00-Core-Pandas-Time-Methods

September 28, 2024

1 Time Methods

1.1 Python Datetime Review

Basic Python outside of Pandas contains a datetime library:

[48]:
```
import numpy as np
import pandas as pd
```

[49]:
```
from datetime import datetime
```

[50]:
```
# To illustrate the order of arguments
my_year = 2017
my_month = 1
my_day = 2
my_hour = 13
my_minute = 30
my_second = 15
```

[51]:
```
# January 2nd, 2017
my_date = datetime(my_year,my_month,my_day)
```

[52]:
```
# Defaults to 0:00
my_date
```

[52]:
```
datetime.datetime(2017, 1, 2, 0, 0)
```

[54]:
```
# January 2nd, 2017 at 13:30:15
my_date_time = datetime(my_year,my_month,my_day,my_hour,my_minute,my_second)
```

[55]:
```
my_date_time
```

[55]:
```
datetime.datetime(2017, 1, 2, 13, 30, 15)
```

You can grab any part of the datetime object you want

[56]:
```
my_date.day
```

[56]:
```
2
```

[57]:
```
my_date_time.hour
```

[57]:
```
13
```

2 Pandas

3 Converting to datetime

Often when data sets are stored, the time component may be a string. Pandas easily converts strings to datetime objects.

[58]:
```
import pandas as pd
```

[59]:
```
myseries = pd.Series(['Nov 3, 2000', '2000-01-01', None])
```

[60]:
```
myseries
```

[60]:
```
0    Nov 3, 2000
1     2000-01-01
2           None
dtype: object
```

[61]:
```
myseries[0]
```

[61]:
```
'Nov 3, 2000'
```

3.0.1 pd.to_datetime()

https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#converting-to-timestamps

[62]:
```
pd.to_datetime(myseries)
```

[62]:
```
0   2000-11-03
1   2000-01-01
2          NaT
dtype: datetime64[ns]
```

[63]:
```
pd.to_datetime(myseries)[0]
```

[63]:
```
Timestamp('2000-11-03 00:00:00')
```

[64]:
```
euro_date = '31-12-2000'
```

[65]:
```
pd.to_datetime(euro_date)
```

```
/tmp/ipykernel_17509/2851784268.py:1: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  pd.to_datetime(euro_date)
```

[65]: `Timestamp('2000-12-31 00:00:00')`

[66]:
```
# 10th of Dec OR 12th of October?
# We may need to tell pandas
euro_date1 = '10-12-2000'
```

[67]:
```
pd.to_datetime(euro_date1)
```

[67]: `Timestamp('2000-10-12 00:00:00')`

[68]:
```
pd.to_datetime(euro_date1,dayfirst=True)
```

[68]: `Timestamp('2000-12-10 00:00:00')`

3.1 Custom Time String Formatting

Sometimes dates can have a non standard format, luckily you can always specify to pandas the format. You should also note this could speed up the conversion, so it may be worth doing even if pandas can parse on its own.

A full table of codes can be found here: https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes

[69]:
```
style_date = '12--Dec--2000'
```

[70]:
```
pd.to_datetime(style_date, format='%d--%b--%Y')
```

[70]: `Timestamp('2000-12-12 00:00:00')`

[71]:
```
strange_date = '12th of Dec 2000'
```

[72]:
```
pd.to_datetime(strange_date)
```

[72]: `Timestamp('2000-12-12 00:00:00')`

3.2 Data

Retail Sales: Beer, Wine, and Liquor Stores

Units: Millions of Dollars, Not Seasonally Adjusted

Frequency: Monthly

U.S. Census Bureau, Retail Sales: Beer, Wine, and Liquor Stores [MRTSSM4453USN], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/MRTSSM4453USN, July 2, 2020.

[73]:
```
sales = pd.read_csv('RetailSales_BeerWineLiquor.csv')
```

[79]:
```
sales.head(2)
```

[79]:
```
         DATE  MRTSSM4453USN
0  1992-01-01           1509
1  1992-02-01           1541
```

[76]:
```
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 340 entries, 0 to 339
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   DATE           340 non-null    object
 1   MRTSSM4453USN  340 non-null    int64 
dtypes: int64(1), object(1)
memory usage: 5.4+ KB
```

[27]:
```
sales.iloc[0]['DATE']
```

[27]:
```
'1992-01-01'
```

[28]:
```
type(sales.iloc[0]['DATE'])
```

[28]:
```
str
```

[80]:
```
sales['DATE'] = pd.to_datetime(sales['DATE'])
```

[81]:
```
sales
```

[81]:
```
           DATE  MRTSSM4453USN
0    1992-01-01           1509
1    1992-02-01           1541
2    1992-03-01           1597
3    1992-04-01           1675
4    1992-05-01           1822
..          …              …
335  2019-12-01           6630
336  2020-01-01           4388
337  2020-02-01           4533
338  2020-03-01           5562
339  2020-04-01           5207
```

```
[340 rows x 2 columns]
```

[82]:
```
sales.iloc[0]['DATE']
```

[82]:
```
Timestamp('1992-01-01 00:00:00')
```

[83]:
```
type(sales.iloc[0]['DATE'])
```

[83]:
```
pandas._libs.tslibs.timestamps.Timestamp
```

3.3 Attempt to Parse Dates Automatically

parse_dates - bool or list of int or names or list of lists or dict, default False The behavior is as follows:

```
boolean. If True -> try parsing the index.

list of int or names. e.g. If [1, 2, 3] -> try parsing columns 1, 2, 3 each as a separate date

list of lists. e.g. If [[1, 3]] -> combine columns 1 and 3 and parse as a single date column.

dict, e.g. {‘foo’ : [1, 3]} -> parse columns 1, 3 as date and call result ‘foo’

If a column or index cannot be represented as an array of datetimes, say because of an unparsea
```

[33]:
```
# Parse Column at Index 0 as Datetime
sales = pd.read_csv('RetailSales_BeerWineLiquor.csv',parse_dates=[0])
```

[87]:
```
sales
```

[87]:

	DATE	MRTSSM4453USN
0	1992-01-01	1509
1	1992-02-01	1541
2	1992-03-01	1597
3	1992-04-01	1675
4	1992-05-01	1822
..	…	…
335	2019-12-01	6630
336	2020-01-01	4388
337	2020-02-01	4533
338	2020-03-01	5562
339	2020-04-01	5207

```
[340 rows x 2 columns]
```

[88]:
```
type(sales.iloc[0]['DATE'])
```

[88]:
```
pandas._libs.tslibs.timestamps.Timestamp
```

3.4 Resample

A common operation with time series data is resampling based on the time series index. Let's see how to use the resample() method. [reference]

[89]:
```
# Our index
sales.index
```

[89]:
```
RangeIndex(start=0, stop=340, step=1)
```

[37]:
```
# Reset DATE to index
```

[93]:
```
sales = pd.read_csv('RetailSales_BeerWineLiquor.csv',parse_dates=[0])
```

[94]:
```
sales = sales.set_index('DATE')
```

[95]:
```
sales
```

[95]:

DATE	MRTSSM4453USN
1992-01-01	1509
1992-02-01	1541
1992-03-01	1597
1992-04-01	1675
1992-05-01	1822
…	…
2019-12-01	6630
2020-01-01	4388
2020-02-01	4533
2020-03-01	5562
2020-04-01	5207

[340 rows x 1 columns]

When calling `.resample()` you first need to pass in a **rule** parameter, then you need to call some sort of aggregation function.

The **rule** parameter describes the frequency with which to apply the aggregation function (daily, monthly, yearly, etc.) It is passed in using an "offset alias" - refer to the table below. [reference]

The aggregation function is needed because, due to resampling, we need some sort of mathematical rule to join the rows (mean, sum, count, etc.)

TIME SERIES OFFSET ALIASES

ALIAS

DESCRIPTION

B

business day frequency

C

custom business day frequency (experimental)

D

calendar day frequency

W

weekly frequency

M

month end frequency

SM

semi-month end frequency (15th and end of month)

BM

business month end frequency

CBM

custom business month end frequency

MS

month start frequency

SMS

semi-month start frequency (1st and 15th)

BMS

business month start frequency

CBMS

custom business month start frequency

Q

quarter end frequency

intentionally left blank

ALIAS

DESCRIPTION

BQ

business quarter endfrequency

QS

quarter start frequency

BQS

business quarter start frequency

A

year end frequency

BA

business year end frequency

AS

year start frequency

BAS

business year start frequency

BH

business hour frequency

H

hourly frequency

T, min

minutely frequency

S

secondly frequency

L, ms

milliseconds

U, us

microseconds

N

nanoseconds

[97]:
```
# Yearly Means
sales.resample(rule='A').mean()
```

[97]:
```
            MRTSSM4453USN
DATE
1992-12-31    1807.250000
1993-12-31    1794.833333
1994-12-31    1841.750000
```

```
1995-12-31    1833.916667
1996-12-31    1929.750000
1997-12-31    2006.750000
1998-12-31    2115.166667
1999-12-31    2206.333333
2000-12-31    2375.583333
2001-12-31    2468.416667
2002-12-31    2491.166667
2003-12-31    2539.083333
2004-12-31    2682.416667
2005-12-31    2797.250000
2006-12-31    3001.333333
2007-12-31    3177.333333
2008-12-31    3292.000000
2009-12-31    3353.750000
2010-12-31    3450.083333
2011-12-31    3532.666667
2012-12-31    3697.083333
2013-12-31    3839.666667
2014-12-31    4023.833333
2015-12-31    4212.500000
2016-12-31    4434.416667
2017-12-31    4602.666667
2018-12-31    4830.666667
2019-12-31    4972.750000
2020-12-31    4922.500000
```

Resampling rule 'A' takes all of the data points in a given year, applies the aggregation function (in this case we calculate the mean), and reports the result as the last day of that year. Note 2020 in this data set was not complete.

4 .dt Method Calls

Once a column or index is ina datetime format, you can call a variety of methods off of the .dt library inside pandas:

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.dt.html

[98]:
```
sales = sales.reset_index()
```

[99]:
```
sales
```

[99]:
```
        DATE  MRTSSM4453USN
0 1992-01-01           1509
1 1992-02-01           1541
2 1992-03-01           1597
3 1992-04-01           1675
```

```
4   1992-05-01           1822
..         …             …
335 2019-12-01           6630
336 2020-01-01           4388
337 2020-02-01           4533
338 2020-03-01           5562
339 2020-04-01           5207

[340 rows x 2 columns]
```

[100]:
```
help(sales['DATE'].dt)
```

```
Help on DatetimeProperties in module pandas.core.indexes.accessors object:

class DatetimeProperties(Properties)
 |  DatetimeProperties(data: 'Series', orig) -> 'None'
 |
 |  Accessor object for datetimelike properties of the Series values.
 |
 |  Examples
 |  --------
 |  >>> seconds_series = pd.Series(pd.date_range("2000-01-01", periods=3,
freq="s"))
 |  >>> seconds_series
 |  0   2000-01-01 00:00:00
 |  1   2000-01-01 00:00:01
 |  2   2000-01-01 00:00:02
 |  dtype: datetime64[ns]
 |  >>> seconds_series.dt.second
 |  0    0
 |  1    1
 |  2    2
 |  dtype: int64
 |
 |  >>> hours_series = pd.Series(pd.date_range("2000-01-01", periods=3,
freq="h"))
 |  >>> hours_series
 |  0   2000-01-01 00:00:00
 |  1   2000-01-01 01:00:00
 |  2   2000-01-01 02:00:00
 |  dtype: datetime64[ns]
 |  >>> hours_series.dt.hour
 |  0    0
 |  1    1
 |  2    2
 |  dtype: int64
 |
```

```
 |  >>> quarters_series = pd.Series(pd.date_range("2000-01-01", periods=3,
freq="q"))
 |  >>> quarters_series
 |  0   2000-03-31
 |  1   2000-06-30
 |  2   2000-09-30
 |  dtype: datetime64[ns]
 |  >>> quarters_series.dt.quarter
 |  0    1
 |  1    2
 |  2    3
 |  dtype: int64
 |
 |  Returns a Series indexed like the original Series.
 |  Raises TypeError if the Series does not contain datetimelike values.
 |
 |  Method resolution order:
 |      DatetimeProperties
 |      Properties
 |      pandas.core.accessor.PandasDelegate
 |      pandas.core.base.PandasObject
 |      pandas.core.accessor.DirNamesMixin
 |      pandas.core.base.NoNewAttributesMixin
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  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
 |          :ref:`frequency aliases <timeseries.offset_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.
```

```
 |
 |      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.
 |
 |      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.
 |
 |      Notes
 |      -----
 |      If the timestamps have a timezone, ceiling will take place relative to
the
 |      local ("wall") time and re-localized to the same timezone. When ceiling
 |      near daylight savings time, use ``nonexistent`` and ``ambiguous`` to
 |      control the re-localization behavior.
 |
 |      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]
 |
 |      When rounding near a daylight savings time transition, use ``ambiguous``
or
 |      ``nonexistent`` to control how the timestamp should be re-localized.
 |
 |      >>> rng_tz = pd.DatetimeIndex(["2021-10-31 01:30:00"],
tz="Europe/Amsterdam")
 |
 |      >>> rng_tz.ceil("H", ambiguous=False)
 |      DatetimeIndex(['2021-10-31 02:00:00+01:00'],
 |                    dtype='datetime64[ns, Europe/Amsterdam]', freq=None)
 |
 |      >>> rng_tz.ceil("H", ambiguous=True)
 |      DatetimeIndex(['2021-10-31 02:00:00+02:00'],
 |                    dtype='datetime64[ns, Europe/Amsterdam]', freq=None)
 |
 |  day_name(self, *args, **kwargs)
 |      Return the day names with specified locale.
 |
 |      Parameters
 |      ----------
 |      locale : str, optional
 |          Locale determining the language in which to return the day name.
 |          Default is English locale.
 |
 |      Returns
 |      -------
 |      Series or Index
 |          Series or Index of day names.
 |
 |      Examples
 |      --------
 |      >>> s = pd.Series(pd.date_range(start='2018-01-01', freq='D',
periods=3))
 |      >>> s
 |      0   2018-01-01
 |      1   2018-01-02
 |      2   2018-01-03
 |      dtype: datetime64[ns]
 |      >>> s.dt.day_name()
 |      0       Monday
 |      1      Tuesday
```

```
 |      2    Wednesday
 |      dtype: object
 |
 |      >>> 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')
 |
 |  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
 |          :ref:`frequency aliases <timeseries.offset_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.
 |
 |      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.
 |
 |      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.
 |
 |      Notes
 |      -----
 |      If the timestamps have a timezone, flooring will take place relative to
the
 |      local ("wall") time and re-localized to the same timezone. When flooring
 |      near daylight savings time, use ``nonexistent`` and ``ambiguous`` to
 |      control the re-localization behavior.
 |
 |      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]
 |
 |      When rounding near a daylight savings time transition, use ``ambiguous``
or
 |      ``nonexistent`` to control how the timestamp should be re-localized.
 |
 |      >>> rng_tz = pd.DatetimeIndex(["2021-10-31 03:30:00"],
tz="Europe/Amsterdam")
 |
 |      >>> rng_tz.floor("2H", ambiguous=False)
 |      DatetimeIndex(['2021-10-31 02:00:00+01:00'],
 |                    dtype='datetime64[ns, Europe/Amsterdam]', freq=None)
```

```
|
|      >>> rng_tz.floor("2H", ambiguous=True)
|      DatetimeIndex(['2021-10-31 02:00:00+02:00'],
|                    dtype='datetime64[ns, Europe/Amsterdam]', freq=None)
|
|  isocalendar(self) -> 'DataFrame'
|      Calculate year, week, and day according to the ISO 8601 standard.
|
|      .. versionadded:: 1.1.0
|
|      Returns
|      -------
|      DataFrame
|          With columns year, week and day.
|
|      See Also
|      --------
|      Timestamp.isocalendar : Function return a 3-tuple containing ISO year,
|          week number, and weekday for the given Timestamp object.
|      datetime.date.isocalendar : Return a named tuple object with
|          three components: year, week and weekday.
|
|      Examples
|      --------
|      >>> ser = pd.to_datetime(pd.Series(["2010-01-01", pd.NaT]))
|      >>> ser.dt.isocalendar()
|         year  week  day
|      0  2009    53     5
|      1  <NA>  <NA>  <NA>
|      >>> ser.dt.isocalendar().week
|      0      53
|      1    <NA>
|      Name: week, dtype: UInt32
|
|  month_name(self, *args, **kwargs)
|      Return the month names with specified locale.
|
|      Parameters
|      ----------
|      locale : str, optional
|          Locale determining the language in which to return the month name.
|          Default is English locale.
|
|      Returns
|      -------
|      Series or Index
|          Series or Index of month names.
|
```

```
|      Examples
|      --------
|      >>> s = pd.Series(pd.date_range(start='2018-01', freq='M', periods=3))
|      >>> s
|      0   2018-01-31
|      1   2018-02-28
|      2   2018-03-31
|      dtype: datetime64[ns]
|      >>> s.dt.month_name()
|      0     January
|      1    February
|      2       March
|      dtype: object
|
|      >>> 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')
|
|  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)
 |
 |  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
 |          :ref:`frequency aliases <timeseries.offset_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.
 |
 |      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.
 |
 |      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.
 |
 |      Notes
 |      -----
 |      If the timestamps have a timezone, rounding will take place relative to
the
 |      local ("wall") time and re-localized to the same timezone. When rounding
 |      near daylight savings time, use ``nonexistent`` and ``ambiguous`` to
 |      control the re-localization behavior.
 |
 |      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]
 |
 |      When rounding near a daylight savings time transition, use ``ambiguous``
or
 |      ``nonexistent`` to control how the timestamp should be re-localized.
 |
 |      >>> rng_tz = pd.DatetimeIndex(["2021-10-31 03:30:00"],
tz="Europe/Amsterdam")
 |
 |      >>> rng_tz.floor("2H", ambiguous=False)
 |      DatetimeIndex(['2021-10-31 02:00:00+01:00'],
 |                    dtype='datetime64[ns, Europe/Amsterdam]', freq=None)
```

```
 |
 |      >>> rng_tz.floor("2H", ambiguous=True)
 |      DatetimeIndex(['2021-10-31 02:00:00+02:00'],
 |                    dtype='datetime64[ns, Europe/Amsterdam]', freq=None)
 |
 |  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 <https://docs.python.org/3/library/datetime.html#strftime-and-
strptime-behavior>`__.
 |
 |      Formats supported by the C `strftime` API but not by the python string
format
 |      doc (such as `"%R"`, `"%r"`) are not officially supported and should be
 |      preferably replaced with their supported equivalents (such as `"%H:%M"`,
 |      `"%I:%M:%S %p"`).
 |
 |      Note that `PeriodIndex` support additional directives, detailed in
 |      `Period.strftime`.
 |
 |      Parameters
 |      ----------
 |      date_format : str
 |          Date format string (e.g. "%Y-%m-%d").
 |
 |      Returns
 |      -------
 |      ndarray[object]
 |          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.
 |      Timestamp.strftime : Format a single Timestamp.
 |      Period.strftime : Format a single Period.
 |
 |      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')
|
|  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' :ref:`offset strings <timeseries.offset_aliases>`
|          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]')
|
|      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]')
|
|  to_pydatetime(self) -> 'np.ndarray'
|      Return the data as an array of :class:`datetime.datetime` objects.
```

```
|
|      Timezone information is retained if present.
|
|      .. warning::
|
|         Python's datetime uses microsecond resolution, which is lower than
|         pandas (nanosecond). The values are truncated.
|
|      Returns
|      -------
|      numpy.ndarray
|          Object dtype array containing native Python datetime objects.
|
|      See Also
|      --------
|      datetime.datetime : Standard library value for a datetime.
|
|      Examples
|      --------
|      >>> s = pd.Series(pd.date_range('20180310', periods=2))
|      >>> s
|      0   2018-03-10
|      1   2018-03-11
|      dtype: datetime64[ns]
|
|      >>> s.dt.to_pydatetime()
|      array([datetime.datetime(2018, 3, 10, 0, 0),
|             datetime.datetime(2018, 3, 11, 0, 0)], dtype=object)
|
|      pandas' nanosecond precision is truncated to microseconds.
|
|      >>> s = pd.Series(pd.date_range('20180310', periods=2, freq='ns'))
|      >>> s
|      0   2018-03-10 00:00:00.000000000
|      1   2018-03-10 00:00:00.000000001
|      dtype: datetime64[ns]
|
|      >>> s.dt.to_pydatetime()
|      array([datetime.datetime(2018, 3, 10, 0, 0),
|             datetime.datetime(2018, 3, 10, 0, 0)], dtype=object)
|
|  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')
 |
 |  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.
 |
 |      This method can also be used to do the inverse -- to create a time
 |      zone unaware object from an aware object. To that end, pass `tz=None`.
 |
 |      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.
|
|    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=None)
|
|    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=None)
|
|    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
|       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   2018-10-28 01:20:00+02:00
|       1   2018-10-28 02:36:00+02:00
|       2   2018-10-28 03:46:00+01:00
|       dtype: datetime64[ns, CET]
|
|       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]
|
|  ----------------------------------------------------------------------
|  Readonly properties defined here:
|
|  freq
|
|  week
|      The week ordinal of the year according to the ISO 8601 standard.
|
|      .. deprecated:: 1.1.0
|
|      Series.dt.weekofyear and Series.dt.week have been deprecated.  Please
|      call :func:`Series.dt.isocalendar` and access the ``week`` column
|      instead.
|
|  weekofyear
|      The week ordinal of the year according to the ISO 8601 standard.
|
|      .. deprecated:: 1.1.0
|
|      Series.dt.weekofyear and Series.dt.week have been deprecated.  Please
|      call :func:`Series.dt.isocalendar` and access the ``week`` column
|      instead.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  date
|      Returns numpy array of python :class:`datetime.date` objects.
|
|      Namely, the date part of Timestamps without time and
|      timezone information.
|
|  day
|      The day of the datetime.
|
|      Examples
|      --------
|      >>> datetime_series = pd.Series(
|      …     pd.date_range("2000-01-01", periods=3, freq="D")
|      … )
|      >>> datetime_series
|      0   2000-01-01
|      1   2000-01-02
|      2   2000-01-03
|      dtype: datetime64[ns]
```

```
|      >>> datetime_series.dt.day
|      0    1
|      1    2
|      2    3
|      dtype: int64
|
|  day_of_week
|      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
|
|  day_of_year
|      The ordinal day of the year.
|
|  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
|
|  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.
|
|  hour
|      The hours of the datetime.
|
|      Examples
|      --------
|      >>> datetime_series = pd.Series(
|      …     pd.date_range("2000-01-01", periods=3, freq="h")
|      … )
|      >>> datetime_series
|      0   2000-01-01 00:00:00
```

```
|      1   2000-01-01 01:00:00
|      2   2000-01-01 02:00:00
|      dtype: datetime64[ns]
|      >>> datetime_series.dt.hour
|      0    0
|      1    1
|      2    2
|      dtype: int64
|
|  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])
|
|      >>> dates_series = 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
|
|  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])
|
|  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]
|      >>> 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])
|
|  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])
|
|  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])
|
|  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])
|
|  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])
|
|  microsecond
|      The microseconds of the datetime.
|
|      Examples
|      --------
|      >>> datetime_series = pd.Series(
|      …     pd.date_range("2000-01-01", periods=3, freq="us")
|      … )
|      >>> datetime_series
|      0   2000-01-01 00:00:00.000000
|      1   2000-01-01 00:00:00.000001
|      2   2000-01-01 00:00:00.000002
|      dtype: datetime64[ns]
|      >>> datetime_series.dt.microsecond
|      0       0
|      1       1
|      2       2
|      dtype: int64
|
|  minute
|      The minutes of the datetime.
|
|      Examples
|      --------
|      >>> datetime_series = pd.Series(
|      …     pd.date_range("2000-01-01", periods=3, freq="T")
|      … )
|      >>> datetime_series
|      0   2000-01-01 00:00:00
|      1   2000-01-01 00:01:00
|      2   2000-01-01 00:02:00
|      dtype: datetime64[ns]
|      >>> datetime_series.dt.minute
|      0    0
|      1    1
|      2    2
|      dtype: int64
|
|  month
|      The month as January=1, December=12.
|
|      Examples
|      --------
|      >>> datetime_series = pd.Series(
|      …     pd.date_range("2000-01-01", periods=3, freq="M")
|      … )
```

```
|      >>> datetime_series
|      0   2000-01-31
|      1   2000-02-29
|      2   2000-03-31
|      dtype: datetime64[ns]
|      >>> datetime_series.dt.month
|      0    1
|      1    2
|      2    3
|      dtype: int64
|
|  nanosecond
|      The nanoseconds of the datetime.
|
|      Examples
|      --------
|      >>> datetime_series = pd.Series(
|      …     pd.date_range("2000-01-01", periods=3, freq="ns")
|      … )
|      >>> datetime_series
|      0   2000-01-01 00:00:00.000000000
|      1   2000-01-01 00:00:00.000000001
|      2   2000-01-01 00:00:00.000000002
|      dtype: datetime64[ns]
|      >>> datetime_series.dt.nanosecond
|      0       0
|      1       1
|      2       2
|      dtype: int64
|
|  quarter
|      The quarter of the date.
|
|  second
|      The seconds of the datetime.
|
|      Examples
|      --------
|      >>> datetime_series = pd.Series(
|      …     pd.date_range("2000-01-01", periods=3, freq="s")
|      … )
|      >>> datetime_series
|      0   2000-01-01 00:00:00
|      1   2000-01-01 00:00:01
|      2   2000-01-01 00:00:02
|      dtype: datetime64[ns]
|      >>> datetime_series.dt.second
|      0    0
```

```
|      1    1
|      2    2
|      dtype: int64
|
|  time
|      Returns numpy array of :class:`datetime.time` objects.
|
|      The time part of the Timestamps.
|
|  timetz
|      Returns numpy array of :class:`datetime.time` objects with timezones.
|
|      The time part of the Timestamps.
|
|  tz
|      Return the timezone.
|
|      Returns
|      -------
|      datetime.tzinfo, pytz.tzinfo.BaseTZInfo, dateutil.tz.tz.tzfile, or None
|          Returns None when the array is tz-naive.
|
|  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
|
|  year
|      The year of the datetime.
|
|      Examples
|      --------
|      >>> datetime_series = pd.Series(
|      …     pd.date_range("2000-01-01", periods=3, freq="Y")
|      … )
|      >>> datetime_series
|      0   2000-12-31
|      1   2001-12-31
|      2   2002-12-31
|      dtype: datetime64[ns]
|      >>> datetime_series.dt.year
|      0    2000
|      1    2001
|      2    2002
|      dtype: int64
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __annotations__ = {}
|
|  ----------------------------------------------------------------------
|  Methods inherited from Properties:
|
|  __init__(self, data: 'Series', orig) -> 'None'
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from pandas.core.accessor.PandasDelegate:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
```

```
|  Methods inherited from pandas.core.base.PandasObject:
|
|  __repr__(self) -> 'str'
|      Return a string representation for a particular object.
|
|  __sizeof__(self) -> 'int'
|      Generates the total memory usage for an object that returns
|      either a value or Series of values
|
|  ----------------------------------------------------------------------
|  Methods inherited from pandas.core.accessor.DirNamesMixin:
|
|  __dir__(self) -> 'list[str]'
|      Provide method name lookup and completion.
|
|      Notes
|      -----
|      Only provide 'public' methods.
|
|  ----------------------------------------------------------------------
|  Methods inherited from pandas.core.base.NoNewAttributesMixin:
|
|  __setattr__(self, key: 'str', value) -> 'None'
|      Implement setattr(self, name, value).
```

[101]:
```
sales['DATE'].dt.month
```

[101]:
```
0       1
1       2
2       3
3       4
4       5
       ..
335    12
336     1
337     2
338     3
339     4
Name: DATE, Length: 340, dtype: int64
```

[47]:
```
sales['DATE'].dt.is_leap_year
```

[47]:
```
0       True
1       True
2       True
3       True
```

```
4       True
        …
335    False
336     True
337     True
338     True
339     True
Name: DATE, Length: 340, dtype: bool
```

[ ]: