Time Series Functions

In [28]:
```
from datetime import datetime
```

In [29]:
```
now=datetime.now()
```

In [30]:
```
now
```

Out[30]:
```
datetime.datetime(2022, 11, 6, 18, 44, 45, 688266)
```

In [4]:
```
now.year,now.month, now.day
```

Out[4]:
```
(2022, 11, 6)
```

In [36]:
```
from datetime import datetime
n=datetime.now()
n.year,n.month,n.day
n.hour
n.minute
n.second
n.date()
```

Out[36]:
```
datetime.date(2022, 11, 6)
```

In [5]:
```
now.hour
```

Out[5]:
```
18
```

In [6]:
```
now.minute
now.second
```

Out[6]:
```
56
```

In [7]:
```
now.date()
```

Out[7]:
```
datetime.date(2022, 11, 6)
```

In [8]:
```
help(datetime.now())
```

```
Help on datetime object:

class datetime(date)
 |  datetime(year, month, day[, hour[, minute[, second[, microsecond[,tzinfo]]]]])
 |
 |  The year, month and day arguments are required. tzinfo may be None, or an
 |  instance of a tzinfo subclass. The remaining arguments may be ints.
 |
 |  Method resolution order:
 |      datetime
 |      date
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __radd__(self, value, /)
 |      Return value+self.
 |
 |  __reduce__(...)
 |      __reduce__() -> (cls, state)
 |
 |  __reduce_ex__(...)
 |      __reduce_ex__(proto) -> (cls, state)
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __rsub__(self, value, /)
 |      Return value-self.
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  __sub__(self, value, /)
 |      Return self-value.
 |
 |  astimezone(...)
```

```
 |      tz -> convert to local time in new timezone tz
 |
 |  ctime(...)
 |      Return ctime() style string.
 |
 |  date(...)
 |      Return date object with same year, month and day.
 |
 |  dst(...)
 |      Return self.tzinfo.dst(self).
 |
 |  isoformat(...)
 |      [sep] -> string in ISO 8601 format, YYYY-MM-DDT[HH[:MM[:SS[.mmm[uuu]]]]][+
HH:MM].
 |      sep is used to separate the year from the time, and defaults to 'T'.
 |      The optional argument timespec specifies the number of additional terms
 |      of the time to include. Valid options are 'auto', 'hours', 'minutes',
 |      'seconds', 'milliseconds' and 'microseconds'.
 |
 |  replace(...)
 |      Return datetime with new specified fields.
 |
 |  time(...)
 |      Return time object with same time but with tzinfo=None.
 |
 |  timestamp(...)
 |      Return POSIX timestamp as float.
 |
 |  timetuple(...)
 |      Return time tuple, compatible with time.localtime().
 |
 |  timetz(...)
 |      Return time object with same time and tzinfo.
 |
 |  tzname(...)
 |      Return self.tzinfo.tzname(self).
 |
 |  utcoffset(...)
 |      Return self.tzinfo.utcoffset(self).
 |
 |  utctimetuple(...)
 |      Return UTC time tuple, compatible with time.localtime().
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  combine(...) from builtins.type
 |      date, time -> datetime with same date and time fields
 |
 |  fromisoformat(...) from builtins.type
 |      string -> datetime from datetime.isoformat() output
 |
 |  fromtimestamp(...) from builtins.type
 |      timestamp[, tz] -> tz's local time from POSIX timestamp.
 |
 |  now(tz=None) from builtins.type
 |      Returns new datetime object representing current time local to tz.
 |
 |        tz
 |          Timezone object.
 |
 |      If no tz is specified, uses local timezone.
 |
 |  strptime(...) from builtins.type
```

```
 |      string, format -> new datetime parsed from a string (like time.strptime
()).
 |
 |  utcfromtimestamp(...) from builtins.type
 |      Construct a naive UTC datetime from a POSIX timestamp.
 |
 |  utcnow(...) from builtins.type
 |      Return a new datetime representing UTC day and time.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  fold
 |
 |  hour
 |
 |  microsecond
 |
 |  minute
 |
 |  second
 |
 |  tzinfo
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  max = datetime.datetime(9999, 12, 31, 23, 59, 59, 999999)
 |
 |  min = datetime.datetime(1, 1, 1, 0, 0)
 |
 |  resolution = datetime.timedelta(microseconds=1)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from date:
 |
 |  __format__(...)
 |      Formats self with strftime.
 |
 |  isocalendar(...)
 |      Return a named tuple containing ISO year, week number, and weekday.
 |
 |  isoweekday(...)
 |      Return the day of the week represented by the date.
 |      Monday == 1 ... Sunday == 7
 |
 |  strftime(...)
 |      format -> strftime() style string.
 |
 |  toordinal(...)
 |      Return proleptic Gregorian ordinal.  January 1 of year 1 is day 1.
 |
 |  weekday(...)
 |      Return the day of the week represented by the date.
 |      Monday == 0 ... Sunday == 6
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from date:
```

```
 |
 |  fromisocalendar(...) from builtins.type
 |      int, int, int -> Construct a date from the ISO year, week number and weekd
ay.
 |
 |      This is the inverse of the date.isocalendar() function
 |
 |  fromordinal(...) from builtins.type
 |      int -> date corresponding to a proleptic Gregorian ordinal.
 |
 |  today(...) from builtins.type
 |      Current date or datetime:  same as self.__class__.fromtimestamp(time.time
()).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from date:
 |
 |  day
 |
 |  month
 |
 |  year
```

In [9]:
```
now.ctime()
```

Out[9]:
```
'Sun Nov  6 18:43:56 2022'
```

In [10]:
```
now.timetuple()
```

Out[10]:
```
time.struct_time(tm_year=2022, tm_mon=11, tm_mday=6, tm_hour=18, tm_min=43, tm_sec
=56, tm_wday=6, tm_yday=310, tm_isdst=-1)
```

In [11]:
```
delta=datetime(2014,8,11)- datetime(2011,6,7)
```

In [12]:
```
delta
```

Out[12]:
```
datetime.timedelta(days=1161)
```

In [13]:
```
delta.seconds
```

Out[13]:
```
0
```

In [14]:
```
from datetime import timedelta
```

In [15]:
```
stdate=datetime(2014,11,10,6,30)
```

In [16]:
```
stdate+timedelta(1200)
```

Out[16]:
```
datetime.datetime(2018, 2, 22, 6, 30)
```

In [17]:
```
stdate+timedelta(12,5,0,0,0,15)
```

Out[17]:
```
datetime.datetime(2014, 11, 22, 21, 30, 5)
```

In [ ]:
```

```

In [18]:
```
help(timedelta)
```

```
Help on class timedelta in module datetime:

class timedelta(builtins.object)
 |  Difference between two datetime values.
 |
 |  timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=
0, weeks=0)
 |
 |  All arguments are optional and default to 0.
 |  Arguments may be integers or floats, and may be positive or negative.
 |
 |  Methods defined here:
 |
 |  __abs__(self, /)
 |      abs(self)
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __bool__(self, /)
 |      True if self else False
 |
 |  __divmod__(self, value, /)
 |      Return divmod(self, value).
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __floordiv__(self, value, /)
 |      Return self//value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mod__(self, value, /)
 |      Return self%value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __neg__(self, /)
 |      -self
 |
 |  __pos__(self, /)
 |      +self
 |
```

```
|  __radd__(self, value, /)
|      Return value+self.
|
|  __rdivmod__(self, value, /)
|      Return divmod(value, self).
|
|  __reduce__(...)
|      __reduce__() -> (cls, state)
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __rfloordiv__(self, value, /)
|      Return value//self.
|
|  __rmod__(self, value, /)
|      Return value%self.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __rsub__(self, value, /)
|      Return value-self.
|
|  __rtruediv__(self, value, /)
|      Return value/self.
|
|  __str__(self, /)
|      Return str(self).
|
|  __sub__(self, value, /)
|      Return self-value.
|
|  __truediv__(self, value, /)
|      Return self/value.
|
|  total_seconds(...)
|      Total seconds in the duration.
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  days
|      Number of days.
|
|  microseconds
|      Number of microseconds (>= 0 and less than 1 second).
|
|  seconds
|      Number of seconds (>= 0 and less than 1 day).
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  max = datetime.timedelta(days=999999999, seconds=86399, microseconds=9...
|
|  min = datetime.timedelta(days=-999999999)
|
```

```
|  resolution = datetime.timedelta(microseconds=1)
```

```
In [37]: Stamp=datetime(2014,3,4,16,17)
```

```
In [38]: str(Stamp)
Out[38]: '2014-03-04 16:17:00'
```

```
In [44]: st=datetime(2020,2,12,17,23,9)
         str(st)
         st.strftime('%Y-%m-%d %I:%M')
Out[44]: '2020-02-12 05:23'
```

```
In [45]: Stamp.strftime('%Y-%m-%d %I:%M')
Out[45]: '2014-03-04 04:17'
```

```
In [55]: Value='2012-2-3 11:33'
```

```
In [56]: datetime.strptime(Value,'%Y-%m-%d %H:%M')
Out[56]: datetime.datetime(2012, 2, 3, 11, 33)
```

```
In [57]: datestrs=['6-5-2013', '3-6-2014']
```

```
In [58]: [datetime.strptime(x,'%d-%m-%Y') for x in datestrs]
Out[58]: [datetime.datetime(2013, 5, 6, 0, 0), datetime.datetime(2014, 6, 3, 0, 0)]
```

```
In [59]: from dateutil.parser import parse
```

```
In [60]: from dateutil.parser import parse
```

Types in datetime module

date: Store calendar date (year, month, day) using the Gregorian calendar time: Store time of day as hours, minutes, seconds, and microseconds datetime: Stores both date and time timedelta: Represents the difference between two datetime values (as days, seconds, and microseconds) tzinfo: Base type for storing time zone information

Datetime format specification (ISO C89 compatible)

%Y Four-digit year %y Two-digit year %m Two-digit month [01, 12] %d Two-digit day [01, 31] %H Hour (24-hour clock) [00, 23] %I Hour (12-hour clock) [01, 12] %M Two-digit minute [00, 59] %S Second [00, 61] (seconds 60, 61 account for leap seconds) %w Weekday as integer [0 (Sunday), 6] %U Week number of the year [00, 53]; Sunday is considered the first day of the week, and days before the first Sunday of the year are “week 0” %W Week number of the year [00, 53]; Monday is considered the first day of the week, and days before the first Monday of the year are “week 0” %z UTC time zone offset as +HHMM or -HHMM;

empty if time zone naive %F Shortcut for %Y-%m-%d (e.g., 2012-4-18) %D Shortcut for %m/%d/%y (e.g., 04/18/12)

In [64]:
```
parse('2012-2-3 11:33')
parse('Feb 20,2022')
```

Out[64]:
```
datetime.datetime(2022, 2, 20, 0, 0)
```

In [62]:
```
parse('Jan 26, 2019 11:33 PM')
```

Out[62]:
```
datetime.datetime(2019, 1, 26, 23, 33)
```

In [63]:
```
parse('16/4/65', dayfirst= True)
```

Out[63]:
```
datetime.datetime(2065, 4, 16, 0, 0)
```

In [65]:
```
datestrs=['6-5-2013 12:00:00', '3-6-2014 00:00:00']
```

In [66]:
```
import pandas as pd
pd.to_datetime(datestrs)
```

Out[66]:
```
DatetimeIndex(['2013-06-05 12:00:00', '2014-03-06 00:00:00'], dtype='datetime64[n
s]', freq=None)
```

In [69]:
```
ind=pd.to_datetime(datestrs+[None])
```

In [71]:
```
ind
```

Out[71]:
```
DatetimeIndex(['2013-06-05 12:00:00', '2014-03-06 00:00:00', 'NaT'], dtype='dateti
me64[ns]', freq=None)
```

In [72]:
```
ind[2]
```

Out[72]:
```
NaT
```

In [73]:
```
pd.isnull(ind)
```

Out[73]:
```
array([False, False,  True])
```

In [74]:
```
from datetime import datetime
```

In [75]:
```
dates=[datetime(2011,3,5), datetime(2012,7,4), datetime(2014,3,5), datetime(2017,5
```

In [76]:
```
import numpy as np
ds=pd.Series(np.random.randn(6),index=dates)
```

In [77]:
```
ds
```

Out[77]:
```
2011-03-05    0.690149
2012-07-04   -1.354414
2014-03-05    2.360318
2017-05-06   -0.711325
2014-07-15    0.689387
2011-05-06    1.001606
dtype: float64
```

In [78]:
```
ds+ds[::2]
```

Out[78]:
```
2011-03-05    1.380298
2011-05-06         NaN
2012-07-04         NaN
2014-03-05    4.720635
2014-07-15    1.378774
2017-05-06         NaN
dtype: float64
```

In [79]:
```
ds[::2]
```

Out[79]:
```
2011-03-05    0.690149
2014-03-05    2.360318
2014-07-15    0.689387
dtype: float64
```

In [80]:
```
ds.index.dtype
```

Out[80]:
```
dtype('<M8[ns]')
```

In [81]:
```
stamp=ds.index[0]
```

In [82]:
```
stamp
```

Out[82]:
```
Timestamp('2011-03-05 00:00:00')
```

In [83]:
```
ds[stamp]
```

Out[83]:
```
0.6901489731216112
```

In [84]:
```
ds['20110305']
```

Out[84]:
```
0.6901489731216112
```

In [86]:
```
import pandas as pd
import numpy as np
long_ts=pd.Series(np.random.randn(1000),index=pd.date_range('1/1/2011',periods=100
```

In [85]:
```
l=pd.Series(np.random.randn(1000),index=pd.date_range('1/1/2011',periods=1000))
l
```

Out[85]:
```
2011-01-01   -1.576413
2011-01-02    0.895353
2011-01-03   -1.028496
2011-01-04   -0.197100
2011-01-05   -0.982859
                ...
2013-09-22   -0.277584
2013-09-23    0.332870
2013-09-24   -0.901054
2013-09-25   -0.314980
2013-09-26   -0.204891
Freq: D, Length: 1000, dtype: float64
```

In [87]:
```
long_ts
```

Out[87]:
```
2011-01-01   -0.343607
2011-01-02   -0.431132
2011-01-03   -0.811779
2011-01-04   -0.859046
2011-01-05   -0.056922
                ...
2013-09-22    0.688521
2013-09-23    0.367926
2013-09-24   -0.802723
2013-09-25    0.051751
2013-09-26   -1.577335
Freq: D, Length: 1000, dtype: float64
```

In [88]:
```
long_ts['2012-07']
```

Out[88]:
```
2012-07-01   -0.290691
2012-07-02   -1.533387
2012-07-03    0.339562
2012-07-04    0.923535
2012-07-05   -1.114503
2012-07-06   -0.431748
2012-07-07    0.309758
2012-07-08   -0.250354
2012-07-09    0.472485
2012-07-10   -1.591920
2012-07-11    0.945053
2012-07-12   -1.846348
2012-07-13   -0.170963
2012-07-14    0.082230
2012-07-15   -0.784211
2012-07-16    1.648397
2012-07-17    0.273452
2012-07-18   -1.466684
2012-07-19   -0.578389
2012-07-20    2.312014
2012-07-21    1.197940
2012-07-22   -0.632694
2012-07-23   -0.306370
2012-07-24    0.181054
2012-07-25    0.200930
2012-07-26   -0.132219
2012-07-27    0.315709
2012-07-28    1.133498
2012-07-29   -1.346844
2012-07-30    0.686577
2012-07-31    1.002125
Freq: D, dtype: float64
```

In [89]:
```
long_ts[datetime(2012,7,30):datetime(2012,10,30)]
```

Out[89]:
```
2012-07-30    0.686577
2012-07-31    1.002125
2012-08-01    0.439218
2012-08-02   -0.297766
2012-08-03   -0.641702
                ...
2012-10-26    0.630695
2012-10-27    1.011546
2012-10-28   -0.487897
2012-10-29   -0.437322
2012-10-30    0.545965
Freq: D, Length: 93, dtype: float64
```

In [90]:
```
long_ts['2012-7-30':'2012-10-30']
```

Out[90]:
```
2012-07-30    0.686577
2012-07-31    1.002125
2012-08-01    0.439218
2012-08-02   -0.297766
2012-08-03   -0.641702
                ...
2012-10-26    0.630695
2012-10-27    1.011546
2012-10-28   -0.487897
2012-10-29   -0.437322
2012-10-30    0.545965
Freq: D, Length: 93, dtype: float64
```

In [91]:
```
long_ts.truncate(after='2012-10-30')
```

Out[91]:
```
2011-01-01   -0.343607
2011-01-02   -0.431132
2011-01-03   -0.811779
2011-01-04   -0.859046
2011-01-05   -0.056922
                ...
2012-10-26    0.630695
2012-10-27    1.011546
2012-10-28   -0.487897
2012-10-29   -0.437322
2012-10-30    0.545965
Freq: D, Length: 669, dtype: float64
```

In [92]:
```
long_ts
```

Out[92]:
```
2011-01-01   -0.343607
2011-01-02   -0.431132
2011-01-03   -0.811779
2011-01-04   -0.859046
2011-01-05   -0.056922
                ...
2013-09-22    0.688521
2013-09-23    0.367926
2013-09-24   -0.802723
2013-09-25    0.051751
2013-09-26   -1.577335
Freq: D, Length: 1000, dtype: float64
```

In [93]:
```
dates=pd.date_range('3/1/2020',periods=100,freq='M')
```

In [94]:
```
dates
```

Out[94]:
```
DatetimeIndex(['2020-03-31', '2020-04-30', '2020-05-31', '2020-06-30',
               '2020-07-31', '2020-08-31', '2020-09-30', '2020-10-31',
               '2020-11-30', '2020-12-31', '2021-01-31', '2021-02-28',
               '2021-03-31', '2021-04-30', '2021-05-31', '2021-06-30',
               '2021-07-31', '2021-08-31', '2021-09-30', '2021-10-31',
               '2021-11-30', '2021-12-31', '2022-01-31', '2022-02-28',
               '2022-03-31', '2022-04-30', '2022-05-31', '2022-06-30',
               '2022-07-31', '2022-08-31', '2022-09-30', '2022-10-31',
               '2022-11-30', '2022-12-31', '2023-01-31', '2023-02-28',
               '2023-03-31', '2023-04-30', '2023-05-31', '2023-06-30',
               '2023-07-31', '2023-08-31', '2023-09-30', '2023-10-31',
               '2023-11-30', '2023-12-31', '2024-01-31', '2024-02-29',
               '2024-03-31', '2024-04-30', '2024-05-31', '2024-06-30',
               '2024-07-31', '2024-08-31', '2024-09-30', '2024-10-31',
               '2024-11-30', '2024-12-31', '2025-01-31', '2025-02-28',
               '2025-03-31', '2025-04-30', '2025-05-31', '2025-06-30',
               '2025-07-31', '2025-08-31', '2025-09-30', '2025-10-31',
               '2025-11-30', '2025-12-31', '2026-01-31', '2026-02-28',
               '2026-03-31', '2026-04-30', '2026-05-31', '2026-06-30',
               '2026-07-31', '2026-08-31', '2026-09-30', '2026-10-31',
               '2026-11-30', '2026-12-31', '2027-01-31', '2027-02-28',
               '2027-03-31', '2027-04-30', '2027-05-31', '2027-06-30',
               '2027-07-31', '2027-08-31', '2027-09-30', '2027-10-31',
               '2027-11-30', '2027-12-31', '2028-01-31', '2028-02-29',
               '2028-03-31', '2028-04-30', '2028-05-31', '2028-06-30'],
              dtype='datetime64[ns]', freq='M')
```

In [95]:
```
help(pd.date_range)
```

```
Help on function date_range in module pandas.core.indexes.datetimes:

date_range(start=None, end=None, periods=None, freq=None, tz=None, normalize: 'boo
l' = False, name: 'Hashable' = None, closed: 'str | None | lib.NoDefault' = <no_de
fault>, inclusive: 'str | None' = None, **kwargs) -> 'DatetimeIndex'
    Return a fixed frequency DatetimeIndex.

    Returns the range of equally spaced time points (where the difference between
any
    two adjacent points is specified by the given frequency) such that they all
    satisfy `start <[=] x <[=] end`, where the first one and the last one are, res
p.,
    the first and last time points in that range that fall on the boundary of ``fr
eq``
    (if given as a frequency string) or that are valid for ``freq`` (if given as a
    :class:`pandas.tseries.offsets.DateOffset`). (If exactly one of ``start``,
    ``end``, or ``freq`` is *not* specified, this missing parameter can be compute
d
    given ``periods``, the number of timesteps in the range. See the note below.)

    Parameters
    ----------
    start : str or datetime-like, optional
        Left bound for generating dates.
    end : str or datetime-like, optional
        Right bound for generating dates.
    periods : int, optional
        Number of periods to generate.
    freq : str or DateOffset, default 'D'
        Frequency strings can have multiples, e.g. '5H'. See
        :ref:`here <timeseries.offset_aliases>` for a list of
        frequency aliases.
    tz : str or tzinfo, optional
        Time zone name for returning localized DatetimeIndex, for example
        'Asia/Hong_Kong'. By default, the resulting DatetimeIndex is
        timezone-naive.
    normalize : bool, default False
        Normalize start/end dates to midnight before generating date range.
    name : str, default None
        Name of the resulting DatetimeIndex.
    closed : {None, 'left', 'right'}, optional
        Make the interval closed with respect to the given frequency to
        the 'left', 'right', or both sides (None, the default).

        .. deprecated:: 1.4.0
           Argument `closed` has been deprecated to standardize boundary inputs.
           Use `inclusive` instead, to set each bound as closed or open.
    inclusive : {"both", "neither", "left", "right"}, default "both"
        Include boundaries; Whether to set each bound as closed or open.

        .. versionadded:: 1.4.0
    **kwargs
        For compatibility. Has no effect on the result.

    Returns
    -------
    rng : DatetimeIndex

    See Also
    --------
    DatetimeIndex : An immutable container for datetimes.
    timedelta_range : Return a fixed frequency TimedeltaIndex.
    period_range : Return a fixed frequency PeriodIndex.
    interval_range : Return a fixed frequency IntervalIndex.
```

```
    Notes
    -----
    Of the four parameters ``start``, ``end``, ``periods``, and ``freq``,
    exactly three must be specified. If ``freq`` is omitted, the resulting
    ``DatetimeIndex`` will have ``periods`` linearly spaced elements between
    ``start`` and ``end`` (closed on both sides).

    To learn more about the frequency strings, please see `this link
    <https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-aliases>`__.

    Examples
    --------
    **Specifying the values**

    The next four examples generate the same `DatetimeIndex`, but vary
    the combination of `start`, `end` and `periods`.

    Specify `start` and `end`, with the default daily frequency.

    >>> pd.date_range(start='1/1/2018', end='1/08/2018')
    DatetimeIndex(['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04',
                   '2018-01-05', '2018-01-06', '2018-01-07', '2018-01-08'],
                  dtype='datetime64[ns]', freq='D')

    Specify `start` and `periods`, the number of periods (days).

    >>> pd.date_range(start='1/1/2018', periods=8)
    DatetimeIndex(['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04',
                   '2018-01-05', '2018-01-06', '2018-01-07', '2018-01-08'],
                  dtype='datetime64[ns]', freq='D')

    Specify `end` and `periods`, the number of periods (days).

    >>> pd.date_range(end='1/1/2018', periods=8)
    DatetimeIndex(['2017-12-25', '2017-12-26', '2017-12-27', '2017-12-28',
                   '2017-12-29', '2017-12-30', '2017-12-31', '2018-01-01'],
                  dtype='datetime64[ns]', freq='D')

    Specify `start`, `end`, and `periods`; the frequency is generated
    automatically (linearly spaced).

    >>> pd.date_range(start='2018-04-24', end='2018-04-27', periods=3)
    DatetimeIndex(['2018-04-24 00:00:00', '2018-04-25 12:00:00',
                   '2018-04-27 00:00:00'],
                  dtype='datetime64[ns]', freq=None)

    **Other Parameters**

    Changed the `freq` (frequency) to ``'M'`` (month end frequency).

    >>> pd.date_range(start='1/1/2018', periods=5, freq='M')
    DatetimeIndex(['2018-01-31', '2018-02-28', '2018-03-31', '2018-04-30',
                   '2018-05-31'],
                  dtype='datetime64[ns]', freq='M')

    Multiples are allowed

    >>> pd.date_range(start='1/1/2018', periods=5, freq='3M')
    DatetimeIndex(['2018-01-31', '2018-04-30', '2018-07-31', '2018-10-31',
                   '2019-01-31'],
                  dtype='datetime64[ns]', freq='3M')
```

```
    `freq` can also be specified as an Offset object.

    >>> pd.date_range(start='1/1/2018', periods=5, freq=pd.offsets.MonthEnd(3))
    DatetimeIndex(['2018-01-31', '2018-04-30', '2018-07-31', '2018-10-31',
                   '2019-01-31'],
                  dtype='datetime64[ns]', freq='3M')

    Specify `tz` to set the timezone.

    >>> pd.date_range(start='1/1/2018', periods=5, tz='Asia/Tokyo')
    DatetimeIndex(['2018-01-01 00:00:00+09:00', '2018-01-02 00:00:00+09:00',
                   '2018-01-03 00:00:00+09:00', '2018-01-04 00:00:00+09:00',
                   '2018-01-05 00:00:00+09:00'],
                  dtype='datetime64[ns, Asia/Tokyo]', freq='D')

    `inclusive` controls whether to include `start` and `end` that are on the
    boundary. The default, "both", includes boundary points on either end.

    >>> pd.date_range(start='2017-01-01', end='2017-01-04', inclusive="both")
    DatetimeIndex(['2017-01-01', '2017-01-02', '2017-01-03', '2017-01-04'],
                  dtype='datetime64[ns]', freq='D')

    Use ``inclusive='left'`` to exclude `end` if it falls on the boundary.

    >>> pd.date_range(start='2017-01-01', end='2017-01-04', inclusive='left')
    DatetimeIndex(['2017-01-01', '2017-01-02', '2017-01-03'],
                  dtype='datetime64[ns]', freq='D')

    Use ``inclusive='right'`` to exclude `start` if it falls on the boundary, and
    similarly ``inclusive='neither'`` will exclude both `start` and `end`.

    >>> pd.date_range(start='2017-01-01', end='2017-01-04', inclusive='right')
    DatetimeIndex(['2017-01-02', '2017-01-03', '2017-01-04'],
                  dtype='datetime64[ns]', freq='D')
```

In [96]:
```
DF1=pd.DataFrame(np.random.randn(100,4), index=dates,columns=['Delhi', 'Mumbai', '
```

In [97]:
```
DF1
```

Out[97]:

	Delhi	Mumbai	Calcutta	Chennai
2020-03-31	1.805688	-1.644654	-0.443906	-0.274933
2020-04-30	1.023040	-0.037232	0.402260	-0.654708
2020-05-31	1.123937	0.281373	-0.908020	0.077146
2020-06-30	-0.544242	-0.905774	0.133154	0.215688
2020-07-31	-0.248010	1.013235	-0.489657	1.178466
...	...	...	...	...
2028-02-29	1.426856	1.196641	1.311469	-0.645135
2028-03-31	-0.523009	0.887019	-0.672808	-0.833629
2028-04-30	-0.391467	0.398324	-0.507522	0.774869
2028-05-31	0.083183	0.636199	0.043913	-0.697511
2028-06-30	0.654763	-0.264198	-0.992724	0.010119

100 rows × 4 columns

Duplicate date indices

In [120…
```
ddates=pd.DatetimeIndex(['1/1/2010','1/2/2010','1/2/2010','1/2/2010','1/3/2010'])
```

In [121…
```
d_ts=pd.Series(np.arange(5),index=ddates)
```

In [122…
```
d_ts
```

Out[122]:
```
2010-01-01    0
2010-01-02    1
2010-01-02    2
2010-01-02    3
2010-01-03    4
dtype: int32
```

In [123…
```
d_ts['1/1/2010']
```

Out[123]:
```
0
```

In [124…
```
d_ts['1/2/2010']
```

Out[124]:
```
2010-01-02    1
2010-01-02    2
2010-01-02    3
dtype: int32
```

In [105…
```
grp=d_ts.groupby(level=0)
```

In [106…
```
grp
```

Out[106]:
```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x000002218B16CDC0>
```

In [107…
```
grp.mean()
```

Out[107]:
```
2010-01-01    0.0
2010-01-02    2.0
2010-01-03    4.0
dtype: float64
```

In [108…
```
grp.count()
```

Out[108]:
```
2010-01-01    1
2010-01-02    3
2010-01-03    1
dtype: int64
```

In [109…
```
dates=[datetime(2011,3,5), datetime(2012,7,4), datetime(2014,3,5), datetime(2017,5,
ds=pd.Series(np.random.randn(6),index=dates)
```

In [110…
```
ds
```

Out[110]:
```
2011-03-05    1.260086
2012-07-04   -0.624688
2014-03-05    1.254635
2017-05-06    0.711369
2014-07-15    0.023494
2011-05-06    1.306218
dtype: float64
```

In [113…
```
res_date=ds.resample('D')
```

```
res_date
```

Out[113]:
```
<pandas.core.resample.DatetimeIndexResampler object at 0x000002218B1BA6D0>
```

In [114…
```
res_date.mean()
```

Out[114]:
```
2011-03-05    1.260086
2011-03-06         NaN
2011-03-07         NaN
2011-03-08         NaN
2011-03-09         NaN
                ...
2017-05-02         NaN
2017-05-03         NaN
2017-05-04         NaN
2017-05-05         NaN
2017-05-06    0.711369
Freq: D, Length: 2255, dtype: float64
```

In [115…
```
res_date.size()
```

Out[115]:
```
2011-03-05    1
2011-03-06    0
2011-03-07    0
2011-03-08    0
2011-03-09    0
             ..
2017-05-02    0
2017-05-03    0
2017-05-04    0
2017-05-05    0
2017-05-06    1
Freq: D, Length: 2255, dtype: int64
```

In [125…
```
index=pd.date_range(start='04/02/2020 09:20:00',periods=10,normalize=True)
```

In [126…
```
index
```

Out[126]:
```
DatetimeIndex(['2020-04-02', '2020-04-03', '2020-04-04', '2020-04-05',
               '2020-04-06', '2020-04-07', '2020-04-08', '2020-04-09',
               '2020-04-10', '2020-04-11'],
              dtype='datetime64[ns]', freq='D')
```

In [130…
```
from pandas.tseries.offsets import Hour,Minute
```

In [131…
```
hour=Hour(4)
```

In [132…
```
hour
```

Out[132]:
```
<4 * Hours>
```

In [133…
```
pd.date_range('1/1/2011','2/1/2011 11:59', freq='1h15min')
```

Out[133]:
```
DatetimeIndex(['2011-01-01 00:00:00', '2011-01-01 01:15:00',
               '2011-01-01 02:30:00', '2011-01-01 03:45:00',
               '2011-01-01 05:00:00', '2011-01-01 06:15:00',
               '2011-01-01 07:30:00', '2011-01-01 08:45:00',
               '2011-01-01 10:00:00', '2011-01-01 11:15:00',
               ...
               '2011-01-31 23:45:00', '2011-02-01 01:00:00',
               '2011-02-01 02:15:00', '2011-02-01 03:30:00',
               '2011-02-01 04:45:00', '2011-02-01 06:00:00',
               '2011-02-01 07:15:00', '2011-02-01 08:30:00',
               '2011-02-01 09:45:00', '2011-02-01 11:00:00'],
              dtype='datetime64[ns]', length=605, freq='75T')
```

In [135…
```
rn_date=pd.date_range('1/1/2021','9/8/2021',freq='WOM-1FRI')
list(rn_date)
rn_date
```

Out[135]:
```
DatetimeIndex(['2021-01-01', '2021-02-05', '2021-03-05', '2021-04-02',
               '2021-05-07', '2021-06-04', '2021-07-02', '2021-08-06',
               '2021-09-03'],
              dtype='datetime64[ns]', freq='WOM-1FRI')
```

Base time series frequencies (not comprehensive)

D Day Calendar daily B BusinessDay Business daily H Hour Hourly T or min Minute Minutely S Second Secondly L or ms Milli Millisecond (1/1,000 of 1 second) U Micro Microsecond (1/1,000,000 of 1 second) M MonthEnd Last calendar day of month BM BusinessMonthEnd Last business day (weekday) of month MS MonthBegin First calendar day of month BMS BusinessMonthBegin First weekday of month W-MON, W-TUE, ... Week Weekly on given day of week (MON, TUE, WED, THU, FRI, SAT, or SUN) WOM-1MON, WOM-2MON, ... WeekOfMonth Generate weekly dates in the first, second, third, or fourth week of the month (e.g., WOM-3FRI for the third Friday of each month) Q-JAN, Q-FEB, ... QuarterEnd Quarterly dates anchored on last calendar day of each month, for year ending in indicated month (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC) WOM-1MON, WOM-2MON, ... WeekOfMonth Generate weekly dates in the first, second, third, or fourth week of the month (e.g., WOM-3FRI for the third Friday of each month) Q-JAN, Q-FEB, ... QuarterEnd Quarterly dates anchored on last calendar day of each month, for year ending in indicated month (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC) BQ-JAN, BQ-FEB, ... BusinessQuarterEnd Quarterly dates anchored on last weekday day of each month, for year ending in indicated month QS-JAN, QS-FEB, ... QuarterBegin Quarterly dates anchored on first calendar day of each month, for year ending in indicated month BQS-JAN, BQS-FEB, ... BusinessQuarterBegin Quarterly dates anchored on first weekday day of each month, for year ending in indicated month A-JAN, A-FEB, ... YearEnd Annual dates anchored on last calendar day of given month (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC) BA-JAN, BA-FEB, ... BusinessYearEnd Annual dates anchored on last weekday of given month AS-JAN, AS-FEB, ... YearBegin Annual dates anchored on first day of given month BAS-JAN, BAS-FEB, ... BusinessYearBegin Annual dates anchored on first weekday of given month

In [136…
```
ts=pd.Series(np.random.randn(10),index=pd.date_range('1/1/2014',periods=10,freq='5
```

In [137…
```
ts
```

Out[137]:
```
2014-01-01   -0.042963
2014-01-06   -0.128384
2014-01-11   -1.123319
2014-01-16   -0.000080
2014-01-21    0.743368
2014-01-26    0.799815
2014-01-31   -0.809785
2014-02-05   -0.419343
2014-02-10    0.724864
2014-02-15   -0.042313
Freq: 5D, dtype: float64
```

In [138…
```
ts.shift(-2)
```

Out[138]:
```
2014-01-01   -1.123319
2014-01-06   -0.000080
2014-01-11    0.743368
2014-01-16    0.799815
2014-01-21   -0.809785
2014-01-26   -0.419343
2014-01-31    0.724864
2014-02-05   -0.042313
2014-02-10         NaN
2014-02-15         NaN
Freq: 5D, dtype: float64
```

In [139…
```
ts/ts.shift(-1)-1
```

Out[139]:
```
2014-01-01       -0.665359
2014-01-06       -0.885710
2014-01-11    14016.290078
2014-01-16       -1.000108
2014-01-21       -0.070576
2014-01-26       -1.987689
2014-01-31        0.931080
2014-02-05       -1.578512
2014-02-10      -18.131185
2014-02-15             NaN
Freq: 5D, dtype: float64
```

In [140…
```
ts.shift(2,freq='M')
```

Out[140]:
```
2014-02-28   -0.042963
2014-02-28   -0.128384
2014-02-28   -1.123319
2014-02-28   -0.000080
2014-02-28    0.743368
2014-02-28    0.799815
2014-03-31   -0.809785
2014-03-31   -0.419343
2014-03-31    0.724864
2014-03-31   -0.042313
dtype: float64
```

In [141…
```
ts.shift(2)
```

Out[141]:
```
2014-01-01         NaN
2014-01-06         NaN
2014-01-11   -0.042963
2014-01-16   -0.128384
2014-01-21   -1.123319
2014-01-26   -0.000080
2014-01-31    0.743368
2014-02-05    0.799815
2014-02-10   -0.809785
2014-02-15   -0.419343
Freq: 5D, dtype: float64
```

In [142…
```
ts
```

Out[142]:
```
2014-01-01   -0.042963
2014-01-06   -0.128384
2014-01-11   -1.123319
2014-01-16   -0.000080
2014-01-21    0.743368
2014-01-26    0.799815
2014-01-31   -0.809785
2014-02-05   -0.419343
2014-02-10    0.724864
2014-02-15   -0.042313
Freq: 5D, dtype: float64
```

In [143…
```
ts.shift(2,freq='D')
```

Out[143]:
```
2014-01-03   -0.042963
2014-01-08   -0.128384
2014-01-13   -1.123319
2014-01-18   -0.000080
2014-01-23    0.743368
2014-01-28    0.799815
2014-02-02   -0.809785
2014-02-07   -0.419343
2014-02-12    0.724864
2014-02-17   -0.042313
Freq: 5D, dtype: float64
```

In [144…
```
ts
```

Out[144]:
```
2014-01-01   -0.042963
2014-01-06   -0.128384
2014-01-11   -1.123319
2014-01-16   -0.000080
2014-01-21    0.743368
2014-01-26    0.799815
2014-01-31   -0.809785
2014-02-05   -0.419343
2014-02-10    0.724864
2014-02-15   -0.042313
Freq: 5D, dtype: float64
```

In [145…
```
ts.shift(4,freq='h')
```

Out[145]:
```
2014-01-01 04:00:00   -0.042963
2014-01-06 04:00:00   -0.128384
2014-01-11 04:00:00   -1.123319
2014-01-16 04:00:00   -0.000080
2014-01-21 04:00:00    0.743368
2014-01-26 04:00:00    0.799815
2014-01-31 04:00:00   -0.809785
2014-02-05 04:00:00   -0.419343
2014-02-10 04:00:00    0.724864
2014-02-15 04:00:00   -0.042313
Freq: 5D, dtype: float64
```

In [146…
```
from pandas.tseries.offsets import Day,MonthEnd
```

In [147…
```
now=datetime(2017,10,17)
```

In [148…
```
now
```

Out[148]:
```
datetime.datetime(2017, 10, 17, 0, 0)
```

In [149…
```
now+3*Day()
```

Out[149]:
```
Timestamp('2017-10-20 00:00:00')
```

In [150…
```
now+MonthEnd(12)
```

Out[150]:
```
Timestamp('2018-09-30 00:00:00')
```

In [151…
```
offset=MonthEnd()
```

In [152…
```
offset.rollforward(now)
```

Out[152]:
```
Timestamp('2017-10-31 00:00:00')
```

In [153…
```
offset.rollback(now)
```

Out[153]:
```
Timestamp('2017-09-30 00:00:00')
```

In [154…
```
ts
```

Out[154]:
```
2014-01-01   -0.042963
2014-01-06   -0.128384
2014-01-11   -1.123319
2014-01-16   -0.000080
2014-01-21    0.743368
2014-01-26    0.799815
2014-01-31   -0.809785
2014-02-05   -0.419343
2014-02-10    0.724864
2014-02-15   -0.042313
Freq: 5D, dtype: float64
```

In [155…
```
ts.groupby(offset.rollforward).mean()
```

Out[155]:
```
2014-01-31   -0.080193
2014-02-28    0.087736
dtype: float64
```

In [156…
```
ts.resample('M').mean()
```

Out[156]:
```
2014-01-31   -0.080193
2014-02-28    0.087736
Freq: M, dtype: float64
```

tm zone

In [157…
```
import numpy as np
import pandas as pd
import pytz
pytz.common_timezones[-5:]
```

Out[157]:
```
['US/Eastern', 'US/Hawaii', 'US/Mountain', 'US/Pacific', 'UTC']
```

In [158…
```
tz1=pytz.timezone('America/New_York')
```

In [159…
```
tz1
```

Out[159]:
```
<DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>
```

In [160…
```
rntz=pd.date_range('4/5/2015 10:10',periods=6,freq='D',tz='UTC')
rntz
```

Out[160]:
```
DatetimeIndex(['2015-04-05 10:10:00+00:00', '2015-04-06 10:10:00+00:00',
               '2015-04-07 10:10:00+00:00', '2015-04-08 10:10:00+00:00',
               '2015-04-09 10:10:00+00:00', '2015-04-10 10:10:00+00:00'],
              dtype='datetime64[ns, UTC]', freq='D')
```

In [161…
```
ts=pd.Series(np.random.randn(len(rntz)),index=rntz)
```

In [162…
```
ts
```

Out[162]:
```
2015-04-05 10:10:00+00:00    1.878185
2015-04-06 10:10:00+00:00    0.529026
2015-04-07 10:10:00+00:00    1.165834
2015-04-08 10:10:00+00:00   -0.531072
2015-04-09 10:10:00+00:00   -0.326039
2015-04-10 10:10:00+00:00    1.907056
Freq: D, dtype: float64
```

In [163…
```
print(ts.index.tz)
```

```
UTC
```

In [164…
```
ts.index
```

Out[164]:
```
DatetimeIndex(['2015-04-05 10:10:00+00:00', '2015-04-06 10:10:00+00:00',
               '2015-04-07 10:10:00+00:00', '2015-04-08 10:10:00+00:00',
               '2015-04-09 10:10:00+00:00', '2015-04-10 10:10:00+00:00'],
              dtype='datetime64[ns, UTC]', freq='D')
```

In [289…
```
rntz=pd.date_range('4/5/2015 10:10',periods=6 ,freq='D')
rntz
```

Out[289]:
```
DatetimeIndex(['2015-04-05 10:10:00', '2015-04-06 10:10:00',
               '2015-04-07 10:10:00', '2015-04-08 10:10:00',
               '2015-04-09 10:10:00', '2015-04-10 10:10:00'],
              dtype='datetime64[ns]', freq='D')
```

In [181…
```
ts1=pd.Series(np.random.randn(len(rntz)),index=rntz)
```

In [182…
```
ts1
```

Out[182]:
```
2015-04-05 10:10:00   -0.218631
2015-04-06 10:10:00   -1.110326
2015-04-07 10:10:00    1.251186
2015-04-08 10:10:00    0.140152
2015-04-09 10:10:00   -0.450883
2015-04-10 10:10:00   -1.524437
Freq: D, dtype: float64
```

In [168…
```
ts_utc=ts1.tz_localize('UTC')
```

In [169…
```
ts_utc
```

Out[169]:
```
2015-04-05 10:10:00+00:00    1.042268
2015-04-06 10:10:00+00:00   -0.518336
2015-04-07 10:10:00+00:00    0.756953
2015-04-08 10:10:00+00:00    0.976125
2015-04-09 10:10:00+00:00   -0.552183
2015-04-10 10:10:00+00:00    1.259548
Freq: D, dtype: float64
```

In [170…
```
ts_utc.tz_convert('America/New_York')
```

Out[170]:
```
2015-04-05 06:10:00-04:00    1.042268
2015-04-06 06:10:00-04:00   -0.518336
2015-04-07 06:10:00-04:00    0.756953
2015-04-08 06:10:00-04:00    0.976125
2015-04-09 06:10:00-04:00   -0.552183
2015-04-10 06:10:00-04:00    1.259548
Freq: D, dtype: float64
```

In [171…
```
ts_utc.tz_convert('Europe/Berlin')
```

Out[171]:
```
2015-04-05 12:10:00+02:00    1.042268
2015-04-06 12:10:00+02:00   -0.518336
2015-04-07 12:10:00+02:00    0.756953
2015-04-08 12:10:00+02:00    0.976125
2015-04-09 12:10:00+02:00   -0.552183
2015-04-10 12:10:00+02:00    1.259548
Freq: D, dtype: float64
```

In [172…
```
ts_utc.tz_convert('Asia/Shanghai')
```

Out[172]:
```
2015-04-05 18:10:00+08:00    1.042268
2015-04-06 18:10:00+08:00   -0.518336
2015-04-07 18:10:00+08:00    0.756953
2015-04-08 18:10:00+08:00    0.976125
2015-04-09 18:10:00+08:00   -0.552183
2015-04-10 18:10:00+08:00    1.259548
Freq: D, dtype: float64
```

periods

In [173…
```
p=pd.Period(2014,freq='A-Dec')
```

In [174…
```
p
```

Out[174]:
```
Period('2014', 'A-DEC')
```

In [175…
```
p+5
```

Out[175]:
```
Period('2019', 'A-DEC')
```

In [176…
```
p-1
```
Out[176]:
```
Period('2013', 'A-DEC')
```

In [284…
```
rng=pd.period_range('1/1/2010','31/3/2010',freq='M')
```

In [285…
```
rng
```
Out[285]:
```
PeriodIndex(['2010-01', '2010-02', '2010-03'], dtype='period[M]')
```

In [184…
```
pd.Series(np.random.randn(3),index=rng)
```
Out[184]:
```
2010-01    0.714331
2010-02    1.159487
2010-03    2.236697
Freq: M, dtype: float64
```

In [185…
```
p=pd.Period('2010',freq='Q-Dec')
```

In [186…
```
p
```
Out[186]:
```
Period('2010Q1', 'Q-DEC')
```

In [187…
```
p.asfreq('M',how='start')
```
Out[187]:
```
Period('2010-01', 'M')
```

In [188…
```
p.asfreq('M',how='end')
```
Out[188]:
```
Period('2010-03', 'M')
```

In [189…
```
import pandas as pd
```

In [190…
```
p=pd.Period('2007',freq='A-JUN')
```

In [191…
```
p
```
Out[191]:
```
Period('2007', 'A-JUN')
```

In [192…
```
p.asfreq('M',how='start')
```
Out[192]:
```
Period('2006-07', 'M')
```

In [193…
```
p.asfreq('M',how='end')
```
Out[193]:
```
Period('2007-06', 'M')
```

In [194…
```
p=pd.Period('Aug-2007','M')
```

In [195…
```
p.asfreq('A-JUN')
```
Out[195]:
```
Period('2008', 'A-JUN')
```

In [196…
```
rng=pd.period_range('2006','2009',freq='A-DEC')
```

In [197…
```
rng
```

Out[197]:
```
PeriodIndex(['2006', '2007', '2008', '2009'], dtype='period[A-DEC]')
```

In [198…]:
```
import numpy as np
ts=pd.Series(np.random.randn(len(rng)),index=rng)
```

In [199…]:
```
ts
```

Out[199]:
```
2006   -0.016256
2007    0.567345
2008    0.684288
2009   -0.508710
Freq: A-DEC, dtype: float64
```

In [200…]:
```
ts.asfreq('M',how='start')
```

Out[200]:
```
2006-01   -0.016256
2007-01    0.567345
2008-01    0.684288
2009-01   -0.508710
Freq: M, dtype: float64
```

In [201…]:
```
ts.asfreq('B',how='end')
```

Out[201]:
```
2006-12-29   -0.016256
2007-12-31    0.567345
2008-12-31    0.684288
2009-12-31   -0.508710
Freq: B, dtype: float64
```

In [202…]:
```
pq=pd.Period('2011Q4',freq='Q-JAN')
```

In [203…]:
```
pq
```

Out[203]:
```
Period('2011Q4', 'Q-JAN')
```

In [204…]:
```
pq.asfreq('D','end')
```

Out[204]:
```
Period('2011-01-31', 'D')
```

In [205…]:
```
ptm=(p.asfreq('B','e')-1).asfreq('T','s')+16*60
```

In [206…]:
```
ptm
```

Out[206]:
```
Period('2007-08-30 16:00', 'T')
```

In [207…]:
```
rng=pd.period_range('2011Q3','2012Q4',freq='Q-JAN')
```

In [208…]:
```
rng
```

Out[208]:
```
PeriodIndex(['2011Q3', '2011Q4', '2012Q1', '2012Q2', '2012Q3', '2012Q4'], dtype='p
eriod[Q-JAN]')
```

In [209…]:
```
tsq=pd.Series(np.arange(len(rng)),index=rng)
```

In [210…]:
```
tsq
```

Out[210]:
```
2011Q3    0
2011Q4    1
2012Q1    2
2012Q2    3
2012Q3    4
2012Q4    5
Freq: Q-JAN, dtype: int32
```

In [211…
```
tm_q=(rng.asfreq('B','e')-1).asfreq('T','s')+16*60
```

In [212…
```
tm_q
```

Out[212]:
```
PeriodIndex(['2010-10-28 16:00', '2011-01-28 16:00', '2011-04-28 16:00',
             '2011-07-28 16:00', '2011-10-28 16:00', '2012-01-30 16:00'],
            dtype='period[T]')
```

In [214…
```
tsq.index=tm_q.to_timestamp()
```

In [215…
```
tsq
```

Out[215]:
```
2010-10-28 16:00:00    0
2011-01-28 16:00:00    1
2011-04-28 16:00:00    2
2011-07-28 16:00:00    3
2011-10-28 16:00:00    4
2012-01-30 16:00:00    5
dtype: int32
```

In [280…
```
rng=pd.date_range('2000-01-01',periods=3,freq='M')
```

In [281…
```
rng
```

Out[281]:
```
DatetimeIndex(['2000-01-31', '2000-02-29', '2000-03-31'], dtype='datetime64[ns]',
freq='M')
```

In [282…
```
ts=pd.Series(np.random.randn(3),index=rng)
```

In [283…
```
ts
```

Out[283]:
```
2000-01-31    0.249529
2000-02-29   -0.171957
2000-03-31   -1.996193
Freq: M, dtype: float64
```

In [220…
```
pts=ts.to_period()
```

In [221…
```
pts
```

Out[221]:
```
2000-01    0.757461
2000-02   -0.870388
2000-03   -1.596932
Freq: M, dtype: float64
```

In [222…
```
rng=pd.date_range('11/29/2000',periods=6,freq='D')
```

In [223…
```
rng
```

Out[223]:
```
DatetimeIndex(['2000-11-29', '2000-11-30', '2000-12-01', '2000-12-02',
               '2000-12-03', '2000-12-04'],
              dtype='datetime64[ns]', freq='D')
```

In [224…
```
ts2=pd.Series(np.random.randn(6),index=rng)
```

In [225…]
```
ts2
```

Out[225]:
```
2000-11-29   -2.283062
2000-11-30    0.133949
2000-12-01    1.196897
2000-12-02    0.689055
2000-12-03    0.425956
2000-12-04   -0.846853
Freq: D, dtype: float64
```

resampling and free conversion

In [226…]
```
rng=pd.date_range('2010-01-01',periods=100,freq='D')
```

In [227…]
```
rng
```

Out[227]:
```
DatetimeIndex(['2010-01-01', '2010-01-02', '2010-01-03', '2010-01-04',
               '2010-01-05', '2010-01-06', '2010-01-07', '2010-01-08',
               '2010-01-09', '2010-01-10', '2010-01-11', '2010-01-12',
               '2010-01-13', '2010-01-14', '2010-01-15', '2010-01-16',
               '2010-01-17', '2010-01-18', '2010-01-19', '2010-01-20',
               '2010-01-21', '2010-01-22', '2010-01-23', '2010-01-24',
               '2010-01-25', '2010-01-26', '2010-01-27', '2010-01-28',
               '2010-01-29', '2010-01-30', '2010-01-31', '2010-02-01',
               '2010-02-02', '2010-02-03', '2010-02-04', '2010-02-05',
               '2010-02-06', '2010-02-07', '2010-02-08', '2010-02-09',
               '2010-02-10', '2010-02-11', '2010-02-12', '2010-02-13',
               '2010-02-14', '2010-02-15', '2010-02-16', '2010-02-17',
               '2010-02-18', '2010-02-19', '2010-02-20', '2010-02-21',
               '2010-02-22', '2010-02-23', '2010-02-24', '2010-02-25',
               '2010-02-26', '2010-02-27', '2010-02-28', '2010-03-01',
               '2010-03-02', '2010-03-03', '2010-03-04', '2010-03-05',
               '2010-03-06', '2010-03-07', '2010-03-08', '2010-03-09',
               '2010-03-10', '2010-03-11', '2010-03-12', '2010-03-13',
               '2010-03-14', '2010-03-15', '2010-03-16', '2010-03-17',
               '2010-03-18', '2010-03-19', '2010-03-20', '2010-03-21',
               '2010-03-22', '2010-03-23', '2010-03-24', '2010-03-25',
               '2010-03-26', '2010-03-27', '2010-03-28', '2010-03-29',
               '2010-03-30', '2010-03-31', '2010-04-01', '2010-04-02',
               '2010-04-03', '2010-04-04', '2010-04-05', '2010-04-06',
               '2010-04-07', '2010-04-08', '2010-04-09', '2010-04-10'],
              dtype='datetime64[ns]', freq='D')
```

In [228…]
```
import numpy as np
ts=pd.Series(np.random.randn(len(rng)),index=rng)
```

In [229…]
```
ts
```

Out[229]:
```
2010-01-01    0.225945
2010-01-02    2.094792
2010-01-03   -1.734555
2010-01-04    0.970358
2010-01-05    0.865721
                ...
2010-04-06   -0.480770
2010-04-07    1.238337
2010-04-08   -0.306074
2010-04-09    0.378287
2010-04-10    2.667155
Freq: D, Length: 100, dtype: float64
```

In [230…]
```
ts.resample('M').mean()
```

Out[230]:
```
2010-01-31    0.223728
2010-02-28   -0.045513
2010-03-31    0.077543
2010-04-30    0.417302
Freq: M, dtype: float64
```

In [231…
```
ts.resample('M',kind='period').mean()
```

Out[231]:
```
2010-01    0.223728
2010-02   -0.045513
2010-03    0.077543
2010-04    0.417302
Freq: M, dtype: float64
```

In [235…
```
rng=pd.date_range('2010-01-01',periods=12,freq='T')
```

In [236…
```
ts=pd.Series(np.arange(12),index=rng)
```

In [237…
```
ts
```

Out[237]:
```
2010-01-01 00:00:00     0
2010-01-01 00:01:00     1
2010-01-01 00:02:00     2
2010-01-01 00:03:00     3
2010-01-01 00:04:00     4
2010-01-01 00:05:00     5
2010-01-01 00:06:00     6
2010-01-01 00:07:00     7
2010-01-01 00:08:00     8
2010-01-01 00:09:00     9
2010-01-01 00:10:00    10
2010-01-01 00:11:00    11
Freq: T, dtype: int32
```

In [238…
```
ts.resample('5min',closed='right').sum()
```

Out[238]:
```
2009-12-31 23:55:00     0
2010-01-01 00:00:00    15
2010-01-01 00:05:00    40
2010-01-01 00:10:00    11
Freq: 5T, dtype: int32
```

In [239…
```
ts.resample('5min',closed='right',label='right',loffset='-1s').sum()
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_19196\1689922366.py:1: FutureWarning:
'loffset' in .resample() and in Grouper() is deprecated.

>>> df.resample(freq="3s", loffset="8H")

becomes:

>>> from pandas.tseries.frequencies import to_offset
>>> df = df.resample(freq="3s").mean()
>>> df.index = df.index.to_timestamp() + to_offset("8H")

  ts.resample('5min',closed='right',label='right',loffset='-1s').sum()
```

Out[239]:
```
2009-12-31 23:59:59     0
2010-01-01 00:04:59    15
2010-01-01 00:09:59    40
2010-01-01 00:14:59    11
Freq: 5T, dtype: int32
```

In [240…
```
ts.resample('5min',closed='left').sum()
```

Out[240]:
```
2010-01-01 00:00:00    10
2010-01-01 00:05:00    35
2010-01-01 00:10:00    21
Freq: 5T, dtype: int32
```

In [245…
```
ts.resample('5min').ohlc()
```

Out[245]:

	open	high	low	close
2010-01-01 00:00:00	0	4	0	4
2010-01-01 00:05:00	5	9	5	9
2010-01-01 00:10:00	10	11	10	11

In [246…
```
frame=pd.DataFrame(np.random.randn(2,4),index=pd.date_range('1/1/2010', periods=2,
```

In [247…
```
frame
```

Out[247]:

	A	B	C	D
2010-01-06	-0.507460	-0.090415	1.651577	0.359781
2010-01-13	2.441199	1.625084	0.535206	-1.840866

In [248…
```
df_daily=frame.resample('D').ffill()
df_daily
```

Out[248]:

	A	B	C	D
2010-01-06	-0.507460	-0.090415	1.651577	0.359781
2010-01-07	-0.507460	-0.090415	1.651577	0.359781
2010-01-08	-0.507460	-0.090415	1.651577	0.359781
2010-01-09	-0.507460	-0.090415	1.651577	0.359781
2010-01-10	-0.507460	-0.090415	1.651577	0.359781
2010-01-11	-0.507460	-0.090415	1.651577	0.359781
2010-01-12	-0.507460	-0.090415	1.651577	0.359781
2010-01-13	2.441199	1.625084	0.535206	-1.840866

In [249…
```
df_daily.resample('W-Wed').ffill()
```

Out[249]:

	A	B	C	D
2010-01-06	-0.507460	-0.090415	1.651577	0.359781
2010-01-13	2.441199	1.625084	0.535206	-1.840866

In [250…
```
df_daily=frame.resample('W-FRI').ffill()
```

In [251…
```
df_daily
```

Out[251]:

	A	B	C	D
2010-01-08	-0.507460	-0.090415	1.651577	0.359781
2010-01-15	2.441199	1.625084	0.535206	-1.840866

In [252…
```
frame=pd.DataFrame(np.random.randn(24,4),index=pd.period_range('1-2010','12-2011',
```

In [253…
```
frame
```

Out[253]:

	A	B	C	D
2010-01	-0.196685	1.030858	0.609206	-1.106616
2010-02	-0.348925	0.659906	-0.278787	-0.503774
2010-03	0.464496	1.156121	-1.265788	-1.904670
2010-04	1.773508	-0.961997	0.610031	-0.557249
2010-05	0.846673	1.059435	1.060110	0.073103
2010-06	-1.525703	-1.877537	-0.916709	0.842509
2010-07	-0.907069	1.283523	0.102073	0.961143
2010-08	-0.476070	0.342375	1.024200	2.326169
2010-09	0.226730	0.194785	0.968480	-1.167262
2010-10	0.487512	-0.481461	0.890352	0.141316
2010-11	0.614603	-1.905556	-0.877312	0.102842
2010-12	0.202772	0.938525	-0.128768	1.136594
2011-01	-0.039739	-1.237386	-0.735527	-1.328682
2011-02	2.226903	-0.065380	0.196212	-0.372764
2011-03	0.778392	0.715364	0.649016	0.341205
2011-04	-1.041785	-3.489955	-1.825042	-1.097427
2011-05	-1.178291	-1.558138	-0.421300	-1.554493
2011-06	-1.083046	-0.255184	-1.571503	-0.058550
2011-07	1.641717	0.162604	2.541272	1.343382
2011-08	-1.273311	1.763216	0.181073	0.923985
2011-09	-0.111058	-0.222709	0.273381	0.854137
2011-10	-0.364227	-1.300867	-1.011528	-1.311209
2011-11	0.746502	-0.559961	-1.194618	-0.078144
2011-12	0.789793	-0.977723	0.569523	0.286644

In [254…
```
annual_frame=frame.resample('A-DEC').mean()
```

In [255…
```
annual_frame
```

Out[255]:

	A	B	C	D
2010	0.096820	0.119915	0.149757	0.028675
2011	0.090988	-0.585510	-0.195753	-0.170993

In [256…
```
annual_frame.resample('Q-DEC').ffill()
```

Out[256]:

	A	B	C	D
2010Q1	0.096820	0.119915	0.149757	0.028675
2010Q2	0.096820	0.119915	0.149757	0.028675
2010Q3	0.096820	0.119915	0.149757	0.028675
2010Q4	0.096820	0.119915	0.149757	0.028675
2011Q1	0.090988	-0.585510	-0.195753	-0.170993
2011Q2	0.090988	-0.585510	-0.195753	-0.170993
2011Q3	0.090988	-0.585510	-0.195753	-0.170993
2011Q4	0.090988	-0.585510	-0.195753	-0.170993

In [257…
```
annual_frame.resample('Q-MAR').ffill()
```

Out[257]:

	A	B	C	D
2010Q4	0.096820	0.119915	0.149757	0.028675
2011Q1	0.096820	0.119915	0.149757	0.028675
2011Q2	0.096820	0.119915	0.149757	0.028675
2011Q3	0.096820	0.119915	0.149757	0.028675
2011Q4	0.090988	-0.585510	-0.195753	-0.170993
2012Q1	0.090988	-0.585510	-0.195753	-0.170993
2012Q2	0.090988	-0.585510	-0.195753	-0.170993
2012Q3	0.090988	-0.585510	-0.195753	-0.170993

moving window

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

In [258…
```
close_px_all = pd.read_csv('stock_px_2.csv',parse_dates=True, index_col=0)
close_px_all
```

Out[258]:

	AAPL	MSFT	XOM	SPX
2003-02-01	7.40	21.11	29.22	909.03
2003-03-01	7.45	21.14	29.24	908.59
2003-06-01	7.45	21.52	29.96	929.01
2003-07-01	7.43	21.93	28.95	922.93
2003-08-01	7.28	21.31	28.83	909.93
...	...	...	...	...
2011-10-10	388.81	26.94	76.28	1194.89
2011-11-10	400.29	27.00	76.27	1195.54
2011-12-10	402.19	26.96	77.16	1207.25
2011-10-13	408.43	27.18	76.37	1203.66
2011-10-14	422.00	27.27	78.11	1224.58

2214 rows × 4 columns

In [259...]:
```
close_px = close_px_all[['AAPL', 'MSFT', 'XOM']]
close_px
```

Out[259]:

	AAPL	MSFT	XOM
2003-02-01	7.40	21.11	29.22
2003-03-01	7.45	21.14	29.24
2003-06-01	7.45	21.52	29.96
2003-07-01	7.43	21.93	28.95
2003-08-01	7.28	21.31	28.83
...	...	...	...
2011-10-10	388.81	26.94	76.28
2011-11-10	400.29	27.00	76.27
2011-12-10	402.19	26.96	77.16
2011-10-13	408.43	27.18	76.37
2011-10-14	422.00	27.27	78.11

2214 rows × 3 columns

In [262...]:
```
close_px = close_px.resample('B').ffill()
```

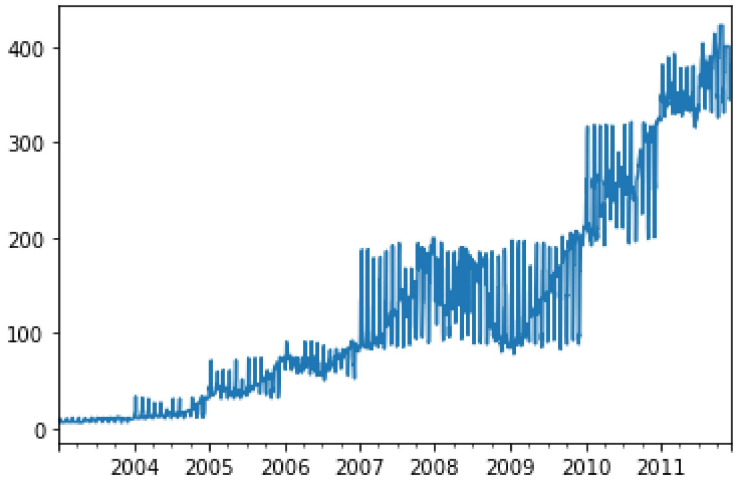
In [263...]:
```
close_px.head(20)
```

Out[263]:

	AAPL	MSFT	XOM
2003-01-03	NaN	NaN	NaN
2003-01-06	7.18	20.28	29.42
2003-01-07	9.55	20.62	30.22
2003-01-08	10.36	20.63	29.51
2003-01-09	10.36	20.63	29.51
2003-01-10	10.40	22.48	31.41
2003-01-13	7.32	22.16	28.91
2003-01-14	7.30	22.39	29.17
2003-01-15	7.22	22.11	28.77
2003-01-16	7.31	21.75	28.90
2003-01-17	7.05	20.22	28.60
2003-01-20	7.05	20.22	28.60
2003-01-21	7.01	20.17	27.94
2003-01-22	6.94	20.04	27.58
2003-01-23	7.09	20.54	27.52
2003-01-24	6.90	19.59	26.93
2003-01-27	7.07	19.32	26.21
2003-01-28	7.29	19.18	26.90
2003-01-29	7.47	19.61	27.88
2003-01-30	7.16	18.95	27.37

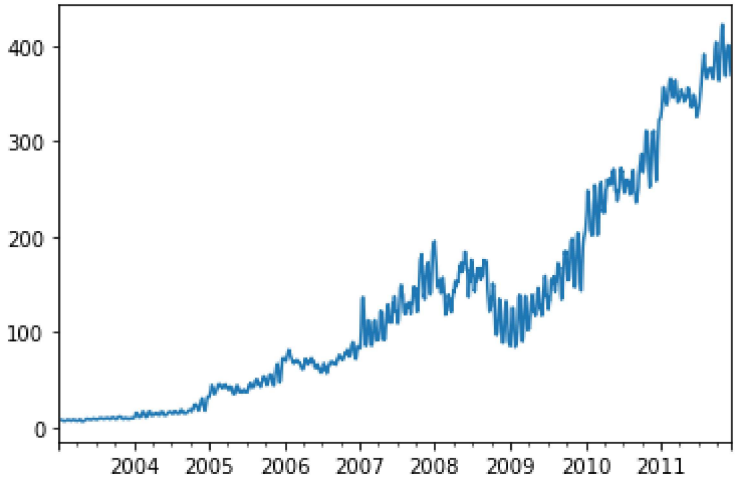
In [264…
```
close_px.AAPL.plot()
```

Out[264]:
```
<AxesSubplot:>
```



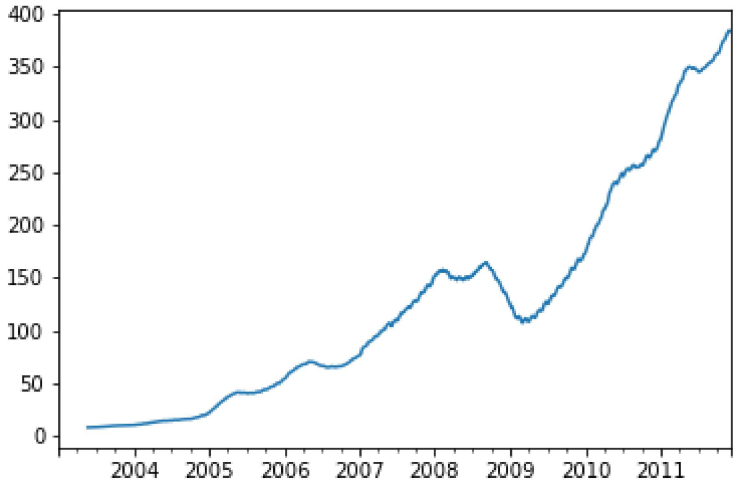
In [265…
```
close_px.AAPL.rolling(10).mean().plot()
```

Out[265]:
```
<AxesSubplot:>
```



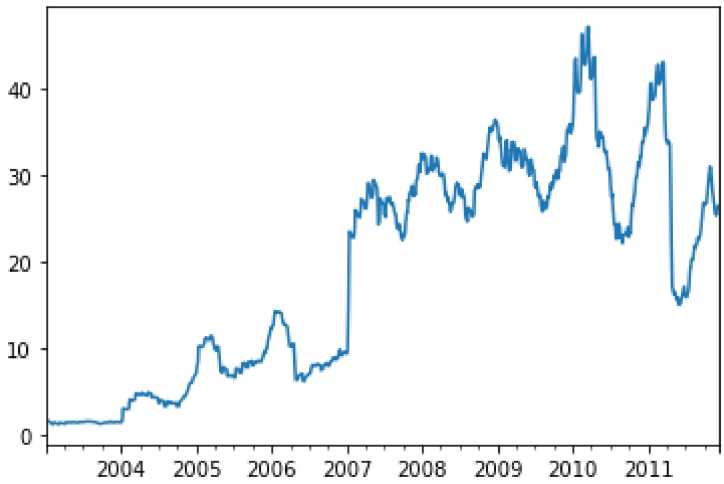
In [266…
```
close_px.AAPL.rolling(100).mean().plot()
```

Out[266]:
```
<AxesSubplot:>
```



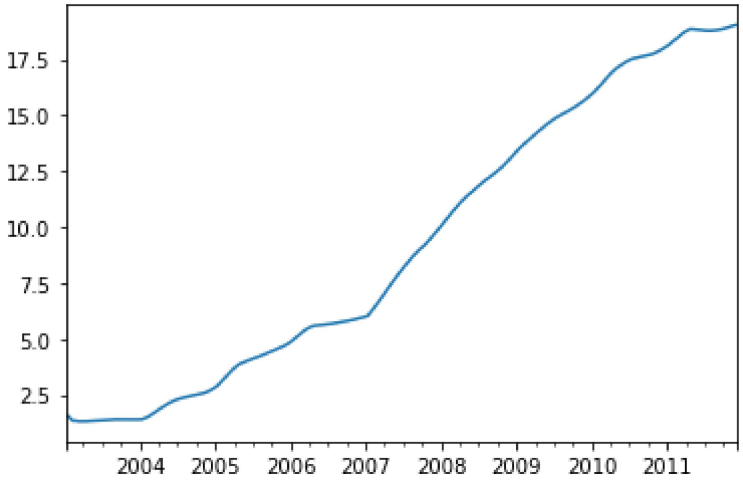
In [267…
```
appl_std100=close_px.AAPL.rolling(100, min_periods=10).std()
appl_std100.plot()
```

Out[267]:
```
<AxesSubplot:>
```



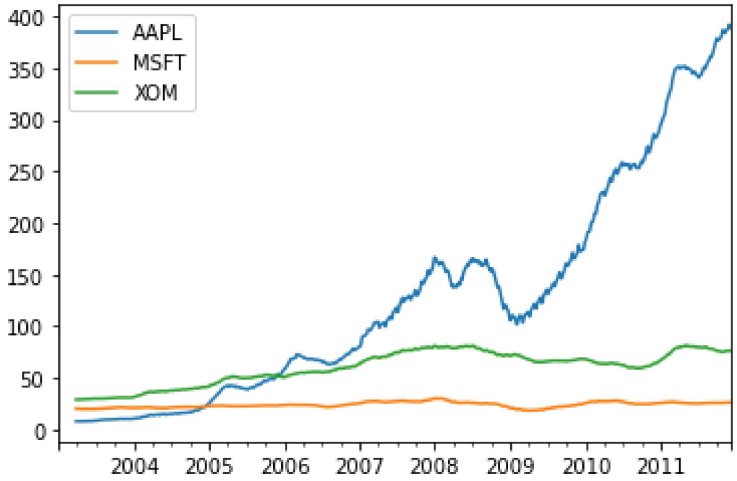
In [268…
```
expanding_mean100=appl_std100.expanding().mean()
expanding_mean100.plot()
```

Out[268]:
```
<AxesSubplot:>
```



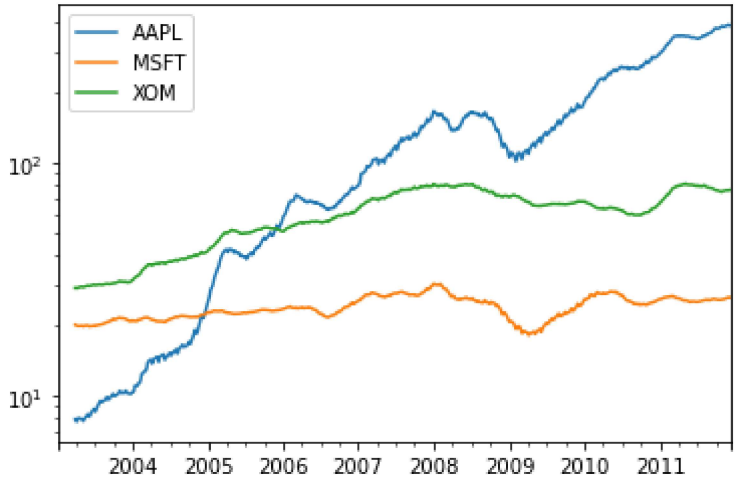
In [269…
```
close_px.rolling(60).mean().plot()
```

Out[269]:
```
<AxesSubplot:>
```



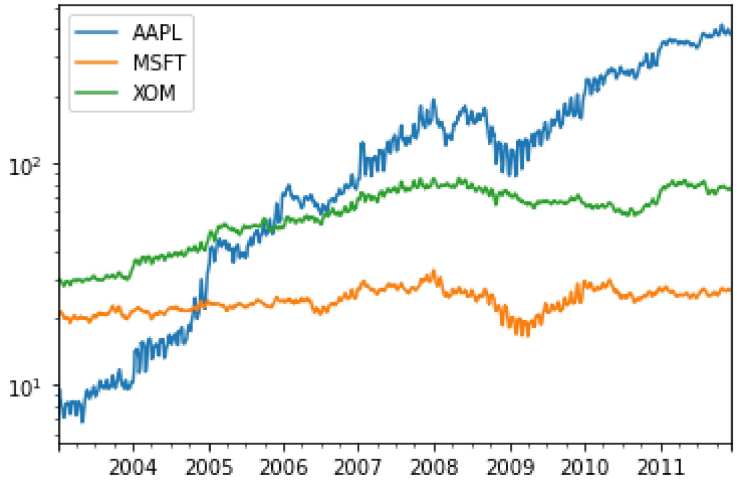
In [273…
```
close_px.rolling(60).mean().plot(logy=True)
```

Out[273]:
```
<AxesSubplot:>
```



In [271…
```
close_px.rolling('20D').mean().plot(logy=True)
```

Out[271]:
```
<AxesSubplot:>
```



In [ ]: