Table 164 – continued from previous page

DatetimeIndex.week	The week ordinal of the year.
DatetimeIndex.dayofweek	The day of the week with Monday=0, Sunday=6.
DatetimeIndex.weekday	The day of the week with Monday=0, Sunday=6.
DatetimeIndex.quarter	The quarter of the date.
DatetimeIndex.tz	Return timezone, if any.
DatetimeIndex.freq	Return the frequency object if it is set, otherwise None.
DatetimeIndex.freqstr	Return the frequency object as a string if its set, otherwise None
DatetimeIndex.is_month_start	Indicates whether the date is the first day of the month.
DatetimeIndex.is_month_end	Indicates whether the date is the last day of the month.
DatetimeIndex.is_quarter_start	Indicator for whether the date is the first day of a quarter.
DatetimeIndex.is_quarter_end	Indicator for whether the date is the last day of a quarter.
DatetimeIndex.is_year_start	Indicate whether the date is the first day of a year.
DatetimeIndex.is_year_end	Indicate whether the date is the last day of the year.
DatetimeIndex.is_leap_year	Boolean indicator if the date belongs to a leap year.
DatetimeIndex.inferred_freq	Tryies to return a string representing a frequency guess, generated by infer_freq.

Selecting

DatetimeIndex.indexer_at_time(self, time[, asof])	Return index locations of index values at particular time of day (e.g.
DatetimeIndex.indexer_between_time(self, ...)	Return index locations of values between particular times of day (e.g., 9:00-9:30AM).

pandas.DatetimeIndex.indexer_at_time

DatetimeIndex.**indexer_at_time**(*self*, *time*, *asof=False*)

Return index locations of index values at particular time of day (e.g. 9:30AM).

Parameters

time [datetime.time or str] datetime.time or string in appropriate format ("%H:%M", "%H%M", "%I:%M%p", "%I%M%p", "%H:%M:%S", "%H%M%S", "%I:%M:%S%p", "%I%M%S%p").

Returns

values_at_time [array of integers]

See also:

indexer_between_time, ***DataFrame.at_time***

pandas.DatetimeIndex.indexer_between_time

DatetimeIndex.**indexer_between_time**(*self*, *start_time*, *end_time*, *include_start=True*, *include_end=True*)

Return index locations of values between particular times of day (e.g., 9:00-9:30AM).

Parameters

start_time, end_time [datetime.time, str] datetime.time or string in appropriate format ("%H:%M", "%H%M", "%I:%M%p", "%I%M%p", "%H:%M:%S", "%H%M%S", "%I:%M:%S%p", "%I%M%S%p").

include_start [bool, default True]

include_end [bool, default True]

Returns

values_between_time [array of integers]

See also:

indexer_at_time*, *DataFrame.between_time

Time-specific operations

DatetimeIndex.normalize(self, *args, **kwargs)	Convert times to midnight.
DatetimeIndex.strftime(self, *args, **kwargs)	Convert to Index using specified date_format.
DatetimeIndex.snap(self[, freq])	Snap time stamps to nearest occurring frequency.
DatetimeIndex.tz_convert(self, *args, **kwargs)	Convert tz-aware Datetime Array/Index from one time zone to another.
DatetimeIndex.tz_localize(self, *args, **kwargs)	Localize tz-naive Datetime Array/Index to tz-aware Datetime Array/Index.
DatetimeIndex.round(self, *args, **kwargs)	Perform round operation on the data to the specified *freq*.
DatetimeIndex.floor(self, *args, **kwargs)	Perform floor operation on the data to the specified *freq*.
DatetimeIndex.ceil(self, *args, **kwargs)	Perform ceil operation on the data to the specified *freq*.
DatetimeIndex.month_name(self, *args, **kwargs)	Return the month names of the DateTimeIndex with specified locale.
DatetimeIndex.day_name(self, *args, **kwargs)	Return the day names of the DateTimeIndex with specified locale.

Conversion

DatetimeIndex.to_period(self, *args, **kwargs)	Cast to PeriodArray/Index at a particular frequency.
DatetimeIndex.to_perioddelta(self, *args, ...)	Calculate TimedeltaArray of difference between index values and index converted to PeriodArray at specified freq.
DatetimeIndex.to_pydatetime(self, *args, ...)	Return Datetime Array/Index as object ndarray of datetime.datetime objects.

continues on next page

Table 167 – continued from previous page

`DatetimeIndex.to_series`(self[, keep_tz, ...])	Create a Series with both index and values equal to the index keys useful with map for returning an indexer based on an index.
`DatetimeIndex.to_frame`(self[, index, name])	Create a DataFrame with a column containing the Index.

Methods

`DatetimeIndex.mean`(self, *args, **kwargs)	Return the mean value of the Array.

3.7.7 TimedeltaIndex

`TimedeltaIndex`([data, unit, freq, closed, ...])	Immutable ndarray of timedelta64 data, represented internally as int64, and which can be boxed to timedelta objects.

pandas.TimedeltaIndex

class pandas.**TimedeltaIndex**(*data=None*, *unit=None*, *freq=None*, *closed=None*, *dtype=dtype('<m8[ns]')*, *copy=False*, *name=None*)

Immutable ndarray of timedelta64 data, represented internally as int64, and which can be boxed to timedelta objects.

Parameters

data [array-like (1-dimensional), optional] Optional timedelta-like data to construct index with.

unit [unit of the arg (D,h,m,s,ms,us,ns) denote the unit, optional] Which is an integer/float number.

freq [str or pandas offset object, optional] One of pandas date offset strings or corresponding objects. The string 'infer' can be passed in order to set the frequency of the index as the inferred frequency upon creation.

copy [bool] Make a copy of input ndarray.

name [object] Name to be stored in the index.

See also:

Index The base pandas Index type.
Timedelta Represents a duration between two dates or times.
DatetimeIndex Index of datetime64 data.
PeriodIndex Index of Period data.
timedelta_range Create a fixed-frequency TimedeltaIndex.

Notes

To learn more about the frequency strings, please see this link.

Attributes

days	Number of days for each element.
seconds	Number of seconds (>= 0 and less than 1 day) for each element.
microseconds	Number of microseconds (>= 0 and less than 1 second) for each element.
nanoseconds	Number of nanoseconds (>= 0 and less than 1 microsecond) for each element.
components	Return a dataframe of the components (days, hours, minutes, seconds, milliseconds, microseconds, nanoseconds) of the Timedeltas.
inferred_freq	Tryies to return a string representing a frequency guess, generated by infer_freq.

pandas.TimedeltaIndex.days

property TimedeltaIndex.**days**
Number of days for each element.

pandas.TimedeltaIndex.seconds

property TimedeltaIndex.**seconds**
Number of seconds (>= 0 and less than 1 day) for each element.

pandas.TimedeltaIndex.microseconds

property TimedeltaIndex.**microseconds**
Number of microseconds (>= 0 and less than 1 second) for each element.

pandas.TimedeltaIndex.nanoseconds

property TimedeltaIndex.**nanoseconds**
Number of nanoseconds (>= 0 and less than 1 microsecond) for each element.

pandas.TimedeltaIndex.components

property TimedeltaIndex.**components**

Return a dataframe of the components (days, hours, minutes, seconds, milliseconds, microseconds, nanoseconds) of the Timedeltas.

Returns

a DataFrame

pandas.TimedeltaIndex.inferred_freq

TimedeltaIndex.**inferred_freq**

Tryies to return a string representing a frequency guess, generated by infer_freq. Returns None if it can't autodetect the frequency.

Methods

to_pytimedelta(self, *args, **kwargs)	Return Timedelta Array/Index as object ndarray of datetime.timedelta objects.
to_series(self[, index, name])	Create a Series with both index and values equal to the index keys.
round(self, *args, **kwargs)	Perform round operation on the data to the specified *freq*.
floor(self, *args, **kwargs)	Perform floor operation on the data to the specified *freq*.
ceil(self, *args, **kwargs)	Perform ceil operation on the data to the specified *freq*.
to_frame(self[, index, name])	Create a DataFrame with a column containing the Index.
mean(self, *args, **kwargs)	Return the mean value of the Array.

pandas.TimedeltaIndex.to_pytimedelta

TimedeltaIndex.**to_pytimedelta**(*self*, **args*, ***kwargs*)

Return Timedelta Array/Index as object ndarray of datetime.timedelta objects.

Returns

datetimes [ndarray]

pandas.TimedeltaIndex.to_series

TimedeltaIndex.**to_series**(*self*, *index=None*, *name=None*)

Create a Series with both index and values equal to the index keys.

Useful with map for returning an indexer based on an index.

Parameters

index [Index, optional] Index of resulting Series. If None, defaults to original index.

name [str, optional] Dame of resulting Series. If None, defaults to name of original index.

Returns

Series The dtype will be based on the type of the Index values.

pandas.TimedeltaIndex.round

TimedeltaIndex.**round**(*self*, **args*, ***kwargs*)

Perform round operation on the data to the specified *freq*.

Parameters

freq [str or Offset] The frequency level to round the index to. Must be a fixed frequency like 'S' (second) not 'ME' (month end). See *frequency aliases* for a list of possible *freq* values.

ambiguous ['infer', bool-ndarray, 'NaT', default 'raise'] Only relevant for DatetimeIndex:

- 'infer' will attempt to infer fall dst-transition hours based on order
- bool-ndarray where True signifies a DST time, False designates a non-DST time (note that this flag is only applicable for ambiguous times)
- 'NaT' will return NaT where there are ambiguous times
- 'raise' will raise an AmbiguousTimeError if there are ambiguous times.

New in version 0.24.0.

nonexistent ['shift_forward', 'shift_backward', 'NaT', timedelta, default 'raise'] A nonexistent time does not exist in a particular timezone where clocks moved forward due to DST.

- 'shift_forward' will shift the nonexistent time forward to the closest existing time
- 'shift_backward' will shift the nonexistent time backward to the closest existing time
- 'NaT' will return NaT where there are nonexistent times
- timedelta objects will shift nonexistent times by the timedelta
- 'raise' will raise an NonExistentTimeError if there are nonexistent times.

New in version 0.24.0.

Returns

DatetimeIndex, TimedeltaIndex, or Series Index of the same type for a DatetimeIndex or TimedeltaIndex, or a Series with the same index for a Series.

Raises

ValueError if the *freq* cannot be converted.

Examples

DatetimeIndex

```
>>> rng = pd.date_range('1/1/2018 11:59:00', periods=3, freq='min')
>>> rng
DatetimeIndex(['2018-01-01 11:59:00', '2018-01-01 12:00:00',
               '2018-01-01 12:01:00'],
              dtype='datetime64[ns]', freq='T')
>>> rng.round('H')
DatetimeIndex(['2018-01-01 12:00:00', '2018-01-01 12:00:00',
               '2018-01-01 12:00:00'],
              dtype='datetime64[ns]', freq=None)
```

Series

```
>>> pd.Series(rng).dt.round("H")
0   2018-01-01 12:00:00
1   2018-01-01 12:00:00
2   2018-01-01 12:00:00
dtype: datetime64[ns]
```

pandas.TimedeltaIndex.floor

TimedeltaIndex.**floor**(*self*, **args*, ***kwargs*)

Perform floor operation on the data to the specified *freq*.

Parameters

freq [str or Offset] The frequency level to floor the index to. Must be a fixed frequency like 'S' (second) not 'ME' (month end). See *frequency aliases* for a list of possible *freq* values.

ambiguous ['infer', bool-ndarray, 'NaT', default 'raise'] Only relevant for DatetimeIndex:

- 'infer' will attempt to infer fall dst-transition hours based on order
- bool-ndarray where True signifies a DST time, False designates a non-DST time (note that this flag is only applicable for ambiguous times)
- 'NaT' will return NaT where there are ambiguous times
- 'raise' will raise an AmbiguousTimeError if there are ambiguous times.

New in version 0.24.0.

nonexistent ['shift_forward', 'shift_backward', 'NaT', timedelta, default 'raise'] A nonexistent time does not exist in a particular timezone where clocks moved forward due to DST.

- 'shift_forward' will shift the nonexistent time forward to the closest existing time
- 'shift_backward' will shift the nonexistent time backward to the closest existing time
- 'NaT' will return NaT where there are nonexistent times
- timedelta objects will shift nonexistent times by the timedelta

- 'raise' will raise an NonExistentTimeError if there are nonexistent times.

New in version 0.24.0.

Returns

DatetimeIndex, TimedeltaIndex, or Series Index of the same type for a DatetimeIndex or TimedeltaIndex, or a Series with the same index for a Series.

Raises

ValueError if the *freq* cannot be converted.

Examples

DatetimeIndex

```
>>> rng = pd.date_range('1/1/2018 11:59:00', periods=3, freq='min')
>>> rng
DatetimeIndex(['2018-01-01 11:59:00', '2018-01-01 12:00:00',
               '2018-01-01 12:01:00'],
              dtype='datetime64[ns]', freq='T')
>>> rng.floor('H')
DatetimeIndex(['2018-01-01 11:00:00', '2018-01-01 12:00:00',
               '2018-01-01 12:00:00'],
              dtype='datetime64[ns]', freq=None)
```

Series

```
>>> pd.Series(rng).dt.floor("H")
0   2018-01-01 11:00:00
1   2018-01-01 12:00:00
2   2018-01-01 12:00:00
dtype: datetime64[ns]
```

pandas.TimedeltaIndex.ceil

`TimedeltaIndex.ceil(self, *args, **kwargs)`

Perform ceil operation on the data to the specified *freq*.

Parameters

freq [str or Offset] The frequency level to ceil the index to. Must be a fixed frequency like 'S' (second) not 'ME' (month end). See *frequency aliases* for a list of possible *freq* values.

ambiguous ['infer', bool-ndarray, 'NaT', default 'raise'] Only relevant for DatetimeIndex:

- 'infer' will attempt to infer fall dst-transition hours based on order
- bool-ndarray where True signifies a DST time, False designates a non-DST time (note that this flag is only applicable for ambiguous times)
- 'NaT' will return NaT where there are ambiguous times
- 'raise' will raise an AmbiguousTimeError if there are ambiguous times.

New in version 0.24.0.

nonexistent ['shift_forward', 'shift_backward', 'NaT', timedelta, default 'raise'] A nonexistent time does not exist in a particular timezone where clocks moved forward due to DST.

- 'shift_forward' will shift the nonexistent time forward to the closest existing time
- 'shift_backward' will shift the nonexistent time backward to the closest existing time
- 'NaT' will return NaT where there are nonexistent times
- timedelta objects will shift nonexistent times by the timedelta
- 'raise' will raise an NonExistentTimeError if there are nonexistent times.

New in version 0.24.0.

Returns

DatetimeIndex, TimedeltaIndex, or Series Index of the same type for a DatetimeIndex or TimedeltaIndex, or a Series with the same index for a Series.

Raises

ValueError if the *freq* cannot be converted.

Examples

DatetimeIndex

```
>>> rng = pd.date_range('1/1/2018 11:59:00', periods=3, freq='min')
>>> rng
DatetimeIndex(['2018-01-01 11:59:00', '2018-01-01 12:00:00',
               '2018-01-01 12:01:00'],
              dtype='datetime64[ns]', freq='T')
>>> rng.ceil('H')
DatetimeIndex(['2018-01-01 12:00:00', '2018-01-01 12:00:00',
               '2018-01-01 13:00:00'],
              dtype='datetime64[ns]', freq=None)
```

Series

```
>>> pd.Series(rng).dt.ceil("H")
0   2018-01-01 12:00:00
1   2018-01-01 12:00:00
2   2018-01-01 13:00:00
dtype: datetime64[ns]
```

pandas.TimedeltaIndex.to_frame

`TimedeltaIndex.`**`to_frame`**(*self*, *index=True*, *name=None*)

Create a DataFrame with a column containing the Index.

New in version 0.24.0.

Parameters

index [bool, default True] Set the index of the returned DataFrame as the original Index.

name [object, default None] The passed name should substitute for the index name (if it has one).

Returns

DataFrame DataFrame containing the original Index data.

See also:

Index.to_series Convert an Index to a Series.

Series.to_frame Convert Series to DataFrame.

Examples

```
>>> idx = pd.Index(['Ant', 'Bear', 'Cow'], name='animal')
>>> idx.to_frame()
       animal
animal
Ant       Ant
Bear     Bear
Cow       Cow
```

By default, the original Index is reused. To enforce a new Index:

```
>>> idx.to_frame(index=False)
    animal
0   Ant
1  Bear
2   Cow
```

To override the name of the resulting column, specify *name*:

```
>>> idx.to_frame(index=False, name='zoo')
    zoo
0   Ant
1  Bear
2   Cow
```

pandas.TimedeltaIndex.mean

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

Return the mean value of the Array.

New in version 0.25.0.

Parameters

skipna [bool, default True] Whether to ignore any NaT elements.

Returns

scalar Timestamp or Timedelta.

See also:

`numpy.ndarray.mean` Returns the average of array elements along a given axis.

Series.mean Return the mean value in a Series.

Notes

mean is only defined for Datetime and Timedelta dtypes, not for Period.

Components

`TimedeltaIndex.days`	Number of days for each element.
`TimedeltaIndex.seconds`	Number of seconds (>= 0 and less than 1 day) for each element.
`TimedeltaIndex.microseconds`	Number of microseconds (>= 0 and less than 1 second) for each element.
`TimedeltaIndex.nanoseconds`	Number of nanoseconds (>= 0 and less than 1 microsecond) for each element.
`TimedeltaIndex.components`	Return a dataframe of the components (days, hours, minutes, seconds, milliseconds, microseconds, nanoseconds) of the Timedeltas.
`TimedeltaIndex.inferred_freq`	Tryies to return a string representing a frequency guess, generated by infer_freq.

Conversion

`TimedeltaIndex.to_pytimedelta`(self, *args, ...)	Return Timedelta Array/Index as object ndarray of datetime.timedelta objects.
`TimedeltaIndex.to_series`(self[, index, name])	Create a Series with both index and values equal to the index keys.
`TimedeltaIndex.round`(self, *args, **kwargs)	Perform round operation on the data to the specified *freq*.
`TimedeltaIndex.floor`(self, *args, **kwargs)	Perform floor operation on the data to the specified *freq*.
`TimedeltaIndex.ceil`(self, *args, **kwargs)	Perform ceil operation on the data to the specified *freq*.
`TimedeltaIndex.to_frame`(self[, index, name])	Create a DataFrame with a column containing the Index.

Methods

`TimedeltaIndex.mean`(self, *args, **kwargs)	Return the mean value of the Array.

3.7.8 PeriodIndex

`PeriodIndex`([data, ordinal, freq, tz, ...])	Immutable ndarray holding ordinal values indicating regular periods in time.

pandas.PeriodIndex

class pandas.**PeriodIndex**(*data=None*, *ordinal=None*, *freq=None*, *tz=None*, *dtype=None*, *copy=False*, *name=None*, ***fields*)

Immutable ndarray holding ordinal values indicating regular periods in time.

Index keys are boxed to Period objects which carries the metadata (eg, frequency information).

Parameters

data [array-like (1d int np.ndarray or PeriodArray), optional] Optional period-like data to construct index with.

copy [bool] Make a copy of input ndarray.

freq [str or period object, optional] One of pandas period strings or corresponding objects

year [int, array, or Series, default None]

month [int, array, or Series, default None]

quarter [int, array, or Series, default None]

day [int, array, or Series, default None]

hour [int, array, or Series, default None]

minute [int, array, or Series, default None]

second [int, array, or Series, default None]

tz [object, default None] Timezone for converting datetime64 data to Periods.

dtype [str or PeriodDtype, default None]

See also:

Index The base pandas Index type.
Period Represents a period of time.
DatetimeIndex Index with datetime64 data.
TimedeltaIndex Index of timedelta64 data.
period_range Create a fixed-frequency PeriodIndex.

Examples

```
>>> idx = pd.PeriodIndex(year=year_arr, quarter=q_arr)
```

Attributes

day	The days of the period.
dayofweek	The day of the week with Monday=0, Sunday=6.
dayofyear	The ordinal day of the year.
days_in_month	The number of days in the month.
daysinmonth	The number of days in the month.
freq	Return the frequency object if it is set, otherwise None.
freqstr	Return the frequency object as a string if its set, otherwise None
hour	The hour of the period.

continues on next page

Table 176 – continued from previous page

is_leap_year	Logical indicating if the date belongs to a leap year.
minute	The minute of the period.
month	The month as January=1, December=12.
quarter	The quarter of the date.
second	The second of the period.
week	The week ordinal of the year.
weekday	The day of the week with Monday=0, Sunday=6.
weekofyear	The week ordinal of the year.
year	The year of the period.

pandas.PeriodIndex.day

property PeriodIndex.**day**
The days of the period.

pandas.PeriodIndex.dayofweek

property PeriodIndex.**dayofweek**
The day of the week with Monday=0, Sunday=6.

pandas.PeriodIndex.dayofyear

property PeriodIndex.**dayofyear**
The ordinal day of the year.

pandas.PeriodIndex.days_in_month

property PeriodIndex.**days_in_month**
The number of days in the month.

pandas.PeriodIndex.daysinmonth

property PeriodIndex.**daysinmonth**
The number of days in the month.

pandas.PeriodIndex.freq

property PeriodIndex.**freq**
Return the frequency object if it is set, otherwise None.

pandas.PeriodIndex.freqstr

property PeriodIndex.**freqstr**
Return the frequency object as a string if its set, otherwise None

pandas.PeriodIndex.hour

property PeriodIndex.**hour**
The hour of the period.

pandas.PeriodIndex.is_leap_year

property PeriodIndex.**is_leap_year**
Logical indicating if the date belongs to a leap year.

pandas.PeriodIndex.minute

property PeriodIndex.**minute**
The minute of the period.

pandas.PeriodIndex.month

property PeriodIndex.**month**
The month as January=1, December=12.

pandas.PeriodIndex.quarter

property PeriodIndex.**quarter**
The quarter of the date.

pandas.PeriodIndex.second

property PeriodIndex.**second**
The second of the period.

pandas.PeriodIndex.week

property PeriodIndex.**week**
The week ordinal of the year.

pandas.PeriodIndex.weekday

property PeriodIndex.**weekday**
The day of the week with Monday=0, Sunday=6.

pandas.PeriodIndex.weekofyear

property PeriodIndex.**weekofyear**
The week ordinal of the year.

pandas.PeriodIndex.year

property PeriodIndex.**year**
The year of the period.

end_time	
qyear	
start_time	

Methods

asfreq(self, *args, **kwargs)	Convert the Period Array/Index to the specified frequency *freq*.
strftime(self, *args, **kwargs)	Convert to Index using specified date_format.
to_timestamp(self, *args, **kwargs)	Cast to DatetimeArray/Index.

pandas.PeriodIndex.asfreq

PeriodIndex.**asfreq**(*self*, **args*, ***kwargs*)
Convert the Period Array/Index to the specified frequency *freq*.

Parameters

freq [str] A frequency.

how [str {'E', 'S'}] Whether the elements should be aligned to the end or start within pa period.

- 'E', 'END', or 'FINISH' for end,
- 'S', 'START', or 'BEGIN' for start.

January 31st ('END') vs. January 1st ('START') for example.

Returns

Period Array/Index Constructed with the new frequency.

Examples

```
>>> pidx = pd.period_range('2010-01-01', '2015-01-01', freq='A')
>>> pidx
PeriodIndex(['2010', '2011', '2012', '2013', '2014', '2015'],
dtype='period[A-DEC]', freq='A-DEC')
```

```
>>> pidx.asfreq('M')
PeriodIndex(['2010-12', '2011-12', '2012-12', '2013-12', '2014-12',
'2015-12'], dtype='period[M]', freq='M')
```

```
>>> pidx.asfreq('M', how='S')
PeriodIndex(['2010-01', '2011-01', '2012-01', '2013-01', '2014-01',
'2015-01'], dtype='period[M]', freq='M')
```

pandas.PeriodIndex.strftime

PeriodIndex.**strftime**(*self*, **args*, ***kwargs*)

Convert to Index using specified date_format.

Return an Index of formatted strings specified by date_format, which supports the same string format as the python standard library. Details of the string format can be found in python string format doc.

Parameters

date_format [str] Date format string (e.g. "%Y-%m-%d").

Returns

ndarray NumPy ndarray of formatted strings.

See also:

to_datetime Convert the given argument to datetime.

DatetimeIndex.normalize Return DatetimeIndex with times to midnight.

DatetimeIndex.round Round the DatetimeIndex to the specified freq.

DatetimeIndex.floor Floor the DatetimeIndex to the specified freq.

Examples

```
>>> rng = pd.date_range(pd.Timestamp("2018-03-10 09:00"),
...                     periods=3, freq='s')
>>> rng.strftime('%B %d, %Y, %r')
Index(['March 10, 2018, 09:00:00 AM', 'March 10, 2018, 09:00:01 AM',
       'March 10, 2018, 09:00:02 AM'],
      dtype='object')
```

pandas.PeriodIndex.to_timestamp

PeriodIndex.**to_timestamp**(*self*, **args*, ***kwargs*)

Cast to DatetimeArray/Index.

Parameters

freq [str or DateOffset, optional] Target frequency. The default is 'D' for week or longer, 'S' otherwise.

how [{'s', 'e', 'start', 'end'}] Whether to use the start or end of the time period being converted.

Returns

DatetimeArray/Index

Properties

PeriodIndex.day	The days of the period.
PeriodIndex.dayofweek	The day of the week with Monday=0, Sunday=6.
PeriodIndex.dayofyear	The ordinal day of the year.
PeriodIndex.days_in_month	The number of days in the month.
PeriodIndex.daysinmonth	The number of days in the month.
PeriodIndex.end_time	
PeriodIndex.freq	Return the frequency object if it is set, otherwise None.
PeriodIndex.freqstr	Return the frequency object as a string if its set, otherwise None
PeriodIndex.hour	The hour of the period.
PeriodIndex.is_leap_year	Logical indicating if the date belongs to a leap year.
PeriodIndex.minute	The minute of the period.
PeriodIndex.month	The month as January=1, December=12.
PeriodIndex.quarter	The quarter of the date.
PeriodIndex.qyear	
PeriodIndex.second	The second of the period.
PeriodIndex.start_time	
PeriodIndex.week	The week ordinal of the year.
PeriodIndex.weekday	The day of the week with Monday=0, Sunday=6.
PeriodIndex.weekofyear	The week ordinal of the year.
PeriodIndex.year	The year of the period.

pandas.PeriodIndex.end_time

property PeriodIndex.**end_time**

pandas.PeriodIndex.qyear

property PeriodIndex.**qyear**

pandas.PeriodIndex.start_time

property PeriodIndex.**start_time**

Methods

PeriodIndex.asfreq(self, *args, **kwargs)	Convert the Period Array/Index to the specified frequency *freq*.
PeriodIndex.strftime(self, *args, **kwargs)	Convert to Index using specified date_format.
PeriodIndex.to_timestamp(self, *args, **kwargs)	Cast to DatetimeArray/Index.

3.8 Date offsets

3.8.1 DateOffset

DateOffset([n, normalize])	Standard kind of date increment used for a date range.

pandas.tseries.offsets.DateOffset

class pandas.tseries.offsets.**DateOffset**(*n=1*, *normalize=False*, ***kwds*)

Standard kind of date increment used for a date range.

Works exactly like relativedelta in terms of the keyword args you pass in, use of the keyword n is discouraged– you would be better off specifying n in the keywords you use, but regardless it is there for you. n is needed for DateOffset subclasses.

DateOffset work as follows. Each offset specify a set of dates that conform to the DateOffset. For example, Bday defines this set to be the set of dates that are weekdays (M-F). To test if a date is in the set of a DateOffset dateOffset we can use the is_on_offset method: dateOffset.is_on_offset(date).

If a date is not on a valid date, the rollback and rollforward methods can be used to roll the date to the nearest valid date before/after the date.

DateOffsets can be created to move dates forward a given number of valid dates. For example, Bday(2) can be added to a date to move it two business days forward. If the date does not start on a valid date, first it is moved to a valid date. Thus pseudo code is:
def __add__(date): date = rollback(date) # does nothing if date is valid return date + <n number of periods>
When a date offset is created for a negative number of periods, the date is first rolled forward. The pseudo code is:
def __add__(date): date = rollforward(date) # does nothing is date is valid return date + <n number of periods>
Zero presents a problem. Should it roll forward or back? We arbitrarily have it rollforward:

date + BDay(0) == BDay.rollforward(date)

Since 0 is a bit weird, we suggest avoiding its use.

Parameters

n [int, default 1] The number of time periods the offset represents.

normalize [bool, default False] Whether to round the result of a DateOffset addition down to the previous midnight.

****kwds** Temporal parameter that add to or replace the offset value.

Parameters that **add** to the offset (like Timedelta):

- years
- months
- weeks
- days
- hours
- minutes
- seconds
- microseconds
- nanoseconds

Parameters that **replace** the offset value:

- year
- month
- day
- weekday
- hour
- minute
- second
- microsecond
- nanosecond.

See also:

`dateutil.relativedelta.relativedelta` The relativedelta type is designed to be applied to an existing datetime an can replace specific components of that datetime, or represents an interval of time.

Examples

```
>>> from pandas.tseries.offsets import DateOffset
>>> ts = pd.Timestamp('2017-01-01 09:10:11')
>>> ts + DateOffset(months=3)
Timestamp('2017-04-01 09:10:11')
```

```
>>> ts = pd.Timestamp('2017-01-01 09:10:11')
>>> ts + DateOffset(months=2)
Timestamp('2017-03-01 09:10:11')
```

Attributes

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

pandas.tseries.offsets.DateOffset.base

property DateOffset.**base**

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

freqstr	
kwds	
name	
nanos	
rule_code	

Methods

apply_index(self, other)	Vectorized apply of DateOffset to DatetimeIndex, raises NotImplentedError for offsets without a vectorized implementation.
rollback(self, dt)	Roll provided date backward to next offset only if not on offset.
rollforward(self, dt)	Roll provided date forward to next offset only if not on offset.

pandas.tseries.offsets.DateOffset.apply_index

DateOffset.**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.DateOffset.rollback

DateOffset.**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.DateOffset.rollforward

DateOffset.**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.

__call__	
apply	
copy	
isAnchored	
is_anchored	
is_on_offset	
onOffset	

Properties

DateOffset.freqstr
DateOffset.kwds
DateOffset.name
DateOffset.nanos
DateOffset.normalize
DateOffset.rule_code

pandas.tseries.offsets.DateOffset.freqstr

DateOffset.**freqstr**

pandas.tseries.offsets.DateOffset.kwds

property DateOffset.**kwds**

pandas.tseries.offsets.DateOffset.name

property DateOffset.**name**

pandas.tseries.offsets.DateOffset.nanos

property DateOffset.**nanos**

pandas.tseries.offsets.DateOffset.normalize

DateOffset.**normalize = False**

pandas.tseries.offsets.DateOffset.rule_code

property DateOffset.**rule_code**

Methods

DateOffset.apply(self, other)	
DateOffset.copy(self)	
DateOffset.isAnchored(self)	
DateOffset.onOffset(self, dt)	
DateOffset.is_anchored(self)	
DateOffset.is_on_offset(self, dt)	
DateOffset.__call__(self, other)	Call self as a function.

pandas.tseries.offsets.DateOffset.apply

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

pandas.tseries.offsets.DateOffset.copy

DateOffset.**copy**(*self*)

pandas.tseries.offsets.DateOffset.isAnchored

DateOffset.**isAnchored**(*self*)

pandas.tseries.offsets.DateOffset.onOffset

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

pandas.tseries.offsets.DateOffset.is_anchored

DateOffset.**is_anchored**(*self*)

pandas.tseries.offsets.DateOffset.is_on_offset

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

pandas.tseries.offsets.DateOffset.__call__

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

3.8.2 BusinessDay

BusinessDay([n, normalize, offset])	DateOffset subclass representing possibly n business days.

pandas.tseries.offsets.BusinessDay

class pandas.tseries.offsets.**BusinessDay**(*n=1*, *normalize=False*, *offset=datetime.timedelta(0)*)
DateOffset subclass representing possibly n business days.

Attributes

base	Returns a copy of the calling offset object with n=1 and all other attributes equal.
offset	Alias for self._offset.

pandas.tseries.offsets.BusinessDay.base

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

pandas.tseries.offsets.BusinessDay.offset

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

freqstr	
kwds	
name	
nanos	
rule_code	

Methods

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

pandas.tseries.offsets.BusinessDay.rollback

BusinessDay.**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.BusinessDay.rollforward

BusinessDay.**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.

__call__	
apply	
apply_index	
copy	
isAnchored	
is_anchored	
is_on_offset	
onOffset	

Properties

BusinessDay.freqstr
BusinessDay.kwds
BusinessDay.name
BusinessDay.nanos
BusinessDay.normalize
BusinessDay.rule_code

pandas.tseries.offsets.BusinessDay.freqstr

BusinessDay.**freqstr**

pandas.tseries.offsets.BusinessDay.kwds

property BusinessDay.**kwds**