待解决问题

#以pylab模式启动ipython

ipython –pylab

# Pandas读取csv文件的前5行

Import pandas as pd

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

Result

#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

# 读取txt文件的第一行 P21

path=’G:\people.txt’

open(path).readline()

《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[0]

#按键取值

records[0][‘name’]

#打印

print (records[0][‘name’])

#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[:5]

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

#取出现次数前2位的省份及其计数值（方法一） 切片使用参数

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

counts2.most\_common(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[‘name’] #返回的是Series对象，有value\_counts方法

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

city\_counts[:5]

#画图 注意：需要用py—lab模式打开ipython才能绘图

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)

#将省份字段中的“省份、出现次序”分离

results=Series([x.split()[0] for x in frame.name.dropna()])

results

#去除确实数据

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

#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

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

category

#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])

#对分组结果进行计数

area\_counts=by\_catagory.size()

area\_counts

# ? 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’)

#将sales和gender合并

data=pd.merge(sales,gender)

#选择第一行（即表头）

data.ix[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

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

#创建ndarray P84。另见P85 数组创建函数

data2=[[1,2,3,4],[5,6,7,8]]

arr2=np.array(data2)

arr2

#数组的维数、大小、类型

arr2.ndim

arr2.shape

arr2.dtype

#创建新数组的其它方法（需传入元祖）

np.zeros(10)

np.zeros((3,6))

np.ones(10)

np.ones((3,6))

np.empty((2,3,2)) #返回未初始化的垃圾值

np.arange(15)

arr3=np.array([1,2,3],dtype=np.float64)

arr4=np.array([1,2,3],dtype=np.int32)

#显示转换dtype

arr5=np.array([1,2,3,4,5])

arr5.dtype

float\_arr=arr.astype(np.float64)

float\_arr.dtype

#二维数组相乘

arr=[[1,2,3,4],[5,6,7,8]]

arr\*arr

#布尔型索引

#numpy.random中的randn函数生成一些正态分布的随机数据

import numpy as np

names=np.array([‘Bob’,’Joe’,’Will’,’Bob’,’Will’,’Joe’,’Joe’])

data=np.random.randn(7,4)

names

data

输出:

array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'],

dtype='<U4')

array([[ 2.02004929, 2.06316172, -0.56521627, -1.6461729 ],

[ 1.13278707, -0.34356274, 1.3636708 , -0.17933299],

[-0.51877519, -1.66929925, -0.04874752, -0.30386502],

[-3.43409282, 0.44679479, 0.56303836, 0.37520325],

[-0.85777394, 0.30959029, 1.06424552, 1.36146412],

[-1.42247169, -1.23412733, -0.41078545, -0.06144812],

[ 1.63737908, -0.02936644, 0.52327628, 0.51490829]])

#生成布尔数组

names==’Bob’

输出：

array([ True, False, False, True, False, False, False], dtype=bool)

#将布尔数组用作索引

data[names=='Bob']

输出：

array([[ 2.02004929, 2.06316172, -0.56521627, -1.6461729 ],

[-3.43409282, 0.44679479, 0.56303836, 0.37520325]])

#布尔型数组与切片组合

data[names=='Bob',2:]

输出：

array([[-0.56521627, -1.6461729 ],

[ 0.56303836, 0.37520325]])

#布尔型数组与整数组合（降维）

data[names=='Bob',3]

输出：

array([-1.6461729 , 0.37520325])

#用!=或者-号进行否定

names!='Bob'

data[-(names=='Bob')]

#用逻辑运算符组合多个布尔条件

mask=(names=='Bob')|(names=='Will')

mask

输出：

array([ True, False, True, True, True, False, False], dtype=bool)

data[mask]

输出：

array([[ 2.02004929, 2.06316172, -0.56521627, -1.6461729 ],

[-0.51877519, -1.66929925, -0.04874752, -0.30386502],

[-3.43409282, 0.44679479, 0.56303836, 0.37520325],

[-0.85777394, 0.30959029, 1.06424552, 1.36146412]])

#通过布尔型数组设置值

data[data<0]=0

data

输出：

array([[ 2.02004929, 2.06316172, 0. , 0. ],

[ 1.13278707, 0. , 1.3636708 , 0. ],

[ 0. , 0. , 0. , 0. ],

[ 0. , 0.44679479, 0.56303836, 0.37520325],

[ 0. , 0.30959029, 1.06424552, 1.36146412],

[ 0. , 0. , 0. , 0. ],

[ 1.63737908, 0. , 0.52327628, 0.51490829]])

#通过一维布尔数组设置整行或者整列的值

data[names!='Joe']=7

#花式索引（即利用整数数组进行索引）

import numpy as np

arr=np.empty((8,4))

for i in range(8):

arr[i]=i

arr

输出：

array([[ 0., 0., 0., 0.],

[ 1., 1., 1., 1.],

[ 2., 2., 2., 2.],

[ 3., 3., 3., 3.],

[ 4., 4., 4., 4.],

[ 5., 5., 5., 5.],

[ 6., 6., 6., 6.],

[ 7., 7., 7., 7.]])

#以特定顺序选取子集，只需传入一个用于指定顺序的整数列表或ndarray即可

arr[[4,3,0,6]]

输出：

array([[ 4., 4., 4., 4.],

[ 3., 3., 3., 3.],

[ 0., 0., 0., 0.],

[ 6., 6., 6., 6.]])

#负数索引

arr[[-3,-5,-7]]

输出：

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

[ 3., 3., 3., 3.],

[ 1., 1., 1., 1.]])

#一次传入多个索引数组，返回一维数组，其中的元素对应各个索引元祖

#arange：创建等差数组 reshape：用来改变数组的形状

import numpy as np

arr=np.arange(32).reshape((8,4))

arr

输出：

array([[ 0, 1, 2, 3],

[ 4, 5, 6, 7],

[ 8, 9, 10, 11],

[12, 13, 14, 15],

[16, 17, 18, 19],

[20, 21, 22, 23],

[24, 25, 26, 27],

[28, 29, 30, 31]])

#选出（1,0）（5，3）（7,1）（2,2）

arr[[1,5,7,2],[0,3,1,2]]

输出：

array([ 4, 23, 29, 10])

#根据特定顺序选取行子集（方法一）

arr[[1,5,7,2]]

输出：

array([[ 4, 5, 6, 7],

[20, 21, 22, 23],

[28, 29, 30, 31],

[ 8, 9, 10, 11]])

#根据特定顺序选取行列子集

arr[[1,5,7,2]] [:,[0,3,1,2]]

输出：

array([[ 4, 7, 5, 6],

[20, 23, 21, 22],

[28, 31, 29, 30],

[ 8, 11, 9, 10]])

#根据特定顺序选取行子集。方法2：np.ix\_函数

arr[np.ix\_([1,5,7,2],[0,3,1,2])]

输出：

array([[ 4, 7, 5, 6],

[20, 23, 21, 22],

[28, 31, 29, 30],

[ 8, 11, 9, 10]])

#数组转置和轴对换

import numpy as np

arr=np.arange(15).reshape((3,5))

arr

输出：

array([[ 0, 1, 2, 3, 4],

[ 5, 6, 7, 8, 9],

[10, 11, 12, 13, 14]])

arr.T

输出：

array([[ 0, 5, 10],

[ 1, 6, 11],

[ 2, 7, 12],

[ 3, 8, 13],

[ 4, 9, 14]])

#对于高维数组，transpose需要得到一个由轴编号组成的元祖才能对这些轴进行转置（比较费脑子）

arr=np.arange(16).reshape((2,2,4))

arr

输出：

array([[[ 0, 1, 2, 3],

[ 4, 5, 6, 7]],

[[ 8, 9, 10, 11],

[12, 13, 14, 15]]])

arr.transpose((1,0,2))

输出：

array([[[ 0, 1, 2, 3],

[ 8, 9, 10, 11]],

[[ 4, 5, 6, 7],

[12, 13, 14, 15]]])

#用ndarray的swapaxes方法进行转置，它需要接受一堆轴编号

arr

输出：

array([[[ 0, 1, 2, 3],

[ 4, 5, 6, 7]],

[[ 8, 9, 10, 11],

[12, 13, 14, 15]]])

array([[[ 0, 4],

[ 1, 5],

[ 2, 6],

[ 3, 7]],

[[ 8, 12],

[ 9, 13],

[10, 14],

[11, 15]]])

#通用函数（ufunc）：快速的元素级数组函数

#可将其看作简单函数（接受一个或多个标量值，并产生一个或多个标量值）的矢量化包装器

arr=np.arange(10)

arr

输出：

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

#平方根（一元ufunc）

np.sqrt(arr)

输出：

array([ 0. , 1. , 1.41421356, 1.73205081, 2. ,

2.23606798, 2.44948974, 2.64575131, 2.82842712, 3. ])

#计算各元素的指数值（一元ufunc）

arr

输出：

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

np.exp(arr)

输出：

array([ 1.00000000e+00, 2.71828183e+00, 7.38905610e+00,

2.00855369e+01, 5.45981500e+01, 1.48413159e+02,

4.03428793e+02, 1.09663316e+03, 2.98095799e+03,

8.10308393e+03])

#二元ufunc：接受2个数组，并返回一个结果数组

import numpy as np

x=np.random.randn(8)

y=np.random.randn(8)

x

输出：

array([ 1.2520982 , -0.05518781, -0.24558432, 0.62698296, -0.13970339,

0.54865848, -0.22517456, -0.32396117])

y

输出：

array([-0.77455229, -0.70097985, -0.99543926, -2.09528636, 0.51910318,

-0.11726021, -0.65378162, 0.96768174])

np.maximum(x,y)

输出：

array([ 1.2520982 , -0.05518781, -0.24558432, 0.62698296, 0.51910318,

0.54865848, -0.22517456, 0.96768174])

#虽然不常见，单有些unfunc的确可以返回多个数组。modf就是一个例子，他是python内置函数divmod的矢量化版本，用于浮点数数组的小数和整数部分。

#modf——将数组的小数和整数部分一两个独立数组的形式返回

arr=np.random.randn(7)\*5

arr

输出：

array([-5.00312417, -8.40191623, 2.67452386, 1.59263246, -5.73674791,

-4.98385499, -3.29161231])

np.modf(arr)

输出：

(array([-0.00312417, -0.40191623, 0.67452386, 0.59263246, -0.73674791,

-0.98385499, -0.29161231]), array([-5., -8., 2., 1., -5., -4., -3.]))

#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

#将条件逻辑表述为数组计算

#numpy.where函数

#目标：根据cond中的值选取xarr和yarr的值。当cond中的值为True时，选取xarr的值，否则从yarr中选取

#方法一：列表推导式——缺点：对大数据的处理速度不是很快（因为所有的工作都是由纯Python完成的）；无法使用多维数组

import numpy as np

xarr=np.array([1.1,1.2,1.3,1.4,1.5])

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

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

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

result

输出：

[1.1000000000000001, 2.2000000000000002, 1.3, 1.3999999999999999, 2.5]

#方法二：np.where

result2=np.where(cond,xarr,yarr)

result2

输出：

array([ 1.1, 2.2, 1.3, 1.4, 2.5])

#数据分析中，where通常用于根据另一个数组而产生一个新的数组。第二个和第三个参数不必是数组，都可以是标量值。传递给where的数组大小可以不相等，甚至可以是标量值。

#例：将所有正值替换为2，所有负值替换为-2

arr=np.random.randn(4,4)

arr

输出：

array([[ 1.17230228, -0.9317545 , -0.90723008, -1.217438 ],

[-0.17728998, 1.10746329, -0.34702151, 0.23148903],

[ 0.97900074, -0.5683452 , 0.5832163 , 0.60391625],

[-1.13154321, 0.85239267, 0.67204922, -0.84364177]])

np.where(arr>0,2,-2)

输出：

array([[ 2, -2, -2, -2],

[-2, 2, -2, 2],

[ 2, -2, 2, 2],

[-2, 2, 2, -2]])

np.where(arr>0,2,arr) #只将正值设置为2

输出：

array([[ 2. , -0.9317545 , -0.90723008, -1.217438 ],

[-0.17728998, 2. , -0.34702151, 2. ],

[ 2. , -0.5683452 , 2. , 2. ],

[-1.13154321, 2. , 2. , -0.84364177]])

#使用where实现更复杂的逻辑

例：两个布尔型数组cond1和cond2，希望根据4种不同的布尔值组合实现不同的赋值操作。

方法一：

result=[]

for I in range(n):

if cond1[i] and cond2[i]:

result.append(0)

elif cond1[i]:

result.append(1)

elif cond2[i]:

result.append(2)

else:

result.append(3)

方法二：利用“布尔值在计算过程中可以被当做0或1处理”

np.where(cond1 & cond2,0

np.where(cond1,1,

np.where(cond2,2,3)))

方法三：

result=1\*(cond1-cond2)+2\*(cond2 & -cond1)+3\*-(cond2 | cond2)

#数学和统计方法

可以通过数组上的一组数学函数对整个数组或某个轴向的数据进行统计计算。sum、mean等聚合计算（约简）既可以当做数组的实例方法调用，也可以当做顶级NumPy函数使用。

arr=np.random.randn(5,4)

arr

输出：

array([[-0.06333384, 0.10382735, 1.01599912, -0.17017985],

[ 1.21328461, 1.25815776, 0.37067357, -1.57187135],

[ 0.8052596 , 0.50071464, -1.04944458, 0.05256228],

[-1.31048739, 1.73380189, 1.72075152, -1.70624861],

[ 1.82011125, -1.93747568, -1.59409209, 0.34773618]])

arr.mean() #效果等同于 np.mean(arr)

输出：

0.076987319431639362

arr.sum() #数组矩阵中每一个数字的和

输出：

1.5397463886327871

#mean和sum这类函数可以接受一个axis参数（用于计算该轴向上的统计值），最终结果是一个少一维的数组。

arr.mean(axis=1) #以列为轴，计算每一行的均值

输出：

array([ 0.22157819, 0.31756115, 0.07727298, 0.10945435, -0.34093008])

arr.sum(0) #以行为轴，计算每一列的和

输出：

array([ 2.46483424, 1.65902596, 0.46388755, -3.04800135])

#不聚合的函数，返回一个由中间结果组成的数组

arr=np.array([[0,1,2],[3,4,5],[6,7,8]])

arr.cumsum(0) #所有元素的累计和。以行为轴

输出：

array([[ 0, 1, 2],

[ 3, 5, 7],

[ 9, 12, 15]], dtype=int32)

arr.cumprod(1) #所有元素的累计积。以列为轴

输出：

array([[ 0, 0, 0],

[ 3, 12, 60],

[ 6, 42, 336]], dtype=int32)

#用于布尔型数组的方法

在上面这些方法中，布尔值会被强制转换为1（True）和0（False），因此，sum经常被用来对布尔型数组中的True值计数。

arr=np.random.randn(100)

(arr>0).sum()

# any/all 方法

boos=np.array([False,False,True,False])

bools.any()

boos.all()

#数学和统计方法

#就地排序

arr=np.random.randn(8)

arr

arr.sort()

arr

#多维数组可以在任何一个轴向上进行排序，只需要将轴编号传给sort即可

arr=np.random.randn(5,3)

arr

输出：

array([[-0.83820852, 1.25207616, -1.21151998],

[-0.40332523, -0.55758606, 1.78325937],

[-1.2052345 , -0.07900297, 1.0095134 ],

[ 0.94679535, 0.69465436, -0.07644921],

[-2.14187762, 0.05923087, -0.01438769]])

arr.sort(1) #以列为轴排序

arr

输出：

array([[-1.21151998, -0.83820852, 1.25207616],

[-0.55758606, -0.40332523, 1.78325937],

[-1.2052345 , -0.07900297, 1.0095134 ],

[-0.07644921, 0.69465436, 0.94679535],

[-2.14187762, -0.01438769, 0.05923087]])

#计算数组分位数

large\_arr=np.random.randn(1000)

large\_arr.sort()

large\_arr[int(0.05\*len(large\_arr))]

#唯一化及其它的集合逻辑（数组的集合运算） P106

#去重·方法一

import numpy as np

names=np.array(['Bob','Joe','Will','Bob','Will','Joe','Joe'])

np.unique(names)

输出：

array(['Bob', 'Joe', 'Will'],

dtype='<U4')

#去重·方法二

sorted(set(names))

#测试“X的元素是否包含于y”

values=np.array([6,0,0,3,2,5,6])

np.in1d(values,[2,3,6])

输出：

array([ True, False, False, True, True, False, True], dtype=bool)

#用于数组的文件输入输出

#将数组以原始二进制格式保存在.npy文件中

arr=np.arange(10)

np.save('some\_array',arr)

#文件保存在C:\Users\zhangjinxiong

np.load('some\_array.npy')

输出：

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

#通过np.save将多个数组保存到一个压缩文件中。

np.savez('array\_archive.npz',a=arr,b=arr)

#文件保存在C:\Users\zhangjinxiong

#加载npz文件时，你回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

arch=np.load(‘array\_archive.npz’)

arch

输出：

<numpy.lib.npyio.NpzFile at 0x4c27a90>

arch[‘b’]

输出：

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

#存取文本文件的函数：np.loadtxt; np.savetxt; np.genformtxt

#线性代数 P109

矩阵乘法：dot函数

numpy.linalg库

#随机数生成与numpy.random模块生成多种概率分布的样本值的函数

#生成标准正态分布的4x4样本

samples=np.random.normal(size=(4,4))

samples

输出：

array([[-1.19193796, -0.63672387, 0.78873424, -1.52772951],

[ 0.77184161, 0.83124287, 0.61238962, 1.20581098],

[ 0.06487975, 1.57101897, 0.78412559, -1.26460437],

[ 0.10819264, 0.06169641, 0.79296622, -1.54144572]])

#随机漫步 P113

第4章 Pandas入门

#Series

#创建带索引的Series

from pandas import Series,DataFrame

import pandas as pd

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

#DataFrame的构造及其索引 P119

#重新索引 P126

# lambda表达式及apply方法

from pandas import Series,DataFrame

import pandas as pd

import numpy as np

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

#应用到列

frame.apply(f,axis=1)

输出：

Utah 0.719537

Ohio 1.359369

Texas 1.230997

Oregon 0.505771

#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 |

#Series应用元素级函数的map方法

frame['e'].map(format)

输出：

Utah 0.62

Ohio -0.85

Texas 0.57

Oregon 1.54

Name: e, dtype: object

第8章 绘图和可视化

保存图片：savefig(‘D:/myfig.jpg’)

第9章 数据聚合与分组运算

#GroupBy技术

#计算经销商结算价的平均值。方法一：describe（）

import pandas as pd

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

data['经销商结算价'].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

Name: 经销商结算价, dtype: float64

#计算经销商结算价的平均值。方法二：mean（）

data['经销商结算价'].mean()

#计算按照省份分组的经销商结算价的平均值。注意：grouped是一个带有新索引的Series。输出结果中的 “车辆所属省份” 即为新索引。 P265

grouped=data['经销商结算价'].groupby(data['车辆所属省份'])

grouped

grouped.mean()

#按月分组

#按照多列进行分组，形成层次化索引。

means=data['经销商结算价'].groupby([data['客户性别'],data['车辆所属省份']]).mean()

means

#或者简化为

data.groupby(['客户性别','车辆所属省份'])['经销商结算价'].mean()

#将上述结果转换为一层索引，转换为DataFrame

**means.unstack()**

#计算8万以下车的数量

pd.value\_counts(data['经销商结算价']<=80000,sort=False)

输出：

False 28464

True 10505

Name: 经销商结算价, dtype: int64

#计算贷款额占车价的百分比

data['pct']=data['融资总额（含超融）']/data['经销商结算价']

data[:5]

#每个月8万以下车的比重（尤其是8万以下车活动上线前后的对比）

#面向列的多函数应用

#计算根据省份分组的 “经销商结算价” 的总和、均值、标准差。也可传入自定义函数

（说白了还是针对同一列）

import pandas as pd

import numpy as np

from pandas import Series,DataFrame

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

grouped=data.groupby(['车辆所属省份'])

grouped\_car=grouped['经销商结算价']

grouped\_car.agg(['sum','mean','std'])

输出：

|  | **sum** | **mean** | **std** |
| --- | --- | --- | --- |
| **车辆所属省份** |  |  |  |
| **上海** | 198964400.0 | 178123.903312 | 115132.277718 |
| **云南** | 127372000.0 | 186489.019034 | 124584.145372 |
| **……** | …… | …… | …… |

#自定义列名称。传入由（name,function）元祖组成的列表

grouped\_car.agg([('车辆总价','sum'),('平均车辆价格','mean')])

输出：

|  | **车辆总价** | **平均车辆价格** |
| --- | --- | --- |
| **车辆所属省份** |  |  |
| **上海** | 198964400.0 | 178123.903312 |
| **云南** | 127372000.0 | 186489.019034 |

#对分组后的不同列应用相同的函数

#计算不同资金渠道，不同业务期限的“首付款金额（元）”和“经销商结算价”的计数、平均值和最大值

import pandas as pd

import numpy as np

from pandas import Series,DataFrame

data=pd.read\_csv('D:\\xin.csv',encoding='gbk')

grouped=data.groupby(['资金渠道','业务期限'])

functions=['count','mean','max']

result=grouped['首付款金额（元）','经销商结算价'].agg(functions)

result

输出：

|  |  | **首付款金额（元）** | | | **经销商结算价** | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **count** | **mean** | **max** | **count** | **mean** | **max** |
| **资金渠道** | **业务期限** |  |  |  |  |  |  |
| **凯枫** | **24个月** | 72 | 89045.694444 | 250800.0 | 72 | 202601.388889 | 498000.0 |
| **36个月** | 162 | 74141.481481 | 258050.0 | 162 | 188726.543210 | 500000.0 |
| **微众** | **21个月** | 1 | 86300.000000 | 86300.0 | 1 | 169000.000000 | 169000.0 |
| **23个月** | 1 | 71800.000000 | 71800.0 | 1 | 140000.000000 | 140000.0 |
| **24个月** | 14560 | 69017.260989 | 464800.0 | 14560 | 154661.964286 | 650000.0 |
| **33个月** | 3 | 71580.000000 | 169300.0 | 3 | 150933.333333 | 330000.0 |
| **34个月** | 1 | 73800.000000 | 73800.0 | 1 | 180000.000000 | 180000.0 |
| **36个月** | 18428 | 65725.851422 | 529930.0 | 18428 | 159706.734317 | 650000.0 |
| **稠州** | **24个月** | 2268 | 63420.427690 | 434800.0 | 2268 | 152855.996473 | 650000.0 |
| **36个月** |  |  |  |  |  |  |

#相当于对各列进行聚合，然后用concat将结果组装到一起

Result[‘经销商结算价’]

输出：

|  |  | **count** | **mean** | **max** |
| --- | --- | --- | --- | --- |
| **资金渠道** | **业务期限** |  |  |  |
| **凯枫** | **24个月** | 72 | 202601.388889 | 498000.0 |
| **36个月** | 162 | 188726.543210 | 500000.0 |
| **微众** | **21个月** | 1 | 169000.000000 | 169000.0 |
| **23个月** | 1 | 140000.000000 | 140000.0 |
| **24个月** | 14560 | 154661.964286 | 650000.0 |
| **33个月** | 3 | 150933.333333 | 330000.0 |
| **34个月** | 1 | 180000.000000 | 180000.0 |
| **36个月** | 18428 | 159706.734317 | 650000.0 |
| **稠州** | **24个月** | 2268 | 152855.996473 | 650000.0 |
| **36个月** | 3469 | 156850.821562 | 638000.0 |

#也可以自定义列名：通过传入带有自定义名称的元组列表

ftuples=[('计数','count'),('平均数','mean')]

grouped['首付款金额（元）','经销商结算价'].agg(ftuples)

输出：

|  |  | **首付款金额（元）** | | **经销商结算价** | |
| --- | --- | --- | --- | --- | --- |
|  |  | **计数** | **平均数** | **计数** | **平均数** |
| **资金渠道** | **业务期限** |  |  |  |  |
| **凯枫** | **24个月** | 72 | 89045.694444 | 72 | 202601.388889 |
| **36个月** | 162 | 74141.481481 | 162 | 188726.543210 |
| **微众** | **21个月** | 1 | 86300.000000 | 1 | 169000.000000 |
| **23个月** | 1 | 71800.000000 | 1 | 140000.000000 |
| **24个月** | 14560 | 69017.260989 | 14560 | 154661.964286 |
| **33个月** | 3 | 71580.000000 | 3 | 150933.333333 |
| **34个月** | 1 | 73800.000000 | 1 | 180000.000000 |
| **36个月** | 18428 | 65725.851422 | 18428 | 159706.734317 |
| **稠州** | **24个月** | 2268 | 63420.427690 | 2268 | 152855.996473 |
| **36个月** | 3469 | 60099.573364 | 3469 | 156850.821562 |

#对不同的列应用不同的函数。具体的办法是想agg传入一个从列名映射到函数的字典。

grouped.agg({'首付款金额（元）':['mean','std'],'经销商结算价':'sum'})

输出：

|  |  | **首付款金额（元）** | | **经销商结算价** |
| --- | --- | --- | --- | --- |
|  |  | **mean** | **std** | **sum** |
| **资金渠道** | **业务期限** |  |  |  |
| **凯枫** | **24个月** | 89045.694444 | 60258.385351 | 1.458730e+07 |
| **36个月** | 74141.481481 | 48140.015744 | 3.057370e+07 |
| **微众** | **21个月** | 86300.000000 | NaN | 1.690000e+05 |
| **23个月** | 71800.000000 | NaN | 1.400000e+05 |
| **24个月** | 69017.260989 | 61395.047093 | 2.251878e+09 |
| **33个月** | 71580.000000 | 84633.003019 | 4.528000e+05 |
| **34个月** | 73800.000000 | NaN | 1.800000e+05 |
| **36个月** | 65725.851422 | 50596.335425 | 2.943076e+09 |
| **稠州** | **24个月** | 63420.427690 | 56350.704781 | 3.466774e+08 |
| **36个月** | 60099.573364 | 42620.798302 | 5.441155e+08 |

#以无索引的形式返回聚合数据——即将层次化索引转为为普通的列

data.groupby(['资金渠道','业务期限'],as\_index=False).mean()

或者：

data.groupby(['资金渠道','业务期限'],as\_index=False)['经销商结算价'].mean()

输出：

|  | **资金渠道** | **业务期限** | **经销商结算价** |
| --- | --- | --- | --- |
| **0** | 凯枫 | 24个月 | 202601.388889 |
| **1** | 凯枫 | 36个月 | 188726.543210 |
| **2** | 微众 | 21个月 | 169000.000000 |
| **3** | 微众 | 23个月 | 140000.000000 |
| **4** | 微众 | 24个月 | 154661.964286 |
| **5** | 微众 | 33个月 | 150933.333333 |
| **6** | 微众 | 34个月 | 180000.000000 |
| **7** | 微众 | 36个月 | 159706.734317 |
| **8** | 稠州 | 24个月 | 152855.996473 |
| **9** | 稠州 | 36个月 | 156850.821562 |

#分组级运算和转换——即“面向分组”的计算

#将大DataFrame拆分为小DataFrame

import pandas as pd

import numpy as np

from pandas import Series,DataFrame

data=pd.read\_csv('D:\\xin.csv',encoding='gbk')

data2=data[['资金渠道','业务期限','首付款金额（元）','经销商结算价']]

#先按照“资金渠道”分组，然后添加一列，为“首付款金额”和“经销商结算价”的平均值

#方法一：先聚合再合并，即将计算的结果加在原来的数据上

new\_col=data2.groupby('资金渠道').mean().add\_prefix('mean\_')

new\_col

输出：

|  | **mean\_首付款金额（元）** | **mean\_经销商结算价** |
| --- | --- | --- |
| **资金渠道** |  |  |
| **凯枫** | 78727.393162 | 192995.726496 |
| **微众** | 67179.909984 | 157480.017579 |
| **稠州** | 61412.401952 | 155271.553077 |

pd.merge(data2,new\_col,left\_on='资金渠道',right\_index=True)

输出：（所有的行）

|  | **资金渠道** | **业务期限** | **首付款金额（元）** | **经销商结算价** | **mean\_首付款金额（元）** | **mean\_经销商结算价** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 微众 | 24个月 | 58750.0 | 113900.0 | 67179.909984 | 157480.017579 |
| **1** | 微众 | 24个月 | 90610.0 | 268000.0 | 67179.909984 | 157480.017579 |
| **2** | 微众 | 36个月 | 215930.0 | 438000.0 | 67179.909984 | 157480.017579 |

#方法二：使用transform。但是这样没显示出对应的主键（解决办法：先将某一列置为索引）

data2.groupby('资金渠道').transform(np.mean)

输出：（所有的行）

|  | **首付款金额（元）** | **经销商结算价** |
| --- | --- | --- |
| **0** | 67179.909984 | 157480.017579 |
| **1** | 67179.909984 | 157480.017579 |
| **2** | 67179.909984 | 157480.017579 |

#使用transform（）应用函数

#距平化函数

data2.index=data2['资金渠道'] #将资金渠道置为索引。以方便后续检验

def demean(arr):

return arr-arr.mean()

demeaned=data2.groupby('业务期限').transform(demean)

demeaned

输出：

|  | **首付款金额（元）** | **经销商结算价** |
| --- | --- | --- |
| **资金渠道** |  |  |
| **微众** | -9601.487574 | -40723.840237 |
| **微众** | 22258.512426 | 113376.159763 |
| **微众** | 151027.133596 | 278529.266966 |
| **微众** | 110898.512426 | 163376.159763 |

检验：

#先将“业务期限”一列加进来——检验失败

pd.merge(demeaned,data2[‘资金渠道’])