Notations：

灰色代表已总结到《DataAnalysis\_V2.0.docx》中

第1章 准备工作

#以pylab模式启动ipython（否则画不出matplotlib图形来）

ipython –pylab

# Pandas读取csv文件的前5行。补充：读取Excel用read\_excel() 函数

方法一：直接取出前5行

Import pandas as pd

Result=pd.read\_csv(‘G:\\xin.csv’,encoding=’gbk’,**nrows=5**)

Result

方法二：取出全部，只展示前5行

Import pandas as pd

Result=pd.read\_csv(‘G:\\xin.csv’,encoding=’gbk’)

Result[:10]

#使用块的方式进行汇总。chunker为TextParser或TextFileReader对象

Import pandas as pd

Chunker=pd.read\_csv(‘G:\\xin.csv’,encoding=’gbk’,chunksize=38970)

tot=pd.Series([])

for piece in chunker:

tot=tot.add(piece[‘客户性别’].value\_counts(),fill\_value=0)

tot=tot.order(ascending=False) #系统显示不推荐使用该种排序

tot

得出结果：

男 29981.0

女 8988.0

dtype：float64

第2章 引言

# 读取txt文件的第一行 P21

path=’G:\people.txt’

open(path).readline()

源文件：《people.txt》

{"name":"中国","province":[{"name":"黑龙江","cities":{"city":["哈尔滨","大庆"]}},{"name":"广东","cities":{"city":["广州","深圳","珠海"]}},{"name":"台湾","cities":{"city":["台北","高雄"]}},{"name":"新疆","cities":{"city":["乌鲁木齐"]}},{"name":"山西","cities":{"city":["太原","大同"]}},{"name":"山西","cities":{"city":["临汾","运城"]}},{"name":"上海","cities":{"city":["浦东"]}},{"name":"上海","cities":{"city":["松江"]}},{"name":"上海","cities":{"city":["嘉定"]}}]}

#用json模块&列表推导式 格式化txt中的json串 P21。records对象为一组python字典 P21

import json

path=’G:\people.txt’

records=[json.loads(line) for line in open(path)] #records的type为列表

records[0]

#按键取值

records[0][‘name’]

输出：

‘中国’

#打印

print (records[0][‘name’])

输出：

中国

#取出省份 用列表推导式和切片，并能够兼容缺省数据 计算结果的长度

cities=[city[‘name’] for city in records[0][‘province’] if ‘name’ in city]

cities

输出：

['黑龙江', '广东', '台湾', '新疆', '山西', '山西', '上海', '上海', '上海']

len（cities）

#计数（只使用标准python库 方法一）计算各省份出现的次数

def get\_counts(sequence):

counts={}

for x in sequence:

if x in counts:

counts[x]+=1

else:

counts[x]=1

return counts

counts=get\_counts(cities)

counts

#计数（只使用标准python库 简洁写法）计算各省份出现的次数

from collections import defaultdict

def get\_counts2(sequence):

counts = defaultdict(int)

for x in sequence:

counts[x]+=1

return counts

counts=get\_counts2(cities)

counts

#计数。取出现次数前2位的省份及其计数值（方法一：提取成为列表，将列表排序，取出最后两个） 补充：在此处count类似于SQL中的聚合函数？

def top\_counts(count\_dict,n=3):

value\_key\_pairs=[(count,name) for name,count in count\_dict.items()]

value\_key\_pairs.sort()

return value\_key\_pairs[-n:]

top\_counts(counts,2)

#计数（使用collections.Counter类）计算各省份出现的次数 并取出现次数前2位的省份

from collections import Counter

counts2=Counter(cities)

counts2

输出：（输出结果的类型是collections.Counter）

Counter({'上海': 3, '台湾': 1, '山西': 2, '广东': 1, '新疆': 1, '黑龙江': 1})

counts2.most\_common(2)

输出：

[('上海', 3), ('山西', 2)]

《city.txt》

[{"name":"山西","city":"太原"},{"name":"山西","city":"临汾"},{"name":"山西","city":"大同"},{"name":"上海","city":"浦东"},{"name":"上海","city":"松江"},{"name":"黑龙江","city":"哈尔滨"},{"name":"浙江","city":"杭州"},{"name":"广东","city":"深圳"}]

#计数（使用Pandas） P24 用fillna()替换缺失值 P26

from pandas import DataFrame,Series

import pandas as pd;import numpy as np;import json

path=’G:\city.txt’

records=[json.loads(line) for line in open(path)]

frame=DataFrame(records[0])

frame

输出：（看来DataFrame自己也解析了JSON串）

|  | **name** | **province** |
| --- | --- | --- |
| **0** | 中国 | {'name': '黑龙江', 'cities': {'city': ['哈尔滨', '大庆... |
| **1** | 中国 | {'name': '广东', 'cities': {'city': ['广州', '深圳',... |
| **2** | 中国 | {'name': '台湾', 'cities': {'city': ['台北', '高雄']}} |
| **3** | 中国 | {'name': '新疆', 'cities': {'city': ['乌鲁木齐']}} |
| **4** | 中国 | {'name': '山西', 'cities': {'city': ['太原', '大同']}} |
| **5** | 中国 | {'name': '山西', 'cities': {'city': ['临汾', '运城']}} |
| **6** | 中国 | {'name': '上海', 'cities': {'city': ['浦东']}} |
| **7** | 中国 | {'name': '上海', 'cities': {'city': ['松江']}} |
| **8** | 中国 | {'name': '上海', 'cities': {'city': ['嘉定']}} |

frame[‘name’]

输出：

0 中国

1 中国

2 中国

3 中国

4 中国

5 中国

6 中国

7 中国

8 中国

Name: name, dtype: object

Frame2=DataFrame(records[0][‘provice’]

输出：

|  | **cities** | **name** |
| --- | --- | --- |
| **0** | {'city': ['哈尔滨', '大庆']} | 黑龙江 |
| **1** | {'city': ['广州', '深圳', '珠海']} | 广东 |
| **2** | {'city': ['台北', '高雄']} | 台湾 |
| **3** | {'city': ['乌鲁木齐']} | 新疆 |
| **4** | {'city': ['太原', '大同']} | 山西 |
| **5** | {'city': ['临汾', '运城']} | 山西 |
| **6** | {'city': ['浦东']} | 上海 |
| **7** | {'city': ['松江']} | 上海 |
| **8** | {'city': ['嘉定']} | 上海 |

# 计数。输出的结果是Series，‘name’成为索引，数量成为值

city\_counts=frame2[‘name’].value\_counts()

city\_counts[:5]

#画图。画条形图 注意：在IPython中，需要用py—lab模式打开ipython才能绘图

#如果在Jupyter Notebook中画图，需要加%matplotlib inline

mpl.rcParams['font.sans-serif'] = ['SimHei'] #用于解决中文显示问题  
mpl.rcParams['axes.unicode\_minus'] = False

city\_counts[:10].plot(kind=’barh’,rot=0) #注意：复制粘贴时去空格

#将省份字段中的“省份、出现次序”分离。并将列表推导式得出的list转换为Series

results=Series([x.split()[0] for x in frame.name.dropna()]) # frame[‘name’]和frame.name都可

results

#去除缺失数据。实现原理：① frame.a.notnull()返回一个bool类型的Series。② DateFrame（Series）是DataFrame的一种筛选方法

cframe=frame[frame.a.notnull()]

cframe

#为**数据打标签（归类数据）**：如果带有“广东”2字，则为“华南区”，否则为“非华南区”；返回的是np数组array——？如果华南有多个省怎么办？

# numpy.where(condition[**,**x**,**y]) —— 类似于列表推导式。可以相辅相成

category=np.where(frame[‘name’].str.contains(‘广东’),’华南区’,’非华南区’)

category

输出：np数组。类似于array([‘华南区’,’非华南区’,’非华南区’,’非华南区’,’华南区’,’非华南区’,’华南区’])

#分组。把省份列表按照华南、非华南进行分组。如果有多个分组条件，用groupby([A,B])

by\_catagory=frame.groupby([category])

输出：无显示输出。输出类型为pandas.core.groupby.groupby.DataFrameGroupBy

#计数。对分组结果进行计数

area\_counts=by\_catagory.size()

area\_counts

输出：类似于

***华南区 3***

***非华南区 4***

# ? 遗留问题：P28的unstack用法和***间接索引数组*** 用法；堆积条形图规范化为“总计为1”的算法

# Pandas读取csv文件（用read\_table）注意：如果不加sep，表格数据只会合并为一列 P32

import pandas as pd

sales=pd.read\_table(‘G:\\xin.csv’,sep=’,’,encoding=’gbk’)

sales[:5]

gender=pd.read\_table(‘G:\\gender.csv’,sep=’,’,encoding=’gbk’)

# 两个DataFrame合并。将sales和gender合并。补充：根据那个主键合并——通过参数制定。DataFrame合并的方法还有append、join、concat

data=pd.merge(sales,gender)

# 关于Index和Select。DataFrame选择表头

data.columns

# 关于Index和Select。DataFrame选择第一行

data.ix[0] 或 data.iloc[0]

#计算每个城市按性别计算首刷金额的平均值 使用pivot\_table方法，返回DataFrame

mean\_payment=data.pivot\_table(‘首付款金额’,index=’车辆所属城市’,columns=’客户性别’,aggfunc=’mean’)

#计算各城市的销量

city\_counts=data[‘车辆所属城市’].value\_counts()

city\_counts

#过滤掉销量不足500的城市。先对城市进行分组，然后利用size（）得到一个含有各城市分组大小的Series对象

sales\_by\_city=data.groupby(‘车辆所属城市’).size()

sales\_by\_city

goodSale\_city=sales\_by\_city.index[sales\_by\_city≥500] #制作索引

goodSale\_city

mean\_payment=mean\_payment.ix[goodSale\_city]

#根据女性客户降序排列

top\_female\_sales=mean\_payment.sort\_values(by=’女’,ascending=False)

top\_female\_sales

#DateFrame中追加一列，表示男女平均额的差值

mean\_payment[‘diff’]=mean\_payment[‘女’]-mean\_payment[‘男’]

mean\_payment

#按照差值排序

sort\_by\_diff=mean\_payment.sort\_values(by=’diff’)

sort\_by\_diff

#对排序结果 ***反序*** 并取出前5行

sort\_by\_diff[::-1][:5]

#计算各城市的***标准差***（不考虑性别因素）

sales\_std\_by\_city=data.groupby(‘车辆所属城市’)[‘首付款金额（元）’].std()

#根据goodSale\_city进行过滤

sales\_std\_by\_city=sale\_std\_by\_city.ix[goodSale\_city]

#根据值对Series进行降序排列

sales\_std\_by\_city.order(ascending=Flase)[:10]

#用pandas加载csv到DataFrame中 P37

import pandas as pd

data=pd.read\_csv(‘G:\\xin.csv’,encoding=’gbk’)

#根据性别，计算抵押贷利息之和

data.groupby(‘客户性别’).抵押贷利息.sum()

#？未进行实验：pandas.concat合并

#根据年份、性别进行利息计算

interest=data.pivot\_table(‘抵押贷利息’,index=’首刷年份’,columns=’客户性别’,aggfunc=sum)

interest

#作图

mpl.rcParams[‘font.sans-serif’]=[‘SemHei’]

mpl.rcParams[‘axes.unicode\_minus’]=False

interest.plot(‘抵押贷利息分组’)

#处理每个城市的抵押贷利息在总抵押贷利息中的占比，并根据“首刷年份”和“客户性别”分组

def add\_prop(group)：

interest=group.抵押贷利息.astype(float)

group[‘prop’]=interest/interest.sum()

return group

data\_prop=data.groupby([‘首刷年份’,’客户性别’]).apply(add\_prop)

data\_prop[:5]

#上述结果的***有效性检查*** P39

np.allclose(data\_prop.groupby([‘首刷年份’，‘客户性别’]).prop.sum(),1)

返回结果：True

#取每对“年份/性别”组合的前1000个抵押贷利息最高的订单。方法一。建议只用前20条做实验，否则屏幕显示不了那么多。

def get\_top1000(group):

return group.sort\_index(by=‘抵押贷利息’,ascending=Fasle)[:3]

grouped=data\_prop.groupby([‘首刷年份’，’客户性别’])

top1000=grouped.apply(get\_top1000)

top1000

#方法二。

pieces=[]

for year,data\_prop in data\_prop.groupby([‘首刷年份’,’客户性别’]):

pieces.append(data\_group.sort\_index(by=‘抵押贷利息’,ascending=False)[:1000])

top1000=pd.concat(pieces,ignore\_index=True)

top1000

#将前1000条数据分为男、女两个部分

boys=top1000[top1000.客户性别 == ‘男’]

girls=top1000[top1000.客户性别 == ‘女’]

#生成按年和省份统计的总透视表

total=top1000.pivot\_table(‘抵押贷利息’,index=‘首刷年份’,columns=’车辆所属省份’,aggfunc=sum)

total

#绘制北京、上海、四川三个城市的曲线图

#取子集

mpl.rcParams['font.sans-serif'] = ['SimHei'] #用于解决中文显示问题  
mpl.rcParams['axes.unicode\_minus'] = False

subset=total[[‘北京’，‘上海’，‘四川’]]

subset.plot(subplots=True,figsize=(12,10),grid=False,title=”城市统计”)

#计算抵押贷利息最高的前1000项在总表中所占的比例

table=top1000.pivot\_table(‘prop’,index=’首刷年份’,columns=’客户性别’,aggfunc=sum)

table.plot(title=”比例统计”,yticks=np.linspace(0,1.2,13),xticks=range(2014,2020,2))

#计算抵押贷利息占前10%的单子的数量，且只考虑2016年的男生

df=boys[boys.首刷年份==2016]

df

#先降序排列。然后通过Numpy的矢量方式，计算prop的累计和cumsum，然后再通过searchsorted方法找出0.1应该被插入在哪个位置才能保证不破坏顺序。

prop\_cumsum=df.sort\_index(by=’prop’,ascending=False).prop.cumsum()

prop\_cumsum[:10]

prop\_cumsum.searchsorted(0.1)

#结果显示array([586],dtype=int64)。和书上结果不同。因为数组索引是从0开始的，因此要对结果加1。

#刚刚先做了单年的计算，现在将算法应用到groupby上。按理说该函数的输出结果应该是DataFrame或者Series才行

def get\_quantile\_count(group,q=0.1):

group=group.sort\_index(by=’prop’,ascending=False)

return group.prop.cumsum().searchsorted(q)+1

diversity=top1000.groupby([‘首刷年份’,’客户性别’]).apply(get\_quantile\_count)

diversity=diversity.unstack(‘sex’)

diversity.head

#显示结果和书上不同。数字被大括号包围。

客户性别 女 男

首刷年份

2016 [195] [587]

2017 [126] [372]

#作图。但是报错提示 Empty ‘DataFrame’:no numric data to plot

diversity.plot(title=”比例统计”)

#取出“客户姓名”中的最后一个字

get\_last\_letter=lambda x:x[-1]

last\_letters=data\_prop.客户姓名.map(get\_last\_letter)

last\_letters.客户姓名 =‘last\_letter’

table\_name=data\_prop.pivot\_table(‘客户年龄’，index=last\_letters,columns=[‘客户性别’,’首刷年份’],aggfunc=sum)

subtable=table\_name.reindex(columns=[2016,2017],lever=’首刷年份’)

subtable.head()

#对总年龄数进行规范化处理，以便计算出各性别各末字占总年龄的比例。

subtable.sum()

letter\_prop=subtable/subtable.sum().astype(float)

#生成条形图

import matplotlib.pyplot as plt

fig,axes=plt.subplots(2,1,figsize=(10,8))

letter\_prop[‘男’].plot(kind=’bar’,rot=0,ax=axes[0],title=’Male’)

letter\_prop[‘女’].plot(kind=’bar’,rot=0,ax=axes[1],title=’Female’,legend=False)

#P45-47页的内容不适用于xin.csv，所以未做尝试。

#用IPython打开外部文件

%run G:\test.py

第4章 Numpy

#布尔型索引

#花式索引（即利用整数数组进行索引）

#数组转置和轴对换

#通用函数（ufunc）：快速的元素级数组函数

#利用数组进行数据处理 · 矢量化

#计算sqrt(X^2+Y^2)

1. np.meshgrid函数接受两个一维数组，并产生两个二维矩阵（对应于两个数组中所有的（x,y）对）

import numpy as np

points=np.arange(-5,5,0.01) #1000个间隔相等的点

xs,ys=np.meshgrid(points,points)

ys

输出：

array([[-5. , -5. , -5. , ..., -5. , -5. , -5. ],

[-4.99, -4.99, -4.99, ..., -4.99, -4.99, -4.99],

[-4.98, -4.98, -4.98, ..., -4.98, -4.98, -4.98],

...,

[ 4.97, 4.97, 4.97, ..., 4.97, 4.97, 4.97],

[ 4.98, 4.98, 4.98, ..., 4.98, 4.98, 4.98],

[ 4.99, 4.99, 4.99, ..., 4.99, 4.99, 4.99]])

#现在，对该函数的求值运算就好办了，把这两个数组当做两个浮点数那样编写表达式即可

%matplotlib inline #将matplotlib集成到jupyter notebook中

import matplotlib.pyplot as plt

z=np.sqrt(xs\*\*2,ys\*\*2)

z

输出：

array([[ 5. , 4.99, 4.98, ..., 4.97, 4.98, 4.99],

[ 5. , 4.99, 4.98, ..., 4.97, 4.98, 4.99],

[ 5. , 4.99, 4.98, ..., 4.97, 4.98, 4.99],

...,

[ 5. , 4.99, 4.98, ..., 4.97, 4.98, 4.99],

[ 5. , 4.99, 4.98, ..., 4.97, 4.98, 4.99],

[ 5. , 4.99, 4.98, ..., 4.97, 4.98, 4.99]])

plt.imshow(z,cmap=plt.cm.gray);plt.colorbar()

plt.title("Image plot of $\sqrt{X^2+Y^2}$ for a grid of values ") #$中间的字符会转换为公式

第9章 数据聚合与分组运算

#GroupBy技术

#面向列的多函数应用

**#分组级运算和转换——即“面向分组”的计算**

#apply:一般性的“拆分-应用-合并” P278

#分位数和桶分析 P281（groupby结合cut和qcut）

#桶（bucket） 分位数（quantile）

#透视表和交叉表

#交叉表（计算分组频率的特殊透视表）

**#示例（应用前面所学到的知识） P292**

#映射map结合lambda、dict.get() P294

第10章 时间序列

P329 面元边界

第11章 金融和经济数据应用

P351 拼接多个数据源：concat combine\_first update

**#分组变换和分析 P355**

P358 分组因子暴露、最小二乘回归

P359 十分位和四分位分析

P366 移动相关系数与线性回归

第12章 NumPy高级应用

数组重塑：reshape/ravel/flatten P370

数组的合并和拆分:np.concatenate/vstack/hstack/split P374

堆叠辅助类：r\_和c\_ P374

元素的重复操作：tile和repeat P375

花式索引（获取和设置数组子集）的等价函数：take和put P377

**#广播 P378**

**#ufunc高级应用 P383（丢开循环编写更简洁的代码）**

ufunc实例方法 P385

自定义ufunc P385

**#结构化和记录式数组 P382**

（类似于自定义数据结构，相当于Pandas的DataFrame的分层索引机制）

**#排序 P388**

ndarray.sort（就地排序）np.sort（副本） P388

间接排序：argsort和lexsort P390

数组排序算法P392

其它总结