第8章 时间序列统计分析

内容

- pandas模块
- statsmodels模块
- datetime模块
- 实际应用举例

金融时间序列数据

- 金融学中遇到的最重要的数据类型之一;
- 是以日期和时间作为索引(index)的数据;
- 例如：股价就表现为金融时间序列数据;
- 类似地，美元-欧元汇率也是金融时间序列数据;
- 时间是重要因素;
- Python中处理时间序列的主要工具是pandas库;
- Pandas模块中DateFrame和Series等基本类的灵感来自于R语言。

pandas模块

pandas基础

- 某种意义上，是在NumPy基础上构建，numpy通用函数通常也可以在pandas对象上工作，首先导入这两个库：

```
import numpy as np
import pandas as pd
```

- 使用DataFrame类第一步（管理带有索引和标签的数据）

```
df = pd.DataFrame([10, 20, 30, 40], columns=['numbers'],
                  index=['a', 'b', 'c', 'd'])
df
```

简单例子说明了DataFrame在存储数据上的主要特性。

	numbers
a	10
b	20
c	30
d	40

pandas基础

DataFrame在存储数据上的特性：

- 数据：可以用不同组成及类型提供（列表、数组、元组、字典对象）；
- 标签：数据组织为列(column)，可以自定义列名；

  df.columns=[]
- 索引（index）：可以采用不同格式（如数值、字符串、时间信息）　　df.index

总体上相当方便和高效，相比之下，常规的ndarray对象更专门化，也更受限制。

pandas基础

DataFrame 对象上典型操作方式：

```
df.index # the index values
df.columns # the column names
df.loc['c'] # selection via index
df.loc[['a', 'd']] # selection of multiple indices
df.loc[df.index[1:3]] # selection via Index object
df.sum() # sum per column

df.apply(lambda x: x ** 2) # square of every element

#Enlarging the DataFrame object in both dimensions is possible:
df['floats'] = (1.5, 2.5, 3.5, 4.5) # new column is generated
df
df['floats'] # selection of column
df['names'] = pd.DataFrame(['Yves', 'Guido', 'Felix', 'Francesc'],
                           index=['d', 'a', 'b', 'c'])
df
df.append({'numbers': 100, 'floats': 5.75, 'names': 'Henry'},
          ignore_index=True)
# temporary object; df not changed

#better to append a DataFrame object, providing the index information
df = df.append(pd.DataFrame({'numbers': 100, 'floats': 5.75,
                             'names': 'Henry'}, index=['z',]))
df
```

pandas基础

- 优势：处理缺失信息

```
#dealing with missing data
df.join(pd.DataFrame([1, 4, 9, 16, 25],
                     index=['a', 'b', 'c', 'd', 'y'],
                     columns=['squares',]))
# lose the value for the index y and have a NaN value

#to preserve both indices
df = df.join(pd.DataFrame([1, 4, 9, 16, 25],
                          index=['a', 'b', 'c', 'd', 'y'],
                          columns=['squares',]),how='outer')
df

#Although missing values, the majority of methods still work
df[['numbers', 'squares']].mean()
     # column-wise mean
df[['numbers', 'squares']].std()
   # column-wise standard deviation
```

pandas基础

使用DataFrame 第二步

- 模拟数据生成ndarray对象，pd.DataFrame()可将其转化成dataframe类型
- 增加DatetimeIndex等功能，以管理时间序列数据。
- pd.date_range函数生成DatetimeIndex

```
a = np.random.standard_normal((9,4))
a.round(6)
df = pd.DataFrame(a)
df
df.columns = ['No1', 'No2', 'No3', 'No4']
df
df['No2'][3] # value in column No2 at index position 3
####handle time  indices
dates = pd.date_range('2015-1-1', periods=9, freq='M')
dates
df.index = dates
df
##transform the dataframe into array
np.array(df).round(6)
```

DataFrame函数参数表：

参数	格式	描述
data	ndarray/字典/DataFrame	DataFrame 数据；字典可以包含序列、ndarray 和列表
index	索引/类似数组	使用的索引；默认为 range(n)
columns	索引/类似数组	使用的列标题；默认为 range(n)
dtype	dtype，默认 None	使用/强制的数据类型；否则通过推导得出
copy	布尔值，默认 None	从输入拷贝数据

date_range函数参数表：

参数	格式	描述
start	字符串/日期时间	生成日期的左界
end	字符串/日期时间	生成日期的右界
periods	整数/None	期数（如果 start 或者 end 空缺）
freq	字符串/日期偏移	频率字符串，例如 5D（5 天）
tz	字符串/None	本地化索引的时区名称
normalize	布尔值，默认 None	将 start 和 end 规范化为午夜
name	字符串，默认 None	结果索引名称

pandas基础

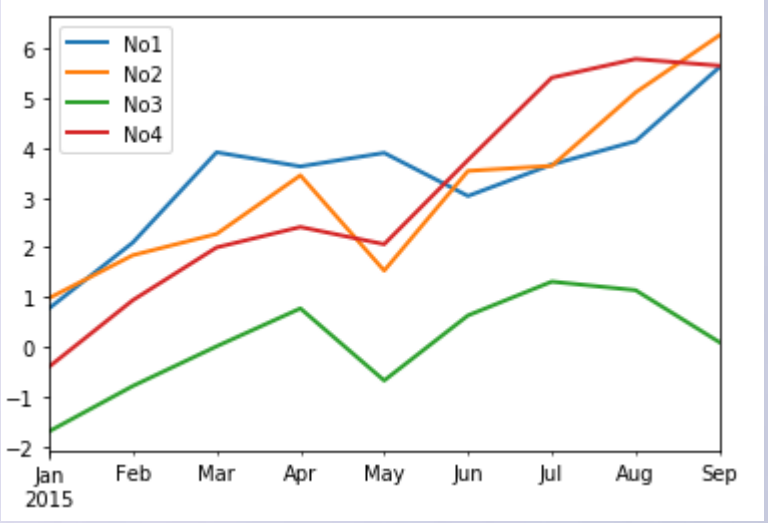
基础分析

- 和numpy数组一样，pandas DataFrame类有多个方便的内建方法。例如：按列总和、平均值和累计总和。
- 还有常用数值数据集的描述统计方法describe()函数。
- 也可对DataFrame对象应用大部分的NumPy通用函数。
- 相当强的容错能力，将不完整的数据集(有缺失数据时)当成完整的用。
- pandas提供matplotlib的一个封装器，专门为DataFrame对象设计。
- Pandas和其他分析库的结合点通常是NumPy数组。要将DataFrame转换为NumPy数组,可以使用.values属性。

pandas基础

df.sum()
df.mean()
df.cumsum()
df.describe()
np.sqrt(df)
np.sqrt(df).sum()

df.cumsum().plot(lw=2.0)

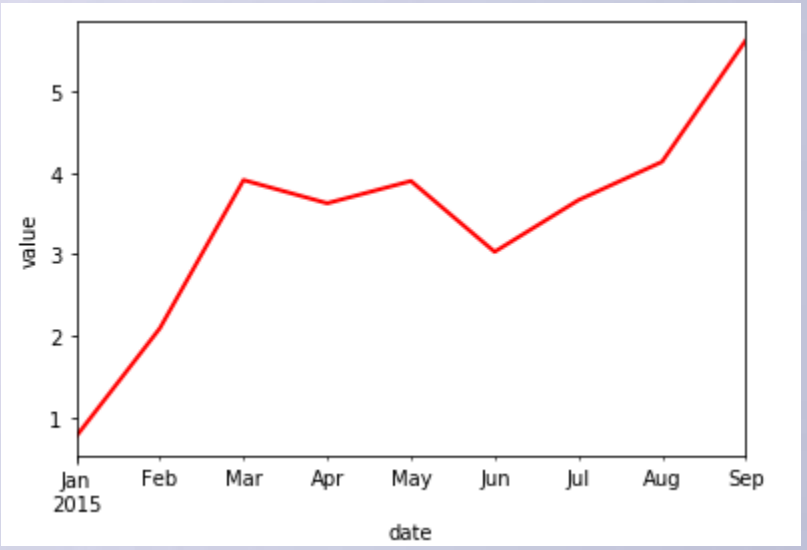


pandas基础

Series类

- 从DataFrame对象中选择一列时就得到一个Series对象;
- DataFrame主要方法也可用于Series对象.
- pd.Series()函数生成一维时间序列

```
df['No1']
type(df['No1'])

import matplotlib.pyplot as plt
df['No1'].cumsum().plot(style='r', lw=2.)
plt.xlabel('date')
plt.ylabel('value')

tsdat=pd.Series(np.random.random(len(dates)),index=dates)
tsdat.cumsum().plot(style='r', lw=2.)
```



pandas基础

GroupBy操作

- 强大的分组功能，类似于Excel中的透视表
- 根据季节进行分组(事先添加季度一列)
- 分组也可以在多列上进行（再添加一列月份的奇偶性）

```
df['Quarter'] = ['Q1', 'Q1', 'Q1', 'Q2', 'Q2', 'Q2',
                 'Q3', 'Q3', 'Q3']
df
groups = df.groupby('Quarter')
groups.mean()
groups.max()
groups.size()
```

Quarter	Odd_Even	No1	No2	No3	No4
Q1	Even	1.322510	0.862718	0.911829	1.337693
	Odd	1.293759	0.703616	-0.449049	0.330864
Q2	Even	-0.574897	1.589618	1.033775	1.049919
	Odd	0.272037	-1.912907	-1.447194	-0.345703
Q3	Even	0.469231	1.473899	-0.172201	0.375268
	Odd	1.058485	0.623849	-0.182769	0.760046

```
df['Odd_Even'] = ['Odd', 'Even', 'Odd', 'Even', 'Odd',
                  'Even','Odd', 'Even', 'Odd']
groups = df.groupby(['Quarter', 'Odd_Even'])
groups.size()
groups.mean()
```

补充：数据结构——字典(dict)

- 字典数据结构

Python中的字典是一种高效查询的数据结构，通过键值对来表示；
键(key)是字符串，必须是独一无二的，每个值（value）对应一个键,通过键可以迅速查询到值;
字典中不存在顺序，使用了哈希算法进行查询。

```
x = {"a":1, "b":5, "c":2}
type(x)
## <class 'dict'>
```

- 字典的使用

字典的主要作用就是根据键来查询，使用get方法可以实现高效的查询。

```
x.get("b")
## 5
```

- 字典信息的查询

items方法返回键值对的视图，keys方法返回所有键，values方法返回所有值。

```
x.items()
## dict_items([('b', 5), ('c', 2), ('a', 1)])
x.keys()
## dict_keys(['b', 'c', 'a'])
x.values()
## dict_values([5, 2, 1])
```

- 字典的删减

pop可以删除某个键，update可以添加某个键值对。

```
x.pop("b")
## {'a': 1, 'c': 2}

x.update({'d':'test' })
x
```

补充：数据结构——集合(set)

- 集合的数据结构

集合可以认为是只包含键的字典，也是使用大括号来创建。会自动清除重复的元素。

```
y = {"a", "b", "c"}
type(y)
## <class 'set'>
y = {"a", "b", "c", "b"}
y
## {'a', 'b', 'c'}
```

- 集合的基础操作

remove可以删除某个元素，update可以添加某个元素。

```
y.update("d")
y
## {'c', 'b', 'a', 'd'}
y.remove("b")
y
## {'c', 'a', 'd'}
```

到目前为止，我们学到的数据结构有：列表与元组；字典和集合；数组与pandas数据框。

读取数据方式

- 从剪贴板输入数据

pd.read_clipboard()

- 从雅虎财经网上读取数据
- 读取txt/csv数据文件

pd.read_csv()

- 读取xlsx数据文件

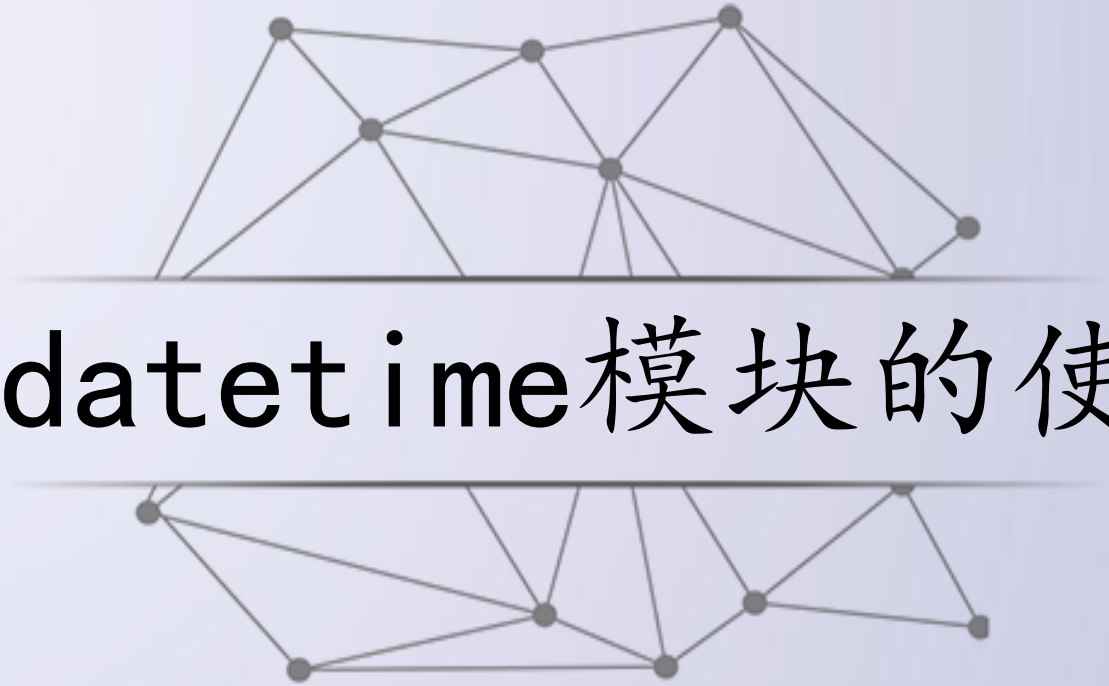
pd.read_excel()

pd.ExcelFile()

```
#1 Inputting data from the clipboard
# pd.read_clipboard()
import pandas as pd
# copy first in your written notebook
data=pd.read_clipboard()

#2 Retrieving historical price data from Yahoo!Finance
import pandas_datareader.data as web
ibm = web.DataReader(name='IBM', data_source='yahoo',
                     start='2013-1-1',end='2013-12-31')

ibm.to_csv("E:/datasets/ibm2013.txt")  #output data

#3 Inputting data from a text file
p1=pd.read_csv("E:/datasets/ibm2013.txt",index_col=0)
#3 Inputting data from a text file
p=pd.read_csv("D:/dataset/ibm.txt",index_col=0)

#4 Inputting data from an Excel file
import xlrd
x1=pd.read_excel("E:/datasets/test.xlsx",index_col=0,header=None)
x1.columns=[['a','b']]
x1.index.name='time'
infile=pd.ExcelFile("E:/datasets/datexcel.xlsx")
x2=infile.parse('Sheet2')
```

pandas输入和输出函数/方法

Format	Input	Output	Remark
CSV	read_csv	to_csv	Text file
XLS/XLSX	read_excel	to_excel	Spreadsheet
HDF	read_hdf	to_hdf	HDF5 database
SQL	read_sql	to_sql	SQL table
JSON	read_json	to_json	JavaScript Object Notation
MSGPACK	read_msgpack	to_msgpack	Portable binary format
HTML	read_html	to_html	HTML code
GBQ	read_gbq	to_gbq	Google Big Query format
DTA	read_stata	to_stata	Formats 104, 105, 108, 113-115, 117
Any	read_clipboard	to_clipboard	E.g., from HTML page
Any	read_pickle	to_pickle	(Structured) Python object



datetime模块的使用

datetime模块概述

以不同格式显示日期和时间是程序中最常用到的功能。Python提供了一个处理时间的标准函数模块datetime，它提供了一系列由简单到复杂的时间处理方法。

datetime模块概述

datetime模块以类的方式提供多种日期和时间表达方式：

- datetime.date：日期表示类，可以表示年、月、日等
- datetime.time：时间表示类，可以表示小时、分钟、秒、毫秒等
- datetime.datetime：日期和时间表示的类，功能覆盖date和time类
- datetime.timedelta：时间间隔有关的类
- datetime.tzinfo：与时区有关的信息表示类

datetime模块解析

使用datetime.now()获得当前日期和时间对象，使用方法如下：

datetime.now()

作用：返回一个datetime类型，表示当前的日期和时间，精确到微秒。

```
from datetime import datetime
datetime.now()
##datetime.datetime(2019, 10, 20, 8, 50, 58, 681000)
```

datetime模块解析

- 使用datetime.utcnow()获得当前日期和时间对应的UTC（世界标准时间）时间对象，使用方法如下：

  datetime.utcnow()

- 作用：返回datetime类型，表示当前日期和时间的UTC表示，精确到微秒。

```
datetime.utcnow()
##datetime.datetime(2019, 10, 20, 1, 2, 30, 55600)
```

datetime模块解析

- datetime.now()和datetime.utcnow()都返回一个datetime类型的对象。
- 也可以直接使用datetime()构造一个日期和时间对象：

  datetime(year,month,day,hour=0,minute=0,second=0,microsecond=0)

作用：返回一个datetime类型，表示指定的日期和时间，可以精确到微秒。

- 举例：调用datetime()函数直接创建一个datetime对象，表示2019年10月20日10:20，10秒20微秒

```
someday=datetime(2019,10,20,10,20,10,20)
someday
##datetime.datetime(2019, 10, 20, 10, 20, 10, 20)
```

- 程序已经有了一个datetime对象，进一步可以利用这个对象的属性显示时间。

datetime模块解析

属性	描述
someday.min	固定返回datetime的最小时间对象，datetime(1,1,1,0,0)
someday.max	固定返回datetime的最大时间对象，datetime(9999,12,31,23,59,59,999999)
someday.year	返回someday包含的年份
someday.month	返回someday包含的月份
someday.day	返回someday包含的日期
someday.hour	返回someday包含的小时
someday.minute	返回someday包含的分钟
someday.second	返回someday包含的秒钟
someday.microsecond	返回someday包含的微秒值

datetime模块解析

- datetime对象有3个常用的时间格式化方法，如表所示

属性	描述
someday.isoformat()	采用ISO 8601标准显示时间
someday.isoweekday()	根据日期计算星期后返回1-7,对应星期一到星期日
someday.strftime(format)	根据格式化字符串format进行格式显示的方法

isoformat()和isoweekday()方法的使用如下：

```
someday.isoformat()
##'2019-10-20T10:20:10'
someday.isoweekday()
##7
```

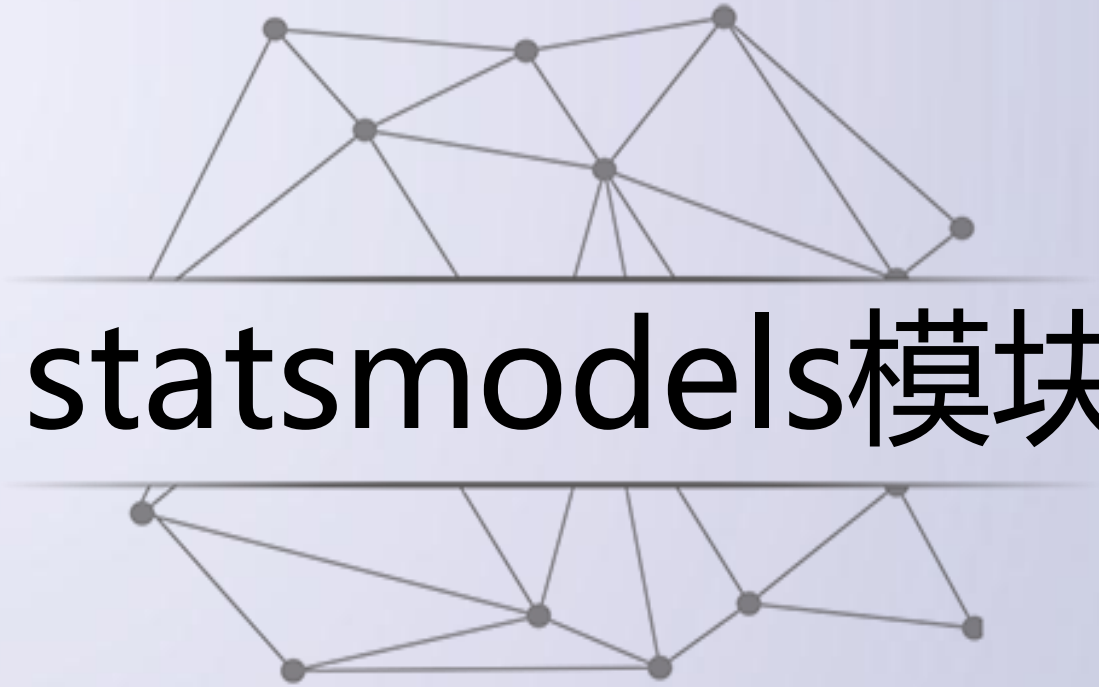
datetime模块解析

- strftime()方法是时间格式化最有效的方法，几乎可以以任何通用格式输出时间。
- strftime()格式化字符串的数字左侧会自动补零，上述格式也可以与print()的格式化函数一起使用。
- strptime()是strftime()反过程；将时间string类型按一定格式format转化为时间datetime.datetime类型。

```
someday.strftime("%Y-%m-%d %H:%M:%S")
someday.strftime("%Y/%m/%d %H:%M:%S")
now = datetime.now()
now.strftime("%Y-%m-%d")
now.strftime("%A, %d. %B %Y %I:%M%p")
print("今天是{0:%Y}年{0:%m}月{0:%d}日".format(now))
S=datetime.strptime('2017/09/30','%Y/%m/%d')
type(S)
print(S)
```

datetime模块解析

格式化字符串	日期/时间	值范围和实例
%Y	年份	0001~9999，例如：1900
%m	月份	01~12，例如：10
%B	月名	January~December，例如：April
%b	月名缩写	Jan~Dec，例如：Apr
%d	日期	01 ~ 31，例如：25
%A	星期	Monday~Sunday，例如：Wednesday
%a	星期缩写	Mon~Sun，例如：Wed
%H	小时（24h制）	00 ~ 23，例如：12
%I	小时（12h制）	01 ~ 12，例如：7
%p	上/下午	AM, PM，例如：PM
%M	分钟	00 ~ 59，例如：26
%S	秒	00 ~ 59，例如：26



statsmodels模块

statsmodels模块简介

statsmodels 是一个统计分析模块，源自斯坦福大学统计学教授Jonathan Taylor利用R语言实现的各类分析模型。Skipper Seabold和Josef Perktold 早在2010年便创建了新的statsmodels项目，自那之后该项目迅速成长。statsmodels包含经典的统计学、经济学算法。包含的模型如下：

- 回归模型：线性回归、广义线性模型、稳健线性模型、线性混合效应模型等；
- 方差分析(ANOVA)；
- 时间序列分析：AR、ARMA、ARIMA、VAR等模型；
- 非参数方法：核密度估计、核回归；
- 统计模型结果可视化

Statsmodels更专注于统计推断，提供不确定性评价和p值参数。而scikit-learn模块更专注于预测。

statsmodels模块简介

Statsmodels是Python的统计建模和计量经济学工具包，包括一些描述统计、统计模型估计和推断.用于拟合多种统计模型，执行统计测试以及数据探索和可视化。有以下用途：

- 线性回归模型 sm.ols
- 广义线型模型，主要用于各种设计的方差分析sm.GLM
- 稳健线性模型 smarm
- Logit模型 sm.Logit
- 时间序列分析 sm.tsa
- 各种统计检验和非参数检验
- 以各种方式输出表格：text，latex，html；读取各种格式的数据

注意：scipy.stats模块功能比较基础，缺了最重要的统计方法体系

statsmodels模块简介

statsmodels中的线性模型有两个不同的主要接口：基于数组和基于公式的。这些接口通过这些API模块导入来访问：

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

普通最小二乘的线性模型

- sm.OLS(Y,X) ：基于数组

其中Y:因变量数据；X:自变量数据（模型是否包含常数项）

- smf.ols(formula,data)：基于公式

其中formula:模型结构，如 y~x1+x2+x3+x4;

data:包含因变量Y和自变量X的数据框

广义最小二乘的线性模型*

- sm.GLS
- smf.gls

statsmodels模块应用

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
import numpy as np
import pandas as pd
x=np.random.standard_normal((20,4))
beta=np.array([[1],[2],[3],[4]])
y=1+np.random.standard_normal((20,1))+np.dot(x,beta)
x=pd.DataFrame(x)
x.columns=['b1','b2','b3','b4']
x=np.array(x)
x=sm.add_constant(x)
results=sm.OLS(y,x).fit()
results.summary()
```

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.1259	0.228	4.938	0.000	0.640	1.612
x1	0.9859	0.263	3.745	0.002	0.425	1.547
x2	1.8130	0.168	10.761	0.000	1.454	2.172
x3	2.8101	0.228	12.309	0.000	2.324	3.297
x4	3.8595	0.211	18.267	0.000	3.409	4.310

Dep. Variable:	y	**R-squared:**	0.973
Model:	OLS	**Adj. R-squared:**	0.965
Method:	Least Squares	**F-statistic:**	133.4
Date:	Sun, 28 Jul 2019	**Prob (F-statistic):**	1.56e-11
Time:	14:15:09	**Log-Likelihood:**	-22.641
No. Observations:	20	**AIC:**	55.28
Df Residuals:	15	**BIC:**	60.26
Df Model:	4		

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
import numpy as np
import pandas as pd
x=np.random.standard_normal((20,4))
beta=np.array([[1],[2],[3],[4]])
e=np.random.standard_normal((20,1))
y=1+np.dot(x,beta)+e
x=pd.DataFrame(x)
x.columns=['x1','x2','x3','x4']
data=pd.DataFrame(x)
data['y']=y
results=smf.ols('y~x1+x2+x3+x4',data=data).fit()
results.summary()
results.predict(data[:5])
```

实际应用

金融数据获取

- 当今的Web本身提供了大量的免费金融信息，如https://finance.yahoo.com/；
- 尽管有些时候不能满足专业需求，但是适合阐述pandas的金融能力；
- 安装pandas_datareader，使用web内建函数DataReader,从雅虎财经读取股价数据，分析数据并生成不同图表。

```
import pandas_datareader.data as web
DAX = web.DataReader(name='^GDAXI', data_source='yahoo',
                     start='2000-1-1')
DAX.info()
#Figure:Historical DAX index levels
DAX['Close'].plot(figsize=(8, 5))
```

金融资产回报率(return)

以P_t表示金融资产在时刻t的价格，那么金融资产回报率可以定义为：

- 净回报率 $R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$
- 总回报率 $\frac{P_t}{P_{t-1}} = R_t + 1$
- 对数回报率 $r_t = \log\left(\frac{P_t}{P_{t-1}}\right) = \log(R_t + 1)$

由日回报率计算月回报率

假设有价格数据$P_0, P_1, \cdots, P_{20}$，其中，P_0为上个月的最后一个交易日的收盘价，P_1是这个月第一个交易日的收盘价，依次为这个月每个交易日的收盘价，P_{20}是这个月最后一个交易日的收盘价。

- 计算这个月的回报率公式为：

$$R_{monthly} = \frac{P_{20} - P_0}{P_0}$$

- 对数回报率：$r_{monthly} = \log\left(\frac{P_{20}}{P_0}\right) = \sum_{i=1}^{20} r_i$

(r_i是每天的 收益率)

用类似的方法，可以由日对数回报率数据计算年对数回报率。

金融数据分析

- 从雅虎财经读取德国DAX指数股价信息，时间设定为2000年1月1日到2014年9月26日；
- 可以对此数据各个变量有个基本描述统计；
- 为了更好地概览指数的变化，绘制日收盘指数水平的时间序列图；
- 根据每天的收盘价，计算日收益率和日对数收益率，并绘图；

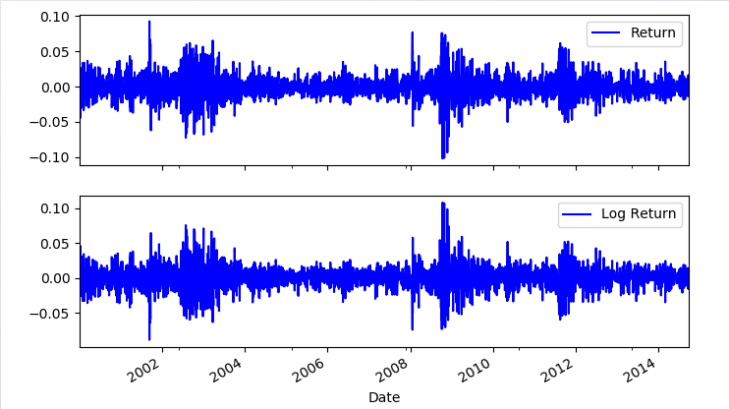
收益率的两种典型事实（stylized facts）

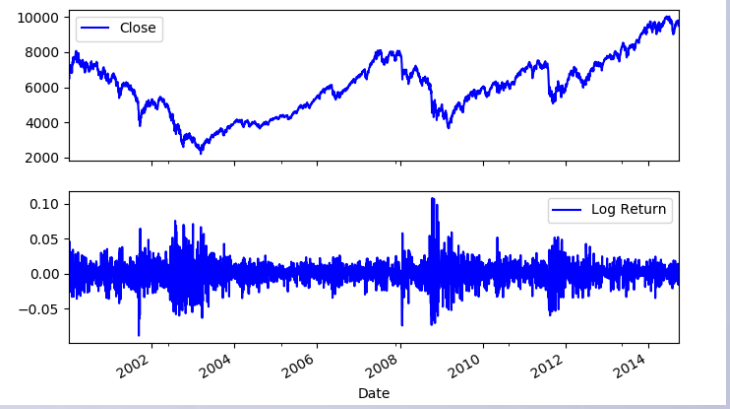
(1)波动率聚集(volatility clustering)

波动率不是恒定的；即有高波动率时期(正收益和负收益都很高)，也有低波动率时期

(2)杠杆效应(leverage effect)

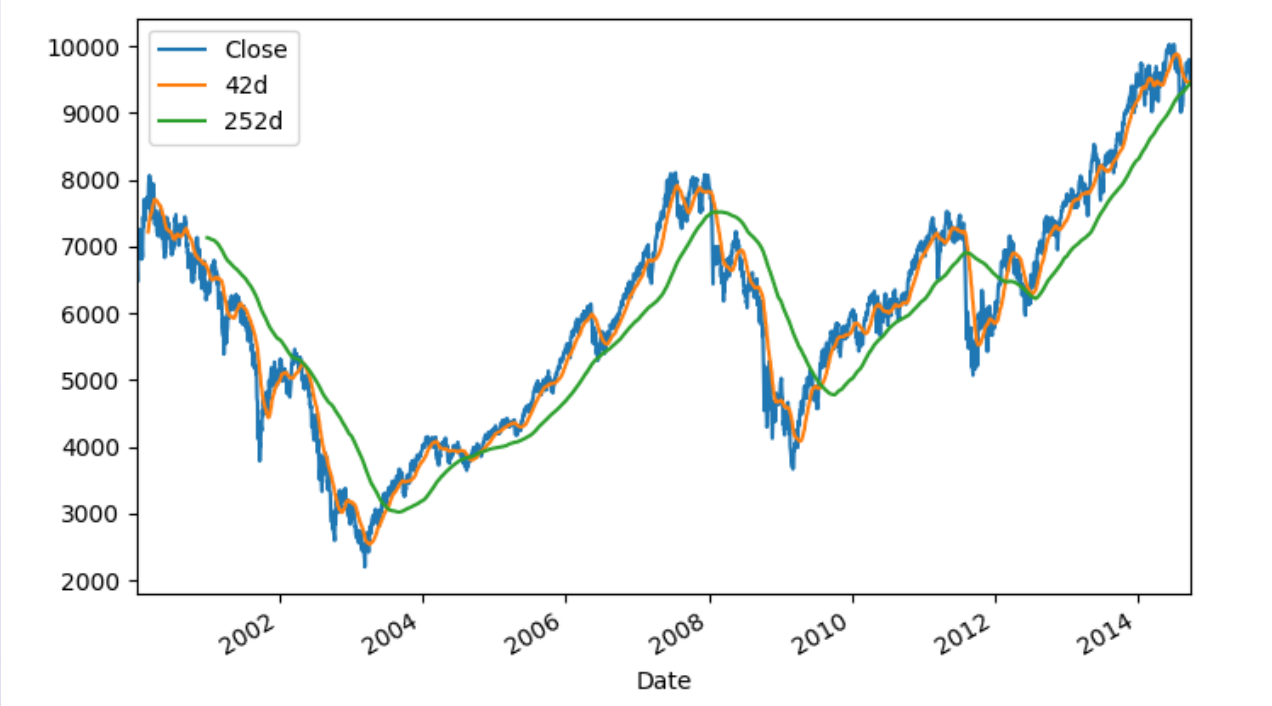
一般来说，波动率和股票市场收益是负相关的，当市场下跌时波动性升高。





金融数据分析

- 计算移动平均收盘价,股票交易者可能对移动平均值（所谓趋势）感兴趣，pd.rolling(42).mean()和pd.rolling(252).mean()分别计算42天的移动平均和252天的移动平均，并绘图；



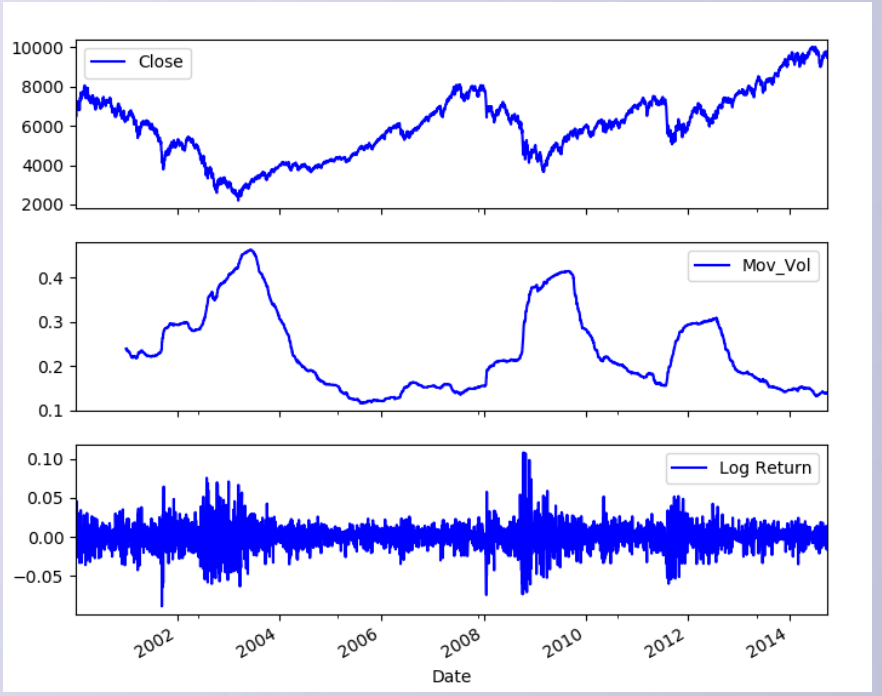
金融数据分析

- 波动率对期权交易者特别重要，计算对数收益率的移动历史标准差-即历史波动率（252天的移动标准差，计算出年化波动率）

- 进一步说明杠杆效应的存在，

市场下跌时历史移动波动率倾向于升高；

而在市场上涨时，波动率下降。



练习：估算市场风险系数(贝塔值)

- 根据CAPM模型，单只股票的收益率（回报率）和市场的回报率线性相关。通常情况下，考虑某只股票的超额回报率与市场的超额回报率之间的关系：

$$R_i - R_f = \alpha + \beta_i(R_m - R_f)$$

- 其中，R_i是该股票的回报率，β_i是斜率，贝塔值，是股票的相关风险的度量指标，反映一只股票在市场中价格上涨或下跌的趋势
- R_m是市场回报率，R_f是无风险利率。
- 从雅虎财经网站下载历史价格数据：股票IBM在1983年1月1日至2013年11月9日期间每日价格数据，并下载代表市场的标准普尔500指数（S&P500，代码^GSPC）数据。
- 利用statsmodels模块做回归，给出每年的贝塔值。

一、回报率数据统计分析

- 对日回报率数据的描述性统计分析
- 对月回报率数据的均值是否为零的检验
- ‘一月效应’的检验（即股票在一月份回报率不同于其他月份，检验方差是否相等）
- 练习：见实验实训四

二、Roll(1984)模型估算买卖价差

- 流动性描述资产或证券在不影响资产价格的情况下在市场上快速买入或卖出的程度。市场流动性是指一个市场（如一个国家的股票市场或一个城市的房地产市场）允许资产以稳定价格买卖的程度。
- 买卖价差(bid-ask spread)是衡量流动性的一个常用指标，理论来自于市场微观结构噪声；一般基于高频数据计算。
- Roll（1984）模型是市场微观结构噪声模型中最简单的情形；基于每天的观测，该模型给出一种方法，可以利用价格变化的自协方差来间接估计买卖价差：

$$S = 2\sqrt{-cov(\Delta P_t, \Delta P_{t-1})}\ , \quad \%\text{spread} = \frac{S}{\bar{P}}$$

- S为买卖价差，P_t为股票在第t天的收盘价，$\bar{P}$是在一段时间内每日收盘价的平均值。
- 明显的缺陷:股票价格变化的协方差可能存在正值，此时，该方法失效。
- 练习：见实验实训五

三、用Amihud(2002)模型估算反流动性指标

- Amihud(2002)认为流动性反映的是一系列订单对价格的影响。其反流动性指标定义如下：

$$illiq_t = \frac{|R_t|}{P_t \times V_t}$$

- R_t为第t天的日回报率，P_t为股票在第t天的收盘价，而V_t为股票在第t天的交易量。
- 反流动性是流动性的对立面，反流动性指标越低，流动性越高。
- 用2013年10月的交易数据估计IBM股票的Amihud反流动性指标，并与同一时期的WMT比较。
- 练习：见实验实训六

四、Pastor和Stambaugh(2003)流动性指标

根据Campbell,Grossman和Wang(1993)， Pastor和Stambaugh(2003)提出了如下模型来衡量个股的流动性和市场流动性：

$$y_t = \alpha + \beta_1 x_{1,t-1} + \beta_2 x_{2,t-1} + \epsilon_t$$

- 其中$y_t = R_t - R_{f,t}$：超额回报率；R_t是个股回报率， $R_{f,t}$是无风险利率；
- $x_{1,t}$是市场回报率；
- $x_{2,t}$=sign$(R_t - R_{f,t}) \times P_t \times V_t$，$P_t$为股票在第t天的收盘价，而$V_t$为股票在第t天的交易量。
- 模型基于一个月的日度数据，每个月可以得到一个β_2的估计值，P&S将其作为流动性指标。

五、Fama-French三因子模型

资本资产定价模型CAPM是一个单因子模型。F-F三因子模型把它扩展到3个因子。单只股票的回报率满足：

$$R_i = R_f + \beta_m(R_m - R_f) + \beta_{SMB} \times \text{SMB} + \beta_{HML} \times \text{HML} + \varepsilon_t$$

- R_i是个股的回报率，R_f是无风险收益率，R_m是市场收益率；
- SMB是小型股的投资组合收益率减去大型股投资组合的收益率；
- HML是高账面市值投资组合的收益率减去低账面市值股投资组合的收益率。

小结

- pandas和statsmodels模块
- 实际应用