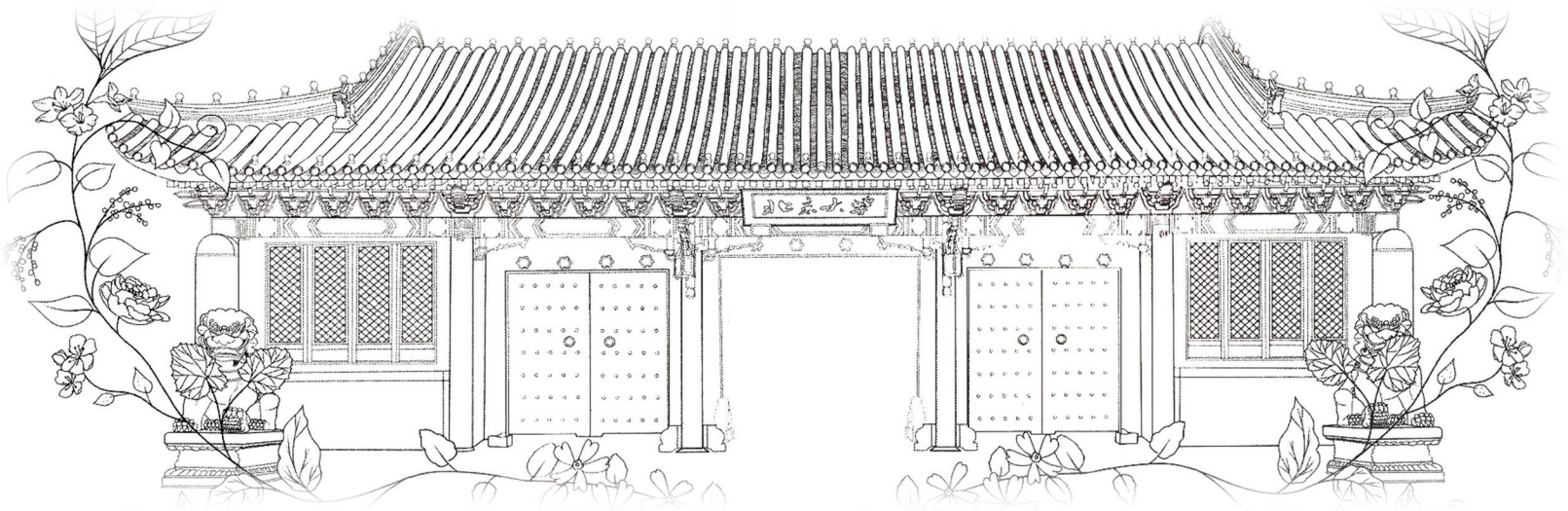
《Python数据分析》
应用篇：时间序列分析基础



三种常见数据

- 时间序列数据（Time series data），是一批按照时间先后顺序排列的统计数据
- 截面数据（Cross-section data），是一批发生在同一时间截面上的数据
  - 工业普查数据、人口普查数据、家庭调查数据等
- 面板数据（Panel data），是时间序列数据与截面数据的合成体。
  - 1978-1999年我国各省市城镇居民消费结构的调查资料

三种常见数据

时间序列数据
截面数据
面板数据

表 1　华东地区各省市 GDP 历史数据　　单位：亿元

	1995	1996	1997	1998	1999
上海	2462.57	2902.20	3360.21	3688.20	4034.96
江苏	5155.25	6004.21	6680.34	7199.95	7697.82
浙江	3524.79	4146.06	4638.24	4987.50	5364.89
安徽	2003.66	2339.25	2669.95	2805.45	2908.59
福建	2191.27	2583.83	3000.36	3286.56	3550.24
江西	1244.04	1517.26	1715.18	1851.98	1962.98
山东	4996.87	5960.42	6650.02	7162.20	7662.10

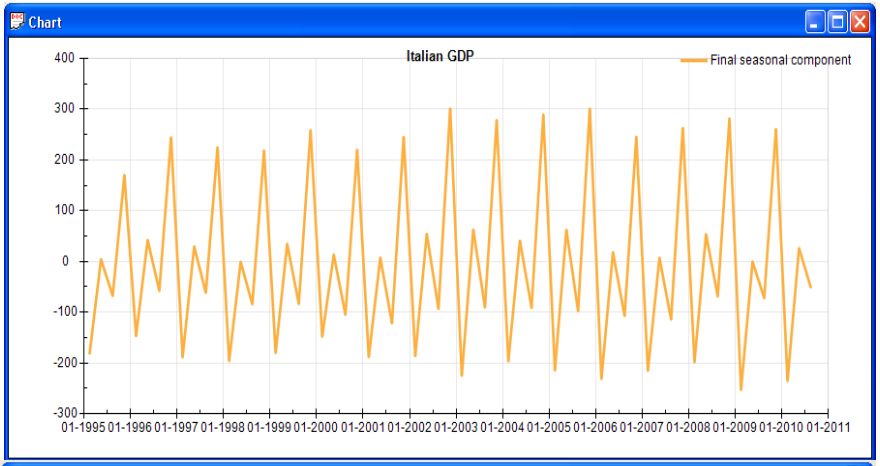
数据来源：中国统计年鉴 1996-2000。

时间序列数据

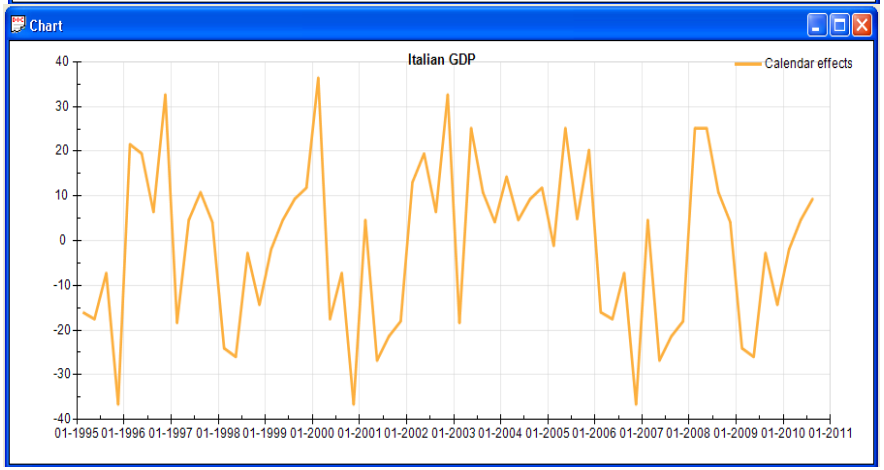
- 形式上由现象所属的时间和现象在不同时间上的观察值两部分组成。
- 时间序列的时间是变化的。常用的时间间隔有：年、季度、月、周、日……
- 时间序列数据有时存在季节变动和序列相关——自相关。

时间序列

- 季度性时间序列



- 日期性时间序列



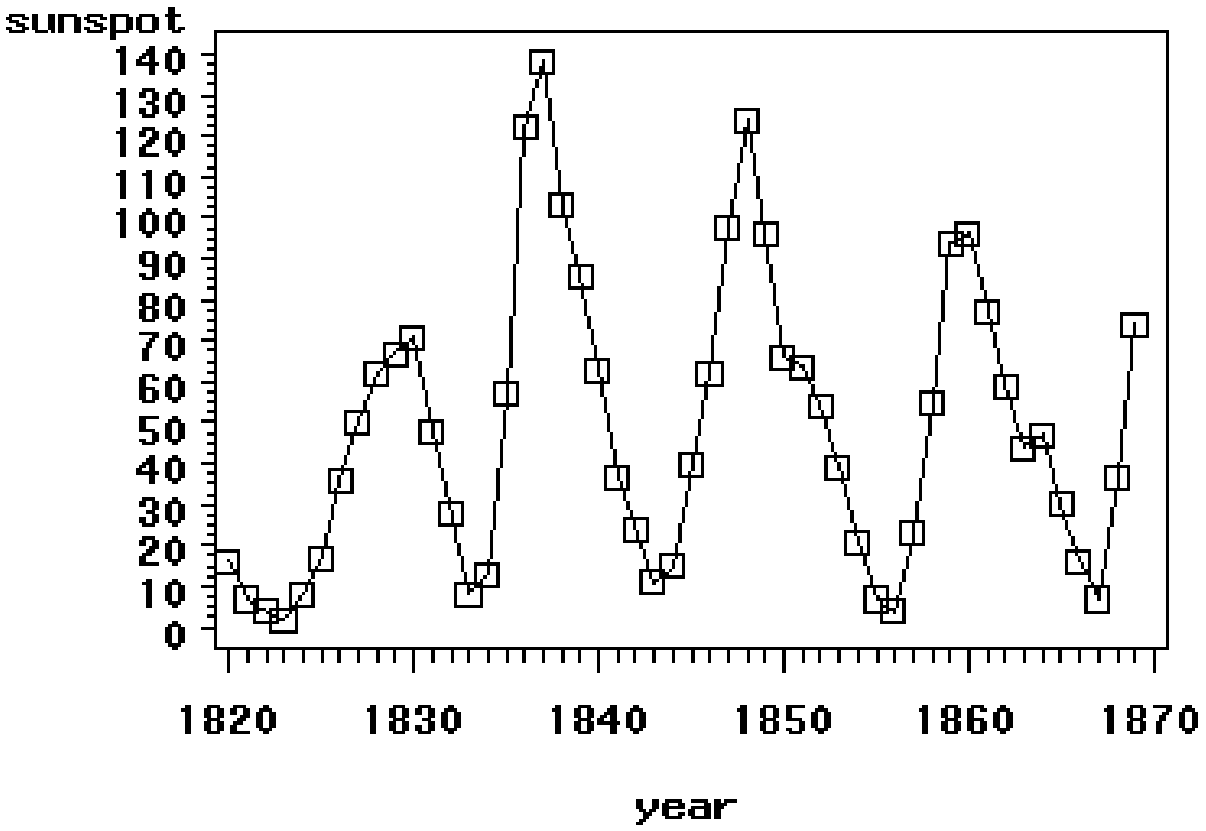
时间序列

- 一个时间序列是依时间顺序生成的观测值的集合
  - 若该集合是连续的 -> 连续型时间序列
  - 若该集合是离散的 -> 离散型时间序列
- 设y_t是时间序列在时刻（或时期）t 的观测值，当在 t = 1，2，3，4，5，6，7，……，n 采样时，得到时间序列：

$$y_1，\ y_2，\ y_3，\ y_4，\ y_5，\ y_6，\ \cdots\cdots，\ y_n$$

时间序列

- 德国业余天文学家施瓦尔发现太阳黑子的活动具有11年左右的周期



目录

- Python中时间和日期的处理
- 时间序列基础
- 日期的范围、频率和转化
- 时间区间和区间算术
- 时间序列的重采样 (resampling)
- 移动窗口函数 (moving window functions)

Python中时间和日期的处理

常用方法和函数?

```
In [10]: from datetime import datetime

In [11]: now = datetime.now()

In [12]: now
Out[12]: datetime.datetime(2017, 9, 25, 14, 5, 52, 72973)

In [13]: now.year, now.month, now.day
Out[13]: (2017, 9, 25)
```

1秒=1000000 微秒(μs)

常用方法和函数

```
In [14]: delta = datetime(2011, 1, 7) - datetime(2008, 6, 24, 8, 15)

In [15]: delta
Out[15]: datetime.timedelta(926, 56700)

In [16]: delta.days
Out[16]: 926

In [17]: delta.seconds
Out[17]: 56700
```

```
In [18]: from datetime import timedelta

In [19]: start = datetime(2011, 1, 7)

In [20]: start + timedelta(12)
Out[20]: datetime.datetime(2011, 1, 19, 0, 0)

In [21]: start - 2 * timedelta(12)
Out[21]: datetime.datetime(2010, 12, 14, 0, 0)
```

now
datetime.datetime(2018, 1, 20, 11, 14, 45, 545000)
now-datetime.timedelta(hours=3,minutes=30)
datetime.datetime(2018, 1, 20, 7, 44, 45, 545000)

常见的几种数据类型

公历

Type	Description
`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

字符串和日期之间的转化

```
In [22]: stamp = datetime(2011, 1, 3)

In [23]: str(stamp)
Out[23]: '2011-01-03 00:00:00'

In [24]: stamp.strftime('%Y-%m-%d')
Out[24]: '2011-01-03'
```

Type	Description
%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)

60、61算作闰秒

字符串和日期之间的转化

```
In [25]: value = '2011-01-03'

In [26]: datetime.strptime(value, '%Y-%m-%d')
Out[26]: datetime.datetime(2011, 1, 3, 0, 0)

In [27]: datestrs = ['7/6/2011', '8/6/2011']

In [28]: [datetime.strptime(x, '%m/%d/%Y') for x in datestrs]
Out[28]:
[datetime.datetime(2011, 7, 6, 0, 0),
 datetime.datetime(2011, 8, 6, 0, 0)]
```

字符串和日期之间的转化

- Datetime.strptime在已知格式时转换日期很好用，但每次都要编写格式代码可能很不方便
- 使用dateutil包的parser.parse方法

```
In [29]: from dateutil.parser import parse

In [30]: parse('2011-01-03')
Out[30]: datetime.datetime(2011, 1, 3, 0, 0)
```

`dateutil` is capable of parsing most human-intelligible date representations:

```
In [31]: parse('Jan 31, 1997 10:45 PM')
Out[31]: datetime.datetime(1997, 1, 31, 22, 45)
```

In international locales, day appearing before month is very common, so you can pass `dayfirst=True` to indicate this:

```
In [32]: parse('6/12/2011', dayfirst=True)
Out[32]: datetime.datetime(2011, 12, 6, 0, 0)
```

字符串和日期之间的转化：pandas

```
In [33]: datestrs = ['2011-07-06 12:00:00', '2011-08-06 00:00:00']

In [34]: pd.to_datetime(datestrs)
Out[34]: DatetimeIndex(['2011-07-06 12:00:00', '2011-08-06 00:00:00'], dtype='dat
etime64[ns]', freq=None)
```

It also handles values that should be considered missing (None, empty string, etc.):

```
In [35]: idx = pd.to_datetime(datestrs + [None])

In [36]: idx
Out[36]: DatetimeIndex(['2011-07-06 12:00:00', '2011-08-06 00:00:00', 'NaT'], dty
pe='datetime64[ns]', freq=None)

In [37]: idx[2]
Out[37]: NaT

In [38]: pd.isnull(idx)
Out[38]: array([False, False,  True], dtype=bool)
```

Not a time

时间序列基础

pandas中的时间序列

- Pandas中的基础时间序列种类是由**时间戳索引的Series**，在pandas外部则通常表示为**Python字符串**或**datetime对象**。

```
In [39]: from datetime import datetime

In [40]: dates = [datetime(2011, 1, 2), datetime(2011, 1, 5),
   ....:          datetime(2011, 1, 7), datetime(2011, 1, 8),
   ....:          datetime(2011, 1, 10), datetime(2011, 1, 12)]

In [41]: ts = pd.Series(np.random.randn(6), index=dates)

In [42]: ts
Out[42]:
2011-01-02   -0.204708
2011-01-05    0.478943
2011-01-07   -0.519439
2011-01-08   -0.555730
2011-01-10    1.965781
2011-01-12    1.393406
dtype: float64
```

```
In [43]: ts.index
Out[43]:
DatetimeIndex(['2011-01-02', '2011-01-05', '2011-01-07', '2011-01-08',
               '2011-01-10', '2011-01-12'],
              dtype='datetime64[ns]', freq=None)
```

pandas中的时间序列

```
In [42]: ts
Out[42]:
2011-01-02   -0.204708
2011-01-05    0.478943
2011-01-07   -0.519439
2011-01-08   -0.555730
2011-01-10    1.965781
2011-01-12    1.393406
dtype: float64
```

- 和其他Series类似，不同索引的时间序列之间的算术运算在日期上**自动对齐**。

```
In [44]: ts + ts[::2]
Out[44]:
2011-01-02   -0.409415
2011-01-05         NaN
2011-01-07   -1.038877
2011-01-08         NaN
2011-01-10    3.931561
2011-01-12         NaN
dtype: float64
```

- Pandas使用Numpy的datetime64数据类型在纳米级分辨率下存储的时间戳：

```
In [45]: ts.index.dtype
Out[45]: dtype('<M8[ns]')
```

pandas中的时间序列

```
In [42]: ts
Out[42]:
2011-01-02   -0.204708
2011-01-05    0.478943
2011-01-07   -0.519439
2011-01-08   -0.555730
2011-01-10    1.965781
2011-01-12    1.393406
dtype: float64
```

- DatetimeIndex的标量值是pandas的Timestamp对象。

```
In [46]: stamp = ts.index[0]

In [47]: stamp
Out[47]: Timestamp('2011-01-02 00:00:00')
```

- 所有使用datetime对象的地方都可以使用Timestamp。
  - Timestamp还可以存储频率信息（如有）并了解如何进行时区转换和其他类型操作。

索引、选择、子序列

```
In [42]: ts
Out[42]:
2011-01-02   -0.204708
2011-01-05    0.478943
2011-01-07   -0.519439
2011-01-08   -0.555730
2011-01-10    1.965781
2011-01-12    1.393406
dtype: float64
```

- 当基于标签进行索引和选择时，时间序列的行为和其他的pandas.Series类似：

```
In [48]: stamp = ts.index[2]

In [49]: ts[stamp]
Out[49]: -0.51943871505673811
```

- 为了方便，还可以传递一个能解释为日期的字符串：

```
In [50]: ts['1/10/2011']
Out[50]: 1.9657805725027142

In [51]: ts['20110110']
Out[51]: 1.9657805725027142
```

索引、选择、子序列

- 对一个长的时间序列，可以传递一个年份或一个年份和periods值来选择数据的切片：

```
In [52]: longer_ts = pd.Series(np.random.randn(1000),
   ....:                       index=pd.date_range('1/1/2000', periods=1000))

In [53]: longer_ts
Out[53]:
2000-01-01    0.092908
2000-01-02    0.281746
2000-01-03    0.769023
2000-01-04    1.246435
2000-01-05    1.007189
2000-01-06   -1.296221
2000-01-07    0.274992
2000-01-08    0.228913
2000-01-09    1.352917
2000-01-10    0.886429
                ...
2002-09-17   -0.139298
2002-09-18   -1.159926
2002-09-19    0.618965
2002-09-20    1.373890
2002-09-21   -0.983505
2002-09-22    0.930944
2002-09-23   -0.811676
2002-09-24   -1.830156
2002-09-25   -0.138730
2002-09-26    0.334088
Freq: D, Length: 1000, dtype: float64
```

```
In [54]: longer_ts['2001']
Out[54]:
2001-01-01    1.599534
2001-01-02    0.474071
2001-01-03    0.151326
2001-01-04   -0.542173
2001-01-05   -0.475496
2001-01-06    0.106403
2001-01-07   -1.308228
2001-01-08    2.173185
2001-01-09    0.564561
2001-01-10   -0.190481
                ...
2001-12-22    0.000369
2001-12-23    0.900885
2001-12-24   -0.454869
2001-12-25   -0.864547
2001-12-26    1.129120
2001-12-27    0.057874
2001-12-28   -0.433739
2001-12-29    0.092698
2001-12-30   -1.397820
2001-12-31    1.457823
Freq: D, Length: 365, dtype: float64
```

索引、选择、子序列

Here, the string '2001' is interpreted as a year and selects that time period. This also works if you specify the month:

```
In [55]: longer_ts['2001-05']
Out[55]:
2001-05-01   -0.622547
2001-05-02    0.936289
2001-05-03    0.750018
2001-05-04   -0.056715
2001-05-05    2.300675
2001-05-06    0.569497
2001-05-07    1.489410
2001-05-08    1.264250
2001-05-09   -0.761837
2001-05-10   -0.331617
                ...
2001-05-22    0.503699
2001-05-23   -1.387874
2001-05-24    0.204851
2001-05-25    0.603705
2001-05-26    0.545680
2001-05-27    0.235477
2001-05-28    0.111835
2001-05-29   -1.251504
2001-05-30   -2.949343
2001-05-31    0.634634
Freq: D, Length: 31, dtype: float64
```

索引、选择、子序列

```
In [42]: ts
Out[42]:
2011-01-02   -0.204708
2011-01-05    0.478943
2011-01-07   -0.519439
2011-01-08   -0.555730
2011-01-10    1.965781
2011-01-12    1.393406
dtype: float64
```

- 也可以使用datetime对象进行切片：

```
In [56]: ts[datetime(2011, 1, 7):]
Out[56]:
2011-01-07   -0.519439
2011-01-08   -0.555730
2011-01-10    1.965781
2011-01-12    1.393406
dtype: float64
```

- 由于大多数时间序列数据是按时间顺序排序的，**可以使用不包含在时间序列中的时间戳进行切片**，以执行范围查询：

```
In [58]: ts['1/6/2011':'1/11/2011']
Out[58]:
2011-01-07   -0.519439
2011-01-08   -0.555730
2011-01-10    1.965781
dtype: float64
```

索引、选择、子序列

```
In [42]: ts
Out[42]:
2011-01-02   -0.204708
2011-01-05    0.478943
2011-01-07   -0.519439
2011-01-08   -0.555730
2011-01-10    1.965781
2011-01-12    1.393406
dtype: float64
```

- 有一个等价实例方法，truncate，它可以在两个日期间对Series进行切片：

```
In [59]: ts.truncate(after='1/9/2011')
Out[59]:
2011-01-02   -0.204708
2011-01-05    0.478943
2011-01-07   -0.519439
2011-01-08   -0.555730
dtype: float64
```

索引、选择、子序列

- 以上的很多操作也都适用于DataFrame，并在其行上进行索引：

```
In [60]: dates = pd.date_range('1/1/2000', periods=100, freq='W-WED')

In [61]: long_df = pd.DataFrame(np.random.randn(100, 4),
   ....:                        index=dates,
   ....:                        columns=['Colorado', 'Texas',
   ....:                                 'New York', 'Ohio'])

In [62]: long_df.loc['5-2001']
Out[62]:
            Colorado     Texas  New York      Ohio
2001-05-02 -0.006045  0.490094 -0.277186 -0.707213
2001-05-09 -0.560107  2.735527  0.927335  1.513906
2001-05-16  0.538600  1.273768  0.667876 -0.969206
2001-05-23  1.676091 -0.817649  0.050188  1.951312
2001-05-30  3.260383  0.963301  1.201206 -1.852001
```

【补充】freq参数

Alias	Description
B	business day frequency
C	custom business day frequency
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
BQ	business quarter end frequency
QS	quarter start frequency
BQS	business quarter start frequency
A, Y	year end frequency
BA, BY	business year end frequency
AS, YS	year start frequency
BAS, BYS	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

含有重复索引的时间序列

```
In [63]: dates = pd.DatetimeIndex(['1/1/2000', '1/2/2000', '1/2/2000',
   ....:                           '1/2/2000', '1/3/2000'])

In [64]: dup_ts = pd.Series(np.arange(5), index=dates)

In [65]: dup_ts
Out[65]:
2000-01-01    0
2000-01-02    1
2000-01-02    2
2000-01-02    3
2000-01-03    4
dtype: int64
```

We can tell that the index is not unique by checking its `is_unique` property:

```
In [66]: dup_ts.index.is_unique
Out[66]: False
```

含有重复索引的时间序列

```
In [65]: dup_ts
Out[65]:
2000-01-01    0
2000-01-02    1
2000-01-02    2
2000-01-02    3
2000-01-03    4
```

- 对上面的Series进行索引，结果是**标量值**还是**Series切片**取决于是否有时间戳是重复的：

```
In [67]: dup_ts['1/3/2000']  # not duplicated
Out[67]: 4

In [68]: dup_ts['1/2/2000']  # duplicated
Out[68]:
2000-01-02    1
2000-01-02    2
2000-01-02    3
dtype: int64
```

含有重复索引的时间序列

```
In [65]: dup_ts
Out[65]:
2000-01-01    0
2000-01-02    1
2000-01-02    2
2000-01-02    3
2000-01-03    4
```

- 假设我们想要聚合含有非唯一时间戳的数据，一种方式就是使用groupby并传递level＝0：

```
In [69]: grouped = dup_ts.groupby(level=0)

In [70]: grouped.mean()
Out[70]:
2000-01-01    0
2000-01-02    2
2000-01-03    4
dtype: int64

In [71]: grouped.count()
Out[71]:
2000-01-01    1
2000-01-02    3
2000-01-03    1
dtype: int64
```

可以用在什么场景呢？

日期的范围、频率和转化

日期的范围

- 再次强调：Pandas.date_range是用于根据特定频率生成指定长度的DateTimeIndex
  - 默认情况下，date_range生成的是每日的时间戳

```
In [74]: index = pd.date_range('2012-04-01', '2012-06-01')

In [75]: index
Out[75]:
DatetimeIndex(['2012-04-01', '2012-04-02', '2012-04-03', '2012-04-04',
               '2012-04-05', '2012-04-06', '2012-04-07', '2012-04-08',
               '2012-04-09', '2012-04-10', '2012-04-11', '2012-04-12',
               '2012-04-13', '2012-04-14', '2012-04-15', '2012-04-16',
               '2012-04-17', '2012-04-18', '2012-04-19', '2012-04-20',
               '2012-04-21', '2012-04-22', '2012-04-23', '2012-04-24',
               '2012-04-25', '2012-04-26', '2012-04-27', '2012-04-28',
               '2012-04-29', '2012-04-30', '2012-05-01', '2012-05-02',
               '2012-05-03', '2012-05-04', '2012-05-05', '2012-05-06',
               '2012-05-07', '2012-05-08', '2012-05-09', '2012-05-10',
               '2012-05-11', '2012-05-12', '2012-05-13', '2012-05-14',
               '2012-05-15', '2012-05-16', '2012-05-17', '2012-05-18',
               '2012-05-19', '2012-05-20', '2012-05-21', '2012-05-22',
               '2012-05-23', '2012-05-24', '2012-05-25', '2012-05-26',
               '2012-05-27', '2012-05-28', '2012-05-29', '2012-05-30',
               '2012-05-31', '2012-06-01'],
              dtype='datetime64[ns]', freq='D')
```

日期的范围

- 如果只传递一个起始或结尾日期，必须传递一个用于生成范围的数字：

```
In [76]: pd.date_range(start='2012-04-01', periods=20)
Out[76]:
DatetimeIndex(['2012-04-01', '2012-04-02', '2012-04-03', '2012-04-04',
               '2012-04-05', '2012-04-06', '2012-04-07', '2012-04-08',
               '2012-04-09', '2012-04-10', '2012-04-11', '2012-04-12',
               '2012-04-13', '2012-04-14', '2012-04-15', '2012-04-16',
               '2012-04-17', '2012-04-18', '2012-04-19', '2012-04-20'],
              dtype='datetime64[ns]', freq='D')

In [77]: pd.date_range(end='2012-06-01', periods=20)
Out[77]:
DatetimeIndex(['2012-05-13', '2012-05-14', '2012-05-15', '2012-05-16',
               '2012-05-17', '2012-05-18', '2012-05-19', '2012-05-20',
               '2012-05-21', '2012-05-22', '2012-05-23', '2012-05-24',
               '2012-05-25', '2012-05-26', '2012-05-27', '2012-05-28',
               '2012-05-29', '2012-05-30', '2012-05-31', '2012-06-01'],
              dtype='datetime64[ns]', freq='D')
```

日期的范围

BusinessMonthEnd，工作日的月底日期

```
In [78]: pd.date_range('2000-01-01', '2000-12-01', freq='BM')
Out[78]:
DatetimeIndex(['2000-01-31', '2000-02-29', '2000-03-31', '2000-04-28',
               '2000-05-31', '2000-06-30', '2000-07-31', '2000-08-31',
               '2000-09-29', '2000-10-31', '2000-11-30'],
              dtype='datetime64[ns]', freq='BM')
```

日期的范围

- 有时候你会获得包含时间信息的开始日期或结束日期，但是你想要生成的是标准化为0:00的时间戳。此时可使用normalize选项实现这个功能：

```
In [80]: pd.date_range('2012-05-02 12:56:31', periods=5, normalize=True)
Out[80]:
DatetimeIndex(['2012-05-02', '2012-05-03', '2012-05-04', '2012-05-05',
               '2012-05-06'],
              dtype='datetime64[ns]', freq='D')
```

没有normalize的时候

```
1  pd.date_range('2012-05-02 12:56:31', periods=5)
```

```
DatetimeIndex(['2012-05-02 12:56:31', '2012-05-03 12:56:31',
               '2012-05-04 12:56:31', '2012-05-05 12:56:31',
               '2012-05-06 12:56:31'],
              dtype='datetime64[ns]', freq='D')
```

日期的频率

- Pandas中的频率是由**基础频率**和**倍数**组成的。
  - 基础频率会有字符串别名，如M表示每月，H表示每小时
  - 对每个基础频率，都有一个对象可以被用于定义日期偏置：

```
In [81]: from pandas.tseries.offsets import Hour, Minute

In [82]: hour = Hour()

In [83]: hour
Out[83]: <Hour>
```

```
In [86]: pd.date_range('2000-01-01', '2000-01-03 23:59', freq='4h')
Out[86]:
DatetimeIndex(['2000-01-01 00:00:00', '2000-01-01 04:00:00',
               '2000-01-01 08:00:00', '2000-01-01 12:00:00',
               '2000-01-01 16:00:00', '2000-01-01 20:00:00',
               '2000-01-02 00:00:00', '2000-01-02 04:00:00',
               '2000-01-02 08:00:00', '2000-01-02 12:00:00',
               '2000-01-02 16:00:00', '2000-01-02 20:00:00',
               '2000-01-03 00:00:00', '2000-01-03 04:00:00',
               '2000-01-03 08:00:00', '2000-01-03 12:00:00',
               '2000-01-03 16:00:00', '2000-01-03 20:00:00'],
              type='datetime64[ns]', freq='4H')
```

日期的频率

```
In [87]: Hour(2) + Minute(30)
Out[87]: <150 * Minutes>

In [88]: pd.date_range('2000-01-01', periods=10, freq='1h30min')
Out[88]:
DatetimeIndex(['2000-01-01 00:00:00', '2000-01-01 01:30:00',
               '2000-01-01 03:00:00', '2000-01-01 04:30:00',
               '2000-01-01 06:00:00', '2000-01-01 07:30:00',
               '2000-01-01 09:00:00', '2000-01-01 10:30:00',
               '2000-01-01 12:00:00', '2000-01-01 13:30:00'],
              dtype='datetime64[ns]', freq='90T')
```

日期的频率：补充一个例子

```
1  rng = pd.date_range('2012-01-01', '2012-09-01', freq='WOM-3FRI')
2  list(rng)
```

```
[Timestamp('2012-01-20 00:00:00', freq='WOM-3FRI'),
 Timestamp('2012-02-17 00:00:00', freq='WOM-3FRI'),
 Timestamp('2012-03-16 00:00:00', freq='WOM-3FRI'),
 Timestamp('2012-04-20 00:00:00', freq='WOM-3FRI'),
 Timestamp('2012-05-18 00:00:00', freq='WOM-3FRI'),
 Timestamp('2012-06-15 00:00:00', freq='WOM-3FRI'),
 Timestamp('2012-07-20 00:00:00', freq='WOM-3FRI'),
 Timestamp('2012-08-17 00:00:00', freq='WOM-3FRI')]
```

“月中某星期”（week of month）表示为WOM，如每月第三个星期五

日期的“移位”

- “移位”是指将日期按时间向前移动或向后移动。
  - Series和DataFrame都有一个shift方法用于进行简单的前、后向移位，而不改变索引。

```
In [91]: ts = pd.Series(np.random.randn(4),
   ....:                index=pd.date_range('1/1/2000', periods=4, freq='M'))

In [92]: ts
Out[92]:
2000-01-31   -0.066748
2000-02-29    0.838639
2000-03-31   -0.117388
2000-04-30   -0.517795
Freq: M, dtype: float64
```

```
In [93]: ts.shift(2)
Out[93]:
2000-01-31         NaN
2000-02-29         NaN
2000-03-31   -0.066748
2000-04-30    0.838639
Freq: M, dtype: float64

In [94]: ts.shift(-2)
Out[94]:
2000-01-31   -0.117388
2000-02-29   -0.517795
2000-03-31         NaN
2000-04-30         NaN
Freq: M, dtype: float64
```

日期的“移位”

```
In [92]: ts
Out[92]:
2000-01-31   -0.066748
2000-02-29    0.838639
2000-03-31   -0.117388
2000-04-30   -0.517795
```

If freq is specified then the index values are shifted but the data is not realigned.

```
In [95]: ts.shift(2, freq='M'
Out[95]:
2000-03-31   -0.066748
2000-04-30    0.838639

2000-05-31   -0.117388
2000-06-30   -0.517795
Freq: M, dtype: float64
```

```
In [96]: ts.shift(3, freq='D')
Out[96]:
2000-02-03   -0.066748
2000-03-03    0.838639
2000-04-03   -0.117388
2000-05-03   -0.517795
dtype: float64
```

```
In [97]: ts.shift(1, freq='90T')
Out[97]:
2000-01-31 01:30:00   -0.066748
2000-02-29 01:30:00    0.838639
2000-03-31 01:30:00   -0.117388
2000-04-30 01:30:00   -0.517795
Freq: M, dtype: float64
```

The T here stands for minutes.

日期的“移位”

- Pandas日期偏置也可以使用datetime或Timestamp对象完成

```
In [98]: from pandas.tseries.offsets import Day, MonthEnd

In [99]: now = datetime(2011, 11, 17)

In [100]: now + 3 * Day()
Out[100]: Timestamp('2011-11-20 00:00:00')


In [101]: now + MonthEnd()
Out[101]: Timestamp('2011-11-30 00:00:00')

In [102]: now + MonthEnd(2)
Out[102]: Timestamp('2011-12-31 00:00:00')
```

日期的“移位”

```
In [99]: now = datetime(2011, 11, 17)
In [103]: offset = MonthEnd()

In [104]: offset.rollforward(now)
Out[104]: Timestamp('2011-11-30 00:00:00')

In [105]: offset.rollback(now)
Out[105]: Timestamp('2011-10-31 00:00:00')
```

日期的“移位”

```
In [106]: ts = pd.Series(np.random.randn(20),
   .....:                index=pd.date_range('1/15/2000', periods=20, freq='4d'))

In [107]: ts
Out[107]:
2000-01-15   -0.116696
2000-01-19    2.389645
2000-01-23   -0.932454
2000-01-27   -0.229331
2000-01-31   -1.140330
2000-02-04    0.439920
2000-02-08   -0.823758
2000-02-12   -0.520930
2000-02-16    0.350282
2000-02-20    0.204395
2000-02-24    0.133445
2000-02-28    0.327905
2000-03-03    0.072153
2000-03-07    0.131678
2000-03-11   -1.297459
2000-03-15    0.997747
2000-03-19    0.870955
2000-03-23   -0.991253
2000-03-27    0.151699
2000-03-31    1.266151
Freq: 4D, dtype: float64
```

```
In [108]: ts.groupby(offset.rollforward).mean()
Out[108]:
2000-01-31   -0.005833
2000-02-29    0.015894
2000-03-31    0.150209
dtype: float64
```

时区

```
In [110]: import pytz

In [111]: pytz.common_timezones[-5:]
Out[111]: ['US/Eastern', 'US/Hawaii', 'US/Mountain', 'US/Pacific', 'UTC']
```

To get a time zone object from `pytz`, use `pytz.timezone`:

```
In [112]: tz = pytz.timezone('America/New_York')

In [113]: tz
Out[113]: <DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>
```

Methods in pandas will accept either time zone names or these objects.

Time Zone Localization and Conversion

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

```
In [114]: rng = pd.date_range('3/9/2012 9:30', periods=6, freq='D')

In [115]: ts = pd.Series(np.random.randn(len(rng)), index=rng)

In [116]: ts
Out[116]:
2012-03-09 09:30:00   -0.202469
2012-03-10 09:30:00    0.050718
2012-03-11 09:30:00    0.639869
2012-03-12 09:30:00    0.597594
2012-03-13 09:30:00   -0.797246
2012-03-14 09:30:00    0.472879
Freq: D, dtype: float64
```

The index's `tz` field is `None`:

```
In [117]: print(ts.index.tz)
None
```

Date ranges can be generated with a time zone set:

```
In [118]: pd.date_range('3/9/2012 9:30', periods=10, freq='D', tz='UTC')
Out[118]:
DatetimeIndex(['2012-03-09 09:30:00+00:00', '2012-03-10 09:30:00+00:00',
                    -03-11 09:30:00+00:00', '2012-03-12 09:30:00+00:00',
                    -03-13 09:30:00+00:00', '2012-03-14 09:30:00+00:00',
                    -03-15 09:30:00+00:00', '2012-03-16 09:30:00+00:00',
                    -03-17 09:30:00+00:00', '2012-03-18 09:30:00+00:00'],
                    'datetime64[ns, UTC]', freq='D')
```

时间区间和区间算术

时间区间

- 时间区间表示的是时间范围，使用Period类：

```
In [149]: p = pd.Period(2007, freq='A-DEC')

In [150]: p
Out[150]: Period('2007', 'A-DEC')

In [151]: p + 5
Out[151]: Period('2012', 'A-DEC')

In [152]: p - 2
Out[152]: Period('2005', 'A-DEC')

In [153]: pd.Period('2014', freq='A-DEC') - p
Out[153]: 7
```

给定月份（DEC）所在月的最后一个工作日所对应的年度日期

两个区间频率相同，差值是它们间的单位数

时间区间

- Period_range函数可以构造规则区间序列：

```
In [154]: rng = pd.period_range('2000-01-01', '2000-06-30', freq='M')

In [155]: rng
Out[155]: PeriodIndex(['2000-01', '2000-02', '2000-03', '2000-04', '2000-05', '20
00-06'], dtype='period[M]', freq='M')

In [156]: pd.Series(np.random.randn(6), index=rng)
Out[156]:
2000-01   -0.514551
2000-02   -0.559782
2000-03   -0.783408
2000-04   -1.797685
2000-05   -0.172670
2000-06    0.680215
Freq: M, dtype: float64
```

Q代表季度为频率,默认的后缀为DEC代表一年以第12个月为结束【最后一个月为12月份】

If you have an array of strings, you can also use the PeriodIndex class:

```
In [157]: values = ['2001Q3', '2002Q2', '2003Q1']

In [158]: index = pd.PeriodIndex(values, freq='Q-DEC')

In [159]: index
Out[159]: PeriodIndex(['2001Q3', '2002Q2', '2003Q1'], dtype='period[Q-DEC]', freq
='Q-DEC')
```

区间频率转化

- 使用asfreq可以将区间和PeriodIndex对象转换为其他的频率。例如，假设我们有一个年度区间，并且想要在一年的开始或结束时将其转换为月度区间：

```
In [160]: p = pd.Period('2007', freq='A-DEC')

In [161]: p
Out[161]: Period('2007', 'A-DEC')

In [162]: p.asfreq('M', how='start')
Out[162]: Period('2007-01', 'M')

In [163]: p.asfreq('M', how='end')
Out[163]: Period('2007-12', 'M')
```

以年为单位，07年6月作为游标。周期的起始点是06年7月，终点为07年6月。

```
In [164]: p = pd.Period('2007', freq='A-JUN')

In [165]: p
Out[165]: Period('2007', 'A-JUN')

In [166]: p.asfreq('M', 'start')
Out[166]: Period('2006-07', 'M')

In [167]: p.asfreq('M', 'end')
Out[167]: Period('2007-06', 'M')
```

区间频率转化

```
In [170]: rng = pd.period_range('2006', '2009', freq='A-DEC')

In [171]: ts = pd.Series(np.random.randn(len(rng)), index=rng)

In [172]: ts
Out[172]:
2006    1.607578
2007    0.200381
2008   -0.834068
2009   -0.302988
Freq: A-DEC, dtype: float64

In [173]: ts.asfreq('M', how='start')
Out[173]:
2006-01    1.607578
2007-01    0.200381
2008-01   -0.834068
2009-01   -0.302988
Freq: M, dtype: float64

In [174]: ts.asfreq('B', how='end')
Out[174]:
2006-12-29    1.607578
2007-12-31    0.200381
2008-12-31   -0.834068
2009-12-31   -0.302988
Freq: B, dtype: float64
```

时间戳和区间之间的转化

```
In [188]: rng = pd.date_range('2000-01-01', periods=3, freq='M')

In [189]: ts = pd.Series(np.random.randn(3), index=rng)

In [190]: ts
Out[190]:
2000-01-31    1.663261
2000-02-29   -0.996206
2000-03-31    1.521760
Freq: M, dtype: float64

In [191]: pts = ts.to_period()

In [192]: pts
Out[192]:
2000-01    1.663261
2000-02   -0.996206
2000-03    1.521760
Freq: M, dtype: float64
```

```
In [193]: rng = pd.date_range('1/29/2000', periods=6, freq='D')

In [194]: ts2 = pd.Series(np.random.randn(6), index=rng)

In [195]: ts2
Out[195]:
2000-01-29    0.244175
2000-01-30    0.423331
2000-01-31   -0.654040
2000-02-01    2.089154
2000-02-02   -0.060220
2000-02-03   -0.167933
Freq: D, dtype: float64

In [196]: ts2.to_period('M')
Out[196]:
2000-01    0.244175
2000-01    0.423331
2000-01   -0.654040
2000-02    2.089154
2000-02   -0.060220
2000-02   -0.167933
Freq: M, dtype: float64
```

时间戳和区间之间的转化

```
In [195]: ts2
Out[195]:
2000-01-29    0.244175
2000-01-30    0.423331
2000-01-31   -0.654040
2000-02-01    2.089154
2000-02-02   -0.060220
2000-02-03   -0.167933
Freq: D, dtype: float64
```

To convert back to timestamps, use to_timestamp:

```
In [197]: pts = ts2.to_period()

In [198]: pts
Out[198]:
2000-01-29    0.244175
2000-01-30    0.423331
2000-01-31   -0.654040
2000-02-01    2.089154
2000-02-02   -0.060220
2000-02-03   -0.167933
Freq: D, dtype: float64

In [199]: pts.to_timestamp(how='end')
Out[199]:
2000-01-29    0.244175
2000-01-30    0.423331
2000-01-31   -0.654040
2000-02-01    2.089154
2000-02-02   -0.060220
2000-02-03   -0.167933
Freq: D, dtype: float64
```

时间序列的重采样 (resampling)

重采样 (Resampling)

- 重采样：将时间序列从一个频率转换为另一个频率的过程。
  - 高频率数据聚合到低频率：下采样（Downsampling）
  - 低频率数据转换到高频率：上采样（Upsampling）
  - 其他：如每周三的数据转化为每周五的数据

重采样 (Resampling)

```
In [208]: rng = pd.date_range('2000-01-01', periods=100, freq='D')

In [209]: ts = pd.Series(np.random.randn(len(rng)), index=rng)

In [210]: ts
Out[210]:
2000-01-01    0.631634
2000-01-02   -1.594313
2000-01-03   -1.519937
2000-01-04    1.108752
2000-01-05    1.255853
2000-01-06   -0.024330
2000-01-07   -2.047939
2000-01-08   -0.272657
2000-01-09   -1.692615
2000-01-10    1.423830
                ...
2000-03-31   -0.007852
2000-04-01   -1.638806
2000-04-02    1.401227
2000-04-03    1.758539
2000-04-04    0.628932
2000-04-05   -0.423776
2000-04-06    0.789740
2000-04-07    0.937568
2000-04-08   -2.253294
2000-04-09   -1.772919
Freq: D, Length: 100, dtype: float64
```

```
In [211]: ts.resample('M').mean()
Out[211]:
2000-01-31   -0.165893
2000-02-29    0.078606
2000-03-31    0.223811
2000-04-30   -0.063643
Freq: M, dtype: float64

In [212]: ts.resample('M', kind='period').mean()
Out[212]:
2000-01   -0.165893
2000-02    0.078606
2000-03    0.223811
2000-04   -0.063643
Freq: M, dtype: float64
```

下采样 (Downsampling)

```
In [213]: rng = pd.date_range('2000-01-01', periods=12, freq='T')

In [214]: ts = pd.Series(np.arange(12), index=rng)

In [215]: ts
Out[215]:
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
```

默认情况下，左箱体边界是包含的，即00:00的值包含在了00:00-00:05内；但传递close='right'后，间隔的闭合端改为了右边。

```
In [216]: ts.resample('5min', closed='right').sum()
Out[216]:
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 [218]: ts.resample('5min', closed='right', label='right').sum()
Out[218]:
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
```

默认情况下，产生的时间序列按照每个箱体左边的时间戳标记；传递label='right'后，将使用右箱体边界时间序列。

下采样 (Downsampling)

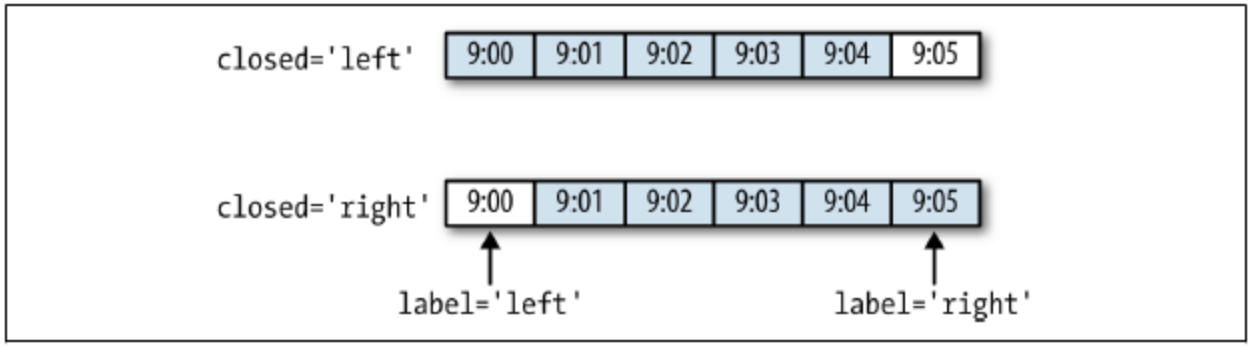


Figure 11-3. Five-minute resampling illustration of closed, label conventions

```
In [219]: ts.resample('5min', closed='right',
   .....:                 label='right', loffset='-1s').sum()
Out[219]:
1999-12-31 23:59:59     0
2000-01-01 00:04:59    15
2000-01-01 00:09:59    40
2000-01-01 00:14:59    11
Freq: 5T, dtype: int64
```

Resample的方法参数

Argument	Description
`freq`	String or DateOffset indicating desired resampled frequency (e.g., 'M', '5min', or `Second(15)`)
`axis`	Axis to resample on; default axis=0
`fill_method`	How to interpolate when upsampling, as in `'ffill'` or `'bfill'`; by default does no interpolation
`closed`	In downsampling, which end of each interval is closed (inclusive), `'right'` or `'left'`
`label`	In downsampling, how to label the aggregated result, with the `'right'` or `'left'` bin edge (e.g., the 9:30 to 9:35 five-minute interval could be labeled `9:30` or `9:35`)
`loffset`	Time adjustment to the bin labels, such as `'-1s'` / `Second(-1)` to shift the aggregate labels one second earlier
`limit`	When forward or backward filling, the maximum number of periods to fill
`kind`	Aggregate to periods (`'period'`) or timestamps (`'timestamp'`); defaults to the type of index the time series has
`convention`	When resampling periods, the convention (`'start'` or `'end'`) for converting the low-frequency period to high frequency; defaults to `'end'`

【扩展】开端-峰值-谷值-结束（OHLC）重采样

- 金融领域：为每个数据箱计算四个值是一种流行的时间序列聚合方法。使用OHLC聚合函数。

```
In [215]: ts
Out[215]:
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
```

Open-High-Low-Close (OHLC) resampling

```
1  ts.resample('5min').ohlc()
```

	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

上采样（Upsampling）和差值（Interpolation）

```
In [221]: frame = pd.DataFrame(np.random.randn(2, 4),
   .....:                      index=pd.date_range('1/1/2000', periods=2,
   .....:                                          freq='W-WED'),
   .....:                      columns=['Colorado', 'Texas', 'New York', 'Ohio'])

In [222]: frame
Out[222]:
            Colorado     Texas  New York      Ohio
2000-01-05 -0.896431  0.677263  0.036503  0.087102
2000-01-12 -0.046662  0.927238  0.482284 -0.867130

In [223]: df_daily = frame.resample('D').asfreq()

In [224]: df_daily
Out[224]:
             Colorado     Texas  New York      Ohio
2000-01-05 -0.896431  0.677263  0.036503  0.087102
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 -0.046662  0.927238  0.482284 -0.867130
```

上采样和差值

```
In [222]: frame
Out[222]:
            Colorado     Texas  New York      Ohio
2000-01-05 -0.896431  0.677263  0.036503  0.087102
2000-01-12 -0.046662  0.927238  0.482284 -0.867130
```

```
In [225]: frame.resample('D').ffill()
Out[225]:
            Colorado     Texas  New York      Ohio
2000-01-05 -0.896431  0.677263  0.036503  0.087102
2000-01-06 -0.896431  0.677263  0.036503  0.087102
2000-01-07 -0.896431  0.677263  0.036503  0.087102
2000-01-08 -0.896431  0.677263  0.036503  0.087102
2000-01-09 -0.896431  0.677263  0.036503  0.087102
2000-01-10 -0.896431  0.677263  0.036503  0.087102
2000-01-11 -0.896431  0.677263  0.036503  0.087102
2000-01-12 -0.046662  0.927238  0.482284 -0.867130
```

```
In [226]: frame.resample('D').ffill(limit=2)
Out[226]:
            Colorado     Texas  New York      Ohio
2000-01-05 -0.896431  0.677263  0.036503  0.087102
2000-01-06 -0.896431  0.677263  0.036503  0.087102
2000-01-07 -0.896431  0.677263  0.036503  0.087102
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 -0.046662  0.927238  0.482284 -0.867130
```

移动窗口函数 (Moving window functions)

移动窗口函数

可用于平滑噪声或粗糙的数据！

移动窗口函数

```
In [235]: close_px_all = pd.read_csv('examples/stock_px_2.csv',
   .....:                            parse_dates=True, index_col=0)

In [236]: close_px = close_px_all[['AAPL', 'MSFT', 'XOM']]

In [237]: close_px = close_px.resample('B').ffill()
```

```
1 close_px
```

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

2292 rows × 3 columns

移动窗口函数

```
In [238]: close_px.AAPL.plot()
Out[238]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2f2570cf98>

In [239]: close_px.AAPL.rolling(250).mean().plot()
```

根据250日滑动窗口分组

移动窗口函数

```
In [241]: appl_std250 = close_px.AAPL.rolling(250, min_periods=10).std()

In [242]: appl_std250[5:12]
Out[242]:
2003-01-09         NaN
2003-01-10         NaN
2003-01-13         NaN
2003-01-14         NaN
2003-01-15    0.077496
2003-01-16    0.074760
2003-01-17    0.112368
Freq: B, Name: AAPL, dtype: float64

In [243]: appl_std250.plot()
```

时间序列的起始位置少于窗口区间，所以有NaN

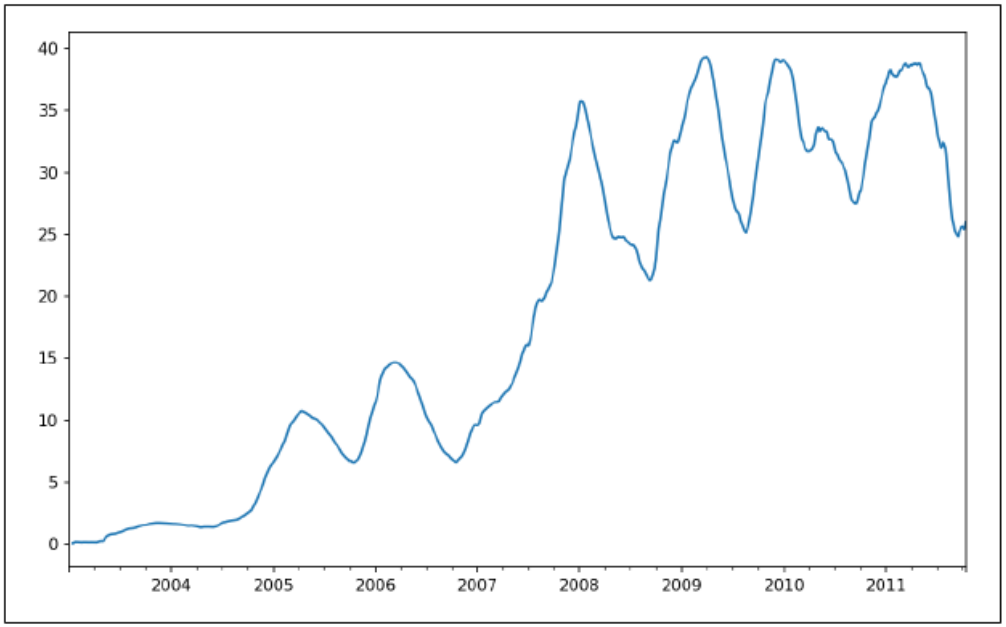


Figure 11-5. Apple 250-day daily return standard deviation

移动窗口函数

- 为了扩展窗口均值，可以使用expanding算子。扩展均值从时间序列起始位置开始时间窗口，并增加窗口的大小，直到它涵盖整个序列。

```
In [244]: expanding_mean = appl_std250.expanding().mean()

In [246]: close_px.rolling(60).mean().plot(logy=True)
```

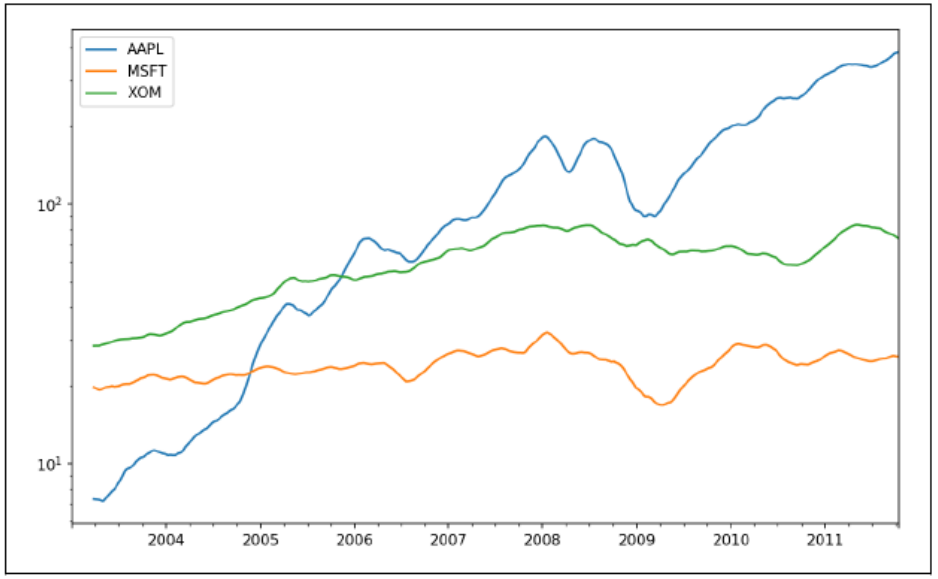


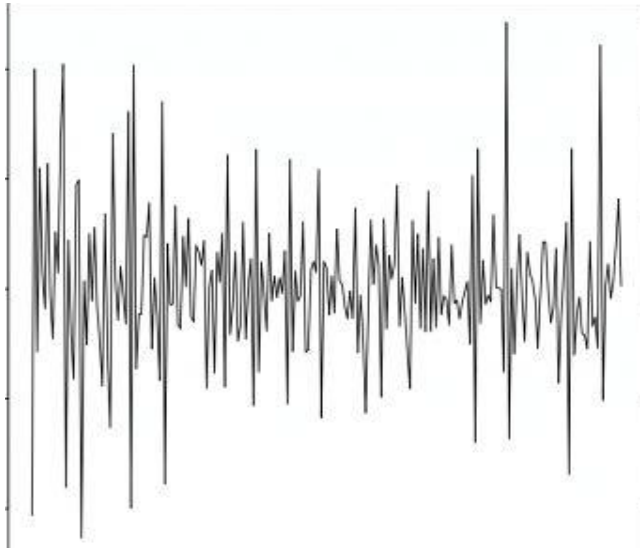
Figure 11-6. Stocks prices 60-day MA (log Y-axis)

其他滑动窗口函数

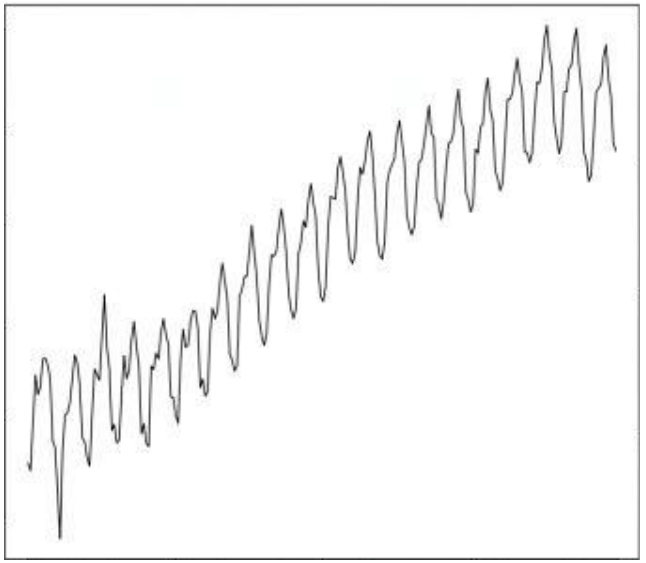
- 指数型滑动平均函数
- 二元移动窗口函数
- 用户自定义的移动窗口函数（使用apply方法等）
- ……

时间序列分析的后续内容

- 时间序列的预处理
- 平稳时间序列的分析（自相关、差分运算）
- 非平稳时间序列的分析
- 预测
- …

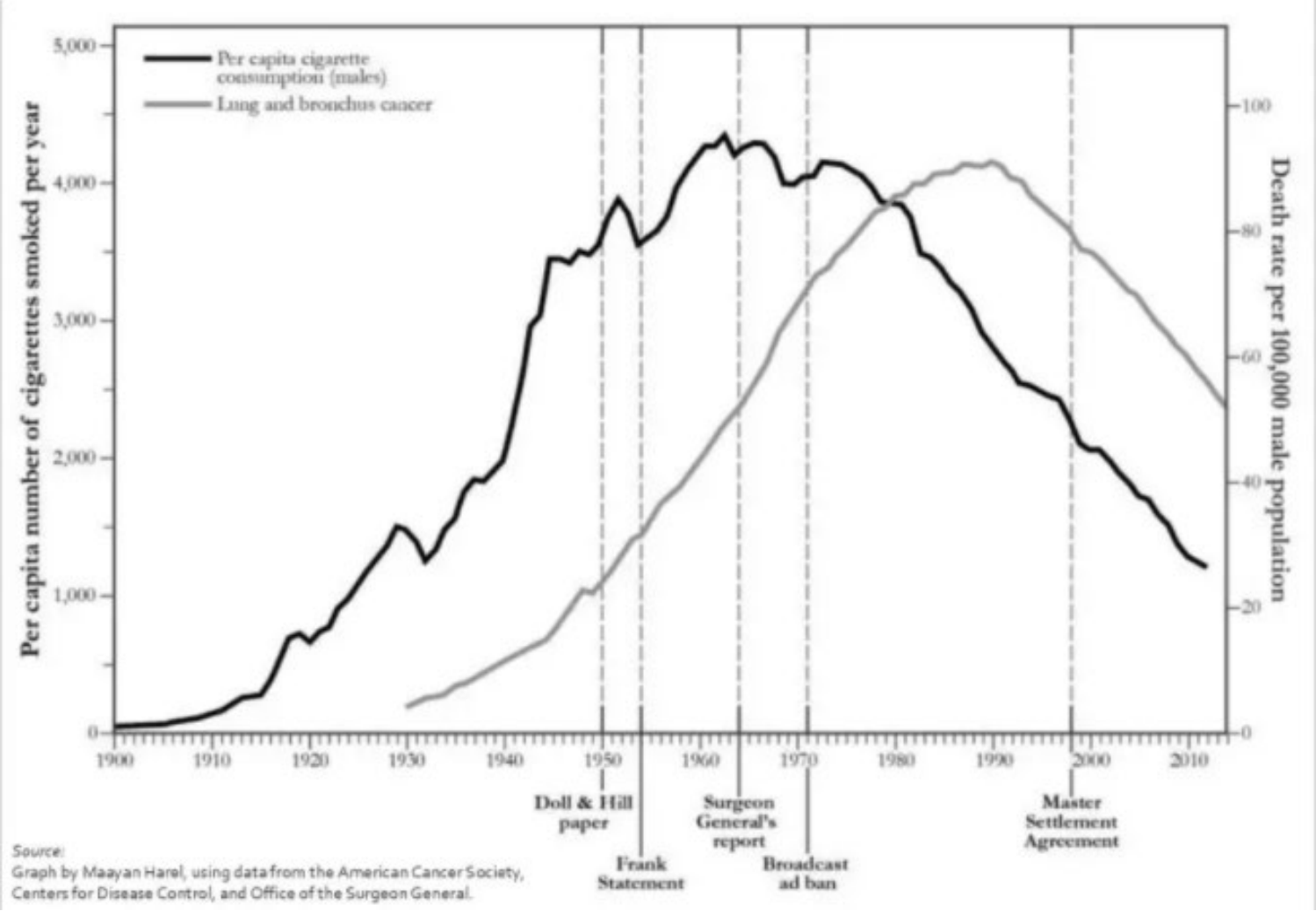


看起来平稳



看起来不平稳

时间序列分析的后续内容



个人作业4（共15分）

- 第一部分：时间序列分析
- 第二部分：图像数据分析
- 第三部分：文本数据分析
- 预计5月31日布置

课堂练习

谢谢！