3.7.6 DatetimeIndex

DatetimeIndex([data, freq, tz, normalize, ...])	Immutable ndarray of datetime64 data, represented internally as int64, and which can be boxed to Timestamp objects that are subclasses of datetime and carry metadata such as frequency information.

pandas.DatetimeIndex

class pandas.**DatetimeIndex**(*data=None*, *freq=None*, *tz=None*, *normalize=False*, *closed=None*, *ambiguous='raise'*, *dayfirst=False*, *yearfirst=False*, *dtype=None*, *copy=False*, *name=None*)

Immutable ndarray of datetime64 data, represented internally as int64, and which can be boxed to Timestamp objects that are subclasses of datetime and carry metadata such as frequency information.

Parameters

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

copy [bool] Make a copy of input ndarray.

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.

tz [pytz.timezone or dateutil.tz.tzfile]

ambiguous ['infer', bool-ndarray, 'NaT', default 'raise'] When clocks moved backward due to DST, ambiguous times may arise. For example in Central European Time (UTC+01), when going from 03:00 DST to 02:00 non-DST, 02:30:00 local time occurs both at 00:30:00 UTC and at 01:30:00 UTC. In such a situation, the *ambiguous* parameter dictates how ambiguous times should be handled.

- 'infer' will attempt to infer fall dst-transition hours based on order
- bool-ndarray where True signifies a DST time, False signifies 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.

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

dayfirst [bool, default False] If True, parse dates in *data* with the day first order.

yearfirst [bool, default False] If True parse dates in *data* with the year first order.

See also:

Index The base pandas Index type.
TimedeltaIndex Index of timedelta64 data.
PeriodIndex Index of Period data.
to_datetime Convert argument to datetime.
date_range Create a fixed-frequency DatetimeIndex.

Notes

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

Attributes

year	The year of the datetime.
month	The month as January=1, December=12.
day	The month as January=1, December=12.
hour	The hours of the datetime.
minute	The minutes of the datetime.
second	The seconds of the datetime.
microsecond	The microseconds of the datetime.
nanosecond	The nanoseconds of the datetime.
date	Returns numpy array of python datetime.date objects (namely, the date part of Timestamps without timezone information).
time	Returns numpy array of datetime.time.
timetz	Returns numpy array of datetime.time also containing timezone information.
dayofyear	The ordinal day of the year.
weekofyear	The week ordinal of the year.
week	The week ordinal of the year.
dayofweek	The day of the week with Monday=0, Sunday=6.
weekday	The day of the week with Monday=0, Sunday=6.
quarter	The quarter of the date.
tz	Return timezone, if any.
freq	Return the frequency object if it is set, otherwise None.
freqstr	Return the frequency object as a string if its set, otherwise None
is_month_start	Indicates whether the date is the first day of the month.
is_month_end	Indicates whether the date is the last day of the month.
is_quarter_start	Indicator for whether the date is the first day of a quarter.
is_quarter_end	Indicator for whether the date is the last day of a quarter.
is_year_start	Indicate whether the date is the first day of a year.
is_year_end	Indicate whether the date is the last day of the year.
is_leap_year	Boolean indicator if the date belongs to a leap year.
inferred_freq	Tryies to return a string representing a frequency guess, generated by infer_freq.

pandas.DatetimeIndex.year

property DatetimeIndex.**year**
The year of the datetime.

pandas.DatetimeIndex.month

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

pandas.DatetimeIndex.day

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

pandas.DatetimeIndex.hour

property DatetimeIndex.**hour**
The hours of the datetime.

pandas.DatetimeIndex.minute

property DatetimeIndex.**minute**
The minutes of the datetime.

pandas.DatetimeIndex.second

property DatetimeIndex.**second**
The seconds of the datetime.

pandas.DatetimeIndex.microsecond

property DatetimeIndex.**microsecond**
The microseconds of the datetime.

pandas.DatetimeIndex.nanosecond

property DatetimeIndex.**nanosecond**
The nanoseconds of the datetime.

pandas.DatetimeIndex.date

property DatetimeIndex.**date**
Returns numpy array of python datetime.date objects (namely, the date part of Timestamps without timezone information).

pandas.DatetimeIndex.time

property DatetimeIndex.**time**
Returns numpy array of datetime.time. The time part of the Timestamps.

pandas.DatetimeIndex.timetz

property DatetimeIndex.**timetz**
Returns numpy array of datetime.time also containing timezone information. The time part of the Timestamps.

pandas.DatetimeIndex.dayofyear

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

pandas.DatetimeIndex.weekofyear

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

pandas.DatetimeIndex.week

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

pandas.DatetimeIndex.dayofweek

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

Return the day of the week. It is assumed the week starts on Monday, which is denoted by 0 and ends on Sunday which is denoted by 6. This method is available on both Series with datetime values (using the *dt* accessor) or DatetimeIndex.

Returns

Series or Index Containing integers indicating the day number.

See also:

Series.dt.dayofweek Alias.

Series.dt.weekday Alias.

Series.dt.day_name Returns the name of the day of the week.

Examples

```
>>> s = pd.date_range('2016-12-31', '2017-01-08', freq='D').to_series()
>>> s.dt.dayofweek
2016-12-31    5
2017-01-01    6
2017-01-02    0
2017-01-03    1
2017-01-04    2
2017-01-05    3
2017-01-06    4
2017-01-07    5
2017-01-08    6
Freq: D, dtype: int64
```

pandas.DatetimeIndex.weekday

property DatetimeIndex.**weekday**

The day of the week with Monday=0, Sunday=6.

Return the day of the week. It is assumed the week starts on Monday, which is denoted by 0 and ends on Sunday which is denoted by 6. This method is available on both Series with datetime values (using the *dt* accessor) or DatetimeIndex.

Returns

Series or Index Containing integers indicating the day number.

See also:

Series.dt.dayofweek Alias.

Series.dt.weekday Alias.

Series.dt.day_name Returns the name of the day of the week.

Examples

```
>>> s = pd.date_range('2016-12-31', '2017-01-08', freq='D').to_series()
>>> s.dt.dayofweek
2016-12-31    5
2017-01-01    6
2017-01-02    0
2017-01-03    1
2017-01-04    2
2017-01-05    3
2017-01-06    4
2017-01-07    5
2017-01-08    6
Freq: D, dtype: int64
```

pandas.DatetimeIndex.quarter

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

pandas.DatetimeIndex.tz

property DatetimeIndex.**tz**
Return timezone, if any.

Returns

datetime.tzinfo, pytz.tzinfo.BaseTZInfo, dateutil.tz.tz.tzfile, or None Returns None when the array is tz-naive.

pandas.DatetimeIndex.freq

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

pandas.DatetimeIndex.freqstr

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

pandas.DatetimeIndex.is_month_start

property DatetimeIndex.**is_month_start**
Indicates whether the date is the first day of the month.

Returns

Series or array For Series, returns a Series with boolean values. For DatetimeIndex, returns a boolean array.

See also:

is_month_start Return a boolean indicating whether the date is the first day of the month.

is_month_end Return a boolean indicating whether the date is the last day of the month.

Examples

This method is available on Series with datetime values under the `.dt` accessor, and directly on DatetimeIndex.

```
>>> s = pd.Series(pd.date_range("2018-02-27", periods=3))
>>> s
0   2018-02-27
1   2018-02-28
2   2018-03-01
dtype: datetime64[ns]
```

(continues on next page)

(continued from previous page)

```
>>> s.dt.is_month_start
0    False
1    False
2    True
dtype: bool
>>> s.dt.is_month_end
0    False
1    True
2    False
dtype: bool
```

```
>>> idx = pd.date_range("2018-02-27", periods=3)
>>> idx.is_month_start
array([False, False, True])
>>> idx.is_month_end
array([False, True, False])
```

pandas.DatetimeIndex.is_month_end

property DatetimeIndex.**is_month_end**

Indicates whether the date is the last day of the month.

Returns

Series or array For Series, returns a Series with boolean values. For DatetimeIndex, returns a boolean array.

See also:

is_month_start Return a boolean indicating whether the date is the first day of the month.

is_month_end Return a boolean indicating whether the date is the last day of the month.

Examples

This method is available on Series with datetime values under the `.dt` accessor, and directly on DatetimeIndex.

```
>>> s = pd.Series(pd.date_range("2018-02-27", periods=3))
>>> s
0   2018-02-27
1   2018-02-28
2   2018-03-01
dtype: datetime64[ns]
>>> s.dt.is_month_start
0    False
1    False
2    True
dtype: bool
>>> s.dt.is_month_end
0    False
1    True
2    False
dtype: bool
```

```
>>> idx = pd.date_range("2018-02-27", periods=3)
>>> idx.is_month_start
array([False, False, True])
>>> idx.is_month_end
array([False, True, False])
```

pandas.DatetimeIndex.is_quarter_start

property DatetimeIndex.**is_quarter_start**

Indicator for whether the date is the first day of a quarter.

Returns

is_quarter_start [Series or DatetimeIndex] The same type as the original data with boolean values. Series will have the same name and index. DatetimeIndex will have the same name.

See also:

quarter Return the quarter of the date.

is_quarter_end Similar property for indicating the quarter start.

Examples

This method is available on Series with datetime values under the `.dt` accessor, and directly on DatetimeIndex.

```
>>> df = pd.DataFrame({'dates': pd.date_range("2017-03-30",
...                   periods=4)})
>>> df.assign(quarter=df.dates.dt.quarter,
...           is_quarter_start=df.dates.dt.is_quarter_start)
       dates  quarter  is_quarter_start
0 2017-03-30        1             False
1 2017-03-31        1             False
2 2017-04-01        2              True
3 2017-04-02        2             False
```

```
>>> idx = pd.date_range('2017-03-30', periods=4)
>>> idx
DatetimeIndex(['2017-03-30', '2017-03-31', '2017-04-01', '2017-04-02'],
              dtype='datetime64[ns]', freq='D')
```

```
>>> idx.is_quarter_start
array([False, False,  True, False])
```

pandas.DatetimeIndex.is_quarter_end

property DatetimeIndex.**is_quarter_end**

Indicator for whether the date is the last day of a quarter.

Returns

is_quarter_end [Series or DatetimeIndex] The same type as the original data with boolean values. Series will have the same name and index. DatetimeIndex will have the same name.

See also:

quarter Return the quarter of the date.

is_quarter_start Similar property indicating the quarter start.

Examples

This method is available on Series with datetime values under the `.dt` accessor, and directly on DatetimeIndex.

```
>>> df = pd.DataFrame({'dates': pd.date_range("2017-03-30",
...                   periods=4)})
>>> df.assign(quarter=df.dates.dt.quarter,
...           is_quarter_end=df.dates.dt.is_quarter_end)
       dates  quarter    is_quarter_end
0 2017-03-30        1             False
1 2017-03-31        1              True
2 2017-04-01        2             False
3 2017-04-02        2             False
```

```
>>> idx = pd.date_range('2017-03-30', periods=4)
>>> idx
DatetimeIndex(['2017-03-30', '2017-03-31', '2017-04-01', '2017-04-02'],
              dtype='datetime64[ns]', freq='D')
```

```
>>> idx.is_quarter_end
array([False,  True, False, False])
```

pandas.DatetimeIndex.is_year_start

property DatetimeIndex.**is_year_start**

Indicate whether the date is the first day of a year.

Returns

Series or DatetimeIndex The same type as the original data with boolean values. Series will have the same name and index. DatetimeIndex will have the same name.

See also:

is_year_end Similar property indicating the last day of the year.

Examples

This method is available on Series with datetime values under the `.dt` accessor, and directly on DatetimeIndex.

```
>>> dates = pd.Series(pd.date_range("2017-12-30", periods=3))
>>> dates
0   2017-12-30
1   2017-12-31
2   2018-01-01
dtype: datetime64[ns]
```

```
>>> dates.dt.is_year_start
0    False
1    False
2    True
dtype: bool
```

```
>>> idx = pd.date_range("2017-12-30", periods=3)
>>> idx
DatetimeIndex(['2017-12-30', '2017-12-31', '2018-01-01'],
              dtype='datetime64[ns]', freq='D')
```

```
>>> idx.is_year_start
array([False, False,  True])
```

pandas.DatetimeIndex.is_year_end

property DatetimeIndex.**is_year_end**

Indicate whether the date is the last day of the year.

Returns

Series or DatetimeIndex The same type as the original data with boolean values. Series will have the same name and index. DatetimeIndex will have the same name.

See also:

is_year_start Similar property indicating the start of the year.

Examples

This method is available on Series with datetime values under the `.dt` accessor, and directly on DatetimeIndex.

```
>>> dates = pd.Series(pd.date_range("2017-12-30", periods=3))
>>> dates
0   2017-12-30
1   2017-12-31
2   2018-01-01
dtype: datetime64[ns]
```

```
>>> dates.dt.is_year_end
0    False
1     True
2    False
dtype: bool
```

```
>>> idx = pd.date_range("2017-12-30", periods=3)
>>> idx
DatetimeIndex(['2017-12-30', '2017-12-31', '2018-01-01'],
              dtype='datetime64[ns]', freq='D')
```

```
>>> idx.is_year_end
array([False,  True, False])
```

pandas.DatetimeIndex.is_leap_year

property DatetimeIndex.**is_leap_year**

Boolean indicator if the date belongs to a leap year.

A leap year is a year, which has 366 days (instead of 365) including 29th of February as an intercalary day. Leap years are years which are multiples of four with the exception of years divisible by 100 but not by 400.

Returns

Series or ndarray Booleans indicating if dates belong to a leap year.

Examples

This method is available on Series with datetime values under the `.dt` accessor, and directly on DatetimeIndex.

```
>>> idx = pd.date_range("2012-01-01", "2015-01-01", freq="Y")
>>> idx
DatetimeIndex(['2012-12-31', '2013-12-31', '2014-12-31'],
              dtype='datetime64[ns]', freq='A-DEC')
>>> idx.is_leap_year
array([ True, False, False], dtype=bool)
```

```
>>> dates = pd.Series(idx)
>>> dates_series
0   2012-12-31
1   2013-12-31
2   2014-12-31
dtype: datetime64[ns]
>>> dates_series.dt.is_leap_year
0     True
1    False
2    False
dtype: bool
```

pandas.DatetimeIndex.inferred_freq

DatetimeIndex.**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

normalize(self, *args, **kwargs)	Convert times to midnight.
strftime(self, *args, **kwargs)	Convert to Index using specified date_format.
snap(self[, freq])	Snap time stamps to nearest occurring frequency.
tz_convert(self, *args, **kwargs)	Convert tz-aware Datetime Array/Index from one time zone to another.
tz_localize(self, *args, **kwargs)	Localize tz-naive Datetime Array/Index to tz-aware Datetime Array/Index.
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_period(self, *args, **kwargs)	Cast to PeriodArray/Index at a particular frequency.
to_perioddelta(self, *args, **kwargs)	Calculate TimedeltaArray of difference between index values and index converted to PeriodArray at specified freq.
to_pydatetime(self, *args, **kwargs)	Return Datetime Array/Index as object ndarray of datetime.datetime objects.
to_series(self[, keep_tz, index, name])	Create a Series with both index and values equal to the index keys useful with map for returning an indexer based on an index.
to_frame(self[, index, name])	Create a DataFrame with a column containing the Index.
month_name(self, *args, **kwargs)	Return the month names of the DateTimeIndex with specified locale.
day_name(self, *args, **kwargs)	Return the day names of the DateTimeIndex with specified locale.
mean(self, *args, **kwargs)	Return the mean value of the Array.

pandas.DatetimeIndex.normalize

DatetimeIndex.**normalize**(*self*, **args*, ***kwargs*)
Convert times to midnight.

The time component of the date-time is converted to midnight i.e. 00:00:00. This is useful in cases, when the time does not matter. Length is unaltered. The timezones are unaffected.

This method is available on Series with datetime values under the `.dt` accessor, and directly on Datetime Array/Index.

Returns

DatetimeArray, DatetimeIndex or Series The same type as the original data. Series

will have the same name and index. DatetimeIndex will have the same name.

See also:

floor Floor the datetimes to the specified freq.

ceil Ceil the datetimes to the specified freq.

round Round the datetimes to the specified freq.

Examples

```
>>> idx = pd.date_range(start='2014-08-01 10:00', freq='H',
...                     periods=3, tz='Asia/Calcutta')
>>> idx
DatetimeIndex(['2014-08-01 10:00:00+05:30',
               '2014-08-01 11:00:00+05:30',
               '2014-08-01 12:00:00+05:30'],
                dtype='datetime64[ns, Asia/Calcutta]', freq='H')
>>> idx.normalize()
DatetimeIndex(['2014-08-01 00:00:00+05:30',
               '2014-08-01 00:00:00+05:30',
               '2014-08-01 00:00:00+05:30'],
              dtype='datetime64[ns, Asia/Calcutta]', freq=None)
```

pandas.DatetimeIndex.strftime

DatetimeIndex.**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.DatetimeIndex.snap

DatetimeIndex.**snap**(*self*, *freq='S'*)

Snap time stamps to nearest occurring frequency.

Returns

DatetimeIndex

pandas.DatetimeIndex.tz_convert

DatetimeIndex.**tz_convert**(*self*, **args*, ***kwargs*)

Convert tz-aware Datetime Array/Index from one time zone to another.

Parameters

tz [str, pytz.timezone, dateutil.tz.tzfile or None] Time zone for time. Corresponding timestamps would be converted to this time zone of the Datetime Array/Index. A *tz* of None will convert to UTC and remove the timezone information.

Returns

Array or Index

Raises

TypeError If Datetime Array/Index is tz-naive.

See also:

DatetimeIndex.tz A timezone that has a variable offset from UTC.

DatetimeIndex.tz_localize Localize tz-naive DatetimeIndex to a given time zone, or remove timezone from a tz-aware DatetimeIndex.

Examples

With the *tz* parameter, we can change the DatetimeIndex to other time zones:

```
>>> dti = pd.date_range(start='2014-08-01 09:00',
...                     freq='H', periods=3, tz='Europe/Berlin')
```

```
>>> dti
DatetimeIndex(['2014-08-01 09:00:00+02:00',
               '2014-08-01 10:00:00+02:00',
               '2014-08-01 11:00:00+02:00'],
              dtype='datetime64[ns, Europe/Berlin]', freq='H')
```

```
>>> dti.tz_convert('US/Central')
DatetimeIndex(['2014-08-01 02:00:00-05:00',
               '2014-08-01 03:00:00-05:00',
               '2014-08-01 04:00:00-05:00'],
              dtype='datetime64[ns, US/Central]', freq='H')
```

With the `tz=None`, we can remove the timezone (after converting to UTC if necessary):

```
>>> dti = pd.date_range(start='2014-08-01 09:00', freq='H',
...                     periods=3, tz='Europe/Berlin')
```

```
>>> dti
DatetimeIndex(['2014-08-01 09:00:00+02:00',
               '2014-08-01 10:00:00+02:00',
               '2014-08-01 11:00:00+02:00'],
               dtype='datetime64[ns, Europe/Berlin]', freq='H')
```

```
>>> dti.tz_convert(None)
DatetimeIndex(['2014-08-01 07:00:00',
               '2014-08-01 08:00:00',
               '2014-08-01 09:00:00'],
               dtype='datetime64[ns]', freq='H')
```

pandas.DatetimeIndex.tz_localize

DatetimeIndex.**tz_localize**(*self*, **args*, ***kwargs*)

Localize tz-naive Datetime Array/Index to tz-aware Datetime Array/Index.

This method takes a time zone (tz) naive Datetime Array/Index object and makes this time zone aware. It does not move the time to another time zone. Time zone localization helps to switch from time zone aware to time zone unaware objects.

Parameters

tz [str, pytz.timezone, dateutil.tz.tzfile or None] Time zone to convert timestamps to. Passing `None` will remove the time zone information preserving local time.

ambiguous ['infer', 'NaT', bool array, default 'raise'] When clocks moved backward due to DST, ambiguous times may arise. For example in Central European Time (UTC+01), when going from 03:00 DST to 02:00 non-DST, 02:30:00 local time occurs both at 00:30:00 UTC and at 01:30:00 UTC. In such a situation, the *ambiguous* parameter dictates how ambiguous times should be handled.

- 'infer' will attempt to infer fall dst-transition hours based on order
- bool-ndarray where True signifies a DST time, False signifies 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.

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

Same type as self Array/Index converted to the specified time zone.

Raises

TypeError If the Datetime Array/Index is tz-aware and tz is not None.

See also:

DatetimeIndex.tz_convert Convert tz-aware DatetimeIndex from one time zone to another.

Examples

```
>>> tz_naive = pd.date_range('2018-03-01 09:00', periods=3)
>>> tz_naive
DatetimeIndex(['2018-03-01 09:00:00', '2018-03-02 09:00:00',
               '2018-03-03 09:00:00'],
              dtype='datetime64[ns]', freq='D')
```

Localize DatetimeIndex in US/Eastern time zone:

```
>>> tz_aware = tz_naive.tz_localize(tz='US/Eastern')
>>> tz_aware
DatetimeIndex(['2018-03-01 09:00:00-05:00',
               '2018-03-02 09:00:00-05:00',
               '2018-03-03 09:00:00-05:00'],
              dtype='datetime64[ns, US/Eastern]', freq='D')
```

With the `tz=None`, we can remove the time zone information while keeping the local time (not converted to UTC):

```
>>> tz_aware.tz_localize(None)
DatetimeIndex(['2018-03-01 09:00:00', '2018-03-02 09:00:00',
               '2018-03-03 09:00:00'],
              dtype='datetime64[ns]', freq='D')
```

Be careful with DST changes. When there is sequential data, pandas can infer the DST time:

```
>>> s = pd.to_datetime(pd.Series(['2018-10-28 01:30:00',
...                               '2018-10-28 02:00:00',
...                               '2018-10-28 02:30:00',
...                               '2018-10-28 02:00:00',
...                               '2018-10-28 02:30:00',
...                               '2018-10-28 03:00:00',
...                               '2018-10-28 03:30:00']))
>>> s.dt.tz_localize('CET', ambiguous='infer')
0   2018-10-28 01:30:00+02:00
```

(continues on next page)

(continued from previous page)

```
1   2018-10-28 02:00:00+02:00
2   2018-10-28 02:30:00+02:00
3   2018-10-28 02:00:00+01:00
4   2018-10-28 02:30:00+01:00
5   2018-10-28 03:00:00+01:00
6   2018-10-28 03:30:00+01:00
dtype: datetime64[ns, CET]
```

In some cases, inferring the DST is impossible. In such cases, you can pass an ndarray to the ambiguous parameter to set the DST explicitly

```
>>> s = pd.to_datetime(pd.Series(['2018-10-28 01:20:00',
...                               '2018-10-28 02:36:00',
...                               '2018-10-28 03:46:00']))
>>> s.dt.tz_localize('CET', ambiguous=np.array([True, True, False]))
0   2015-03-29 03:00:00+02:00
1   2015-03-29 03:30:00+02:00
dtype: datetime64[ns, Europe/Warsaw]
```

If the DST transition causes nonexistent times, you can shift these dates forward or backwards with a timedelta object or *'shift_forward'* or *'shift_backwards'*.

```
>>> s = pd.to_datetime(pd.Series(['2015-03-29 02:30:00',
...                               '2015-03-29 03:30:00']))
>>> s.dt.tz_localize('Europe/Warsaw', nonexistent='shift_forward')
0   2015-03-29 03:00:00+02:00
1   2015-03-29 03:30:00+02:00
dtype: datetime64[ns, 'Europe/Warsaw']
>>> s.dt.tz_localize('Europe/Warsaw', nonexistent='shift_backward')
0   2015-03-29 01:59:59.999999999+01:00
1   2015-03-29 03:30:00+02:00
dtype: datetime64[ns, 'Europe/Warsaw']
>>> s.dt.tz_localize('Europe/Warsaw', nonexistent=pd.Timedelta('1H'))
0   2015-03-29 03:30:00+02:00
1   2015-03-29 03:30:00+02:00
dtype: datetime64[ns, 'Europe/Warsaw']
```

pandas.DatetimeIndex.round

DatetimeIndex.**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.DatetimeIndex.floor

DatetimeIndex.**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')
```

(continues on next page)

(continued from previous page)

```
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.DatetimeIndex.ceil

DatetimeIndex.**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.DatetimeIndex.to_period

DatetimeIndex.**to_period**(*self*, **args*, ***kwargs*)

Cast to PeriodArray/Index at a particular frequency.

Converts DatetimeArray/Index to PeriodArray/Index.

Parameters

freq [str or Offset, optional] One of pandas' *offset strings* or an Offset object. Will be inferred by default.

Returns

PeriodArray/Index

Raises

ValueError When converting a DatetimeArray/Index with non-regular values, so that a frequency cannot be inferred.

See also:

PeriodIndex Immutable ndarray holding ordinal values.

DatetimeIndex.to_pydatetime Return DatetimeIndex as object.

Examples

```
>>> df = pd.DataFrame({"y": [1, 2, 3]},
...                   index=pd.to_datetime(["2000-03-31 00:00:00",
...                                         "2000-05-31 00:00:00",
...                                         "2000-08-31 00:00:00"]))
>>> df.index.to_period("M")
PeriodIndex(['2000-03', '2000-05', '2000-08'],
            dtype='period[M]', freq='M')
```

Infer the daily frequency

```
>>> idx = pd.date_range("2017-01-01", periods=2)
>>> idx.to_period()
PeriodIndex(['2017-01-01', '2017-01-02'],
            dtype='period[D]', freq='D')
```

pandas.DatetimeIndex.to_perioddelta

DatetimeIndex.**to_perioddelta**(*self*, **args*, ***kwargs*)

Calculate TimedeltaArray of difference between index values and index converted to PeriodArray at specified freq. Used for vectorized offsets.

Parameters

freq [Period frequency]

Returns

TimedeltaArray/Index

pandas.DatetimeIndex.to_pydatetime

DatetimeIndex.**to_pydatetime**(*self*, **args*, ***kwargs*)

Return Datetime Array/Index as object ndarray of datetime.datetime objects.

Returns

datetimes [ndarray]

pandas.DatetimeIndex.to_series

DatetimeIndex.**to_series**(*self*, *keep_tz=<object object at 0x7f5374a06320>*, *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

keep_tz [optional, defaults True] Return the data keeping the timezone.

If keep_tz is True:

If the timezone is not set, the resulting Series will have a datetime64[ns] dtype.

Otherwise the Series will have an datetime64[ns, tz] dtype; the tz will be preserved.

If keep_tz is False:

Series will have a datetime64[ns] dtype. TZ aware objects will have the tz removed.

Changed in version 1.0.0: The default value is now True. In a future version, this keyword will be removed entirely. Stop passing the argument to obtain the future behavior and silence the warning.

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

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

Returns

Series

pandas.DatetimeIndex.to_frame

`DatetimeIndex.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.DatetimeIndex.month_name

DatetimeIndex.**month_name**(*self*, **args*, ***kwargs*)

Return the month names of the DateTimeIndex with specified locale.

New in version 0.23.0.

Parameters

locale [str, optional] Locale determining the language in which to return the month name. Default is English locale.

Returns

Index Index of month names.

Examples

```
>>> idx = pd.date_range(start='2018-01', freq='M', periods=3)
>>> idx
DatetimeIndex(['2018-01-31', '2018-02-28', '2018-03-31'],
              dtype='datetime64[ns]', freq='M')
>>> idx.month_name()
Index(['January', 'February', 'March'], dtype='object')
```

pandas.DatetimeIndex.day_name

DatetimeIndex.**day_name**(*self*, **args*, ***kwargs*)

Return the day names of the DateTimeIndex with specified locale.

New in version 0.23.0.

Parameters

locale [str, optional] Locale determining the language in which to return the day name. Default is English locale.

Returns

Index Index of day names.

Examples

```
>>> idx = pd.date_range(start='2018-01-01', freq='D', periods=3)
>>> idx
DatetimeIndex(['2018-01-01', '2018-01-02', '2018-01-03'],
              dtype='datetime64[ns]', freq='D')
>>> idx.day_name()
Index(['Monday', 'Tuesday', 'Wednesday'], dtype='object')
```

pandas.DatetimeIndex.mean

DatetimeIndex.**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.

Time/Date components

DatetimeIndex.year	The year of the datetime.
DatetimeIndex.month	The month as January=1, December=12.
DatetimeIndex.day	The month as January=1, December=12.
DatetimeIndex.hour	The hours of the datetime.
DatetimeIndex.minute	The minutes of the datetime.
DatetimeIndex.second	The seconds of the datetime.
DatetimeIndex.microsecond	The microseconds of the datetime.
DatetimeIndex.nanosecond	The nanoseconds of the datetime.
DatetimeIndex.date	Returns numpy array of python datetime.date objects (namely, the date part of Timestamps without timezone information).
DatetimeIndex.time	Returns numpy array of datetime.time.
DatetimeIndex.timetz	Returns numpy array of datetime.time also containing timezone information.
DatetimeIndex.dayofyear	The ordinal day of the year.
DatetimeIndex.weekofyear	The week ordinal of the year.

continues on next page