利用python进行数据分析

与scikit-learn比较，statsmodels包含经典统计学和经济计量学的算法。包括如下子模块：

- 回归模型：线性回归，广义线性模型，健壮线性模型，线性混合效应模型等等。
- 方差分析（ANOVA）。
- 时间序列分析：AR，ARMA，ARIMA，VAR和其它模型。
- 非参数方法： 核密度估计，核回归。
- 统计模型结果可视化。

statsmodels更关注与统计推断，提供不确定估计和参数p-值。相反的，scikit-learn注重预测。

Chapter 2

当你使用 `%run` 命令，IPython会同样执行指定文件中的代码，结束之后，还可以与结果交互：

你可以用 `%run` 命令运行所有的Python程序。 结果和普通的运行方式 `python script.py` 相同。文件中所有定义的变量（import、函数和全局变量，除非抛出异常），都可以在IPython shell中随后访问。如果一个Python脚本需要命令行参数（在 `sys.argv` 中查找），可以在文件路径之后传递，就像在命令行上运行一样。

笔记：如果想让一个脚本访问IPython已经定义过的变量，可以使用 `%run -i` 。

在Jupyter notebook中，你也可以使用 `%load` ，它将脚本导入到一个代码格中：

键盘快捷键

快捷键	说明
Ctrl-P 或↑箭头	用当前输入的文本搜索之前的命令
Ctrl-N 或↓箭头	用当前输入的文本搜索之后的命令
Ctrl-R	Readline 方式翻转历史搜索（部分匹配）
Ctrl-Shift-V	从剪贴板粘贴文本
Ctrl-C	中断运行的代码
Ctrl-A	将光标移动到一行的开头
Ctrl-E	将光标移动到一行的末尾
Ctrl-K	删除光标到行尾的文本
Ctrl-U	删除当前行的所有文本
Ctrl-F	光标向后移动一个字符
Ctrl-B	光标向前移动一个字符
Ctrl-L	清空屏幕

魔术命令

魔术命令可以被看做IPython中运行的命令行 。

命令	说明
%quickref	显示 IPython 的快速参考。
%magic	显示所有魔术命令的详细文档。
%debug	在出现异常的语句进入调试模式。
%hist	打印命令的输入（可以选择输出）历史。
%pdb	出现异常时自动进入调试。
%paste	执行剪贴板中的代码。
%cpaste	开启特别提示，手动粘贴待执行代码。
%reset	删除所有命名空间中的变量和名字。
%page OBJECT	美化打印对象，分页显示。
%run script.py	运行代码。
%prun statement	用 CProfile 运行代码，并报告分析器输出。
%time statement	报告单条语句的执行时间。
%timeit statement	多次运行一条语句，计算平均执行时间。适合执行时间短的代码。
%who, %who_ls, %whos	显示命名空间中的变量，三者显示的信息级别不同。
%xdel variable	删除一个变量，并清空任何对它的引用。

Numpy

数组跟列表最重要的区别在于，数组切片是原始数组的视图。这意味着数据不会被复制，视图上的任何修改都会直接反映到源数组上。

```
In [66]: arr_slice = arr[5:8]

In [67]: arr_slice
Out[67]: array([12, 12, 12])

In [68]: arr_slice[1] = 12345

In [69]: arr
Out[69]: array([    0,     1,     2,     3,     4,    12, 12345,    12,     8,
         9])
```

由于NumPy的设计目的是处理大数据，所以你可以想象一下，假如NumPy坚持要将数据复制来复制去的话会产生何等的性能和内存问题。

注意：如果你想要得到的是ndarray切片的一份副本而非视图，就需要明确地进行复制操作，例如 `arr[5:8].copy()` 。

可以对各个元素进行递归访问，但这样需要做的事情有点多。你可以传入一个以逗号隔开的索引列表来选取单个元素。也就是说，下面两种方式是等价的：

```
In [74]: arr2d[0][2]
Out[74]: 3

In [75]: arr2d[0, 2]
Out[75]: 3
```

花式索引

为了以特定顺序选取行子集，只需传入一个用于指定顺序的整数列表或ndarray即可：

好用的zip函数：

```
In [165]: xarr = np.array([1.1, 1.2, 1.3, 1.4, 1.5])

In [166]: yarr = np.array([2.1, 2.2, 2.3, 2.4, 2.5])

In [167]: cond = np.array([True, False, True, True, False])

In [168]: result = [(x if c else y)
   .....:           for x, y, c in zip(xarr, yarr, cond)]

In [169]: result
Out[169]: [1.1000000000000001, 2.2000000000000002, 1.3,
1.3999999999999999, 2.5]
```

可以用np.where快速实现：

```
In [170]: result = np.where(cond, xarr, yarr)

In [171]: result
Out[171]: array([ 1.1,  2.2,  1.3,  1.4,  2.5])
```

```
bools.any()
bools.all()
```

通过np.savez可以将多个数组保存到一个未压缩文件中，将数组以关键字参数的形式传入即可：

```
In [216]: np.savez('array_archive.npz', a=arr, b=arr)
```

加载.npz文件时，你会得到一个类似字典的对象，该对象会对各个数组进行延迟加载：

```
In [217]: arch = np.load('array_archive.npz')

In [218]: arch['b']
Out[218]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

@符（类似Python 3.5）也可以用作中缀运算符，进行矩阵乘法：（这个很方便啊）

```
In [230]: x @ np.ones(3)
Out[230]: array([  6.,  15.])
```

numpy.linalg

表4-7：常用的numpy.linalg函数

函数	说明
diag	以一维数组的形式返回方阵的对角线（或非对角线）元素，或将一维数组转换为方阵（非对角线元素为0）
dot	矩阵乘法
trace	计算对角线元素的和
det	计算矩阵行列式
eig	计算方阵的本征值和本征向量
inv	计算方阵的逆
pinv	计算矩阵的Moore-Penrose伪逆
qr	计算QR分解
svd	计算奇异值分解（SVD）
solve	解线性方程组Ax = b，其中A为一个方阵
lstsq	计算Ax = b的最小二乘解

伪随机数生成

表4-8：部分numpy.random函数

函数	说明
seed	确定随机数生成器的种子
permutation	返回一个序列的随机排列或返回一个随机排列的范围
shuffle	对一个序列就地随机排列
rand	产生均匀分布的样本值
randint	从给定的上下限范围内随机选取整数
randn	产生正态分布（平均值为0，标准差为1）的样本值，类似于MATLAB接口
binomial	产生二项分布的样本值
normal	产生正态（高斯）分布的样本值
beta	产生Beta分布的样本值

表4-8：部分numpy.random函数（续）

函数	说明
chisquare	产生卡方分布的样本值
gamma	产生Gamma分布的样本值
uniform	产生在[0, 1)中均匀分布的样本值

Pandas

虽然pandas采用了大量的NumPy编码风格，但二者最大的不同是pandas是专门为处理表格和混杂数据设计的。而NumPy更适合处理统一的数值数组数据。

数据结构：Series

Series是一种类似于一维数组的对象，它由一组数据（各种NumPy数据类型）以及一组与之相关的数据标签（即索引）组成。仅由一组数据即可产生最简单的Series：

```
obj = pd.Series([4, 7, -5, 3])
obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])



obj2[['c', 'a', 'd']]
```

['c', 'a', 'd']是索引列表，即使它包含的是字符串而不是整数。

Series比较类似于字典——

```
In [26]: sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah':
5000}

In [27]: obj3 = pd.Series(sdata)

In [28]: obj3
```

DataFrame

DataFrame是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔值等）。

DataFrame既有行索引也有列索引，它可以被看做由Series组成的字典（共用同一个索引）。

DataFrame中的数据是以一个或多个二维块存放的（而不是列表、字典或别的一维数据结构）。

通过类似字典标记的方式或属性的方式，可以将DataFrame的列获取为一个Series

```
In [51]: frame2['state']
```

```
In [51]: frame2['state']
Out[51]:
one       Ohio
two       Ohio
three     Ohio
four     Nevada
five     Nevada
six      Nevada
Name: state, dtype: object


In [52]: frame2.year
Out[52]:
one      2000
two      2001
three    2002
four     2001
five     2002
six      2003
```

行也可以通过位置或名称的方式进行获取，比如用loc属性 。

如果嵌套字典传给DataFrame，pandas就会被解释为：外层字典的键作为列，内层键则作为行索引。

```
pop = {'Nevada': {2001: 2.4, 2002: 2.9},
....:       'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}

In [66]: frame3 = pd.DataFrame(pop)
```

```
In [67]: frame3
Out[67]:
     Nevada  Ohio
2000    NaN   1.5
2001    2.4   1.7
2002    2.9   3.6

In [68]: frame3.T
Out[68]:
      2000  2001  2002
Nevada   NaN   2.4   2.9
Ohio     1.5   1.7   3.6
```

构造DataFrame的方法：

表5-1：可以输入给DataFrame构造器的数据

类型	说明
二维ndarray	数据矩阵，还可以传入行标和列标
由数组、列表或元组组成的字典	每个序列会变成DataFrame的一列。所有序列的长度必须相同
NumPy的结构化/记录数组	类似于“由数组组成的字典”
由Series组成的字典	每个Series会成为一列。如果没有显式指定索引，则各Series的索引会被合并成结果的行索引
由字典组成的字典	各内层字典会成为一列。键会被合并成结果的行索引，跟“由Series组成的字典”的情况一样
字典或Series的列表	各项将会成为DataFrame的一行。字典键或Series索引的并集将会成为DataFrame的列标
由列表或元组组成的列表	类似于“二维ndarray”
另一个DataFrame	该DataFrame的索引将会被沿用，除非显式指定了其他索引
NumPy的MaskedArray	类似于“二维ndarray”的情况，只是掩码值在结果DataFrame会变成NA/缺失值

values属性也会以二维ndarray的形式返回DataFrame中的数据 。

pandas的索引对象负责管理轴标签和其他元数据（比如轴名称等）。构建Series或DataFrame时，所用到的任何数组或其他序列的标签都会被转换成一个Index 。除了类似于数组，Index的功能也类似一个固定大小的集合，但可以包含重复的标签。

Index对象是不可变的，因此用户不能对其进行修改 。

reindex

对于时间序列这样的有序数据，重新索引时可能需要做一些插值处理。method选项即可达到此目的，例如，使用ffill可以实现前向值填充 。

```
In [97]: obj3.reindex(range(6), method='ffill')
Out[97]:
0    blue
1    blue
2  purple
3  purple
4  yellow
5  yellow
```

索引

利用标签的切片运算与普通的Python切片运算不同，其末端是包含的 。

```
In [125]: obj['b':'c']
Out[125]:
b    1.0
c    2.0
dtype: float64
```

切片[:2] 默认选取行。

loc基于标签的索引，iloc基于整数的索引。

类型	说明
df[val]	从 DataFrame 选取单列或一组列；在特殊情况下比较便利：布尔型数组（过滤行）、切片（行切片）、或布尔型 DataFrame（根据条件设置值）
df.loc[val]	通过标签，选取 DataFrame 的单个行或一组行
df.loc[:, val]	通过标签，选取单列或列子集
df.loc[val1, val2]	通过标签，同时选取行和列
df.iloc[where]	通过整数位置，从 DataFrame 选取单个行或行子集
df.iloc[:, where]	通过整数位置，从 DataFrame 选取单个列或列子集
df.iloc[where_i, where_j]	通过整数位置，同时选取行和列
df.at[label_i, label_j]	通过行和列标签，选取单一的标量
df.iat[i, j]	通过行和列的位置（整数），选取单一的标量
reindex	通过标签选取行或列
get_value, set_value	通过行和列标签选取单一值

对于Series和DataFrame的索引，为了更准确，请使用loc（标签）或iloc（整数）。

默认情况下，DataFrame和Series之间的算术运算会将Series的索引**匹配到DataFrame的列**，然后沿着行一直向下广播。如果某个索引值在DataFrame的列或Series的索引中找不到，则参与运算的两个对象就会被重新索引以形成并集。

如果你希望匹配行且在列上广播，则必须使用算术运算方法。

```
frame.sub(series3, axis='index')
```

传入的轴号就是希望匹配的轴。 在本例中，我们的目的是匹配DataFrame的行索引（axis='index' or axis=0）并进行广播。

NumPy的ufuncs（元素级数组方法）也可用于操作pandas对象

表4-3：一元ufunc

函数	说明
abs、fabs	计算整数、浮点数或复数的绝对值。对于非复数值，可以使用更快的fabs
sqrt	计算各元素的平方根。相当于arr ** 0.5
square	计算各元素的平方。相当于arr ** 2
exp	计算各元素的指数e^x
log、log10、log2、log1p	分别为自然对数（底数为e）、底数为10的log、底数为2的log、log(1 + x)
sign	计算各元素的正负号：1（正数）、0（零）、－1（负数）
ceil	计算各元素的ceiling值，即大于等于该值的最小整数
floor	计算各元素的floor值，即小于等于该值的最大整数
rint	将各元素值四舍五入到最接近的整数，保留dtype
modf	将数组的小数和整数部分以两个独立数组的形式返回
isnan	返回一个表示“哪些值是NaN（这不是一个数字）”的布尔型数组
isfinite、isinf	分别返回一个表示“哪些元素是有穷的（非inf，非NaN）”或“哪些元素是无穷的”的布尔型数组
cos、cosh、sin、sinh、tan、tanh	普通型和双曲型三角函数

表4-3：一元ufunc（续）

函数	说明
arccos、arccosh、arcsin、arcsinh、arctan、arctanh	反三角函数
logical_not	计算各元素not x的真值。相当于-arr

表4-4：二元ufunc

函数	说明
add	将数组中对应的元素相加
subtract	从第一个数组中减去第二个数组中的元素
multiply	数组元素相乘
divide、floor_divide	除法或向下圆整除法（丢弃余数）
power	对第一个数组中的元素A，根据第二个数组中的相应元素B，计算A^B
maximum、fmax	元素级的最大值计算。fmax将忽略NaN
minimum、fmin	元素级的最小值计算。fmin将忽略NaN
mod	元素级的求模计算（除法的余数）
copysign	将第二个数组中的值的符号复制给第一个数组中的值
greater、greater_equal、less、less_equal、equal、not_equal	执行元素级的比较运算，最终产生布尔型数组。相当于中缀运算符>、>=、<、<=、==、!=
logical_and、logical_or、logical_xor	执行元素级的真值逻辑运算。相当于中缀运算符&、|、^

将函数应用到由各列或行所形成的一维数组上。DataFrame的apply方法即可实现此功能。

```
In [127]: f = lambda x: x.max() - x.min()
```

```
In [193]: f = lambda x: x.max() - x.min()

In [194]: frame.apply(f)
   #如果传递axis='columns'到apply，这个函数会在每行执行
Out[194]:
b    1.802165
d    1.684034
e    2.689627
```

元素级的Python函数也是可以用的，DataFrame中使用applymap即可，而Series有一个用于应用元素级函数的map方法。

要对行或列索引进行排序（按字典顺序），可使用**sort_index**方法，它将返回一个已排序的新对象。

```
frame.sort_index(axis=1, ascending=False)
# 列索引，降序
```

若要按值对Series进行排序，可使用其sort_values方法 。当排序一个DataFrame， 将一个或多个列的名字传递给sort_values的by参数即可按照该列排序

rank

rank是通过“为各组分配一个平均排名”的方式破坏平级关系的。

```
In [215]: obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
In [216]: obj.rank()
Out[216]:

0    6.5
```

```
0    6.5
1    1.0
2    6.5
3    4.5
4    3.0
5    2.0
6    4.5

# 也可以根据值在原数据中出现的顺序给出排名
In [217]: obj.rank(method='first')
Out[217]:
0    6.0
1    1.0
2    7.0
3    4.0
4    3.0
5    2.0
6    5.0
dtype: float64
```

Chapter 6 数据加载、存储与文件格式

函数	说明
read_csv	从文件、URL、文件型对象中加载带分隔符的数据。默认分隔符为逗号
read_table	从文件、URL、文件型对象中加载带分隔符的数据。默认分隔符为制表符('\t')
read_fwf	读取定宽列格式数据（也就是说，没有分隔符）
read_clipboard	读取剪贴板中的数据，可以看做 read_table 的剪贴板版。再将网页转换为表格时很有用
read_excel	从 Excel XLS 或 XLSX file 读取表格数据
read_hdf	读取 pandas 写的 HDF5 文件
read_html	读取 HTML 文档中的所有表格
read_json	读取 JSON (JavaScript Object Notation)字符串中的数据
read_msgpack	二进制格式编码的 pandas 数据
read_pickle	读取 Python pickle 格式中存储的任意对象
read_sas	读取存储于 SAS 系统自定义存储格式的 SAS 数据集
read_sql	（使用 SQLAlchemy）读取 SQL 查询结果为 pandas 的 DataFrame
read_stata	读取 Stata 文件格式的数据集
read_feather	读取 Feather 二进制文件格式

如果希望将多个列做成一个层次化索引，只需传入由列编号或列名组成的列表即可：

```
parsed = pd.read_csv('examples/csv_mindex.csv',index_col=['key1', 'key2'])
```

用正则表达式表示分隔符：

```
result = pd.read_table('examples/ex3.txt', sep='\s+')
```

可以调节pandas的显示：

```
pd.options.display.max_rows = 10
```

利用DataFrame的to_csv方法，我们可以将数据写到文件中：

```
data.to_csv(sys.stdout, sep='|')
```

二进制数据格式

实现数据的高效二进制格式存储最简单的办法之一是使用Python内置的pickle序列化。

```
In [87]: frame = pd.read_csv('examples/ex1.csv')
In [88]: frame
Out[88]:
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo

In [89]: frame.to_pickle('examples/frame_pickle')
# 你可以通过pickle直接读取被pickle化的数据，或是使用更为方便pandas.read_pickle
In [90]: pd.read_pickle('examples/frame_pickle')
Out[90]:
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

Chapter 7 数据清洗和准备

处理缺失数据

方法	说明
dropna	根据各标签的值中是否存在缺失数据对轴标签进行过滤，可通过阈值调节对缺失值的容忍度
fillna	用指定值或插值方法（如ffill或bfill）填充缺失数据
isnull	返回一个含有布尔值的对象，这些布尔值表示哪些值是缺失值/NA，该对象的类型与源类型一样
notnull	isnull的否定式

对于一个Series，dropna返回一个仅含非空数据和索引值的Series.

而对于DataFrame对象,dropna默认丢弃任何含有缺失值的行.传入how='all'将只丢弃全为NA的那些行.

```
data.dropna(how='all')
data.dropna(axis=1, how='all')

df.dropna(thresh=2) # 保留含Na小于2的行
```

数据转换

map()函数

DataFrame实现了data['x'].map(function()来将函数作用在每一个元素上的功能。

raplace()函数

```
data.replace({-999: np.nan, -1000: 0})
```

跟Series一样，轴索引也有一个map方法。

如果想要创建数据集的转换版（而不是修改原始数据），比较实用的方法是rename。

数据离散化

分bin

```
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]



cats = pd.cut(ages, bins)
```

pandas返回的是一个特殊的Categorical对象。结果展示了pandas.cut划分的bin

```
# 等长bin
pd.cut(data, 4, precision=2)

# 等元素bin
cats = pd.qcut(data, 4)
```

排列和随即采样

利用numpy.random.permutation函数可以轻松实现对Series或DataFrame的列的排列工作。通过需要排列的轴的长度调用permutation，可产生一个表示新顺序的整数数组：

```
In [100]: df = pd.DataFrame(np.arange(5 * 4).reshape((5, 4)))

In [101]: sampler = np.random.permutation(5)

In [102]: sampler

df.take(sampler)

df.iloc(sampler)
```

如果不想用替换的方式选取随机子集，可以在Series和DataFrame上使用sample方法。

要通过替换的方式产生样本（允许重复选择），可以传递replace=True到sample （**应该是有放回的抽样**）

如果DataFrame的某一列中含有k个不同的值，则get_dummies函数可以派生出一个k列矩阵或DataFrame（其值全为1和0）。

```
dummies = pd.get_dummies(df['key'], prefix='key')
```

字符串操作

以逗号分隔的字符串可以用split拆分成数段。

```
```

```
In [134]: val = 'a,b,  guido'
In [135]: val.split(',')
Out[135]: ['a', 'b', '  guido']
```

split常常与strip一起使用，以去除空白符（包括换行符）

```
In [136]: pieces = [x.strip() for x in val.split(',')]

In [137]: pieces
Out[137]: ['a', 'b', 'guido']
# 向字符串"::"的join方法传入一个列表或元组
In [140]: '::'.join(pieces)
Out[140]: 'a::b::guido'
```

replace用于将指定模式替换为另一个模式。通过传入空字符串，它也常常用于删除模式：

模式匹配、替换以及拆分 。match(findall,match只匹配开头)、replace、split

Chapter 8 数据规整：聚合、合并和重塑

层次化索引

层次化索引在数据重塑和基于分组的操作（如透视表生成）中扮演着重要的角色。例如，可以通过unstack方法将这段数据重新安排到一个DataFrame中：

```
In [16]: data.unstack()
# 其逆运算为stack()
```

对于一个DataFrame，每条轴都可以有分层索引,每一层都可以有名字，用names设3置。

重排与分级排序

swaplevel接受两个级别编号或名称，并返回一个互换了级别的新对象。而sort_index则根据单个级别中的值对数据进行排序。交换级别时，常常也会用到sort_index，这样最终结果就是按照指定顺序进行字母排序了。

根据级别汇总排序

许多对DataFrame和Series的描述和汇总统计都有一个level选项，它用于指定在某条轴上求和的级别。

```
In [27]: frame.sum(level='key2')
Out[27]:
state  Ohio     Colorado
color Green Red    Green
key2
1       6   8      10
2      12  14      16
```

DataFrame的set_index函数会将其一个或多个列转换为行索引，并创建一个新的DataFrame，即把列索引变为行索引。

reset_index的功能跟set_index刚好相反，层次化索引的级别会被转移到列里面。

合并数据集

pandas.merge可根据一个或多个键将不同DataFrame中的行连接起来。 如果没有指定，merge就会将重叠列的列名当做键。 不过，最好明确指定一下：

```
pd.merge(df1, df2, on='key')
```

如果两个对象的列名不同，也可以分别进行指定

```
pd.merge(df3, df4, left_on='lkey', right_on='rkey')

# 默认内连接，手选外连接
pd.merge(df1, df2, how='outer')
```

DataFrame还有一个便捷的join实例方法，它能更为方便地实现按索引合并。它还可用于合并多个带有相同或相似索引的DataFrame对象，但要求没有重叠的列。

轴向连接

对对象调用concat可以将值和索引粘合在一起

```
pd.concat([s1, s2, s3])
```

默认情况下，concat是在axis=0上工作的，最终产生一个新的Series。如果传入axis=1，则结果就会变成一个DataFrame。传入join='inner'即可得到它们的交集。

合并重叠数据

对于DataFrame，combine_first用传递对象中的数据为调用对象的缺失数据“打补丁 ”。

重塑(reshape)和轴向旋转(pivot)

```
In [121]: data
Out[121]:
number    one  two  three
state
Ohio       0    1     2
Colorado   3    4     5
```

对该数据使用stack方法即可将列转换为行，得到一个Series.

```
In [122]: result = data.stack()
In [123]: result
Out[123]:
state    number
Ohio     one      0
         two      1
         three    2
Colorado  one     3
         two      4
         three    5
```

对于一个层次化索引的Series，你可以用unstack将其重排为一个DataFrame.传入分层级别的编号或名称即可对其它级别进行unstack操作.

将“长格式”旋转为“宽格式”

前两个传递的值分别用作行和列索引，最后一个可选值则是用于填充DataFrame的数据列。

```
pivoted = ldata.pivot('date', 'item', 'value')
```

旋转DataFrame的逆运算是pandas.melt。它不是将一列转换到多个新的DataFrame，而是合并多个列成为一个，产生一个比输入长的DataFrame。

Chapter 9 绘图与可视化

通常引入约定是

```
import matplotlib.pyplot as plt
```

matplotlib的图像都位于Figure对象中。你可以用plt.figure创建一个新的Figure。

不能通过空Figure绘图。必须用add_subplot创建一个或多个subplot才行

```
ax1 = fig.add_subplot(2, 2, 1)
#这条代码的意思是：图像应该是2×2的（即最多4张图），且当前选中的是4个subplot中的第一个（编号从1开始）
```

如果这时执行一条绘图命令，matplotlib就会在最后一个用过的subplot（如果没有则创建一个）上进行绘制。上面那些由fig.add_subplot所返回的对象是AxesSubplot对象，直接调用它们的实例方法就可以在其它空着的格子里面画图了 。

```
_ = ax1.hist(np.random.randn(100), bins=20, color='k', alpha=0.3)

ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
```

创建包含subplot网格的figure是一个非常常见的任务，matplotlib有一个更为方便的方法plt.subplots，它可以创建一个新的Figure，并返回一个含有已创建的subplot对象的NumPy数组。

```
fig, axes = plt.subplots(2, 3)
```

你必须调用plt.legend（或使用ax.legend，如果引用了轴的话）来创建图例，无论你绘图时是否传递label标签选项。

刻度、标签和图例

pyplot接口的设计目的就是交互式使用，含有诸如xlim、xticks和xticklabels之类的方法。它们分别控制图表的范围、刻度位置、刻度标签等。其使用方式有以下两种：

- 调用时不带参数，则返回当前的参数值
- 调用时带参数，则设置参数值

所有这些方法都是对当前或最近创建的AxesSubplot起作用的。

轴的类有集合方法，可以批量设定绘图选项。

```
props = {
   'title': 'My first matplotlib plot',
   'xlabel': 'Stages'
}
ax.set(**props)
```

matplotlib有一些表示常见图形的对象。这些对象被称为块（patch）。

要在图表中添加一个图形，你需要创建一个块对象shp，然后通过ax.add_patch(shp)将其添加到subplot中。

```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

rect = plt.Rectangle((0.2, 0.75), 0.4, 0.15, color='k', alpha=0.3)
circ = plt.Circle((0.7, 0.2), 0.15, color='b', alpha=0.3)
pgon = plt.Polygon([[0.15, 0.15], [0.35, 0.4], [0.2, 0.6]],
                   color='g', alpha=0.5)

ax.add_patch(rect)
ax.add_patch(circ)
ax.add_patch(pgon)
```

利用plt.savefig可以将当前图表保存到文件。

matplotlib配置

一种Python编程方式配置系统的方法是使用rc方法。 例如，要将全局的图像默认大小设置为10×10，你可以执行

```
plt.rc('figure', figsize=(10, 10))
```

rc的第一个参数是希望自定义的对象，如 'figure' 、'axes' 、'xtick' 、'ytick' 、'grid' 、'legend' 等。其后可以跟上一系列的关键字参数。一个简单的办法是将这些选项写成一个字典：

```
font_options = {'family' : 'monospace',
                'weight' : 'bold',
                'size'   : 'small'}
plt.rc('font', **font_options)
```

matplotlib实际上是一种比较低级的工具。要绘制一张图表，你得组装一些基本组件

使用pandas和seaborn

引入seaborn会修改matplotlib默认的颜色方案和绘图类型，以提高可读性和美观度。

线形图

Series和DataFrame都有一个用于生成各类图表的plot方法。默认情况下，它们所生成的是线型图。

柱状图

plot.bar()和plot.barh()分别绘制水平和垂直的柱状图。设置stacked=True即可为DataFrame生成堆积柱状图。

直方图和密度图

seaborn的distplot方法绘制直方图和密度图更加简单，还可以同时画出直方图和连续密度估计图。

```
In [96]: comp1 = np.random.normal(0, 1, size=200)

In [97]: comp2 = np.random.normal(10, 2, size=200)

In [98]: values = pd.Series(np.concatenate([comp1, comp2]))

In [99]: sns.distplot(values, bins=100, color='k')
```

散布图或点图

可以使用seaborn的regplot方法，它可以做一个散布图，并加上一条线性回归的线。

在探索式数据分析工作中，同时观察一组变量的散布图是很有意义的，这也被称为散布图矩阵（scatter plot matrix）。纯手工创建这样的图表很费工夫，所以seaborn提供了一个便捷的pairplot函数，它支持在对角线上放置每个变量的直方图或密度估计。

```
sns.pairplot(trans_data, diag_kind='kde', plot_kws={'alpha': 0.2})
```

分面网格和类型数据

seaborn有一个有用的内置函数factorplot，可以简化制作多种分面图。

```
sns.factorplot(x='day', y='tip_pct', hue='time', col='smoker',kind='bar',
data=tips[tips.tip_pct < 1])
```

factorplot支持其它的绘图类型。例如，盒图（它可以显示中位数，四分位数，和异常值）就是一个有用的可视化类型。（kind = "box"）

对于创建用于打印或网页的静态图形，建议默认使用matplotlib和附加的库，比如pandas和seaborn。

Chapter 10 数据聚合与分组运算

GroupBy机制

```
In [10]: df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
   ....:                    'key2' : ['one', 'two', 'one', 'two', 'one'],
   ....:                    'data1' : np.random.randn(5),
   ....:                    'data2' : np.random.randn(5)})

grouped = df['data1'].groupby(df['key1'])
# 将data1这一列数据按照key1聚合
```

size()返回剔除缺失值后的聚合分组大小。

你可以对这些数据片段做任何操作。有一个你可能会觉得有用的运算：将这些数据片段做成一个字典

```
pieces = dict(list(df.groupby('key1')))
```

groupby默认是在axis=0上进行分组的，通过设置也可以在其他任何轴上进行分组。

```
                                                    
```

```
grouped = df.groupby(df.dtypes, axis=1)

for dtype, group in grouped:
  ....:     print(dtype)
  ....:     print(group)
```

对于由DataFrame产生的GroupBy对象，如果用一个（单个字符串）或一组（字符串数组）列名对其进行索引，就能实现选取部分列进行聚合的目的。

```
df.groupby(['key1', 'key2'])[['data2']].mean()
```

可以将这个字典传给groupby，来构造数组，但我们可以直接传递字典。

```
by_column = people.groupby(mapping, axis=1)
# 可以实现本来不是一类的通过字典认为规定应该聚为一类的功能。
```

Series也有同样的功能，它可以被看做一个固定大小的映射。

```
map_series = pd.Series(mapping)
people.groupby(map_series, axis=1).count()
```

比起使用字典或Series，使用Python函数是一种更原生的方法定义分组映射。任何被当做分组键的函数都会在各个索引值上被调用一次，其返回值就会被用作分组名称。

要根据级别分组，使用level关键字传递级别序号或名字 。

数据聚合

如果要使用你自己的聚合函数，只需将其传入aggregate或agg方法即可。

```
def peak_to_peak(arr):
    return arr.max() - arr.min()
grouped.agg(peak_to_peak)
```

如果传入的是一个由(name,function)元组组成的列表，则各元组的第一个元素就会被用作DataFrame的列名。

```
# 自定义列名
grouped_pct.agg([('foo', 'mean'), ('bar', np.std)])



#应用于多列
ftuples = [('Durchschnitt', 'mean'),('Abweichung', np.var)]
grouped['tip_pct', 'total_bill'].agg(ftuples)
# 不同列应用不同种函数
grouped.agg({'tip' : np.max, 'size' : 'sum'})
```

apply：一般性的“拆分 - 应用 - 合并”

首先，编写一个选取指定列具有最大值的行的函数。

```
```

```
def top(df, n=5, column='tip_pct'):
  ....:     return df.sort_values(by=column)[-n:]
# 对smoker分组并用该函数调用apply
tips.groupby('smoker').apply(top)

# 这里，top函数在DataFrame的各个片段上调用，然后结果由pandas.concat组装到一起，并以分组名称进行了标记。
```

由cut返回的Categorical对象可直接传递到groupby。

实例

```
#先分组，对组内的缺失值用组的均值填充
fill_mean = lambda g: g.fillna(g.mean())
data.groupby(group_key).apply(fill_mean)
```

总之groupby()里面可以传一个匿名函数，后面apply也可以传一个匿名函数，非常powerful.

透视表和交叉表

透视表：根据一个或多个键对数据进行聚合，并根据行和列上的分组键将数据分配到各个矩形区域中。

```
tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
  .....:                columns='smoker')
# 还可以对这个表作进一步的处理，传入margins=True添加分项小计。

# 要使用其他的聚合函数，将其传给aggfunc即可。
tips.pivot_table('tip_pct', index=['time', 'smoker'],
columns='day',aggfunc=len, margins=True)

# crosstab也可以，比pivot_table更方便一点。
pd.crosstab(data.Nationality, data.Handedness, margins=True)
```

时间序列

时间序列的常见类型：

- 时间戳（timestamp），特定的时刻。
- 固定时期（period），如2007年1月或2010年全年。
- 时间间隔（interval），由起始和结束时间戳表示。时期（period）可以被看做间隔（interval）的特例。
- 实验或过程时间，每个时间点都是相对于特定起始时间的一个度量。其索引可能是一个整数或浮点数（表示从实验开始算起已经过去的时间）（timedeltas的指数，它可以有效代表实验或经过的时间。）

最简单也最常见的时间序列都是用时间戳进行索引的。

pandas

pandas最基本的时间序列类型就是以时间戳为索引的Series。

索引、选取和子集构造

当你根据标签索引选取数据时，时间序列和其它的pandas.Series很像：

```
stamp = ts.index[2]
ts[stamp]

ts['1/10/2011']

# 对于较长的时间序列，只需传入“年”或“年月”即可轻松选取数据的切片
# 选取2001年的
longer_ts['2001']
# 选取2001年5月的
longer_ts['2001-05']
# datatime对象也可以用于切片
ts[datetime(2011, 1, 7):]
# 由于大部分时间序列数据都是按照时间先后排序的，因此你也可以用不存在于该时间序列中的时间戳对其进行切片，即范围查询
```

非唯一时间戳

假设你想要对具有非唯一时间戳的数据进行聚合。一个办法是使用groupby，并传入level=0。

```
grouped = dup_ts.groupby(level=0)
grouped.mean()
grouped.count()
```

日期的范围、频率以及移动

```
# 生成
index = pd.date_range('2012-04-01', '2012-06-01')
```

默认情况下，date_range会产生按天计算的时间点。如果只传入起始或结束日期，那就还得传入一个表示一段时间的数字。

```
pd.date_range(start='2012-04-01', periods=20)
# freq参数控制频率
```

别名	偏移量类型	说明
D	Day	每日历日
B	BusinessDay	每工作日
H	Hour	每小时
T或min	Minute	每分
S	Second	每秒
L或ms	Milli	每毫秒（即每千分之一秒）
U	Micro	每微秒（即每百万分之一秒）
M	MonthEnd	每月最后一个日历日
BM	BusinessMonthEnd	每月最后一个工作日
MS	MonthBegin	每月第一个日历日

pandas中的频率是由一个基础频率（base frequency）和一个乘数组成的。

```
four_hours = Hour(4)
```

一般来说，无需明确创建这样的对象，只需使用诸如"H"或"4H"这样的字符串别名即可。偏移量对象可通过加法进行连接

```
pd.date_range('2000-01-01', '2000-01-03 23:59', freq='4h')
# 同理，你也可以传入频率字符串（如"2h30min"）
```

移动（shifting）指的是沿着时间轴将**数据前移或后移。**Series和DataFrame都有一个shift方法用于执行单纯的前移或后移操作，**保持索引不变**

```
ts.shift(2)
# 计算一个时间序列或多个时间序列
ts / ts.shift(1) - 1
# 如果频率已知，则可以将其传给shift以便实现对时间戳进行位移而不是对数据进行简单位移
ts.shift(2, freq='M')
```

锚点偏移量，第一次增量会将原日期向前滚动到符合频率规则的下一个日期。

```
now + MonthEnd()
# 通过锚点偏移量的rollforward和rollback方法，可明确地将日期向前或向后“滚动”
offset.rollforward(now)

offset.rollback(now)
```

时期和算术运算

时期（period）表示的是时间区间，比如数日、数月、数季、数年等。Period类所表示的就是这种数据类型，其构造函数需要用到一个字符串或整数以及频率

```
p = Period(2007, freq='A-DEC')
```

```
p = pd.Period(2007, freq='A-DEC')
```

period_range函数可用于创建规则的时期范围

```
rng = pd.period_range('2000-01-01', '2000-06-30', freq='M')
```

asfreq可以将period对象转换为别的频率，比如，希望将其转换为当年年初或年末的一个月度时期。

```
p.asfreq('M', how='start')
```

通过使用to_period方法，可以将由时间戳索引的Series和DataFrame对象转换为以时期索引。

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

通过数组创建periodindex

固定频率的数据集通常会将时间信息分开存放在多个列中。通过通过将这些数组以及一个频率传入PeriodIndex，就可以将它们合并成DataFrame的一个索引。

```
index = pd.PeriodIndex(year=data.year, quarter=data.quarter， freq='Q-DEC')
```

重采样及频率转换

重采样（resampling）指的是将时间序列从一个频率转换到另一个频率的处理过程。将高频率数据聚合到低频率称为降采样（downsampling），而将低频率数据转换到高频率则称为升采样（upsampling）。

pandas对象都带有一个resample方法，它是各种频率转换工作的主力函数。resample有一个类似于groupby的API，调用resample可以分组数据，然后会调用一个聚合函数。

```
# 每月最后一天
ts.resample('M').mean()
2000-01-31  -0.165893
2000-02-29   0.078606
2000-03-31   0.223811
2000-04-30  -0.063643
# 每月period
ts.resample('M', kind='period').mean()
```

降采样——聚合

```
ts.resample('5min', closed='right', label='right').sum()
# 区间右侧闭合，名称以右侧为准
```

从右边界减去一秒以便更容易明白该时间戳到底表示的是哪个区间，只需通过loffset设置一个字符串或日期偏移量即可实现。

传入how='ohlc'即可得到一个含有这四种聚合值的DataFrame，金融常用

```
```

```
In [220]: ts.resample('5min').ohlc()
Out[220]:
                     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
```

升采样——插值

```
frame.resample('D').ffill(limit=2)
```

移动窗口函数

在移动窗口（可以带有指数衰减权数）上计算的各种统计函数也是一类常见于时间序列的数组变换。这样可以圆滑噪音数据或断裂数据。

rolling函数也可以接受一个指定固定大小时间补偿字符串，而不是一组时期。这样可以方便处理不规律的时间序列。这些字符串也可以传递给resample。

指数加权函数

```
ewma60 = aapl_px.ewm(span=30).mean()
```

apply函数使你能够在移动窗口上应用自己设计的数组函数

```
result = returns.AAPL.rolling(250).apply(score_at_2percent)
```

分类

pandas有一个特殊的分类类型，用于保存使用整数分类表示法的数据。说白了就是某些具体的数据用0,1...代替，并且用一个字典记录下来。

pandas.Categorical实例

```
fruit_cat = df['fruit'].astype('category')
```

分类操作使得groupby操作明显加快，尤其是大数据集。

分类的特殊使用方法

特别的cat属性提供了分类方法的入口：

```
cat_s.cat.codes
cat_s.cat.categories

# 假设我们知道这个数据的实际分类集，超出了数据中的四个值。我们可以使用set_categories方法改变它们

cat_s2 = cat_s.cat.set_categories(actual_categories)
```

与apply类似，transform的函数会返回Series，但是结果必须与输入大小相同。

分组的时间重采样

要对每个key值进行相同的重采样，我们引入pandas.TimeGrouper对象。

```
time_key = pd.TimeGrouper('5min')
resampled = (df2.set_index('time')
  ....:              .groupby(['key', time_key])
  ....:              .sum())
                       value
key time
a   2017-05-20 00:00:00   30.0
    2017-05-20 00:05:00  105.0
    2017-05-20 00:10:00  180.0
b   2017-05-20 00:00:00   35.0
    2017-05-20 00:05:00  110.0
    2017-05-20 00:10:00  185.0
c   2017-05-20 00:00:00   40.0
    2017-05-20 00:05:00  115.0
    2017-05-20 00:10:00  190.0
```

使用TimeGrouper的限制是时间必须是Series或DataFrame的索引。

链式编程技术

DataFrame.assign方法是一个df[k] = v形式的函数式的列分配方法。它不是就地修改对象，而是返回新的修改过的DataFrame。下面语句等价：

```
# Usual non-functional way
df2 = df.copy()
df2['k'] = v

# Functional assign way
df2 = df.assign(k=v)



# 链式编程技术，其中外括号是为了更方便添加换行符
result = (df2.assign(col1_demeaned=df2.col1 - df2.col2.mean())
          .groupby('key')
          .col1_demeaned.std())
```

使用链式编程时要注意，你可能会需要涉及临时对象。在前面的例子中，我们不能使用load_data的结果，直到它被赋值给临时变量df。为了这么做，assign和许多其它pandas函数可以接收类似函数的参数，即可调用对象（callable）。

```
# 传递到[]的函数被绑定到了对象
df = (load_data()
      [lambda x: x['col2'] < 0])
```

管道方法

有如下函数调用。

```
a = f(df, arg1=v1)
b = g(a, v2, arg3=v3)
c = h(b, arg4=v4)
```

当使用接收、返回Series或DataFrame对象的函数式，你可以调用pipe将其重写：

```
result = (df.pipe(f, arg1=v1)
          .pipe(g, v2, arg3=v3)
          .pipe(h, arg4=v4))
```

f(df)和df.pipe(f)是等价的，但是pipe使得链式声明更容易。

python建模库介绍

pandas与其它分析库通常是靠NumPy的数组联系起来的。将DataFrame转换为NumPy数组，可以使用.values属性。

```
data.values
```

对于一些模型，你可能只想使用列的子集。我建议你使用loc，用values作索引

```
model_cols = ['x0', 'x1']
data.loc[:, model_cols].values
```

patsy

Patsy是Python的一个库，使用简短的字符串“公式语法”描述统计模型（尤其是线性模型），可能是受到了R和S统计编程语言的公式语法的启发。

就是类似于 `y=x1+x2` 这样的。

patsy.build_design_matrices函数可以使用原始样本数据集的保存信息，来转换新数据。

```
new_X = patsy.build_design_matrices([X.design_info], new_data)
```

当你在模型中使用多个分类名，事情就会变复杂，因为会包括key1:key2形式的相交部分，它可以用在方差（ANOVA）模型分析中。

Statsmodels

statsmodels是Python进行拟合多种统计模型、进行统计试验和数据探索可视化的库。Statsmodels包含许多经典的统计方法，但没有贝叶斯方法和机器学习模型。

两种接口：基于数组，和基于公式

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

sm.add_constant函数可以添加一个截距的列到现存的矩阵：

```
X_model = sm.add_constant(X)

model = sm.OLS(y, X)
results = model.fit()
results.params

# 对结果使用summary方法可以打印模型的详细诊断结果（很像R）
print(results.summary())
```

公式API

```
results = smf.ols('y ~ col0 + col1 + col2', data=data).fit()
```

时间序列分析

statsmodels的另一模型类是进行时间序列分析，包括自回归过程、卡尔曼滤波和其它态空间模型，和多元自回归模型。

```
In [82]: MAXLAGS = 5

In [83]: model = sm.tsa.AR(values)

In [84]: results = model.fit(MAXLAGS)
```

数据分析案例

numpy高级功能

reshape

作为参数的形状的其中一维可以是 - 1，它表示该维度的大小由数据本身推断而来

```
arr.reshape((5, -1))
```

与reshape将一维数组转换为多维数组的运算过程相反的运算通常称为扁平化（flattening）或散开（raveling）

```
arr.ravel()
```

concatenate

numpy.concatenate可以按指定轴将一个由数组组成的序列（如元组、列表等）连接到一起。对于常见的连接操作，NumPy提供了一些比较方便的方法（如vstack和hstack）。 分别是行数增加和列数增加。

split()进行拆分。

r_ 和c_，相当于vstack和hstack。

repeat

对数组进行重复以产生更大数组的工具主要是repeat和tile这两个函数。

```
arr.repeat([2, 3], axis=1)
```

tile的功能是沿指定轴向堆叠数组的副本，可以形象地将其想象成“铺瓷砖”。第二个参数是瓷砖的数量。对于标量，瓷砖是水平铺设的，而不是垂直铺设。它可以是一个表示“铺设”布局的元组。

```
In [70]: arr.take(inds)
Out[70]: array([700, 100, 200, 600])

In [71]: arr.put(inds, 42)

In [72]: arr
Out[72]: array([  0,  42,  42, 300, 400, 500,  42,  42,800, 900])

In [73]: arr.put(inds, [40, 41, 42, 43])

In [74]: arr
Out[74]: array([  0,  41,  42, 300, 400, 500,  43,  40, 800, 900])
```

用Numba快速编写NumPy函数

写一个普通函数，然后

```
import numba as nb
# 添加装饰器
@nb.jit
```