课件代码：请结合课件一起看

2. Series对象

import pandas as pd

from pandas import Series,DataFrame

import numpy as np

# 创建Series对象并省略索引

'''

index 参数是可省略的，你可以选择不输入这个参数。

如果不带 index 参数，Pandas 会自动用默认 index 进行索引，类似数组，索引值是 [0, ..., len(data) - 1]

'''

sel = Series([1,2,3,4])

print(sel)

# 通常我们会自己创建索引

# sel = Series(data = [1,2,3,4], index = ['a','b','c','d'])

sel = Series(data = [1,2,3,4], index = list('abcd'))

print(sel)

# 获取内容

print(sel.values)

# 获取索引

print(sel.index)

# 获取索引和值对

print(list(sel.iteritems()))

# 将字典转换为Series

dict={"red":100,"black":400,"green":300,"pink":900}

se3=Series(dict)

print(se3)

# Series数据获取

sel = Series(data = [1,2,3,4], index = list('abcd'))

print(sel)

# Series对象同时支持位置和标签两种方式获取数据

print('索引下标',sel['c'])

print('位置下标',sel[2])

# 获取不连续的数据

print('索引下标',sel[['a','c']])

print('位置下标',sel[[1,3]])

# 可以使用切片或取数据

print('位置切片',sel[1:3])# 左包含右不包含

print('索引切片',sel['b':'d'])# 左右都包含

# 重新赋值索引的值

sel.index = list('dcba')

print(sel)

# ReIndex重新索引,会返回一个新的Series(调用reindex将会重新排序，缺失值则用NaN填补)

print(sel.reindex(['b','a','c','d','e']))

# Drop丢弃指定轴上的项

se1=pd.Series(range(10,15))

print(se1)

print(se1.drop([2,3]))

```

\*\*Series 进行算术运算操作\*\*

'''

对 Series 的算术运算都是基于 index 进行的。

我们可以用加减乘除（+ - \* /）这样的运算符对两个 Series 进行运算，

Pandas 将会根据索引 index，对响应的数据进行计算，结果将会以浮点数的形式存储，以避免丢失精度。

如果 Pandas 在两个 Series 里找不到相同的 index，对应的位置就返回一个空值 NaN

'''

series1 = pd.Series([1,2,3,4],['London','HongKong','Humbai','lagos'])

series2 = pd.Series([1,3,6,4],['London','Accra','lagos','Delhi'])

print(series1-series2)

print(series1+series2)

print(series1\*series2)

# 同样也支持numpy的数组运算

sel = Series(data = [1,6,3,5], index = list('abcd'))

print(sel[sel>3]) # 布尔数组过滤

print(sel\*2) # 标量乘法

print(np.square(sel)) # 可以直接加入到numpy的数学函数

```

## 3. DataFrame的创建

### 3.1 DataFrame的创建

# 1. 创建DataFrame

# 使用二维数组

df1 = DataFrame(np.random.randint(0,10,(4,4)),index=[1,2,3,4],columns=['a','b','c','d'])

print(df1)

# 使用字典创建(行索引由index决定，列索引由字典的键决定)

dict={

'Province': ['Guangdong', 'Beijing', 'Qinghai', 'Fujian'],

'pop': [1.3, 2.5, 1.1, 0.7],

'year': [2018, 2018, 2018, 2018]}

df2=pd.DataFrame(dict,index=[1,2,3,4])

print(df2)

# 使用from\_dict

dict2={"a":[1,2,3],"b":[4,5,6]}

df6=pd.DataFrame.from\_dict(dict2)

print(df6)

#索引相同的情况下，相同索引的值会相对应，缺少的值会添加NaN

data = {

'Name':pd.Series(['zs','ls','we'],index=['a','b','c']),

'Age':pd.Series(['10','20','30','40'],index=['a','b','c','d']),

'country':pd.Series(['中国','日本','韩国'],index=['a','c','b'])

}

df = pd.DataFrame(data)

print(df)

# to\_dict()方法将DataFrame对象转换为字典

dict = df.to\_dict()

print(dict)

```

### 3.2 DataFrame对象常用属性

import pandas as pd

from pandas import Series,DataFrame

import numpy as np

# dataframe常用属性

df\_dict = {

'name':['James','Curry','Iversion'],

'age':['18','20','19'],

'national':['us','China','us']

}

df = pd.DataFrame(data=df\_dict,index=['0','1','2'])

print(df)

# 获取行数和列数

print(df.shape)

# # 获取行索引

print(df.index.tolist())

# # 获取列索引

print(df.columns.tolist())

# 获取数据的类型

print(df.dtypes)

# 获取数据的维度

print(df.ndim)

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

print(df.values)

# 展示df的概览

print(df.info())

# 显示头几行,默认显示5行

print(df.head(2))

# 显示后几行

print(df.tail(1))

# 获取DataFrame的列

print(df['name'])

#因为我们只获取一列，所以返回的就是一个 Series

print(type(df['name']))

# 如果获取多个列，那返回的就是一个 DataFrame 类型：

print(df[['name','age']])

print(type(df[['name','age']]))

# 获取一行

print(df[0:1])

# 去多行

print(df[1:3])

# 取多行里面的某一列（不能进行多行多列的选择）

print(df[1:3][['name','age']])

# 注意： df[]只能进行行选择，或列选择，不能同时多行多列选择。

'''

df.loc 通过标签索引行数据

df.iloc 通过位置获取行数据

'''

# 获取某一行某一列的数据

print(df.loc['0','name'])

# 一行所有列

print(df.loc['0',:])

# 某一行多列的数据

print(df.loc['0',['name','age']])

# 选择间隔的多行多列

print(df.loc[['0','2'],['name','national']])

# 选择连续的多行和间隔的多列

print(df.loc['0':'2',['name','national']])

# 取一行

print(df.iloc[1])

# 取连续多行

print(df.iloc[0:2])

# 取间断的多行

print(df.iloc[[0,2],:])

# 取某一列

print(df.iloc[:,1])

# 某一个值

print(df.iloc[1,0])

# 修改值

df.iloc[0,0]='panda'

print(df)

# dataframe中的排序方法

df = df.sort\_values(by='age',ascending=False)

# ascending=False ： 降序排列，默认是升序

print(df)

```

### 3.3 dataframe修改index、columns

df1 = pd.DataFrame(np.arange(9).reshape(3, 3), index = ['bj', 'sh', 'gz'], columns=['a', 'b', 'c'])

print(df1)

# 修改 df1 的 index

print(df1.index) # 可以打印出print的值，同时也可以为其赋值

df1.index = ['beijing', 'shanghai', 'guangzhou']

print(df1)

# 自定义map函数（x是原有的行列值）

def test\_map(x):

return x+'\_ABC'

# inplace：布尔值，默认为False。指定是否返回新的DataFrame。如果为True，则在原df上修改，返回值为None。

print(df1.rename(index=test\_map, columns=test\_map, inplace=True))

# 同时，rename 还可以传入字典，为某个 index 单独修改名称

df3 = df1.rename(index={'bj':'beijing'}, columns = {'a':'aa'})

print(df3)

# 列转化为索引

df1=pd.DataFrame({'X':range(5),'Y':range(5),'S':list("abcde"),'Z':[1,1,2,2,2]})

print(df1)

# 指定一列为索引 (drop=False 指定同时保留作为索引的列)

result = df1.set\_index('S',drop=False)

result.index.name=None

print(result)

# 行转为列索引

result = df1.set\_axis(df1.iloc[0],axis=1,inplace=False)

result.columns.name=None

print(result)

```

### 3.4 添加数据

# 增加数据

df1 = pd.DataFrame([['Snow','M',22],['Tyrion','M',32],['Sansa','F',18],['Arya','F',14]],

columns=['name','gender','age'])

# 在数据框最后加上score一列

df1['score']=[80,98,67,90] # 增加列的元素个数要跟原数据列的个数一样

print(df1)

# 在具体某个位置插入一列可以用insert的方法

# 语法格式：列表.insert(index, obj)

# index --->对象 obj 需要插入的索引位置。

# obj ---> 要插入列表中的对象（列名）

col\_name=df1.columns.tolist() # 将数据框的列名全部提取出来存放在列表里

col\_name.insert(2,'city') # 在列索引为2的位置插入一列,列名为:city，刚插入时不会有值，整列都是NaN

df1=df1.reindex(columns=col\_name) # DataFrame.reindex() 对原行/列索引重新构建索引值

print(df1)

df1['city']=['北京','山西','湖北','澳门'] # 给city列赋值

print(df1)

# df中的insert,插入一列

'''

df.insert(iloc,column,value)

iloc:要插入的位置

colunm:列名

value:值

'''

df1.insert(2,'score',[80,98,67,90])

print(df1)

# 插入一行

row=['111','222','333']

df1.iloc[1]=row

print(df1)

# 增加数据

df1 = pd.DataFrame([['Snow','M',22],['Tyrion','M',32],['Sansa','F',18],['Arya','F',14]],

columns=['name','gender','age'])

# 先创建一个DataFrame，用来增加进数据框的最后一行

new=pd.DataFrame({'name':'lisa',

'gender':'F',

'age':19

},index=[0])

print(new)

# print("-------在原数据框df1最后一行新增一行，用append方法------------")

df1=df1.append(new,ignore\_index=True) # ignore\_index=False,表示不按原来的索引，从0开始自动递增

print(df1)

# 合并

'''

objs:合并对象

axis:合并方式，默认0表示按列合并，1表示按行合并

ignore\_index:是否忽略索引

'''

df1 = pd.DataFrame(np.arange(6).reshape(3,2),columns=['four','five'])

df2 = pd.DataFrame(np.arange(6).reshape(2,3),columns=['one','two','three'])

print(df2)

# # 按行合并

result = pd.concat([df1,df2],axis=1)

print(result)

# # 按列合并

result = pd.concat([df1,df2],axis=0,ignore\_index=True)

print(result)

# DataFrame的删除

'''

lables：要删除数据的标签

axis：0表示删除行，1表示删除列，默认0

inplace:是否在当前df中执行此操作

'''

df2 = pd.DataFrame(np.arange(9).reshape(3,3),columns=['one','two','three'])

print(df2)

df3=df2.drop(['one'],axis=1, inplace=True)

# df3=df2.drop([0,1],axis=0, inplace=False)

print(df2)

print(df3)

## 4. 数据处理

```python

from numpy import nan as NaN

# 通过\*\*dropna()\*\*滤除缺失数据：

se=pd.Series([4,NaN,8,NaN,5])

# print(se)

# print(se.dropna())

# print(se.notnull())

# print(se.isnull())

# # 通过布尔序列也能滤除：

# print(se[se.notnull()])

# 2.2 处理DataFrame对象

df1=pd.DataFrame([[1,2,3],[NaN,NaN,2],[NaN,NaN,NaN],[8,8,NaN]])

# print(df1)

# 默认滤除所有包含NaN：

# print(df1.dropna())

# 传入how=‘all’滤除全为NaN的行：

# print(df1.dropna(how='all')) # 默认情况下是how='any'，只要有nan就删除

# 传入axis=1滤除列：

# print(df1.dropna(axis=1,how="all"))

#传入thresh=n保留至少有n个非NaN数据的行：

# print(df1.dropna(thresh=1))

# 2.3 填充缺失数据

df1=pd.DataFrame([[1,2,3],[NaN,NaN,2],[NaN,NaN,NaN],[8,8,NaN]])

# print(df1)

# 用常数填充fillna

# print(df1.fillna(0))

#传入inplace=True直接修改原对象：

# df1.fillna(0,inplace=True)

# print(df1)

# 通过字典填充不同的常数

# print(df1.fillna({0:10,1:20,2:30}))

# 填充平均值

print(df1.fillna(df1.mean()))

# 如果只填充一列

print(df1.iloc[:,1].fillna(5,inplace = True))

print(df1)

# 传入method=” “改变插值方式：

df2=pd.DataFrame(np.random.randint(0,10,(5,5)))

df2.iloc[1:4,3]=NaN

df2.iloc[2:4,4]=NaN

# print(df2)

#用前面的值来填充ffill 用后面的值来填充bfill

# print(df2.fillna(method='ffill'))

# 传入limit=” “限制填充行数：

# print(df2.fillna(method='bfill',limit=1))

# 传入axis=” “修改填充方向：

# print(df2.fillna(method="ffill",limit=1,axis=1))

# 2.4 移除重复数据

'''

DataFrame中经常会出现重复行，利用duplicated()函数返回每一行判断是否重复的结果（重复则为True)

'''

df1=pd.DataFrame({'A':[1,1,1,2,2,3,1],'B':list("aabbbca")})

print(df1)

# 判断每一行是否重复(结果是bool值，TRUE代表重复的)

# print(df1.duplicated())

# 去除全部的重复行

# print(df1.drop\_duplicates())

# # 指定列去除重复行

# print(df1.drop\_duplicates(['A']))

# 保留重复行中的最后一行

# print(df1.drop\_duplicates(['A'],keep='last'))

# 去除重复的同时改变DataFrame对象

# df1.drop\_duplicates(['A','B'],inplace=True)

# print(df1)

```

## 5. 数据合并

```python

# 使用join合并，着重关注的是行的合并

import pandas as pd

df3=pd.DataFrame({'Red':[1,3,5],'Green':[5,0,3]},index=list('abc'))

df4=pd.DataFrame({'Blue':[1,9,8],'Yellow':[6,6,7]},index=list('cde'))

print(df3)

print(df4)

# 简单合并（默认是left左连接,以左侧df3为基础）

df3.join(df4,how='left')

# 右链接

df3.join(df4,how='right')

# 外链接

df3.join(df4,how='outer')

# 合并多个DataFrame对象

# df5=pd.DataFrame({'Brown':[3,4,5],'White':[1,1,2]},index=list('aed'))

# df3.join([df4,df5])

```

```python

# 使用merge，着重关注的是列的合并

df1=pd.DataFrame({'名字':list('ABCDE'),'性别':['男','女','男','男','女'],'职称':['副教授','讲师','助教','教授','助教']},index=range(1001,1006))

df1.columns.name='学院老师'

df1.index.name='编号'

print(df1)

df2=pd.DataFrame({'名字':list('ABDAX'),'课程':['C++','计算机导论','汇编','数据结构','马克思原理'],'职称':['副教授','讲师','教授','副教授','讲师']},index=[1001,1002,1004,1001,3001])

df2.columns.name='课程'

df2.index.name='编号'

print(df2)

# 默认下是根据左右对象中出现同名的列作为连接的键，且连接方式是how=’inner’

# print(pd.merge(df1,df2))# 返回匹配的

# 指定列名合并

pd.merge(df1,df2,on='名字',suffixes=['\_1','\_2'])# 返回匹配的

# 连接方式，根据左侧为准

pd.merge(df1,df2,how='left')

# 根据左侧为准

pd.merge(df1,df2,how='right')

# 所有

# pd.merge(df1,df2,how='outer')

# 根据多个键进行连接

pd.merge(df1,df2,on=['职称','名字'])

```

\*\*拓展\*\*

```python

# 轴向连接-Concat

# 1. Series对象的连接

# s1=pd.Series([1,2],index=list('ab'))

# s2=pd.Series([3,4,5],index=list('bde'))

# print(s1)

# print(s2)

# pd.concat([s1,s2])

#横向连接

# pd.concat([s1,s2],axis=1)

# 用内连接求交集(连接方式，共有’inner’,’left’,right’,’outer’)

# pd.concat([s1,s2],axis=1,join='inner')

# 指定部分索引进行连接

# pd.concat([s1,s2],axis=1,join\_axes=[list('abc')])

# 创建层次化索引

# pd.concat([s1,s2],keys=['A','B'])

#当纵向连接时keys为列名

# pd.concat([s1,s2],keys=['A','D'],axis=1)

# 2. DataFrame对象的连接

df3=pd.DataFrame({'Red':[1,3,5],'Green':[5,0,3]},index=list('abd'))

df4=pd.DataFrame({'Blue':[1,9],'Yellow':[6,6]},index=list('ce'))

print(df3)

print(df4)

# pd.concat([df3,df4])

# pd.concat([df3,df4],axis=1,keys=['A','B'])

# 用字典的方式连接同样可以创建层次化列索引

# pd.concat({'A':df3,'B':df4},axis=1)

```

## 6. 多层索引（拓展）

\*\*创建多层索引\*\*

```python

import numpy as np

import pandas as pd

from pandas import Series,DataFrame

# Series也可以创建多层索引

# s = Series(np.random.randint(0,150,size=6),index=list('abcdef'))

# print(s)

# s = Series(np.random.randint(0,150,size=6),

# index=[['a','a','b','b','c','c'],['期中','期末','期中','期末','期中','期末']])

# print(s)

# DataFrame创建多层索引

# df1 = DataFrame(np.random.randint(0,150,size=(6,4)),

# columns = ['zs','ls','ww','zl'],

# index = [['python','python','math','math','En','En'],['期中','期末','期中','期末','期中','期末']])

# print(df1)

# 2. 特定结构

# class1=['python','python','math','math','En','En']

# class2=['期中','期末','期中','期末','期中','期末']

# m\_index2=pd.MultiIndex.from\_arrays([class1,class2])

# df2=DataFrame(np.random.randint(0,150,(6,4)),index=m\_index2)

# print(df2)

# class1=['期中','期中','期中','期末','期末','期末']

# class2=['python','math','En','python','math','En']

# m\_index2=pd.MultiIndex.from\_arrays([class1,class2])

# df2=DataFrame(np.random.randint(0,150,(6,4)),index=m\_index2)

# print(df2)

# 3. product构造

class1=['python','math','En']

class2=['期中','期末']

m\_index2=pd.MultiIndex.from\_product([class1,class2])

df2=DataFrame(np.random.randint(0,150,(6,4)),index=m\_index2)

print(df2)

```

\*\*多层索引对象的索引\*\*

```python

#多层索引对象的索引操作

# series

# s = Series(np.random.randint(0,150,size=6),

# index=[['a','a','b','b','c','c'],['期中','期末','期中','期末','期中','期末']])

# print(s)

# 取一个第一级索引

# print(s['a'])

# 取多个第一级索引

# print(s[['a','b']])

# 根据索引获取值

# print(s['a','期末'])

# loc方法取值

# print(s.loc['a'])

# print(s.loc[['a','b']])

# print(s.loc['a','期末'])

# iloc方法取值(iloc计算的事最内层索引)

# print(s.iloc[1])

# print(s.iloc[1:4])

# dataframe

class1=['python','math','En']

class2=['期中','期末']

m\_index2=pd.MultiIndex.from\_product([class1,class2])

df2=DataFrame(np.random.randint(0,150,(6,4)),index=m\_index2)

print(df2)

# 获取列

# print(df2[0])

# 一级索引

# print(df2.loc['python'])

# 多个一级索引

# print(df2.loc[['python','math']])

# 取一行

# print(df2.loc['python','期末'])

# 取一值

# print(df2.loc['python','期末'][0])

# iloc是只取最内层的索引的

#print(df2.iloc[0])

```

## 7. 时间序列

```python

import pandas as pd

import numpy as np

# 1. 生成一段时间范围

'''

该函数主要用于生成一个固定频率的时间索引，在调用构造方法时，必须指定start、end、periods中的两个参数值，否则报错。

时间序列频率：

D 日历日的每天

B 工作日的每天

H 每小时

T或min 每分钟

S 每秒

L或ms 每毫秒

U 每微秒

M 日历日的月底日期

BM 工作日的月底日期

MS 日历日的月初日期

BMS 工作日的月初日期

'''

# date = pd.date\_range(start='20190501',end='20190530')

# print(date)

# freq：日期偏移量，取值为string, 默认为'D'， freq='1h30min' freq='10D'

# periods：固定时期，取值为整数或None

# date = pd.date\_range(start='20190501',periods=10,freq='10D')

# print(date)

'''

根据closed参数选择是否包含开始和结束时间closed=None，left包含开始时间，不包含结束时间，right与之相反。

'''

data\_time =pd.date\_range(start='2019-01-09',end='2019-01-14',closed='left')

print(data\_time)

# 2. 时间序列在dataFrame中的作用

# 可以将时间作为索引

index = pd.date\_range(start='20190101',periods=10)

df = pd.Series(np.random.randint(0,10,size = 10),index=index)

print(df)

# truncate这个函数将before指定日期之前的值全部过滤出去,after指定日期之前的值全部过滤出去.

after = df.truncate(after='2019-01-8')

print(after)

long\_ts = pd.Series(np.random.randn(1000),index=pd.date\_range('1/1/2019',periods=1000))

# print(long\_ts)

# 根据年份获取

# result = long\_ts['2020']

# print(result)

# 年份和日期获取

# result = long\_ts['2020-05']

# print(result)

# 使用切片

# result = long\_ts['2020-05-01':'2020-05-06']

# print(result)

# 通过between\_time()返回位于指定时间段的数据集

# index=pd.date\_range("2018-03-17","2018-03-30",freq="2H")

# ts = pd.Series(np.random.randn(157),index=index)

# print(ts.between\_time("7:00","17:00"))

# 这些操作也都适用于dataframe

# index=pd.date\_range('1/1/2019',periods=100)

# df = pd.DataFrame(np.random.randn(100,4),index=index)

# print(df.loc['2019-04'])

# 6. 移位日期

ts = pd.Series(np.random.randn(10),index=pd.date\_range('1/1/2019',periods=10))

print(ts)

# 移动数据，索引不变，默认由NaN填充

# periods: 移动的位数 负数是向上移动

# fill\_value: 移动后填充数据

# freq： 日期偏移量

ts.shift(periods=2,fill\_value=100，freq='D')

# 通过tshift()将索引移动指定的时间：

ts.tshift(2)

# 将时间戳转化成时间根式

pd.to\_datetime(1554970740000,unit='ms')

# utc是协调世界时,时区是以UTC的偏移量的形式表示的,但是注意设置utc=True,是让pandas对象具有时区性质,对于一列进行转换的,会造成转换错误

# unit='ms' 设置粒度是到毫秒级别的

# 时区名字

# import pytz

# print(pytz.common\_timezones)

2019-5-1

pd.to\_datetime(1554970740000,unit='ms').tz\_localize('UTC').tz\_convert('Asia/Shanghai')

# 处理一列

df = pd.DataFrame([1554970740000, 1554970800000, 1554970860000],columns = ['time\_stamp'])

pd.to\_datetime(df['time\_stamp'],unit='ms').dt.tz\_localize('UTC').dt.tz\_convert('Asia/Shanghai')#先赋予标准时区,再转换到东八区

# 处理中文

pd.to\_datetime('2019年10月10日',format='%Y年%m月%d日')

```

## 8. 分组聚合

```python

# 分组

import pandas as pd

import numpy as np

df=pd.DataFrame({

'name':['BOSS','Lilei','Lilei','Han','BOSS','BOSS','Han','BOSS'],

'Year':[2016,2016,2016,2016,2017,2017,2017,2017],

'Salary':[999999,20000,25000,3000,9999999,999999,3500,999999],

'Bonus':[100000,20000,20000,5000,200000,300000,3000,400000]

})

# print(df)

# 根据name这一列进行分组

# group\_by\_name=df.groupby('name')

# print(type(group\_by\_name))

# 查看分组

# print(group\_by\_name.groups)

# 分组后的数量

# print(group\_by\_name.count())

# 产看分组的情况

# for name,group in group\_by\_name:

# print(name)# 组的名字

# print(group)# 组具体内容

# 可以选择分组

# print(group\_by\_name.get\_group('BOSS'))

# 按照某一列进行分组, 将name这一列作为分组的键，对year进行分组

# group\_by\_name=df['Year'].groupby(df['name'])

# print(group\_by\_name.count())

# 按照多列进行分组

# group\_by\_name\_year=df.groupby(['name','Year'])

# for name,group in group\_by\_name\_year:

# print(name)# 组的名字

# print(group)# 组具体内容

# 可以选择分组

# print(group\_by\_name\_year.get\_group(('BOSS',2016)))

# 将某列数据按数据值分成不同范围段进行分组（groupby）运算

df = pd.DataFrame({'Age': np.random.randint(20, 70, 100),

'Sex': np.random.choice(['M', 'F'], 100),

})

age\_groups = pd.cut(df['Age'], bins=[19,40,65,100])

# print(age\_groups)

print(df.groupby(age\_groups).count())

# 按‘Age’分组范围和性别（sex）进行制作交叉表

pd.crosstab(age\_groups, df['Sex'])

## 聚合

'''聚合函数

mean 计算分组平均值

count 分组中非NA值的数量

sum 非NA值的和

median 非NA值的算术中位数

std 标准差

var 方差

min 非NA值的最小值

max 非NA值的最大值

prod 非NA值的积

first 第一个非NA值

last 最后一个非NA值

mad 平均绝对偏差

mode 模

abs 绝对值

sem 平均值的标准误差

skew 样品偏斜度（三阶矩）

kurt 样品峰度（四阶矩）

quantile 样本分位数（百分位上的值）

cumsum 累积总和

cumprod 累积乘积

cummax 累积最大值

cummin 累积最小值

'''

df1=pd.DataFrame({'Data1':np.random.randint(0,10,5),

'Data2':np.random.randint(10,20,5),

'key1':list('aabba'),

'key2':list('xyyxy')})

print(df1)

# 按key1分组，进行聚合计算

# 注意：当分组后进行数值计算时，不是数值类的列（即麻烦列）会被清除

print(df1.groupby('key1').sum())

# 只算data1

# print(df1['Data1'].groupby(df1['key1']).sum())

# print(df1.groupby('key1')['Data1'].sum())

# 使用agg()函数做聚合运算

# print(df1.groupby('key1').agg('sum'))

# 可以同时做多个聚合运算

# print(df1.groupby('key1').agg(['sum','mean','std']))

# 可自定义函数，传入agg方法中 grouped.agg(func)

def peak\_range(df):

"""

返回数值范围

"""

return df.max() - df.min()

# print(df1.groupby('key1').agg(peak\_range))

# 同时应用多个聚合函数

# print(df1.groupby('key1').agg(['mean', 'std', 'count', peak\_range])) # 默认列名为函数名

# print(df1.groupby('key1').agg(['mean', 'std', 'count', ('range', peak\_range)])) # 通过元组提供新的列名

# 给每列作用不同的聚合函数

dict\_mapping = {

'Data1':['mean','max'],

'Data2':'sum'

}

df1.groupby('key1').agg(dict\_mapping)

```

拓展apply（）函数

```python

# 拓展apply函数

# apply函数是pandas里面所有函数中自由度最高的函数

df1=pd.DataFrame({'sex':list('FFMFMMF'),'smoker':list('YNYYNYY'),'age':[21,30,17,37,40,18,26],'weight':[120,100,132,140,94,89,123]})

print(df1)

def bin\_age(age):

if age >=18:

return 1

else:

return 0

# 抽烟的年龄大于等18的

# print(df1['age'].apply(bin\_age))

# df1['age'] = df1['age'].apply(bin\_age)

# print(df1)

# 取出抽烟和不抽烟的体重前二

def top(smoker,col,n=5):

return smoker.sort\_values(by=col)[-n:]

df1.groupby('smoker').apply(top,col='weight',n=2)

```

## 9.分组案例

```python

# 读取数据

data = pd.read\_csv('movie\_metadata.csv')

# print('数据的形状：', data.shape)

print(data.head())

# 2、处理缺失值

data = data.dropna(how='any')

# print(data.head())

# 查看票房收入统计

# 导演vs票房总收入

group\_director = data.groupby(by='director\_name')['gross'].sum()

# ascending升降序排列，True升序

result = group\_director.sort\_values()

print(type(result))

print(result[])

# 电影产量年份趋势

from matplotlib import pyplot as plt

import random

from matplotlib import font\_manager

movie\_years = data.groupby('title\_year')['movie\_title']

print(movie\_years.count().index.tolist())

print(movie\_years.count().values)

x = movie\_years.count().index.tolist()

y = movie\_years.count().values

plt.figure(figsize=(20,8),dpi=80)

plt.plot(x,y)

plt.show()

```