Time Series

how you mark and refer to time series data depends on the application, and you may have one of the following

- Timestamps, specific instants in time
- fixed periods, such as the month jan 2007 or the full year 2010
- intervals of time, indicated by a start and end timestamp. Periods can be thought of as special cases of intervals
- Experiment or elapsed time; each timestamp is a measure of time relative to a particular start time(e.g. the diameter of a cookie baking each seocond since being placed in the oven)

In [44]:
```
import pandas as pd
```

In [45]:
```
from datetime import datetime
now = datetime.now()
now
```

Out[45]:
```
datetime.datetime(2020, 8, 4, 15, 45, 15, 197133)
```

In [46]:
```
now.year, now.month, now.day
```

Out[46]:
```
(2020, 8, 4)
```

In [47]:
```
delta = datetime(2011, 1, 7) - datetime(2008, 6, 24, 8, 15)
delta
```

Out[47]:
```
datetime.timedelta(926, 56700)
```

In [48]:
```
from datetime import timedelta
start = datetime(2011, 1, 7)
start + 2 * timedelta(12)
```

Out[48]:
```
datetime.datetime(2011, 1, 31, 0, 0)
```

Converting Between String and Datetime

In [49]:
```
stamp = datetime(1998, 12, 28)
str(stamp)
```

Out[49]:
```
'1998-12-28 00:00:00'
```

In [50]:
```
stamp.strftime('%Y-%m-%d')
```

Out[50]:
```
'1998-12-28'
```

Datetime format specification

%Y -- four
%y -- Two-digit year
%m -- Two-digit month
%d -- Two-digit day
%H -- Hour (24 hour clock)
%I -- House (12 hour clock)
%M -- Two-digit minute
%S -- Second [00,61] 61 account for leap second
%w -- Weekday as integer [0(sunday), 6]
%U -- Week number of the year [0-53]; 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
%D -- Shortcut for %m/%d/%y

you can use the same methods to convert strings to dates using datetime.strptime

In [51]:
```
value = '2011-01-03'
datetime.strptime(value, '%Y-%m-%d')
```

Out[51]:
```
datetime.datetime(2011, 1, 3, 0, 0)
```

In [52]:
```
import numpy as np
dates = [datetime(2011, 1, 2), datetime(2011, 1, 5), datetime(2011, 1,
ts = pd.Series(np.random.randn(6), index=dates)
ts
```

Out[52]:
```
2011-01-02   -0.596279
2011-01-05   -0.815065
2011-01-07    1.743168
2011-01-08   -0.773380
2011-01-10   -0.306357
2011-01-12    1.100062
dtype: float64
```

In [53]:
```
ts.index
```

Out[53]:
```
DatetimeIndex(['2011-01-02', '2011-01-05', '2011-01-07', '2011-01-08
',
               '2011-01-10', '2011-01-12'],
              dtype='datetime64[ns]', freq=None)
```

In [54]:
```
ts + ts[::2] # selects every other point and doubles them.
```

Out[54]:
```
2011-01-02   -1.192558
2011-01-05         NaN
2011-01-07    3.486336
2011-01-08         NaN
2011-01-10   -0.612714
2011-01-12         NaN
dtype: float64
```

In [55]:
```
stamp = ts.index[0]
stamp
```

Out[55]:
```
Timestamp('2011-01-02 00:00:00')
```

In [56]:
```
ts[stamp]
```

Out[56]:
```
-0.5962792122800337
```

In [57]:
```
ts['2011/01/02'] # can pass a date as a string
```

Out[57]:
```
-0.5962792122800337
```

In [58]:
```
#for larger series sets
longer_ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/
longer_ts.tail()
```

Out[58]:
```
2002-09-22   -0.493384
2002-09-23   -0.207827
2002-09-24   -0.886046
2002-09-25    0.237732
2002-09-26    1.460096
Freq: D, dtype: float64
```

In [59]:
```
longer_ts['2001'].tail()
```

Out[59]:
```
2001-12-27   -0.744843
2001-12-28   -0.074248
2001-12-29   -0.944912
2001-12-30    1.131164
2001-12-31   -0.408839
Freq: D, dtype: float64
```

In [60]:
```
longer_ts['2002-05'].tail()
```

Out[60]:
```
2002-05-27    0.718844
2002-05-28    1.360635
2002-05-29   -2.496779
2002-05-30    0.412370
2002-05-31    1.176583
Freq: D, dtype: float64
```

In [61]:
```
ts[datetime(2009, 5, 5):]
```

Out[61]:
```
2011-01-02   -0.596279
2011-01-05   -0.815065
2011-01-07    1.743168
2011-01-08   -0.773380
2011-01-10   -0.306357
2011-01-12    1.100062
dtype: float64
```

In [62]:
```
ts['1/6/2011':'1/11/2011']
```

Out[62]:
```
2011-01-07    1.743168
2011-01-08   -0.773380
2011-01-10   -0.306357
dtype: float64
```

In [63]:
```
ts.truncate(after='1/9/2011')
```

Out[63]:
```
2011-01-02   -0.596279
2011-01-05   -0.815065
2011-01-07    1.743168
2011-01-08   -0.773380
dtype: float64
```

Time Series with Duplicate Indices

In [64]:
```
dates = pd.DatetimeIndex(['1/1/2002', '1/2/2002', '1/3/2002', '1/2/200
dup_ts = pd.Series(np.random.randn(5), index=dates)
dup_ts
```

Out[64]:
```
2002-01-01   -1.410824
2002-01-02    0.881724
2002-01-03    1.002362
2002-01-02    0.173960
2002-01-02    0.180727
dtype: float64
```

In [65]:
```
dup_ts.index.unique()
```

Out[65]:
```
DatetimeIndex(['2002-01-01', '2002-01-02', '2002-01-03'], dtype='dat
etime64[ns]', freq=None)
```

In [66]:
```
# suppose you wanted to group non-unique dates
non_unique = dup_ts.groupby(level=0)
non_unique.mean()
```

Out[66]:
```
2002-01-01   -1.410824
2002-01-02    0.412137
2002-01-03    1.002362
dtype: float64
```

In [67]:
```
non_unique.count()
```

Out[67]:
```
2002-01-01    1
2002-01-02    3
2002-01-03    1
dtype: int64
```

Date Ranges Frequencies and shifting

often you time series may have non-uniform date ranges. For most purposes this is fine but should you whish to have equidistant dates you may whish to use the following

In [68]:
```
ts
```

Out[68]:
```
2011-01-02   -0.596279
2011-01-05   -0.815065
2011-01-07    1.743168
2011-01-08   -0.773380
2011-01-10   -0.306357
2011-01-12    1.100062
dtype: float64
```

In [69]:
```
resampler = ts.resample('D') #D for daily
resampler.mean()
```

Out[69]:
```
2011-01-02   -0.596279
2011-01-03         NaN
2011-01-04         NaN
2011-01-05   -0.815065
2011-01-06         NaN
2011-01-07    1.743168
2011-01-08   -0.773380
2011-01-09         NaN
2011-01-10   -0.306357
2011-01-11         NaN
2011-01-12    1.100062
Freq: D, dtype: float64
```

In [70]:
```
index = pd.date_range('2009-04-01', '2009-05-01')
index
```

Out[70]:
```
DatetimeIndex(['2009-04-01', '2009-04-02', '2009-04-03', '2009-04-04
',
               '2009-04-05', '2009-04-06', '2009-04-07', '2009-04-08
',
               '2009-04-09', '2009-04-10', '2009-04-11', '2009-04-12
',
               '2009-04-13', '2009-04-14', '2009-04-15', '2009-04-16
',
               '2009-04-17', '2009-04-18', '2009-04-19', '2009-04-20
',
               '2009-04-21', '2009-04-22', '2009-04-23', '2009-04-24
',
               '2009-04-25', '2009-04-26', '2009-04-27', '2009-04-28
',
               '2009-04-29', '2009-04-30', '2009-05-01'],
              dtype='datetime64[ns]', freq='D')
```

In [71]:
```
index2 = pd.date_range(start='1998-12-28', periods=365) #can also usec
# should you have wanted a index that just referenced the last busines
index2 = pd.date_range(start='1998-12-28', end='2020-12-28', freq='BM'
```

Base time series frequencies

D -- Day
B -- BusinessDay
H -- Hour
T/min -- Minute
S -- Second
L/ms -- milli
U -- Micro
M -- MonthEnd
BM -- BusinessMonthEnd
MS -- MonthBegin
BMS -- BusinessMonthBegin
W-MON, W-TUE -- Week
WOM-1MON/WOM-2MON -- Generates dates in first/second/third/fourth week of the month.
Q-JAN/Q-FEB -- Quater end anchord on last calendar name of each month
BQ-JAN/BQ-FEB -- Business Quater End. Quaterly dates anchored on last weekday of each month
QS-JAN/QS-FEB -- Quaterly dates anchored on first calendar day of each month.
BQS-JAN/BQS-FEB -- Quarterly dates anchored onf irst weekday of each mont, for year ending in indicated month.
A-JAN/A-FEB -- Annual dates anchored on last calendar day of given month
BA-JAN/BA-FEB -- Annual dates anchored on last weekday of given month
AS-JAN/AS-FEB -- Annual dates anchored on first daty of given month
BAS-JAN/BAS-FEB -- Annual dates anchored on frist weekday of given month.

In [72]:
```
pd.date_range('2001-11-23 12:56:31', periods=5)
```

Out[72]:
```
DatetimeIndex(['2001-11-23 12:56:31', '2001-11-24 12:56:31',
               '2001-11-25 12:56:31', '2001-11-26 12:56:31',
               '2001-11-27 12:56:31'],
              dtype='datetime64[ns]', freq='D')
```

In [73]:
```
pd.date_range('2001-11-23 12:56:31', periods=5, normalize=True) #shoul
```

Out[73]:
```
DatetimeIndex(['2001-11-23', '2001-11-24', '2001-11-25', '2001-11-26
',
               '2001-11-27'],
              dtype='datetime64[ns]', freq='D')
```

Shifting Data

moving information around by shifting date

In [74]:
```
dates = pd.date_range('1998-12-05', periods = 6)
ts = pd.Series(np.random.randn(6), index=dates)
ts
```

Out[74]:
```
1998-12-05    0.484211
1998-12-06   -1.271930
1998-12-07    0.467360
1998-12-08    0.952550
1998-12-09    0.371530
1998-12-10   -1.454242
Freq: D, dtype: float64
```

In [75]:
```
ts.shift(3)
```

Out[75]:
```
1998-12-05         NaN
1998-12-06         NaN
1998-12-07         NaN
1998-12-08    0.484211
1998-12-09   -1.271930
1998-12-10    0.467360
Freq: D, dtype: float64
```

In [76]:
```
ts.shift(-2)
```

Out[76]:
```
1998-12-05    0.467360
1998-12-06    0.952550
1998-12-07    0.371530
1998-12-08   -1.454242
1998-12-09         NaN
1998-12-10         NaN
Freq: D, dtype: float64
```

In [77]:
```
ts.shift(1, freq='90S')
```

Out[77]:
```
1998-12-05 00:01:30    0.484211
1998-12-06 00:01:30   -1.271930
1998-12-07 00:01:30    0.467360
1998-12-08 00:01:30    0.952550
1998-12-09 00:01:30    0.371530
1998-12-10 00:01:30   -1.454242
Freq: D, dtype: float64
```

Time Zone Handling

working with time zones is considered to be a pain. As a result many time series used choose to with with UTC, the succsesor to GMT. IT is the current international standard. Time zones are expressed as offsets from UTC. In python, time zone information comes from pytz lirary. You can install it with pip or conda. pandas wraps pytz's functionalty so you can ignore its API outside of the time zone names. Time zone names can be found interactively and in the docs.

In [78]:
```
import pytz
pytz.common_timezones[-5:]
```

Out[78]:
```
['US/Eastern', 'US/Hawaii', 'US/Mountain', 'US/Pacific', 'UTC']
```

In [79]:
```
# To get a time zone object from pytz, use pytz.timezone
tz = pytz.timezone('America/New_York')
tz
```

Out[79]:
```
<DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>
```

Time Zone Localization and Conversion

By default, Time series in pandas are time zone naive. For example, consider the following time series

In [80]:
```
rng = pd.date_range('3/9/2012 9:20', periods=7, freq='D')
rng
```

Out[80]:
```
DatetimeIndex(['2012-03-09 09:20:00', '2012-03-10 09:20:00',
               '2012-03-11 09:20:00', '2012-03-12 09:20:00',
               '2012-03-13 09:20:00', '2012-03-14 09:20:00',
               '2012-03-15 09:20:00'],
              dtype='datetime64[ns]', freq='D')
```

In [81]:
```
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ts
```

Out[81]:
```
2012-03-09 09:20:00   -2.070686
2012-03-10 09:20:00    1.916341
2012-03-11 09:20:00   -0.393583
2012-03-12 09:20:00    0.893082
2012-03-13 09:20:00    1.851536
2012-03-14 09:20:00   -0.172357
2012-03-15 09:20:00   -0.370833
Freq: D, dtype: float64
```

In [82]:
```
print(ts.index.tz) # Shows the series does not have an associated time
```

```
None
```

In [83]:
```
pd.date_range('3/9/2012 9:30', periods=10, freq='D', tz='UTC') #can pa
```

Out[83]:
```
DatetimeIndex(['2012-03-09 09:30:00+00:00', '2012-03-10 09:30:00+00:
00',
               '2012-03-11 09:30:00+00:00', '2012-03-12 09:30:00+00:
00',
               '2012-03-13 09:30:00+00:00', '2012-03-14 09:30:00+00:
00',
               '2012-03-15 09:30:00+00:00', '2012-03-16 09:30:00+00:
00',
               '2012-03-17 09:30:00+00:00', '2012-03-18 09:30:00+00:
00'],
              dtype='datetime64[ns, UTC]', freq='D')
```

In [84]:
```
ts_utc = ts.tz_localize('UTC') #localises time series to UTC
ts_utc
```

Out[84]:
```
2012-03-09 09:20:00+00:00   -2.070686
2012-03-10 09:20:00+00:00    1.916341
2012-03-11 09:20:00+00:00   -0.393583
2012-03-12 09:20:00+00:00    0.893082
2012-03-13 09:20:00+00:00    1.851536
2012-03-14 09:20:00+00:00   -0.172357
2012-03-15 09:20:00+00:00   -0.370833
Freq: D, dtype: float64
```

In [85]:
```
ts_utc.index
```

Out[85]:
```
DatetimeIndex(['2012-03-09 09:20:00+00:00', '2012-03-10 09:20:00+00:
00',
               '2012-03-11 09:20:00+00:00', '2012-03-12 09:20:00+00:
00',
               '2012-03-13 09:20:00+00:00', '2012-03-14 09:20:00+00:
00',
               '2012-03-15 09:20:00+00:00'],
              dtype='datetime64[ns, UTC]', freq='D')
```

In [86]:
```
ts_utc.tz_convert('America/New_York') # converts it to another time zo
```

Out[86]:
```
2012-03-09 04:20:00-05:00   -2.070686
2012-03-10 04:20:00-05:00    1.916341
2012-03-11 05:20:00-04:00   -0.393583
2012-03-12 05:20:00-04:00    0.893082
2012-03-13 05:20:00-04:00    1.851536
2012-03-14 05:20:00-04:00   -0.172357
2012-03-15 05:20:00-04:00   -0.370833
Freq: D, dtype: float64
```

In [87]:
```
ts_eastern = ts.tz_localize('America/New_York') #localise to America/N
ts_eastern.tz_convert('UTC') #Converts to UTC
```

Out[87]:
```
2012-03-09 14:20:00+00:00   -2.070686
2012-03-10 14:20:00+00:00    1.916341
2012-03-11 13:20:00+00:00   -0.393583
2012-03-12 13:20:00+00:00    0.893082
2012-03-13 13:20:00+00:00    1.851536
2012-03-14 13:20:00+00:00   -0.172357
2012-03-15 13:20:00+00:00   -0.370833
Freq: D, dtype: float64
```

In [88]:
```
ts_eastern.tz_convert('Europe/Berlin')
```

Out[88]:
```
2012-03-09 15:20:00+01:00   -2.070686
2012-03-10 15:20:00+01:00    1.916341
2012-03-11 14:20:00+01:00   -0.393583
2012-03-12 14:20:00+01:00    0.893082
2012-03-13 14:20:00+01:00    1.851536
2012-03-14 14:20:00+01:00   -0.172357
2012-03-15 14:20:00+01:00   -0.370833
Freq: D, dtype: float64
```

Operations Between Different Time Zones

If two Series with different time zones are combined, the result will be UTC. Since the timestamps are stored under the hood in UTC.

In [89]:
```
rng = pd.date_range('3/7/2012 9:30', periods=10, freq='B')
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ts
```

Out[89]:
```
2012-03-07 09:30:00    0.605051
2012-03-08 09:30:00    0.481053
2012-03-09 09:30:00   -0.330596
2012-03-12 09:30:00    0.510970
2012-03-13 09:30:00   -0.340682
2012-03-14 09:30:00   -0.271859
2012-03-15 09:30:00    1.490435
2012-03-16 09:30:00   -1.887412
2012-03-19 09:30:00   -0.395401
2012-03-20 09:30:00    0.802360
Freq: B, dtype: float64
```

In [90]:
```
ts1 = ts[:7].tz_localize('Europe/London')
ts2 = ts1[2:].tz_convert('Europe/Moscow')
result = ts1 + ts2
result.index # Resultant time zone is UTC
```

Out[90]:
```
DatetimeIndex(['2012-03-07 09:30:00+00:00', '2012-03-08 09:30:00+00:
00',
               '2012-03-09 09:30:00+00:00', '2012-03-12 09:30:00+00:
00',
               '2012-03-13 09:30:00+00:00', '2012-03-14 09:30:00+00:
00',
               '2012-03-15 09:30:00+00:00'],
              dtype='datetime64[ns, UTC]', freq='B')
```

Periods and Period Arithmetic

Periods represent timespanes, like days, months, quaters or years. The period is represented by the Period class

In [91]:
```
p = pd.Period(2007, freq='A-Dec')
p
```

Out[91]:
```
Period('2007', 'A-DEC')
```

In [92]:
```
# addidng and subtracting shift the periods date
p + 5
```

Out[92]:
```
Period('2012', 'A-DEC')
```

In [93]:
```
# regular ranges or periods can be constructed with period_range
rng = pd.period_range('2000-01-01', '2000-06-30', freq='M')
rng
```

Out[93]:
```
PeriodIndex(['2000-01', '2000-02', '2000-03', '2000-04', '2000-05',
'2000-06'], dtype='period[M]', freq='M')
```

In [94]:
```
pd.Series(np.random.randn(len(rng)), index=rng) #can use as an axis ir
```

Out[94]:
```
2000-01    0.651785
2000-02    0.750168
2000-03   -0.866046
2000-04    0.470224
2000-05   -0.949654
2000-06   -1.326554
Freq: M, dtype: float64
```

period Freqency Conversion

for example shoudl you have a period index of a year and whish to convert it into months

In [95]:
```
p = pd.Period('2007', freq='A-DEC')
p
```

Out[95]:
```
Period('2007', 'A-DEC')
```

In [96]:
```
p.asfreq('M', how='start')
```

Out[96]:
```
Period('2007-01', 'M')
```

In [97]:
```
p.asfreq('M', how='end')
```

Out[97]:
```
Period('2007-12', 'M')
```

PeriodIndex objects or time series can be similarly converted with the same sematincs

In [98]:
```
rng = pd.period_range('2006','2012', freq='A-DEC')
p = pd.Series(np.random.randn(len(rng)), index=rng)
p
```

Out[98]:
```
2006   -1.061012
2007    0.163543
2008    0.666401
2009    2.197572
2010    1.363657
2011   -0.417862
2012   -0.852959
Freq: A-DEC, dtype: float64
```

In [99]:
```
p.asfreq('B', how='end') # last business day of each year
```

Out[99]:
```
2006-12-29   -1.061012
2007-12-31    0.163543
2008-12-31    0.666401
2009-12-31    2.197572
2010-12-31    1.363657
2011-12-30   -0.417862
2012-12-31   -0.852959
Freq: B, dtype: float64
```

Converting Timestamps to Peridos (and Back)

Series and DataFrame objects indexed by timestamps can be converted to periods with the to_period method

In [100]:
```
rng = pd.date_range('2000-01-01', periods=3, freq='M')
ts = pd.Series(np.random.randn(3), index=rng)
ts
```

Out[100]:
```
2000-01-31    1.408165
2000-02-29    0.693272
2000-03-31    0.189881
Freq: M, dtype: float64
```

In [101]:
```
pts = ts.to_period()
pts
```

Out[101]:
```
2000-01    1.408165
2000-02    0.693272
2000-03    0.189881
Freq: M, dtype: float64
```

Since periods refer to non-overlapping timespans, a timestamp can only belong to a single period. There however is no issue with having multiple numbers of the same period.

In [102]:
```
rng = pd.date_range('1/29/2000', periods=6, freq='D')
ts2 = pd.Series(np.random.randn(6), index=rng)
ts2
```

Out[102]:
```
2000-01-29   -0.695630
2000-01-30   -1.227850
2000-01-31    0.655502
2000-02-01   -0.434818
2000-02-02    0.157314
2000-02-03    0.174238
Freq: D, dtype: float64
```

In [103]:
```
pts2 = ts2.to_period('M')
pts2
```

Out[103]:
```
2000-01   -0.695630
2000-01   -1.227850
2000-01    0.655502
2000-02   -0.434818
2000-02    0.157314
2000-02    0.174238
Freq: M, dtype: float64
```

In [104]:
```
# To convert back to timestamp use to_timestamp()
```

In [105]:
```
ts3 = pts2.to_timestamp(how='end')
ts3
```

Out[105]:
```
2000-01-31 23:59:59.999999999   -0.695630
2000-01-31 23:59:59.999999999   -1.227850
2000-01-31 23:59:59.999999999    0.655502
2000-02-29 23:59:59.999999999   -0.434818
2000-02-29 23:59:59.999999999    0.157314
2000-02-29 23:59:59.999999999    0.174238
dtype: float64
```

Resampling and Frequency conversion

resampling referes to the process of converting a time series from one frequncy to antoher. downsampling, converts to lower frequency where upsampling refers to higher frequency. the resample method is the main workhorse for all frequency conversion.

In [106]:
```
rng = pd.date_range('2000-01-01', periods=100, freq='D')
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ts.head(8)
```

Out[106]:
```
2000-01-01   -1.066628
2000-01-02   -1.394048
2000-01-03    0.161837
2000-01-04   -1.373027
2000-01-05    0.055547
2000-01-06   -1.287104
2000-01-07   -0.352793
2000-01-08    1.503156
Freq: D, dtype: float64
```

In [107]:
```
ts.resample('M').mean()
```

Out[107]:
```
2000-01-31   -0.400625
2000-02-29   -0.156533
2000-03-31    0.042490
2000-04-30    0.069311
Freq: M, dtype: float64
```

In [108]:
```
ts.resample('M', kind='period').mean()
```

Out[108]:
```
2000-01   -0.400625
2000-02   -0.156533
2000-03    0.042490
2000-04    0.069311
Freq: M, dtype: float64
```

Resample method arguments

freq -- String or DateOffset indicating desired resampled frequency axis -- Axis to resample on; default=0
fill_method -- How to interpolate when upsampling as in 'ffill', or 'bfill';
closed -- In downsampling, which ened of each interval is closed 'right' or 'left'.
label -- In downsampling, how to lavel the aggregated result with the 'right' or 'left' bin edge
limit -- When foward or backward filling, the maximum number of periods to fill.
kind -- Aggregate to periods or timestamps 'period' or 'timestamp'
convention -- When resampling periods, the convention('start' or'end' for converting the low-frequncy period to high frequency'; defaults to 'start'.

Downsampling

when downsampling you need to thingk about

- Which side of each interval is closed
- How to label each aggregated bin, either with the start of the interval or the end.

In [110]:
```
rng = pd.date_range('2000-01-01', periods=12, freq='T')
ts = pd.Series(np.arange(len(rng)), index=rng)
ts
```

Out[110]:
```
2000-01-01 00:00:00     0
2000-01-01 00:01:00     1
2000-01-01 00:02:00     2
2000-01-01 00:03:00     3
2000-01-01 00:04:00     4
2000-01-01 00:05:00     5
2000-01-01 00:06:00     6
2000-01-01 00:07:00     7
2000-01-01 00:08:00     8
2000-01-01 00:09:00     9
2000-01-01 00:10:00    10
2000-01-01 00:11:00    11
Freq: T, dtype: int64
```

In [111]:
```
ts.resample('5min', closed='right').sum()
```

Out[111]:
```
1999-12-31 23:55:00     0
2000-01-01 00:00:00    15
2000-01-01 00:05:00    40
2000-01-01 00:10:00    11
Freq: 5T, dtype: int64
```

In [113]:
```
ts.resample('5min', closed='right', label='right').sum() # changes the
```

Out[113]:
```
2000-01-01 00:00:00     0
2000-01-01 00:05:00    15
2000-01-01 00:10:00    40
2000-01-01 00:15:00    11
Freq: 5T, dtype: int64
```

Open-High-Low-Close resampling

in finance a popular way to aggregate a time seires is to compute four values for each bucket. By using ohlc aggregate function you will obtain a DataFrame having columns containing these four aggregates

In [114]:
```
ts.resample('5min').ohlc()
```

Out[114]:

	open	high	low	close
2000-01-01 00:00:00	0	4	0	4
2000-01-01 00:05:00	5	9	5	9
2000-01-01 00:10:00	10	11	10	11

Upsampling and Interpolation

when converting from a low frequency to a higher frequency no aggretation is needed.

In [118]:
```
frame = pd.DataFrame(np.random.randn(2,4), index=pd.date_range('1/1/20
frame
```

Out[118]:

	Colorado	Texas	New York	Ohio
2000-01-05	1.312761	-0.186424	1.127892	-0.630408
2000-01-12	-0.880612	-0.578471	0.799949	1.848771

In [120]:
```
df_daily = frame.resample('D').asfreq()
df_daily
```

Out[120]:

	Colorado	Texas	New York	Ohio
2000-01-05	1.312761	-0.186424	1.127892	-0.630408
2000-01-06	NaN	NaN	NaN	NaN
2000-01-07	NaN	NaN	NaN	NaN
2000-01-08	NaN	NaN	NaN	NaN
2000-01-09	NaN	NaN	NaN	NaN
2000-01-10	NaN	NaN	NaN	NaN
2000-01-11	NaN	NaN	NaN	NaN
2000-01-12	-0.880612	-0.578471	0.799949	1.848771

In [123]:
```
frame.resample('D').ffill() # fill carries the data foward and resampl
# as ffill reduces the time frequency and therefore rows
```

Out[123]:

	Colorado	Texas	New York	Ohio
2000-01-05	1.312761	-0.186424	1.127892	-0.630408
2000-01-06	1.312761	-0.186424	1.127892	-0.630408
2000-01-07	1.312761	-0.186424	1.127892	-0.630408
2000-01-08	1.312761	-0.186424	1.127892	-0.630408
2000-01-09	1.312761	-0.186424	1.127892	-0.630408
2000-01-10	1.312761	-0.186424	1.127892	-0.630408
2000-01-11	1.312761	-0.186424	1.127892	-0.630408
2000-01-12	-0.880612	-0.578471	0.799949	1.848771

Resampling with Periods

Resampling data indexed by periods is similar to timestamps

In [125]:
```
frame = pd.DataFrame(np.random.randn(24, 4), index=pd.period_range('1-
frame.head(3)
```

Out[125]:

	Colorado	Texas	New York	Ohio
2000-01	-0.952643	-0.856533	0.024391	0.108535
2000-02	0.446466	-0.460336	-2.619437	-0.252802
2000-03	0.735194	-0.892583	1.422746	0.216618

In [126]:
```
annual_frame = frame.resample('A-Dec').mean()
annual_frame
```

Out[126]:

	Colorado	Texas	New York	Ohio
2000	0.164265	-0.109949	0.040561	-0.008732
2001	0.472907	0.223241	0.018972	-0.343108

In [127]:
```
# Upsampling is more nuanced, as you must make a decision about which
# to place the values before resampling
# Q-DEC: quaterly, year ending in December
annual_frame.resample('Q-DEC').ffill()
```

Out[127]:

	Colorado	Texas	New York	Ohio
2000Q1	0.164265	-0.109949	0.040561	-0.008732
2000Q2	0.164265	-0.109949	0.040561	-0.008732
2000Q3	0.164265	-0.109949	0.040561	-0.008732
2000Q4	0.164265	-0.109949	0.040561	-0.008732
2001Q1	0.472907	0.223241	0.018972	-0.343108
2001Q2	0.472907	0.223241	0.018972	-0.343108
2001Q3	0.472907	0.223241	0.018972	-0.343108
2001Q4	0.472907	0.223241	0.018972	-0.343108

In [129]:
```
annual_frame.resample('Q-DEC', convention='end').ffill()
```

Out[129]:

	Colorado	Texas	New York	Ohio
2000Q4	0.164265	-0.109949	0.040561	-0.008732
2001Q1	0.164265	-0.109949	0.040561	-0.008732
2001Q2	0.164265	-0.109949	0.040561	-0.008732
2001Q3	0.164265	-0.109949	0.040561	-0.008732
2001Q4	0.472907	0.223241	0.018972	-0.343108

In downsampling, the target frequency must be a subperiod of the source frequency.

In upsampling, the target frequency must be a superperiod fo the source frequency.

In [ ]:
```

```