*. Not relevant for *Series*.

****kwargs** Unused.

Returns

Series or DataFrame Returned object type is determined by the caller of the rolling calculation.

See also:

Series.rolling Calling object with Series data.
DataFrame.rolling Calling object with DataFrames.
Series.corr Equivalent method for Series.
DataFrame.corr Equivalent method for DataFrame.
rolling.cov Similar method to calculate covariance.
numpy.corrcoef NumPy Pearson's correlation calculation.

Notes

This function uses Pearson's definition of correlation (https://en.wikipedia.org/wiki/Pearson_correlation_coefficient).

When *other* is not specified, the output will be self correlation (e.g. all 1's), except for *DataFrame* inputs with *pairwise* set to *True*.

Function will return `NaN` for correlations of equal valued sequences; this is the result of a 0/0 division error.

When *pairwise* is set to *False*, only matching columns between *self* and *other* will be used.

When *pairwise* is set to *True*, the output will be a MultiIndex DataFrame with the original index on the first level, and the *other* DataFrame columns on the second level.

In the case of missing elements, only complete pairwise observations will be used.

Examples

The below example shows a rolling calculation with a window size of four matching the equivalent function call using `numpy.corrcoef()`.

```
>>> v1 = [3, 3, 3, 5, 8]
>>> v2 = [3, 4, 4, 4, 8]
>>> # numpy returns a 2X2 array, the correlation coefficient
>>> # is the number at entry [0][1]
>>> print(f"{np.corrcoef(v1[:-1], v2[:-1])[0][1]:.6f}")
0.333333
>>> print(f"{np.corrcoef(v1[1:], v2[1:])[0][1]:.6f}")
0.916949
>>> s1 = pd.Series(v1)
```

(continues on next page)

(continued from previous page)

```
>>> s2 = pd.Series(v2)
>>> s1.rolling(4).corr(s2)
0         NaN
1         NaN
2         NaN
3    0.333333
4    0.916949
dtype: float64
```

The below example shows a similar rolling calculation on a DataFrame using the pairwise option.

```
>>> matrix = np.array([[51., 35.], [49., 30.], [47., 32.],    [46., 31.], [50.,␣
↪36.]])
>>> print(np.corrcoef(matrix[:-1,0], matrix[:-1,1]).round(7))
[[1.         0.6263001]
 [0.6263001  1.       ]]
>>> print(np.corrcoef(matrix[1:,0], matrix[1:,1]).round(7))
[[1.         0.5553681]
 [0.5553681  1.       ]]
>>> df = pd.DataFrame(matrix, columns=['X','Y'])
>>> df
      X     Y
0  51.0  35.0
1  49.0  30.0
2  47.0  32.0
3  46.0  31.0
4  50.0  36.0
>>> df.rolling(4).corr(pairwise=True)
           X         Y
0 X       NaN       NaN
  Y       NaN       NaN
1 X       NaN       NaN
  Y       NaN       NaN
2 X       NaN       NaN
  Y       NaN       NaN
3 X  1.000000  0.626300
  Y  0.626300  1.000000
4 X  1.000000  0.555368
  Y  0.555368  1.000000
```

pandas.core.window.rolling.Rolling.cov

`Rolling.cov`(*self*, *other=None*, *pairwise=None*, *ddof=1*, ***kwargs*)

Calculate the rolling sample covariance.

Parameters

other [Series, DataFrame, or ndarray, optional] If not supplied then will default to self and produce pairwise output.

pairwise [bool, default None] If False then only matching columns between self and other will be used and the output will be a DataFrame. If True then all pairwise combinations will be calculated and the output will be a MultiIndexed DataFrame in the case of DataFrame inputs. In the case of missing elements, only complete pairwise observations will be used.

ddof [int, default 1] Delta Degrees of Freedom. The divisor used in calculations is `N` −

ddof, where N represents the number of elements.

****kwargs** Keyword arguments to be passed into func.

Returns

Series or DataFrame Return type is determined by the caller.

See also:

Series.rolling Series rolling.
DataFrame.rolling DataFrame rolling.

pandas.core.window.rolling.Rolling.skew

Rolling.**skew**(*self*, ***kwargs*)

Unbiased rolling skewness.

Parameters

****kwargs** Keyword arguments to be passed into func.

Returns

Series or DataFrame Return type is determined by the caller.

See also:

Series.rolling Series rolling.
DataFrame.rolling DataFrame rolling.

pandas.core.window.rolling.Rolling.kurt

Rolling.**kurt**(*self*, ***kwargs*)

Calculate unbiased rolling kurtosis.

This function uses Fisher's definition of kurtosis without bias.

Parameters

****kwargs** Under Review.

Returns

Series or DataFrame Returned object type is determined by the caller of the rolling calculation.

See also:

Series.rolling Calling object with Series data.
DataFrame.rolling Calling object with DataFrames.
Series.kurt Equivalent method for Series.
DataFrame.kurt Equivalent method for DataFrame.
scipy.stats.skew Third moment of a probability density.
scipy.stats.kurtosis Reference SciPy method.