Python for Data Analysis

Author : Yunqiu Xu

`DataAnalysis` `Python` `numpy` `pandas` `matplotlib`

- References:
  - Book: Python For Data Analysis
  - python数据分析入门
  - 利用python进行数据分析 FDU
  - numpy 官方文档
  - pandas 官方文档
  - scipy 官方文档
- numpy,pandas,matplotlib,scipy

Chapter 1 Introduction

- Import the data;CSV
- Data transformation;
- Statistical description;
- Hypothesis test (t test);
- Visualization;
- Function definition;

1.1 导入数据

```
open(file) #内置函数open()

import pandas as pd
df=pd.read_table()

unames=['uid','gender','age','occupation','zip']
users=pd.read_table('users.dat',sep='::',header=None,names=unames)
df=pd.read_csv() #csv
df=pd.DataFrame() #将已经打开的txt转化为数据框架形式
```

1.2 数据转换

1.2.1 基本转换方式

```
df.head()/df.tail() #这个类似R，只不过R给出6行，Python给出5行
df.columns/df.index #相当于R里的colnames/rownames
df.T #等价于numpy.transpose
df.ix('colname') #提取特定的某列数据，注意python索引从0开始
    df.ix[:,0].head() #提取第1列前5行
    df.ix[10:20,0:3] #11-20行[10,19)的前三列数据[0,3)
df.drop(df.columns[[1,2]],axis=1)#舍弃前两列，axis=0为舍弃行
```

1.2.2 透视表 `pandas+numpy`

```
pd.pivot_table(df,
        index=["Name","Rep"], #设置行名
        values=["price"], #设置列名，未指定则默认其余所有内容为列值
        aggfunc=[numpy.sum], #函数，未指定则默认展示出各列列值
        columns=['Product'], #也是设置列名，但aggfunc只会作用于value
        fill_value=0, #将缺失值填充为0
        margins=True, #汇总数据，默认为False
        )
```

1.3 统计描述

```
df.describe() #给出count/mean/std/min/25%/50%/75%/max
```

1.4 假设检验

```
from scipy import stats as ss
```

- 举个栗子：检验Abra列均值，假定总产量均值为15000

```
ss.ttest_1samp(a = df.ix[:, 'Abra'], popmean = 15000)
>>>print后返回tuple，包括t,p,通过p即可验证假设是否成立
```

1.5 可视化

1.5.1 .plot() in pandas

```
table.plot(title='Sum of table1k.prop by year & sex',
        yticks=np.linspace(0, 1.2, 13),
        xticks= range(1880, 2020, 10))
```

1.5.2 matplotlib

```
import matplotlib.pyplot as plt
fig=plt.figure #给出画布
ax=plt.add_subplot(111)#将画布分成1行1列，获取从上到下，从左到右第1块小画布
add_subplot(3,4,6) #将画布分成3行4列要在第6块画布作画
ax.plot(dfx,dfy,kind='box')
plt.show
```

- flatten: 数据扁平化(只可用于array/matrix,list不可用)

```
a = array([[1,3],[2,4],[3,5]])
a.flatten#[1,3,2,4,3,5]
```

Chapter 2 数据分析实例：1880-2010 年间 全美婴儿姓名的趋势

2.1 导入并处理1880年的数据

- 该数据集可分为三列：名字;性别;每个名字的婴儿数量

```
import pandas as pd
path='C:/xyq data/py projects/pydata/names/yob1880.txt'
names1880 = pd.read_csv(path, names= ['name', 'sex', 'births'])
names1880.groupby('sex').births.sum() #统计当年出生的男女数量(打乱名字只按性别分配，同时累加第三列)
```

2.2 整合多个年份的数据

- 将1880 - 2010 的数据统计文件整合进一个数据集

```
years = range(1880, 2011 )
```

```
pieces = [] #将每个数据集作为pieces的一个元素
columns = names = ['name', 'sex', 'births']

for year in years:
    path = 'names/yob%d.txt' % year
    frame  = pd.read_csv(path, names = columns)
    frame['year'] = year
    pieces.append(frame)
    #得到的pieces是一个长度为131的list，每个元素都是一年的数据

#使用pd.concat()将各年数据整合进一个数据集
names = pd.concat(pieces, ignore_index = True) #忽略read_csv返回的原始行号
```

2.3 数据处理

2.3.1 统计每年(行)出生的男婴和女婴数量

```
from pandas import pivot_table
total_births = pivot_table(names,
        values='births', #列值
        index= 'year',  #行值
        columns='sex',  #列值，但不受aggfunc影响
        aggfunc=sum #求和函数，作用于values
        )
total_births.tail() #数据预览

import matplotlib.pyplot as plt
fig=plt.figure()
ax=fig.add_subplot(111)
ax.plot(x,y,title='Total births by sex and year')
plt.show()
```

2.3.2 插入新列prop，记录指定婴儿数量相对于总出生人数的比例

```
def add_prop(group):
    births = group.births.astype(float) #注意不可用整除
    group['prop']=births/births.sum()
    return group
names = names.groupby(['year', 'sex']).apply(add_prop)
#apply函数可用于调用参数已经存在于元组或字典的某个函数
```

2.3.3 选取每年排名前1000的姓进行统计(sex/year的前1000名)

- `sort_index in DataFrame`

```
from pandas import DataFrame as df
def get_top1k(group):
    return group.sort_index(by= 'births', ascending = False)[:1000] #前1000名，降序排列

grouped = names.groupby(['year', 'sex'])
top1k = grouped.apply(get_top1k)
```

2.4 分析命名趋势

2.4.1 按性别分类并建立透视表

```
from pandas import pivot_table
boys=top1k[top1k.sex=='M']
girls=top1k[top1k.sex=='F']

total_births = pivot_table(top1k, values='births', index= 'year', columns = 'name', aggfunc = sum)
```

2.4.2 评估某几个姓名使用率随年份的变化

```
subset = total_births[['John', 'Harry', 'Mary', 'Marilyn' ]]
subset.plot(subplots=True,
        figsize = (12,10),
        grid= False,
        title='Number of births per year')
```

2.4.3 评估命名多样性的增长（计算前1000个名字在总数中的prop）

```
import numpy as np
table = pivot_table(top1k,values='prop', index='year', columns='sex', aggfunc = sum)
table.plot(title='Sum of table1k.prop by year & sex',
        yticks=np.linspace(0, 1.2, 13),
        xticks= range(1880, 2020, 10))
```

Chapter 3 Numpy

```
from numpy import *
```

3.1 创建Array

```
arr=array([[1,2,3],[4,5,6]],dtype=' ') #创建一个2*3 array
a=array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])#2*2*3
zeros((3,4))/empty((3,4)) #零矩阵
ones( (2,3,4), dtype=int16 ) #2*3*4阵列，元素全为1
eye()/identity() #单位矩阵
```

```
arange(10,30,5) #[10,30)间隔为5, array([10,15,20,25])
arange().reshape() #常用创建矩阵方法，先arange再reshape
linspace(0,2,9) #[0,2)中均匀选取9个数字
```

```
from np import random #对内置的random进行补充
samples = np.random.normal(size=(4, 4)) #4*4 normal distribution
random.random((2,12)) #2*12 随机分布
#若样本量很大，np.random的速度更快
```

3.2 查看矩阵properties

3.2.1 基本查看方式

```
arr.ndim #dimension,此处返回2
arr.shape #给出矩阵某个维度的长度，此处返回(2L, 3L)表示2行3列
shape[0] #某个矩阵的第一维长度
shape(3) #单独数字，返回为空()
shape([3]) #一维矩阵，返回(1L,)
arr.size #元素数量

arr.dtype.name #查看数据类型,默认为int32
    arr.dtype('float64')#设置矩阵类型
arr.itemsize #每个元素的大小(bytes)
    float64=64/8=8
    int32=32/8=4
```

3.2.2 nonzero

```
nonzero(x) #返回x阵列中非零元素的位置
```

```
x[nonzero(x)] #返回x阵列中非零元素值
```

- 举个栗子

```
x=eye(3)
nonzero(x) #(array([0, 1, 2]), array([0, 1, 2])) 即非零元素位置为(0,0),(1,1),(2,2)
x[nonzero(x)] #(array([0, 1, 1])
```

- 另一种形式`transpose`

```
np.transpose(np.nonzero(x)) #array([[0, 0],[1, 1],[2, 2]])
```

3.3 基本操作

```
A+B #相加,等价于add(A,B)
B**2 #每个元素平方
A<35 #各元素返回为boolean
arrA*arrB #同位置元素相乘，类似octave A .* B
dot(arrA,arrB) #矩阵乘法
b+=a #等价于b=b+a
a*=3 #a=a*3
exp()/sqrt()/abs()/square()/log()/log10()/log2() #每个元素
sign()#各元素正负号
isnan() #各元素是否为NaN
.sum()/.min()/.max() #给出指定的元素

A.sum(axis=0) #返回每列的sum
A.min(axis=1) #返回每行的min
A.cumsum(axis=0) #从上到下累加（第二行为新值+上一行值）
```

- 举个栗子:

```
b=array([[0,1,2,3],[4,5,6,7],[8,9,10,11]])
b.cumsum(axis=1) #行方向:从左到右累加
>>>array([[0,1,3,6],[4,9,15,12],[8,17,27,38]])
```

- 再举个栗子:

```
xxx=np.arange(0,12).reshape((2,3,2)) #2个3*2矩阵
```

```
xxx
>>>array([[[ 0,  1],
           [ 2,  3],
           [ 4,  5]],

           [[ 6,  7],
            [ 8,  9],
            [10, 11]]])

xxx.sum(axis=0)
>>>array([[ 6,  8],
          [10, 12],
          [14, 16]])

xxx.sum(axis=1)
>>>array([[ 6,  9],
          [24, 27]]) #两个矩阵每列的和

xxx.sum(axis=2)
>>>array([[ 1,  5,  9],
          [13, 17, 21]]) #两个矩阵每行的和
```

3.4 Indexing, Slicing and Iterating

```
argsort(a) #排序，返回元素在原阵列的位置
    argsort([1, 2, 4, 9, 5, 3])->[0,1,5,2,4,3]
    argsort(-a) #反向排列(从大到小)

arr[-1]/arr[5:8,1]/arr[:,2] #可以查询或赋值某个元素,单数字默认为行
    a[:6:2]=-1000 #[0,6),每两个元素赋值为1000
    a[::-1]#反向排列
```

- `(...)` 在保证默认条件的前提下包含所有其他维度

```
#假定a为2*2*3array
a[1,...] #阵列中第二个矩阵的全部行列
a[...,2] #阵列中所有矩阵的第三列

for x in a.flat:
    print x #扁平化阵列：将所有元素打印为一列
```

3.5 变换阵列形式

3.5.1 改变行列

```
arange(24).reshape(4,3,2) #将一行向量转化为4*3*2阵列

#另一种方法：先设置shape再transpose()
arr.shape=(4,3,2)
arr.transpose() #这个也可以用于转置

arr.reshape(3,-1) #指定行数，自动计算列数
arr.resize((2,6)) #同reshape，但直接改变原array，类似sort/sorted的关系
```

3.5.2 组合阵列

```
vstack((a,b))/hstack((a,b)) #垂直/水平组合
column_stack((a,b)) #新增一列b
row_stack((a,b)) #新增一行

arr[:,newaxis] #增加新维度
arr=array([[1,2,3],[4,5,6]]) #(2L, 3L)
arr[:,newaxis]=array([[[1,2,3]],[[4,5,6]]]) #(2L, 1L, 3L)
```

- 举个栗子:

```
a=array([1,2,3,4]) #4L
b=array([5,6,7,8])
c=a[:,newaxis] #4*1 array([[1],[2],[3],[4]])
d=b[:,newaxis] #array([[5],[6],[7],[8]])
column_stack((c,d)) #4*2
```

3.5.3 拆分阵列

```
hsplit(a,3) #水平方向均匀拆分a为3个阵列（行不变，拆分列）
hsplit(a,(3,4)) #分割点为第四列（前三列;第四列；剩下列）
vsplit()#垂直方向，列不变，拆分行
```

3.6 Copy and View

- 赋值/浅复制/深复制

```
b=a #赋值：相当于根本没复制，改变了b，a也会变化
c=a.view() #shallow copy只拷贝父对象，不会拷贝对象的内部的子对象（子对象相当于赋值）
d=a.copy() #deep copy什么都复制了，从此再怎么变化都不会对副本造成影响
```

3.7 使用阵列查找阵列元素

```
a = arange(12)**2
i = array([1,1,3,8,5])
a[i] #a at the positions i
>>>array([1,1,9,64,25])

j = array([[3,4],[9,7]])
a[j]
>>>等价于array([[ a[3], a[4]],[a[9], a[7]]])
```

3.8 布尔阵列

```
a = arange(12).reshape(3,4)
b = a > 4 #b中元素为True/False
a[b] #得到a中令b为True(>4)的元素并组合成array
a[b]=0 #此时查看a会发现这些元素被赋值为0
```

3.9 ix_() function

- 组合不同向量用于进行矩阵运算
- 个人感觉应该和pandas.ix()类似

3.10 Linear Algebra

```
from numpy import *
from numpy.linalg import *
```

3.10.1 构造阵列

```
a = array([[1.0, 2.0], [3.0, 4.0]]) #2*2
print a #展示阵列a
```

```
a.transpose() #转置,等价于pandas.T
inv(a) #inversal matrix
```

3.10.2 求trace迹

```
u=eye(2)
trace(u) #trace (主对角线元素之和)
```

3.10.3 求解线性方程组

```
y=array([[5.],[7.]])
solve(a,y)#解为array([[-3.],[ 4.]])
```

3.10.4 直接构造矩阵

```
A=matrix('1.0 2.0;3.0 4.0') #2*2
type(A)
>>> <class 'numpy.matrixlib.defmatrix.matrix'>

A.T #transpose()
A.I #inv()

Y=matrix('5.0;7.0') #2*1
print A*Y #矩阵乘法，最后得到2*1矩阵

solve(A,Y) #和之前一样，求解线性方程组
```

Chapter 4 Pandas

- 两个重要的数据结构：Series and DataFrame

4.1 Series

```
from pandas import Series
#Series: 一维类似数组的对象，包含一列NumPy数据及一列标签索引
```

4.1.1 创建Series

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

```
>>>d  4
b  7
a  -5
c  3
dtype: int64


#基于dict创建(会自动排序)
obj=Series(dictionary,index=newindex) #索引可以赋值成新索引，如果没有就用原dict里的key

obj.values #获取数据，这里可以回顾下Chapter2里的pivot_table
obj.index #未指定则默认[0,1,2,3],也可以赋值新索引
obj[['a','b']] #返回对应值,也可以对该索引进行赋值
```

4.1.2 类numpy操作

- 由于和NumPy共享数据类型，可以延续numpy的操作

```
obj2[obj2 > 0]
obj2 * 2
np.exp(obj2)
'b' in obj2 #True
#算术运算时，不同索引数据会自动对齐
```

4.2 DataFrame

- 包含一个索引以及与这些索引联合在一起的Series

4.2.1 创建DataFrame

```
from pandas import DataFrame
data={'key1':[],'key2':[],'key3':[]}
frame=DataFrame(data)

frame
>>> Empty DataFrame
Columns: [key1, key2, key3]
Index: []
#index是行名，keys是列名！

DataFrame(data,columns=[],index=[]) #重新指定col,index
#DF中的每一个column可以被认为是一个Series
```

4.2.2 查看元素并赋值

```
frame['state']/frame.year #查看某列
frame.ix['index'] #查看某行

#对某列赋值，如果值数和列长度不等，值会被随机填充，空位为NaN

frame['eastern'] = frame.state == 'Ohio' #增加一条名为eastern的新列，列值为boolean

del frame['eastern'] #删除列
```

4.3 Index Objects

```
index = obj.index #给出每行的名字
index
>>>index([a,b,c],dtype=object)
index[1:]
>>>b,c
#注意index是不可变的，赋值会返回错误

2003 in frame3.index #类似查询value,查询是否存在名为2003的index(注意2003不是value)
```

4.4 Reindexing

- 创建带有新索引的新对象，相当于增加新行

```
obj=Series([4.5,7.2,-5.3,3.6],index=['d','b','a','c'])
obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e']) #'e'=NaN

#替换缺失值为0
obj.reindex(['a', 'b', 'c', 'd', 'e'], fill_value=0)
#当然,可以直接赋值以增加新行
obj["e"]="asshole"

DataFrame(np.arange(9).reshape((3, 3)),
        index=['a', 'c', 'd'],
        columns=['Ohio', 'Texas', 'California'])
frame2 = frame.reindex(['a', 'b', 'c','d']) #默认为重新索引行
frame.reindex(columns=states) #重新索引列
```

```
#可以将上面两个结合在一起同时索引行列
```

4.5 删除条目

```
new_obj = obj.drop(['c','d']) #删除'c','d'这两行

#对于DataFrame,可以从任意坐标轴删除索引
data.drop('Ohio') #删除某行
data.drop('two') #删除某列，等价于del frame[' ']
```

4.6 Indexing , selection and filtering

```
#Series索引类似NumPy
obj['b']/obj[1]/obj[2:4]/obj[obj<2]

#注意，切片跟python不同！两边均包含而非[)
obj['b':'c'] #索引b与c的值都会返回

obj.ix[xxx] #DataFrame中某行
obj.ix[:,xxx] #某列
obj.ix[xxx,yyy] #行列
```

4.7 Sorting and Ranking

4.7.1 按index排序

```
seri.sort_index(
        axis=1, #按照某个维度排序
        ascending=False #降序排列
        )
df.sort_index(by=['a','b']) #对于数据框架，指定按哪一列索引排序
```

4.7.2 按value排序

```
obj.order()
```

4.7.3 Rank

- 与排序不同，Rank会把对象的 values 替换成名次(不一定是整数)

- 对于平级项，可选method:`average/min/max/first`
- 默认`ascending=True`从小到大
- 举个栗子：

```
obj = Series([7, -5, 7, 4, 2, 0, 4])
obj.rank()
>>>6.5 1.0 6.5 4.5 3.0 2.0 4.5

obj.rank(method='first')
>>>6 1 7 4 3 2 5
#7的名次,-5的名次,...

obj.rank(ascending=False, method='max')
>>>2 7 2 4 5 6 4

#对于DataFrame，多了个axis项，其余类似
frame.rank(axis=1)
```

4.8 Summarizing and Computing Descriptive Statistics

```
df.sum(axis=1)#numpy.sum
df.mean(axis=1, skipna=False) #每行平均值
df.idxmax() #max的索引值
df.argmax() #max的索引位置
cumsum/cummin/cummax #累计和/最小最大值
var/std/quantile
```

4.9 Handle Missing Values

```
data.isnull() #查询缺失值

data.dropna() #删除缺失值，等价于data[data.notnull()]
    data.dropna(how='all') #删除全是NA的行
    data.dropna(axis=1, how='all') #同理，删除全是NA的列

df.fillna(0) #填充缺失值，返回新object
    df.fillna({1: 0.5, 3: -1}) #不同列不同填充值
    data.fillna(data.mean()) #使用平均值来填充NA
    df.fillna(0,inplace=True) #不返回新object，直接在原位修改
```

Chapter 5 Data Loading , Storage and File Formats

5.1 Reading

```
pd.read_table(path,names,sep='')

pd.read_csv(
        header=None, #设置默认header,没有则将第一行数据作为header
        na_values=['NULL'], #缺失值显示为'NULL'
        nrows=5, #仅仅读取前5行
        )

pd.from_csv()
np.loadtxt(delimiter) #np.array(pd.read_csv())
```

5.2 Writing

```
data.to_csv(
        "filename",
        sep="|",
        index=False,
        header=False,
        na_rep="NULL",
        cols=["a","b","c"]
        ) #将DataFrame/Series 另存为csv
```

5.3 Other Kinds of Files / Modules

```
import csv
import json
```

- For HTML/XML : `urllib2/BeautifulSoup/requests`

5.4 Binary Data Formats

```
frame=pd.read_csv()
frame.save("xxx_pickle") #另存为二进制格式
```

```
pd.load("xxx_pickle") #使用python读取该文件
```

5.5 Excel:

```
xls_file=pd.ExcelFile("data.xls")
table=xls_file.parse('sheet1') #转化为DataFrame
```

Chapter 6 Data Wrangling: Clean , Transform , Merge , Reshape

6.1 Combining and Merging

```
pd.merge()
pd.concat()
combine_first()
```

6.1.1 Database-style DataFrame Merges

```
pd.merge(df1,df2,on='key') #基于key组合两个DataFrame
pd.merge(df3,df4,left_on='key1',right_on='key2') #基于不同的key
```

6.1.2 Merging on Index

```
pd.merge(df1,df2,left_on='key',right_index=True)
```

6.1.3 Concatenating Along on Axis

```
arr=np.arange(12).reshape((3,4))
np.concatenate([arr,arr],axis=1) #横向组合
pd.concat([s1, s2, s3], axis=1) #横向组合,axis=0纵向组合
```

6.1.4 Combining Data with Overlap

```
df1.combine_first(df2) #重复数据df1优先
```

6.2 Reshaping and Pivoting

6.2.1 stack/unstack

```
data=pd.DataFrame(
        np.arange(6).reshape((2,3)),                                    index=
pd.Index(['Ohio','Colorado'],name='state'),          columns=pd.Index(['
1','2','3'],name='number')
        ) #构造一个表格
result=data.stack() #解构表格为Series
result.unstack() #返回解构前的DataFrame
result.unstack(0)/result.unstack('state') #返回但行列互换
```

6.2.2 Pivot

```
pivotData=data.pivot('date','item','value')
#date为行名(index), item为列名(key/column), value为pivot中各元素的值(value)
```

6.3 Data Transformation

6.3.1 Removing Duplicates

```
data.duplicated() #第一次出现False,再次出现True(保留第一次)
data.drop_duplicated() #返回去除重复值的DF
    data.drop_duplicated(['col1']) #基于col列去除重复值
    data.drop_duplicated(take_last=True) #保留最后一次
```

6.3.2 Mapping

```
data['xxx'].map(functionName)
```

6.3.3 Replacing Values

```
data.replace('sb','asshole')
```

6.3.4 Renaming Axis Indexes

```
data.index.map(str.upper) #行名全大写
data.rename(index=str.title,columns=str.upper) #行名首字母大写，列名全大写
```

- 后面略了一些内容

6.4 String Manipulation

6.4.1 String Object Methods

```
val.split(',')
val.strip()
"::".join(pieces)
val.count(',')
```

6.4.2 Regular Expressions, 见python笔记

6.4.3 Vectorized string functions in pandas

```
data.str.contains('xxx') #类似data.find('xxx')
data.str.findall(pattern)
data.str.match(pattern)
```

Chapter 7 Visualization

7.1 简单作图

7.1.1 事例

```
import matplotlib.pyplot as plt
fig=plt.figure() #create a new figure
ax1=fig.add_subplot(2,2,1) #2行2列，在位置1添加坐标系
ax4=fig.add_subplot(2,2,4) #2行2列，在位置4添加坐标系

from numpy.random import randn
ax1.hist(randn(100),bins=20,color='k',alpha=0.3)
ax4.scatter(np.arange(30),np.arange(30)+3*randn(30))
```

7.1.2 subplot function

```
fig,axes=plt.subplots(2,3) #直接2*3图层，6个坐标系
```

- other parameters:

```
nrows,
ncols,
```

```
sharex #same X-axis ticks
sharey #same Y-axis ticks
```

7.1.3 subplot_adjust

7.2 Colors, Markers and Line Styles

```
plt.plot(randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post')
plt.legend(loc='best') #add a legend
```

7.3 Add a shape to a plot

```
rect=plt.Rectangle((0.2, 0.75), #起点坐标
        0.4, 0.15, #长度x,高度y
        color='k', alpha=0.3)
circ = plt.Circle((0.7, 0.2),  #圆心坐标
        0.15, #短轴/长轴
        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) #三顶点坐标

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

7.4 Save to a file

```
plt.savefig('filename.png',
        dpi=400,
        facecolor/edgecolor,
        format='png'/'pdf'/'svg'...
        bbox_inches='tight' #the portion of the figure to save
        )
```

- P.S. There are also some plot functions in pandas

Chapter 8 Data Aggregation and Group Operations

8.1 GroupBy Mechanics

- split into subgroups-->apply function-->combine into group

8.2 GroupBy Example

```
df=pd.DataFrame({'key1':['a','a','b','b','a'],'key2':['1','2','1','2','1'],'data1':np.random.randn(5),'data2':np.random.randn(5)})

means = df['data1'].groupby([df['key1'], df['key2']]).mean()

means.unstack() #得到split但未进行apply function的数据
```

8.3 Iterating Over Groups

```
for name,group in df.groupby('key1'):
    print name
    print group
```

8.4 Aggregate function

```
quant=df.groupby('key1').quantile(0.9)
ag=df.groupby('key1').agg(yourOwnFunction)
```

8.5 Pivot Tables and Cross-Tabulation

```
tips.pivot_table('tip_pct',
        rows=['sex', 'smoker'],
        cols='day',
        aggfunc=len,
        margins=True)
```

- Cross-Tabulation:交叉分析（就是计数的透视表）

```
pd.crosstab(data.Gender,data.Handedness,margins=True)
#行为gender,列为handness,数据为count
```

Chapter 9 Time Series

9.1 概念:

- 时间序列: 用时间排序的一组随机变量
- 特征:unstable, 波动幅度随时间变化

9.2 Date and Time Data Types and Tools

```
from datetime import datetime
now = datetime.now()
now.year/month/day

delta=datetime(2011,1,7)-datetime(2008,6,24,8,15)
#返回天数差.days与秒数差.seconds

starttime=datetime(2016,5,9)
start+timedelta(12) #+12 days
```

9.3 Converting between string and datetime

- datetime --> strings

```
stamp=datetime(2016,5,9)
str(stamp) #convert to default strings "2016-05-09 00:00:00"
stamp.strftime("%Y-%m-%d") #"2011-01-03"
```

- strings --> dates

```
datetime.strptime(strings,"%Y-%m-%d")
```

- parse:更改不同类型的日期格式

```
parse('6/12/2011',dayfirst=True) #(2011,12,6,0,0)
```

9.4 Time Series Basics

```
dates=[datetime(xxxx),datetime(xxxx),...]
timeSeries=pd.Series(np.random.randn(x),index=dates)
```

9.4.1 Indexing,Selection and Subsetting

9.4.2 Time Series with Duplicate Indices

9.5 Date Ranges,Frequencies and Shifting

Chapter 10 Financial and Economic Data Applications

Chapter 11 Advanced Numpy