说明：

1. 如下内容，是在《DataAnalysis.docx》的基础上， 按照知识点的内容进行排列，并加入《利用Python进行数据分析》之外的知识点。
2. 使用的数据主要来源于《Data\_big.xlsx》和《Data\_small.xlsx》

《Data\_big.xlsx》包括21个字段，211291条数据：

订单编号、客户id、客户姓名、客户手机号、客户身份证、客户性别、客户年龄、车辆编号、车辆名称、车辆价格、车辆所属城市、商家ID、商家名称、首次上牌日期、首付刷卡日期、首付刷卡时间、业务期限、首付款金额、融资总额、首付车款、利息

《Data\_small.xlsx》包括20个字段，36548条数据（首付刷卡日期为2017年第三季度）：

订单编号、客户id、客户姓名、客户身份证、客户性别、客户年龄、车辆编号、车辆名称、车辆价格、车辆所属城市、商家ID、商家名称、首次上牌日期、首付刷卡日期、首付刷卡时间、业务期限、首付款金额、融资总额、首付车款、利息

任务：

1. 提取车辆品牌
2. 取出车辆价格排前10%的订单

**关于DataFrame**

#DataFrame的构造及其索引 P119

#重新索引 P126

# Pandas读取Excel

from pandas import DataFrame,Series

import pandas as pd

df1=pd.read\_excel('D:\\Data\_small.xlsx',encoding='gbk')

df1[:20]

输出：df1的数据格式为DataFrame

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 订单编号 | 客户id | …… | 利息 |
| 0 |  |  |  |  |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| …… |  |  |  |  |

# 选择DataFrame的某几列，形成新的DataFrame

df2=df1[['客户姓名','客户性别']]

df2

|  |  |  |
| --- | --- | --- |
|  | 客户姓名 | 客户性别 |
| 0 |  |  |
| 1 |  |  |
| 2 |  |  |

# 选择表头

df1.columns

*Index(['订单编号', '客户id', '客户姓名', '客户身份证', '客户性别', '客户年龄', '车辆编号', '车辆名称', '车辆价格','车辆所属城市', '商家ID', '商家名称', '首次上牌日期', '首付刷卡日期', '首付刷卡时间', '业务期限', '首付款金额','融资总额', '首付车款', '利息'],dtype='object')*

# 选择第一行或者多行

data.ix[0] ——> 注意：ix在新版本的Pandas中已经不建议使用

data.iloc[0]

data.iloc[0:5]

# 去除缺失值。

举例：如果a列有缺失值，则将a列有缺失值的行删除。

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

**# 数据聚合与分组运算**

# 描述统计

举例：计算关于车辆价格的描述统计

df1['车辆价格'].describe()

输出：

count 38965.000000

mean 157368.140639

std 107051.443548

min 30000.000000

25% 78900.000000

50% 125000.000000

75% 198000.000000

max 650000.000000

# 计算某列的平均值

df1['车辆价格'].mean()

# 分类计算。求和。

df1.groupby('首付刷卡日期').融资总额.sum()

# 分类计算。平均值

举例：计算不同性别的车辆价格平均值

price\_mean\_bysex=df1['车辆价格'].groupby(df1['客户性别']).mean()

price\_mean\_bysex

# 分类计算。计数

举例：计算各城市的销量。

city\_counts=df1['车辆所属城市'].value\_counts()

city\_counts

|  |  |
| --- | --- |
| 乌鲁木齐 | 1280 |
| 银川 | 1103 |
| …… | …… |
| 阿拉善左旗 | 1 |

举例：计算不同客户性别的销量。思路：实际上是Series的值计数。

# 分类计算。标准差

举例：计算各城市车辆价格的标准差。输出的结果是Series

price\_std\_by\_city=df1.groupby('车辆所属城市')['车辆价格'].std()

price\_std\_by\_city

# Series排序。和DataFrame的方法一样，只是参数不同。

sales\_std\_by\_city.sort\_values(ascending=False)

# 过滤

选出购买客户是女性的订单。

思路：使用DataFrame[ bool型Series] 的方法

order\_female=df1[df1.客户性别=='女']

order\_female

# 过滤

举例：从city\_counts的结果中，过滤掉销量不足500的城市。

方法：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。输出结果和value\_counts()方法的不同在于，没有进行排序。——此方法也适用于DataFrame以及数据透视表

sales\_by\_city=df1.groupby('车辆所属城市').size()

sales\_by\_city

|  |  |
| --- | --- |
| 万州 | 25 |
| 三亚 | 25 |
| …… | …… |
| 齐齐哈尔 | 99 |
| 龙岩 | 30 |

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

goodSale\_city

*Index(['东莞', '乌鲁木齐', '包头', '北京', '南京', '南宁', '合肥', '呼和浩特', '唐山', '太原', '宁波','成都', '武汉', '沈阳', '深圳', '石家庄', '西安', '贵阳', '赣州', '银川', '长春', '长沙'],dtype='object', name='车辆所属城市')*

city\_counts=city\_counts.loc[goodSale\_city] # 过滤

city\_counts

|  |  |
| --- | --- |
| 东莞 | 715 |
| 乌鲁木齐 | 1280 |
| …… | …… |
| 长沙 | 608 |

# 过滤

生成实验用数据：

from pandas import DataFrame,Series

import pandas as pd

cencus\_df=pd.DataFrame({'city':['北京','上海','广州','深圳','宁波'],'cencus':[3000,25000,2000,1500,500]})

cencus\_df

|  |  |  |
| --- | --- | --- |
|  | city | cencus |
| 0 | 北京 | 3000 |
| 1 | 上海 | 2500 |
| 2 | 广州 | 2000 |
| 3 | 深圳 | 1500 |
| 4 | 宁波 | 500 |

举例：选出人口大于1500万成的城市

a\_df = cencus\_df[ cencus\_df['cencus'] > 1500 ]

|  |  |  |
| --- | --- | --- |
|  | city | cencus |
| 0 | 北京 | 3000 |
| 1 | 上海 | 2500 |
| 2 | 广州 | 2000 |

# 选择

举例：取出车辆价格排前10%的订单

思路：

**# 应用函数·apply()**

frame=DataFrame(np.random.randn(4,3),columns=list('bde'),index=['Utah','Ohio','Texas','Oregon'])

frame

输出：

|  | **b** | **d** | **e** |
| --- | --- | --- | --- |
| **Utah** | -0.101401 | -0.026205 | 0.618136 |
| **Ohio** | -2.209868 | -1.829319 | -0.850499 |
| **Texas** | -0.664304 | 0.107059 | 0.566694 |
| **Oregon** | 1.046030 | 1.037165 | 1.542936 |

#引用到行

f=lambda x:x.max()-x.min()

frame.apply(f)

输出：

b 3.255898

d 2.866484

e 2.393435

dtype: float64

备注：类似于groupby.apply()，也是将属于一行行传入函数进行处理

#应用到列

frame.apply(f,axis=1)

输出：

Utah 0.719537

Ohio 1.359369

Texas 1.230997

Oregon 0.505771

#传递给apply的函数，除了返回标量值，还可以返回由多个值组成的Series

#除了标量值外，传递给apply的函数还可以返回由多个值组成的Series

def f(x):

return Series([x.min(),x.max()],index=['min','max'])

frame.apply(f)

输出：

|  | **b** | **d** | **e** |
| --- | --- | --- | --- |
| **min** | -2.209868 | -1.829319 | -0.850499 |
| **max** | 1.046030 | 1.037165 | 1.542936 |

**# 应用函数·applymap()**

#应用元素级的python函数和applymap方法。将数字格式化到小数点后两位

format=lambda x:'%.2f' %x

frame.applymap(format)

输出：

|  | **b** | **d** | **e** |
| --- | --- | --- | --- |
| **Utah** | -0.10 | -0.03 | 0.62 |
| **Ohio** | -2.21 | -1.83 | -0.85 |
| **Texas** | -0.66 | 0.11 | 0.57 |
| **Oregon** | 1.05 | 1.04 | 1.54 |

**# 应用函数比较apply、applymap、map**

apply 用在dataframe上，用于对row或者column进行计算

applymap 用于dataframe上，是元素级别的操作

map （其实是python自带的）用于series上，是元素级别的操作

**# 数据透视表**

举例：计算每个城市按性别计算车辆价格的平均值。使用pivot\_table方法，返回DataFrame

mean\_price = df1.pivot\_table('车辆价格',index='车辆所属城市',columns='客户性别',aggfunc='mean')

mean\_price

|  |  |  |
| --- | --- | --- |
|  | 女 | 男 |
| 万州 | 134800.000000 | 95250.000000 |
| 三亚 | 138062.500000 | 83341.176471 |
| …… | …… | …… |
| 龙岩 | 83800.000000 | 82020.689655 |

# 延伸：取出女性消费比男性消费高的城市

思路：属于DataFrame的过滤。可以先制作bool型Series，然后应用到DataFrame。

female\_higher = (price\_sum['女']-price\_sum['男']) > 0 # 原理：> 和加减乘除一样都是操作符

female\_higher\_city = mean\_price[female\_higher]

female\_higher\_city

# 延伸：女性消费高、男性消费高的城市计数

思路：Series的值计数

female\_higher\_count = female\_higher.value\_counts()

female\_higher\_count

|  |  |
| --- | --- |
| True | 198 |
| False | 46 |

# 排序

sort\_values() ——

举例：在mean\_price的基础上，按照女性客户降序排列。——适用于一般的DataFrame

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

top\_female\_sales

# 反序。

举例：对top\_female\_sales排序结果进行反序，并取出前5行。

top\_female\_sales[::-1][:5]

# 排序。复杂排序

方法一：apply函数

取出每个城市车价最高的3个订单——？如果一个城市没有3个订单咋办

def get\_top3(data):

return data.sort\_values(by='车辆价格',ascending=False)[:3]

top3=df1.groupby('车辆所属城市').apply(get\_top3)

top3

方法二：

利用groupby对象的可迭代性

pieces=[]

for name,data in df1.groupby('车辆所属城市'):

pieces.append(data.sort\_values(by='车辆价格',ascending=False)[:3])

top3=pd.concat(pieces,ignore\_index=True)

top3

取出其中北京的数据

top3[top3.车辆所属城市=='北京']

# DataFrame追加列。将计算结果追加为一列。思路：可将Series追加为列。

举例：DateFrame中追加一列，表示男女平均额的差值

mean\_price['差值']=mean\_price['女']-mean\_price['男']

mean\_price

|  |  |  |  |
| --- | --- | --- | --- |
|  | 女 | 男 | 差值 |
| 万州 | 134800.000000 | 95250.000000 | 39550.000000 |
| 三亚 | 138062.500000 | 83341.176471 | 54721.323529 |
| …… | …… | …… | …… |
| 龙岩 | 83800.000000 | 82020.689655 | 1779.310345 |

**# 应用函数·groupby.apply()**

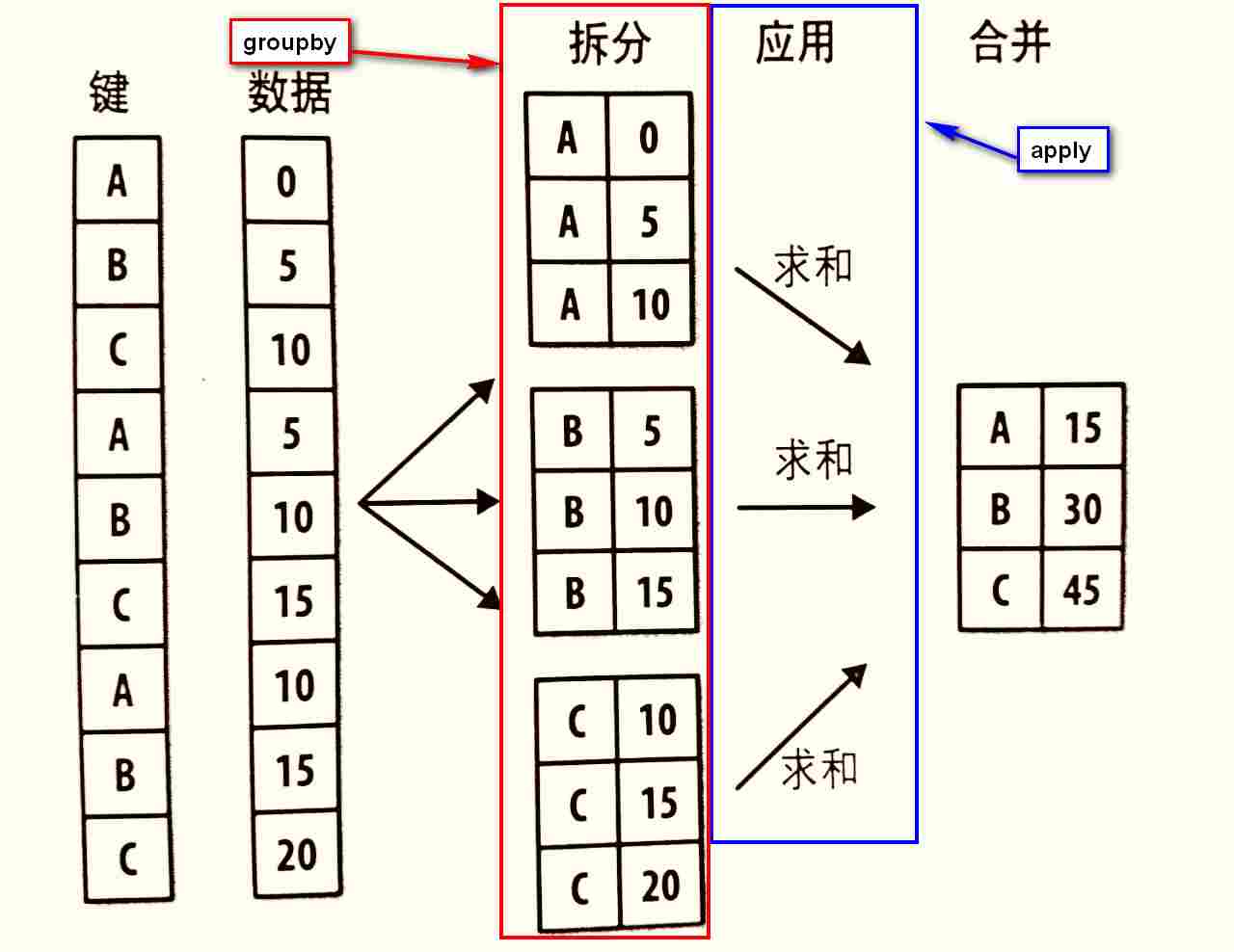
注意：此处的apply函数不是Python的apply函数，而是Pandas的apply函数

apply 是 pandas 库的一个很重要的函数，多和 groupby 函数一起用，也可以直接用于 DataFrame 和 Series 对象。主要用于数据聚合运算，可以很方便的对分组进行现有的运算和自定义的运算。参考：<https://blog.csdn.net/u010099080/article/details/53019844>

函数的传入参数根据axis来定，比如axis = 1，就会把一行数据作为Series的数据结构传入给自己实现的函数中，我们在函数中实现对Series不同属性之间的计算，返回一个结果，则apply函数会自动遍历每一行DataFrame的数据，最后将所有结果组合成一个Series数据结构并返回。

更准确的解释见官方文档：应用apply()的函数，第一个参数必须是DataFrame。返回的结果则是DataFrame、Series或者标量数据。然后将结果拼接起来。

理解：将data的每一行看作一个array，groupby()后如果不指定计算的字段的话，那么f(group)中的group默认传递的是data的每一行，所以书中例子输出结果是在整个names的dataframe形式上再加一列，1690784行5列而非我理解的262行3列了。



举例：自定义函数，为DataFrame新增一列，代表融资额占总融资额（指该城市的总融资额）的比例，然后返回更新后的DataFrame

df2=df1[['车辆所属城市','车辆价格','融资总额']][:100]

df2

|  |  |  |  |
| --- | --- | --- | --- |
|  | 车辆所属城市 | 车辆价格 | 融资总额 |
| 0 | 烟台 | 170000.0 | 146460.0 |
| 1 | 承德 | 135000.0 | 116310.0 |
| …… | …… | …… | …… |
| 99 | 太原 | 62900.0 | 8300.0 |

def add\_prop(data):

loan = data.融资总额.astype(float)

data['prop']=loan/loan.sum()

return data

df3 = df2.groupby('车辆所属城市').apply(add\_prop)

df3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 车辆所属城市 | 车辆价格 | 融资总额 | prop |
| 0 | 烟台 | 170000.0 | 146460.0 | 1.000000 |
| 1 | 承德 | 135000.0 | 116310.0 | 1.000000 |
| …… | …… | …… | …… | …… |
| 99 | 太原 | 62900.0 | 8300.0 | 0.678240 |

解析①：prop=1代表该DataFrame中，该城市的数据就此一行。

如果想求该行数据在总数据（而不是本城市数据）中的比值，则用如下方式

df4=add\_prop(df2)

df4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 车辆所属城市 | 车辆价格 | 融资总额 | prop |
| 0 | 烟台 | 170000.0 | 146460.0 | 0.014831 |
| 1 | 承德 | 135000.0 | 116310.0 | 0.011778 |
| …… | …… | …… | …… | …… |
| 99 | 太原 | 62900.0 | 8300.0 | 0.006369 |

验证：

df4['prop'].sum()

out：1.0000000000000002

解析②：

def add\_prop2(data):

return data

test\_2 = df2.groupby('车辆所属城市').apply(add\_prop2)

test\_2

|  |  |  |  |
| --- | --- | --- | --- |
|  | 车辆所属城市 | 车辆价格 | 融资总额 |
| 0 | 烟台 | 170000.0 | 146460.0 |
| 1 | 承德 | 135000.0 | 116310.0 |
| …… | …… | …… | …… |
| 99 | 太原 | 62900.0 | 8300.0 |

解析③：

def add\_prop3(data):

loan = data.融资总额.astype(float)

return loan

test3=df2.groupby('车辆所属城市').apply(add\_prop3)

test3 （数据格式是Series，大小是100）

|  |  |  |
| --- | --- | --- |
| 上海 | 84 | 115210.0 |
|  | 85 | 131750.0 |
| 东莞 | 98 | 30650.0 |
| 临沂 | 27 | 58580.0 |
|  | 92 | 60308.0 |
| …… | …… | …… |
| 长沙 | 2 | 125538.0 |
|  | 12 | 92170.0 |
|  | 24 | 95460.0 |
|  | 44 | 85980.0 |

备注：说明如果参数只传入groupby()，则输出的还是原数据。但是如果传入计算值，即groupby().字段名，则输出结果采用的是groupby()后的形式。根据解析④，可以知道55个城市，每个城市一个Series

解析④：

def add\_prop4(data):

loan = data.融资总额.astype(float)

return type(loan)

test4=df2.groupby('车辆所属城市').apply(add\_prop4)

test4 （数据格式是Series，大小是55）

|  |  |
| --- | --- |
| 上海 | <class 'pandas.core.series.Series'> |
| 东莞 | <class 'pandas.core.series.Series'> |
| 临沂 | <class 'pandas.core.series.Series'> |
| …… | …… |
| 长春 | <class 'pandas.core.series.Series'> |
| 长沙 | <class 'pandas.core.series.Series'> |

备注：结果共55个城市

解析⑤：

def add\_prop5(data):

loan = data.融资总额.astype(float)

return loan.sum()

test5=df2.groupby('车辆所属城市').apply(add\_prop5)

test5 （数据格式是Series，大小是55）

|  |  |
| --- | --- |
| 上海 | 246960.0 |
| 东莞 | 30650.0 |
| 临沂 | 118888.0 |
| …… | …… |
| 长沙 | 399148.0 |

解析⑥：

prop=test3/test5

prop （数据格式是Series，长度是100）

|  |  |  |
| --- | --- | --- |
| 上海 | 84 | 0.466513 |
|  | 85 | 0.533487 |
| 东莞 | 98 | 1.000000 |
| 临沂 | 27 | 0.492733 |
|  | 92 | 0.507267 |
| …… | …… | …… |
| 长沙 | 2 | 0.314515 |
|  | 12 | 0.230917 |
|  | 24 | 0.239159 |
|  | 44 | 0.215409 |

备注：猜测——长Series/短Series=短Series

解析⑦

loan2=df2.groupby('车辆所属城市').融资总额.astype(float)

loan2

报错提示：

Cannot access callable attribute 'astype' of 'SeriesGroupBy' objects, try using the 'apply' method

备注：说明astype等方法不能应用在groupby()上，只用应用在groupby()里的Series上。而groupby转换为Series，只有传入apply()才行

**# 关于groupby()**

**包括SeriesGroupBy,DataFrameGrouby**

# groupby之后的数据并不是DataFrame格式的数据，而是特殊的groupby类型，此时，可以通过size()方法返回分组后的记录数目统计结果，该结果是Series类型

# 通过groupby完成对数据的分组后，可以通过get\_group方法来获取某一制定分组的结果

对于多层分组后的数据，如果需要使用多重索引，此处需要传入一个元组

# 对于分组后的数据，如果还需要进行一系列查询，可以使用apply方法，传入一个处理的函数

# agg函数提供基于列的聚合操作。而groupby可以看做是基于行，或者说index的聚合操作

# GroupBy对象是一个迭代对象，每次迭代结果是一个元组，元组的第一个元素是该组的名称(就是groupby的列的元素名称)，第二个元素是该组的具体信息，是一个数据框，索引是以前的数据框的总索引

解析：

gb\_test = df2.groupby('车辆所属城市')

for name,data in test:

print(name) # 数据格式是 str

print(data) # 数据格式是 DataFrame

**关于Series**

#创建Series

通过列表创建。创建带索引的Series

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

obj

输出：

d 4

b 7

a -5

c 3

#通过字典来创建Series

sdata={'Ohio':35000,'Texas':71000,'Oregon':16000,'Utah':5000}

obj2=Series(sdata)

obj2

输出：

Ohio 35000

Oregeon 16000

Texas 71000

Utah 5000

#通过字典创建Series时，可以单独提供索引

states=['Califonia','Ohio','Oregon','Texas']

obj3=Series(sdata,index=states)

obj3

输出：

Califonia NaN

Ohio 35000.0

Oregon 16000.0

Texas 71000.0

#pandas的isnull和notnull函数

pd.isnull(obj3)

输出：

Califonia True

Ohio False

Oregon False

Texas False

**# 应用函数·map()** 有点类似于列表推导式

map() 会根据提供的函数对指定序列做映射。第一个参数 function 以参数序列中的每一个元素调用 function 函数，返回包含每次 function 函数返回值的新列表。

举例：取出客户的姓（假设无复姓）

get\_surname=lambda x:x[0]

surname=df1.客户姓名.map(get\_surname)

surname

**应用举例**

1. Excel中有一列‘对话’，统计‘对话’中包含‘再见’字样的总数

|  |  |
| --- | --- |
| **姓名** | 对话 |
| 林炳科 | 机器人：您好，请问您是机主本人吗 |
| 林炳科 | 客户：啊 喂 |
| 林炳科 | 机器人：这里是玖富借贷服务平台，您在我平台申请的万卡借款已经逾期。想问一下，您是什么原因没有处理欠款呢 |
| 林炳科 | 客户：我已经知道了 我等下到家了 十一点多就处理掉。 |

方法一：列表推导式

Import panda as pd

Result = pd.read\_excel(“D:\\test.xlsx”,encoding=’gbk’)

bye = [dialogue for dialogue in result['对话'] if '再见' in dialogue]

len(bye)

方法二：groupby

contain\_bye = np.where(df2['对话'].str.contains('再见'),'包含','不包含')

contain\_bye

输出：array(['不包含', '不包含', '不包含', ..., '不包含', '不包含', '包含'], dtype='<U3')

group\_bye = df2.groupby([contain\_bye])

group\_bye

输出：<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x00000145D5833E48>

bye\_counts = group\_bye.size()

bye\_counts

输出：

不包含 16363

包含 3015