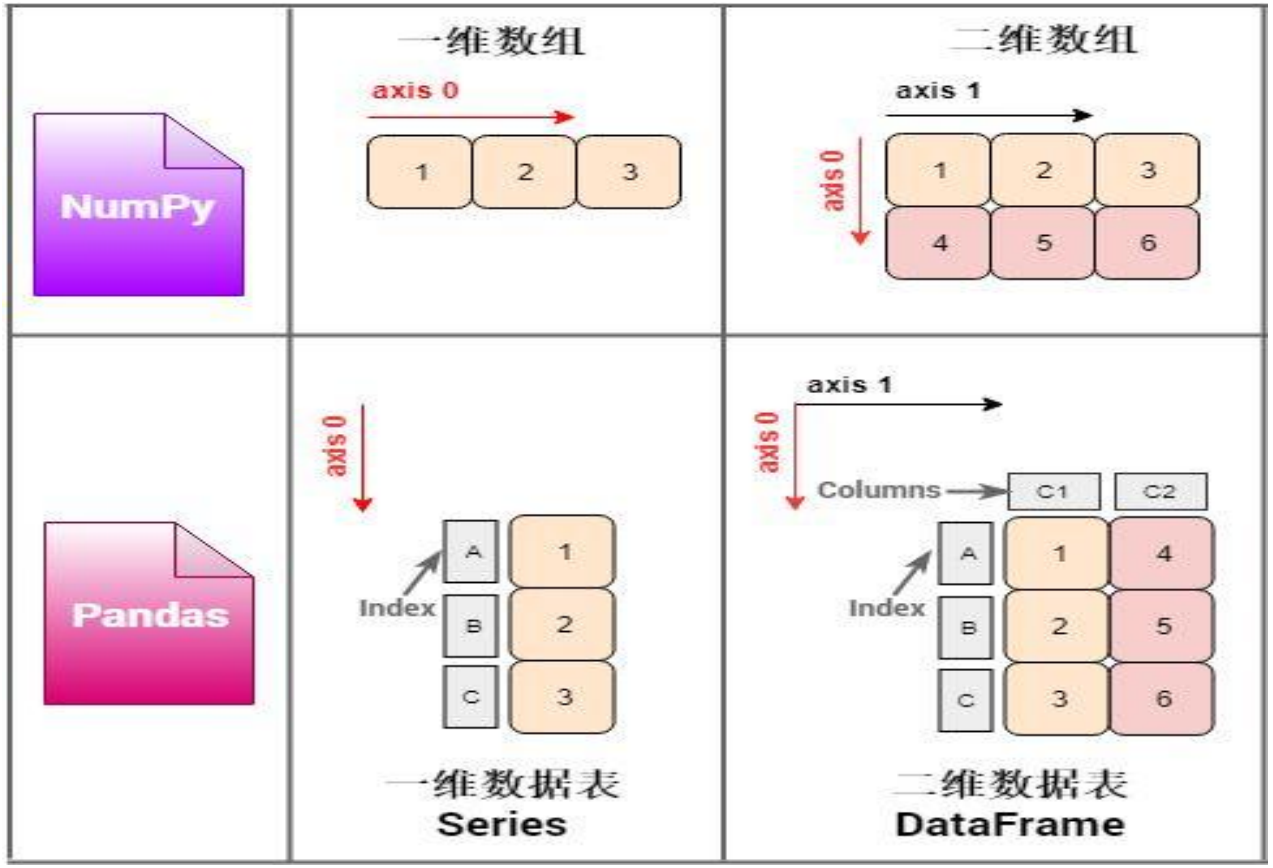
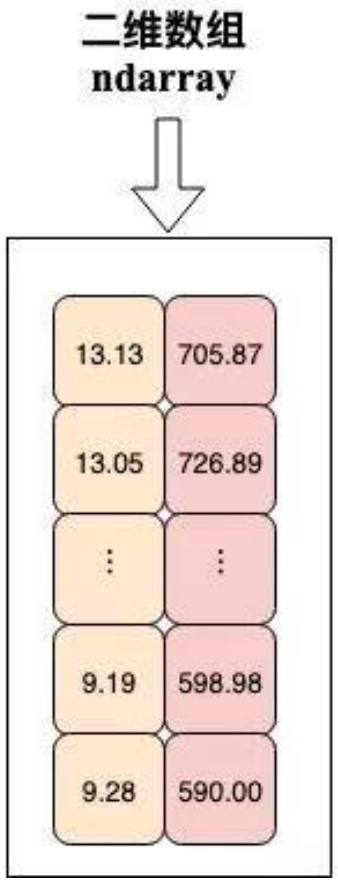
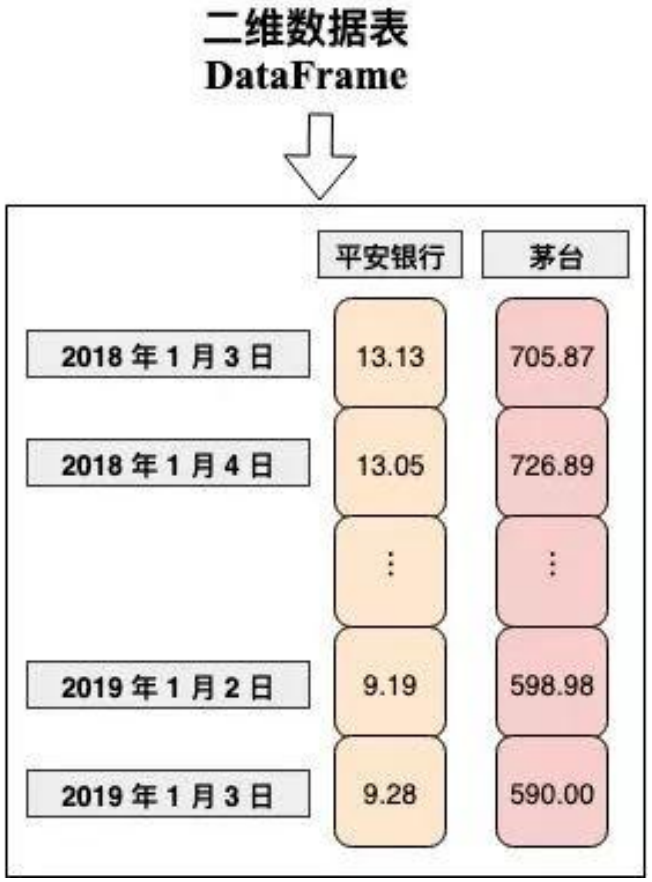
pandas统计分析基础

2020/6/11

目录


NumPy
一维数组
axis 0
1
2
3
二维数组
axis 1
axis 0
1
2
3
4
5
6
Pandas
axis 0
Index
A
B
C
1
2
3
一维数据表
Series
axis 1
axis 0
Columns
C1
C2
Index
A
B
C
1
4
2
5
3
6
二维数据表
DataFrame


二维数组
ndarray
13.13
705.87
13.05
726.89
9.19
598.98
9.28
590.00


二维数据表
DataFrame
平安银行
茅台
2018 年 1 月 3 日
13.13
705.87
2018 年 1 月 4 日
13.05
726.89
2019 年 1 月 2 日
9.19
598.98
2019 年 1 月 3 日
9.28
590.00

Pandas的Series对象

Pandas对象可以看成增强版的结构化Numpy数组，行列不再是简单的整数索引，还可以带上标签便于实现更加灵活的数据处理与分析任务。

```
import numpy as np
import pandas as pd
```

Series对象带索引的一维数组	返回值
data = pd.Series([0.25, 0.5, 0.75, 1.0]) data	0 0.25 1 0.50 2 0.75 3 1.00 dtype: float64
data.values #返回一维数组	array([0.25, 0.5 , 0.75, 1.])
data.index #pd.index类数组对象	RangeIndex(start=0, stop=4, step=1)
data[1]	0.5
data[1:3]	1 0.50 2 0.75 dtype: float64

1、Series是通用的np数组

Series与np的差异是隐式索引与显示索引，可以用任意类型定义索引。类似于列表与字典

Series对象带索引的一维数组	返回值
data = pd.Series([0.25, 0.5, 0.75, 1.0], index=['a', 'b', 'c', 'd']) data	a 0.25 b 0.50 c 0.75 d 1.00 dtype: float64
pd.Series(np.random.rand(5),np.arange(5))	0 0.406734 1 0.587088 2 0.988648 3 0.308711 4 0.985670 dtype: float64
data['b']	0.5
data['a':'c']	a 0.25 b 0.50 c 0.75 dtype: float64

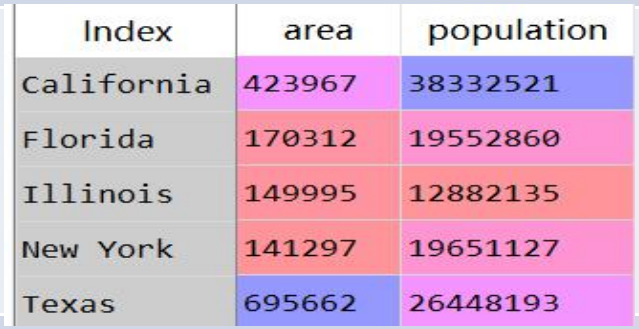
2、Series是特殊的字典

Series是一种将一种类型的键映射到一组类型值的数据结构。比字典更高效

命令	返回值
population_dict = {'California': 38332521, 'Texas': 26448193, 'New York': 19651127, 'Florida': 19552860, 'Illinois': 12882135} population = pd.Series(population_dict) population	California 38332521 Florida 19552860 Illinois 12882135 New York 19651127 Texas 26448193 dtype: int64
population['California':'Illinois']#分片 字典没有	California 38332521 Florida 19552860 Illinois 12882135 dtype: int64
pd.Series([2, 4, 6])	产生默认数字索引
pd.Series(5, index=[100, 200, 300])	根据索引序列扩展匹配
pd.Series({2:'a', 1:'b', 3:'c'})	根据字典创建
pd.Series({2:'a', 1:'b', 3:'c'}, index=[3, 2])	

1、DataFrame对象也是通用Numpy数组

Series比为带灵活索引的一维数组。DataFrame对象既有行索引又有列名的二维数组

命令	返回值
area_dict = {'California': 423967, 'Texas': 695662, 'New York': 141297, 'Florida': 170312, 'Illinois': 149995} area = pd.Series(area_dict) area	California 423967 Florida 170312 Illinois 149995 New York 141297 Texas 695662 dtype: int64
states = pd.DataFrame({'population': population, 'area': area}) states #Series对象作为字典值，可建立二维字典	 Index area population California 423967 38332521 Florida 170312 19552860 Illinois 149995 12882135 New York 141297 19651127 Texas 695662 26448193
states.index #存放索引行标签	Index(['California', 'Florida', 'Illinois', 'New York', 'Texas'], dtype='object')
states.columns #存放列标签	Index(['area', 'population'], dtype='object')

2、DataFrame是特殊的字典

字典是键值映射，DataFrame是映射到一个Series序列pd.DataFrame([population,area])

命令	返回值
states['area'] #列索引先出现，不同于二维数组 #把DataFrame看做通用的字典，而不是通用数组	California 423967 Florida 170312 Illinois 149995 New York 141297 Texas 695662 Name: area, dtype: int64
states['area']['Texas']	695662
data = pd.DataFrame(area)#直接由一个Series生成，列名取默认0	
data = pd.DataFrame([area]) #序列 列表	
pd.DataFrame([{'a': 1, 'b': 2}, {'b': 3, 'c': 4}])	
pd.DataFrame(np.random.rand(3, 2), columns=['foo', 'bar'], index=['a', 'b', 'c'])	

Index	California	Florida	Illinois	New York	Texas
0	38332521	19552860	12882135	19651127	26448193

Index	a	b	c
0	1	2	nan
1	nan	3	4

可以由**字典或Series 列表**产生

Index	foo	bar
a	0.575687	0.514053
b	0.998677	0.498061
c	0.203784	0.962496

3.pandas文件读取

- DataFrame 可以被保存为 Excel, csv, SQL 和 HDF5 格式，其语句一看就懂，用 to_数据格式，具体如下：
- to_excel()
- to_csv()
- to_sql()
- to_hdf()
- 如果要加载某种格式的数据到 DataFrame 里，用 read_数据格式，具体如下：
- read_excel()
- read_csv()
- read_sql()
- read_hdf()
- 我们只用 csv 格式举例。

- csv 格式
- 用 pd.to_csv 函数将 DataFrame 保存为 .csv 格式，注意如果 index 没有特意设定，最后不要把 index 值存到 csv 文件中。具体写法如下：
- pd.to_csv('文件名'，index=False)
- data = {'Code': ['BABA', '00700.HK', 'AAPL', '600519.SH'],
- 'Name': ['阿里巴巴', '腾讯', '苹果', '茅台'],
- 'Market': ['US', 'HK', 'US', 'SH'],
- 'Price': [185.35, 380.2, 197, 900.2],
- 'Currency': ['USD', 'HKD', 'USD', 'CNY']}
- df = pd.DataFrame(data)
- df.to_csv('pd_csv.csv', index=False)

	Unnamed: 0	Code	Name	Market	Price	Currency
0	0	BABA	阿里巴巴	US	185.35	USD
1	1	00700.HK	腾讯	HK	380.20	HKD
2	2	AAPL	苹果	US	197.00	USD
3	3	600519.SH	茅台	SH	900.20	CNY

- 用 pd.read_csv('文件名') 即可加载该文件并存成 DataFrame 形式
- df2 = pd.read_csv('pd_csv.csv')
- df2
- 如果一开始储存 df 的时候用 index=True，你会发现加载完后的 df2 是以下的样子。

目录

1、数据取值与选择

1.1 Series数据选择方法 Series对象与一维Numpy数组和标准Python字典很类似。

对Numpy数据的访问，**取值arr[2,1] 切片 arr[:,1:5] 掩码 arr[arr>0] 组合 arr[:,[1,5]]** ,这里也基本适用

(1) 将Series看作字典

命令	返回值
data = pd.Series([0.25, 0.5, 0.75, 1.0], index=['a', 'b', 'c', 'd'])	a 0.25 b 0.50 c 0.75 d 1.00 dtype: float64
data['b']	0.5
'a' in data	True
data.keys()	Index(['a', 'b', 'c', 'd'], dtype='object')
list(data.items())	[('a', 0.25), ('b', 0.5), ('c', 0.75), ('d', 1.0)]

1、数据取值与选择

1.1 Series数据选择方法

(2) 将Series看作一维数组，索引 掩码 花哨索引 都可用

命令	返回值
data['a':'c'] #显示索引，左右闭合	a 0.25 b 0.50 c 0.75 dtype: float64
data[0:2] #隐式索引，左闭右开	a 0.25 b 0.50 dtype: float64
data[(data > 0.3) & (data < 0.8)]#掩码	b 0.50 c 0.75 dtype: float64
data[['a', 'e']]#花式索引	a 0.25 e 1.25 dtype: float64

1、数据取值与选择

1.1 Series数据选择方法

(3) loc与iloc索引器　　显式和隐式索引可能会造成混乱，当使用显式整数索引

data = pd.Series(['a', 'b', 'c'], index=[1, 3, 5])

data[1]='a' #取值使用显式索引 data[1:3] # 3 b 5 c 切片使用隐式索引

命令	返回值
data.loc[1] #loc取值和切片都是显式	‘a’
data.loc[1:3]	1 a 3 b dtype: object
data.iloc[1]#iloc取值和切片都是隐式	'b'
data.iloc[1:3]	3 b 5 c dtype: object

1、数据取值与选择

1.2 DataFrame数据选择方法 二维数组或字典对比理解

(1) 将DataFrame看作字典

```
area = pd.Series({'California': 423967, 'Texas': 695662,
                  'New York': 141297, 'Florida': 170312,
                  'Illinois': 149995})
pop = pd.Series({'California': 38332521, 'Texas': 26448193,
                 'New York': 19651127, 'Florida': 19552860,
                 'Illinois': 12882135})
data = pd.DataFrame({'area':area, 'pop':pop})
data
```

Index	area	pop
California	423967	38332521
Florida	170312	19552860
Illinois	149995	12882135
New York	141297	19651127
Texas	695662	26448193

1、数据取值与选择

1.2 DataFrame数据选择方法 二维数组或字典对比理解

(1) 将DataFrame看作字典

命令	返回值
data['area'] data.area	California 423967 Florida 170312 Illinois 149995 New York 141297 Texas 695662 Name: area, dtype: int64
data['density'] = data['pop'] /data['area'] data #可以通过运算增加一列	Index area pop density California 423967 38332521 90.4139 Florida 170312 19552860 114.806 Illinois 149995 12882135 85.8838 New York 141297 19651127 139.077 Texas 695662 26448193 38.0187

1、数据取值与选择

1.2DataFrame数据选择方法

(2)将DataFrame看作二维数组

DataFrame属性	返回值
values	元素
index	索引
columns	列名
dtypes	类型
size	元素个数
ndim	维度数
shape	数据形状（行列数目）

1、数据取值与选择

1.2DataFrame数据选择方法

(2)将DataFrame看作二维数组

DataFrame属性	返回值
data.values	array([[4.23967000e+05, 3.83325210e+07, 9.04139261e+01, 2.00000000e+02], [1.70312000e+05, 1.95528600e+07, 1.14806121e+02, 2.00000000e+02], [1.49995000e+05, 1.28821350e+07, 8.58837628e+01, 2.00000000e+02], [1.41297000e+05, 1.96511270e+07, 1.39076746e+02, 2.00000000e+02], [6.95662000e+05, 2.64481930e+07, 3.80187404e+01, 2.00000000e+02]])
data.values[0] #第一行	array([4.23967000e+05, 3.83325210e+07, 9.04139261e+01])
data['area']	获取一列数据
data.iloc[:3, :2]	隐式
data.loc[:'Illinois', :'pop']	显式
data.T	行列转置

Index	area	pop
California	423967	38332521
Florida	170312	19552860
Illinois	149995	12882135

Index	area	pop
California	423967	38332521
Florida	170312	19552860
Illinois	149995	12882135

1、数据取值与选择

1.2DataFrame数据选择方法

(2)将DataFrame看作二维数组 drop('索引名',axis=0 1) 可以删除对应行或列

命令	返回值
data.loc[data.density > 100, ['pop', 'density']] #loc索引可用掩码和花式	pop density Florida 19552860 114.806121 New York 19651127 139.076746
data.iloc[0, 2] = 90	将0行2列元素重新赋值
data['Florida':'Illinois'] #单个标签选列，分片选行	area pop density Florida 170312 19552860 114.806121 Illinois 149995 12882135 85.883763
data.loc[data.density > 100] data[data.density > 100]#掩码筛选	area pop density Florida 170312 19552860 114.806121 New York 141297 19651127 139.076746
data[3:4] data['New York':'New York'] #选择一行	area pop density New York 141297 19651127 139.076746

2、Pandas数值运算方法

2.1 Numpy的基本能力是可以快速对每个元素运算，包括基本运算及复杂函数运算

Pandas一元运算可以保留索引和列标签

命令	返回值	
rng = np.random.RandomState(42) ser = pd.Series(rng.randint(0, 10, 4)) ser np.exp(ser)　#指数函数e^x	0　6 1　3 2　7 3　4 dtype: int64	0　403.428793 1　20.085537 2　1096.633158 3　54.598150 dtype: float64
df = pd.DataFrame(rng.randint(0, 10, (3, 4)), columns=['A', 'B', 'C', 'D']) df df-int(np.average(df))	A B C D 0 6 3 7 4 1 6 9 2 6 2 7 4 3 7	A B C D 0 1 -2 2 -1 1 1 4 -3 1 2 2 -1 -2 2

2、Pandas数值运算方法

2.2 二元运算，Pandas对齐索引

命令	返回值	
area = pd.Series({'Alaska': 1723337, 'Texas': 695662, 'California': 423967}, name='area') population = pd.Series({'California': 38332521, 'Texas': 26448193, 'New York': 19651127}, name='population') population / area #索引缺失用NaN填充	Alaska NaN California 90.413926 New York NaN Texas 38.018740 dtype: float64	
A = pd.Series([2, 4, 6], index=[0, 1, 2]) B = pd.Series([1, 3, 5], index=[1, 2, 3]) A +B A.add(B, fill_value=0) #等价于A+B，缺失值用0填充	0 NaN 1 5.0 2 9.0 3 NaN dtype: float64	0 2.0 1 5.0 2 9.0 3 5.0 dtype: float64

2、Pandas数值运算方法

2.3 DataFrame 与 Series 运算：默认按行广播

<table>
<tr><th>命令</th><th colspan="2">返回值</th></tr>
<tr><td>A = rng.randint(10, size=(3, 4))
A
A - A[0] #根据广播规则，按行进行计算。</td><td>array([[3, 8, 2, 4],
[2, 6, 4, 8],
[6, 1, 3, 8]])</td><td>array([[0, 0, 0, 0],
[-1, -2, 2, 4],
[3, -7, 1, 4]])</td></tr>
<tr><td>df = pd.DataFrame(A, columns=list('QRST'))
df - df.iloc[0] #同Numpy运算规则

df.subtract(df['R'], axis=0) #按列计算需要axis参数</td><td>
<table>
<tr><th>Index</th><th>Q</th><th>R</th><th>S</th><th>T</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>6</td><td>-5</td><td>2</td></tr>
<tr><td>2</td><td>1</td><td>1</td><td>-4</td><td>3</td></tr>
</table>
</td><td>
<table>
<tr><th>Index</th><th>Q</th><th>R</th><th>S</th><th>T</th></tr>
<tr><td>0</td><td>3</td><td>0</td><td>4</td><td>1</td></tr>
<tr><td>1</td><td>-3</td><td>0</td><td>-7</td><td>-3</td></tr>
<tr><td>2</td><td>3</td><td>0</td><td>-1</td><td>3</td></tr>
</table>
</td></tr>
</table>

3、检测与处理重复、缺失值

删除重复值

- pandas提供了一个名为drop_duplicates的去重方法。该方法只对DataFrame或者Series类型有效。这种方法不会改变数据原始排列，并且兼具代码简洁和运行稳定的特点。该方法不仅支持单一特征的数据去重，还能够依据DataFrame的其中一个或者几个特征进行去重操作。

*pandas.DataFrame(Series).**drop_duplicates**(self, subset=None, keep='first', inplace=False)*

参数名称	说明
subset	接收string或sequence。表示进行去重的列。默认为None，表示全部列。
keep	接收特定string。表示重复时保留第几个数据。First：保留第一个。Last：保留最后一个。False：只要有重复都不保留。默认为first。
inplace	接收boolean。表示是否在原表上进行操作。默认为False。
df6['test']=[1,2,1] df6.drop_duplicates('test')	food name test 0 fish Peter 1 1 beans Paul 2 2 bread Mary 1 food name test 0 fish Peter 1 1 beans Paul 2

3、检测与处理重复、缺失值

利用isnull或notnull找到缺失值

- 数据中的某个或某些特征的值是不完整的，这些值称为缺失值。
- pandas提供了识别缺失值的方法isnull以及识别非缺失值的方法notnull，这两种方法在使用时返回的都是布尔值True和False。
- 结合sum函数和isnull、notnull函数，可以检测数据中缺失值的分布以及数据中一共含有多少缺失值。

```
data = pd.Series([1, np.nan, 'hello', None])
data.isnull()
data[data.notnull()]#掩码
```

```
0       1
1     NaN
2   hello
3    None
dtype: object
```

```
0       1
2   hello
dtype: object
```

3、检测与处理重复、缺失值

删除缺失值

- pandas中提供了简便的删除缺失值的方法dropna，该方法既可以删除观测记录，亦可以删除特征。

*pandas.DataFrame.**dropna**(self, axis=0, how='any', thresh=None, subset=None, inplace=False)*

常用参数及其说明如下。

参数名称	说明
axis	接收0或1。表示轴向，0为删除观测记录（行），1为删除特征（列）。默认为0。
how	接收特定string。表示删除的形式。any表示只要有缺失值存在就执行删除操作。all表示当且仅当全部为缺失值时执行删除操作。默认为any。
subset	接收类array数据。表示进行去重的列/行。默认为None，表示所有列/行。
inplace	接收boolean。表示是否在原表上进行操作。默认为False。
data.dropna()	0 1 2 hello dtype: object

3、检测与处理重复、缺失值

删除缺失值

参数名称	说明		
df = pd.DataFrame([[1, np.nan, 2], [2, 3, 5], [np.nan, 4, 6]]) df.dropna() #默认删除有缺失值的行 s#删除列	0 1 2 0 1.0 NaN 2 1 2.0 3.0 5 2 NaN 4.0 6	0 1 2 1 2.0 3.0 5	2 0 2 1 5 2 6
df[3] = np.nan#增加一个缺失值列	0 1 2 3 0 1.0 NaN 2 NaN 1 2.0 3.0 5 NaN 2 NaN 4.0 6 NaN	0 1 2 0 1.0 NaN 2 1 2.0 3.0 5 2 NaN 4.0 6	
df.dropna(axis='columns', how='all')			

3、检测与处理重复、缺失值

（4）填充缺失值

参数名称	说明		
data = pd.Series([1, np.nan, 2, None, 3], index=list('abcde')) data.fillna(0)#用0填充 data.fillna(method='ffill')#用前值填充	a 1.0 b NaN c 2.0 d NaN e 3.0 dtype: float64	a 1.0 b 0.0 c 2.0 d 0.0 e 3.0 dtype: float64	a 1.0 b 1.0 c 2.0 d 2.0 e 3.0 dtype: float64
df.fillna(method='ffill', axis=0)	0 1 2 3 0 1.0 1.0 2.0 2.0 1 2.0 3.0 5.0 5.0 2 NaN 4.0 6.0 6.0	0 1 2 3 0 1.0 1.0 2.0 2.0 1 2.0 3.0 5.0 5.0 2 2.0 4.0 6.0 6.0	

5、基于关系代数的主键合并数据

主键合并——merge函数

- 和数据库的join一样，merge函数也有左连接（left）、右连接（right）、内连接（inner）和外连接（outer），但比起数据库SQL语言中的join和merge函数还有其自身独到之处，例如可以在合并过程中对数据集中的数据进行排序等。

  *pandas.**merge**(left, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=True, indicator=False)*

- 可根据merge函数中的参数说明，并按照需求修改相关参数，就可以多种方法实现主键合并。

Merge

df1

	代号	价格
0	JD	25.95
1	BABA	176.92
2	PDD	22.50

df2

	代号	雇员
0	JD	多
1	BABA	中
2	BIDU	多

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

	代号	价格	雇员
0	JD	25.95	多
1	BABA	176.92	中
2	PDD	22.50	NaN

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

	代号	价格	雇员
0	JD	25.95	多
1	BABA	176.92	中
2	BIDU	NaN	多

```
pd.merge( df1, df2, how='inner'
          on='代号' )
```

	代号	价格	雇员
0	JD	25.95	多
1	BABA	176.92	中

```
pd.merge( df1, df2, how='outer'
          on='代号' )
```

	代号	价格	雇员
0	JD	25.95	多
1	BABA	176.92	中
2	PDD	22.50	NaN
3	BIDU	NaN	多

5、基于关系代数的主键合并数据

常用参数及其说明

参数名称	说明
left	接收DataFrame或Series。表示要添加的新数据。无默认。
right	接收DataFrame或Series。表示要添加的新数据。无默认。。
how	接收inner，outer，left，right。表示数据的连接方式。默认为inner。
on	接收string或sequence。表示两个数据合并的主键（必须一致）。默认为None。
left_on	接收string或sequence。表示left参数接收数据用于合并的主键。默认为None。
right_on	接收string或sequence。表示right参数接收数据用于合并的主键。默认为None。
left_index	接收boolean。表示是否将left参数接收数据的index作为连接主键。默认为False。
right_index	接收boolean。表示是否将right参数接收数据的index作为连接主键。默认为False。
sort	接收boolean。表示是否根据连接键对合并后的数据进行排序。默认为False。
suffixes	接收接收tuple。表示用于追加到left和right参数接收数据重叠列名的尾缀默认为('_x', '_y')。

5、基于关系代数的主键合并数据

pd.merge()可 以实现三种数据连接：一对一、多对一、多对多。

命令	返回值	

df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})
 	
df5 = pd.DataFrame({'group': ['Accounting', 'Accounting','Engineering', 'Engineering', 'HR', 'HR'],'skills': ['math', 'spreadsheets', 'coding', 'linux','spreadsheets', 'organization']})
 	
#df1 df5自动通过'group'索引连接。多对多
pd.merge(df1, df5)
 	
pd.merge(df1,df5,on='group')#on参数可以省略
	employee group
	0 Bob Accounting
	1 Jake Engineering
	2 Lisa Engineering
	3 Sue HR
	
	group skills
	0 Accounting math
	1 Accounting spreadsheets
	2 Engineering coding
	3 Engineering linux
	4 HR spreadsheets
	5 HR organization
		employee group skills
		0 Bob Accounting math
		1 Bob Accounting spreadsheet
		2 Jake Engineering coding
		3 Jake Engineering linux
		4 Lisa Engineering coding
		5 Lisa Engineering linux
		6 Sue HR spreadsheets
		7 Sue HR organization

5、基于关系代数的主键合并数据

pd.merge()不同列索引合并可以设置合并的键。left_on right_on

命令	返回值	
df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],'group': ['Accounting', 'Engineering', 'Engineering', 'HR']}) df3 = pd.DataFrame({'name': ['Bob', 'Jake', 'Lisa', 'Sue'],'salary': [70000, 80000, 120000, 90000]}) pd.merge(df1, df3, left_on="employee", right_on="name")') pd.merge(df1, df3, left_on="employee", right_on="name")').drop('name',axis=1) #drop删除重复列	employee group 0 Bob Accounting 1 Jake Engineering 2 Lisa Engineering 3 Sue HR name salary 0 Bob 70000 1 Jake 80000 2 Lisa 120000 3 Sue 90000	employee group name salary 0 Bob Accounting Bob 70000 1 Jake Engineering Jake 80000 2 Lisa Engineering Lisa 120000 3 Sue HR Sue 90000

5、基于关系代数的主键合并数据

pd.merge() 可以按索引合并。left_index right_index

命令	返回值	
df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],'group': ['Accounting', 'Engineering', 'Engineering', 'HR']}) df3 = pd.DataFrame({'name': ['Bob', 'Jake', 'Lisa', 'Sue'],'salary': [70000, 80000, 120000, 90000]}) df1a = df1.set_index('employee') df3a = df3.set_index(name') pd.merge(df1, df3, left_index=True, right_index=True) df1a.join(df3a)#按索引进行join #也可以left_on 与right_index混用	goup employee Bob Accounting Jake Engineering Lisa Engineering Sue HR salary name Bob 70000 Jake 80000 Lisa 120000 Sue 90000	employee group name salary 0 Bob Accounting Bob 70000 1 Jake Engineering Jake 80000 2 Lisa Engineering Lisa 120000 3 Sue HR Sue 90000 group salary employee Bob Accounting 70000 Jake Engineering 80000 Lisa Engineering 120000 Sue HR 90000

6、美国各州数据统计案例

- pop = pd.read_csv('data/state-population.csv') #读取csv文件到Dataframe
- areas = pd.read_csv('data/state-areas.csv')
- abbrevs = pd.read_csv('data/state-abbrevs.csv')
- pop.head()
- areas.head()
- abbrevs.head()')。

pop.head()

	state/region	ages	year	population
0	AL	under18	2012	1117489.0
1	AL	total	2012	4817528.0
2	AL	under18	2010	1130966.0
3	AL	total	2010	4785570.0
4	AL	under18	2011	1125763.0

area()

	state	area (sq. mi)
0	Alabama	52423
1	Alaska	656425
2	Arizona	114006
3	Arkansas	53182
4	California	163707

	state	abbreviation
0	Alabama	AL
1	Alaska	AK
2	Arizona	AZ
3	Arkansas	AR
4	California	CA

6、美国各州数据统计案例

- 任务：计算各州人口密度。
- 首先多对一合并 pop与abbrevs得到州名全称，还要通过how=”outer”保证无丢失。

merged = pd.merge(pop, abbrevs, how='outer', left_on='state/region',right_on='abbreviation') merged = merged.drop('abbreviation', 1) # 删除无用列 merged.head()	state/region ages year population state 0 AL under18 2012 1117489.0 Alabama 1 AL total 2012 4817528.0 Alabama 2 AL under18 2010 1130966.0 Alabama
merged.isnull().any() #查看那一列数据有缺失 #结果显示人口 和州全称有缺失	state/region False ages False year False population True state True dtype: bool

6、美国各州数据统计案例

merged[merged['population'].isnull()].head() #发现问题，PR（波多黎各）2000以前没有统计过人口，同时州名缩写也缺失。 merged.loc[merged['state'].isnull(),'state/region'].unique()	state/region ages year population state 2448 PR under18 1990 NaN NaN 2449 PR total 1990 NaN NaN 2450 PR total 1991 NaN NaN array(['PR', 'USA'], dtype=object) #全国 与波多

merged.loc[merged['state/region'] == 'PR', 'state']
= 'Puerto Rico' #补全PR的州全称
merged.loc[merged['state/region'] == 'USA', 'state']
= 'United States' #补全美国全称
merged.isnull().any()
 	
经过上述处理后，state列已经没有缺失
	state/region False
	ages False
	year False
	population True
	state False
	dtype: bool

7、美国各州数据统计案例

采用同样的处理方法合并面积数据，进而处理面积缺失。

```
final = pd.merge(merged, areas, on='state',
                 how='left')
final.head()
```

```
  state/region     ages  year  population    state  area (sq. mi)
0           AL  under18  2012   1117489.0  Alabama         52423.0
1           AL    total  2012   4817528.0  Alabama         52423.0
2           AL  under18  2010   1130966.0  Alabama         52423.0
3           AL    total  2010   4785570.0  Alabama         52423.0
4           AL  under18  2011   1125763.0  Alabama         52423.0
```

```
final.isnull().any()  #得到area有缺失值
final['state'][final['area (sq. mi)'].isnull()].
                        unique()
#final.loc[final['area (sq. mi)'].isnull(),'state']
#注意区分上述两个命令，先行后列 先列后行
```

```
state/region     False
ages             False
year             False
population        True
state            False
area (sq. mi)     True
dtype: bool
```

```
final.dropna(inplace=True)#这里我们删除
```

经过上述处理后，area列已经没有缺失

```
array(['United States'], dtype=object)
#全国面积数据缺失，可以用各州面积和填充
```

6、美国各州数据统计案例

选择2000各州人口及面积数据

data2010 = final.query("year == 2010 & ages == 'total'")
data2010.set_index('state', inplace=True) #设置州名为新索引。
density = data2010['population'] / data2010['area (sq. mi)']#新Series
 	
density.sort_values(ascending=False, inplace=True)#排序
density.head()
	state
	District of Columbia 8898.897059
	Puerto Rico 1058.665149
	New Jersey 1009.253268
	Rhode Island 681.339159
	Connecticut 645.600649
	dtype: float64

density.tail() #人口密度最低的几个州 **思考：** **data2010=data2010.loc[:,['state','population',**'area (sq. mi)']]	

6、美国各州数据统计案例：可视化

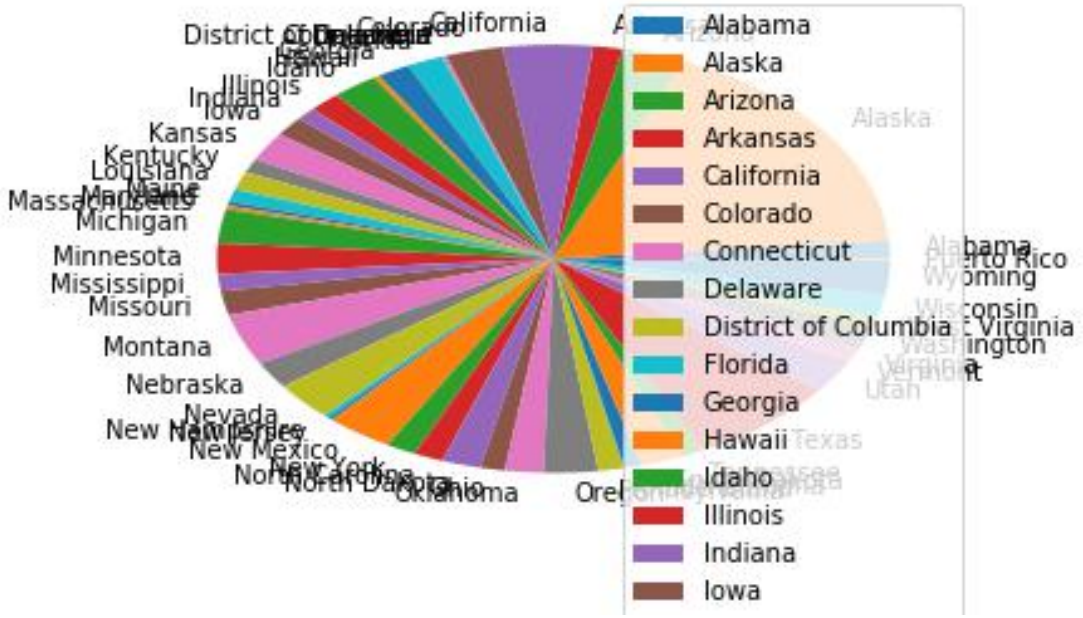
#绘制可视化，各州面积 分布饼图 。人口分布柱形图 ， 人口密度折线图。

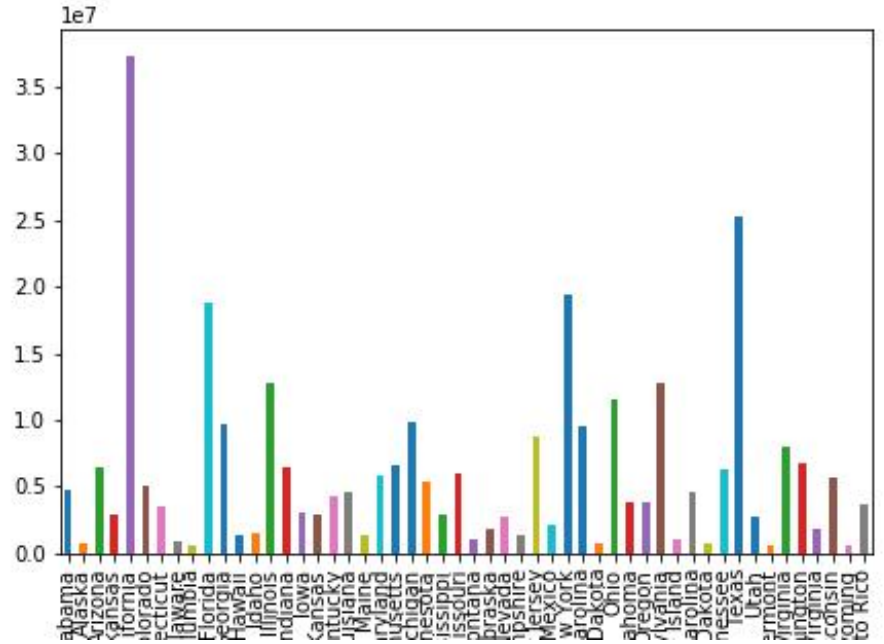
plt.pie(data2010['area (sq. mi)'],labels=data2010.index)

#data2010['area (sq. mi)'].plot(kind='pie')

plt.legend()

data2010['population'].plot(kind='bar')





7、Pandas统计与排序

(1)数值型特征的描述性统计——NumPy中的描述性统计函数

- 数值型数据的描述性统计主要包括了计算数值型数据的完整情况、最小值、均值、中位数、最大值、四分位数、极差、标准差、方差、协方差和变异系数等。在NumPy库中一些常用的统计学函数如下表所示。
- pandas库基于NumPy，自然也可以用这些函数对数据框进行描述性统计。

函数名称	说明	函数名称	说明
np.min	最小值	np.max	最大值
np.mean	均值	np.ptp	极差
np.median	中位数	np.std	标准差
np.var	方差	np.cov	协方差

7、Pandas统计与排序

(2)数值型特征的描述性统计—— pandas描述性统计方法

方法名称	说明	方法名称	说明
min	最小值	max	最大值
mean	均值	ptp	极差
median	中位数	std	标准差
var	方差	cov	协方差
sem	标准误差	mode	众数
skew	样本偏度	kurt	样本峰度
quantile	四分位数	count	非空值数目
describe	描述统计	mad	平均绝对离差

7、Pandas统计与排序

方法名称	说明	
rng = np.random.RandomState(42) ser = pd.Series(rng.rand(5)) ser df = pd.DataFrame({'A': rng.rand(5), 'B': rng.rand(5)}) df	0 0.374540 1 0.950714 2 0.731994 3 0.598658 4 0.156019 dtype: float64	A B 0 0.374540 0.155995 1 0.950714 0.058084 2 0.731994 0.866176 3 0.598658 0.601115 4 0.156019 0.708073
ser.sum()	2.8119254917081569	
ser.mean()	0.5623850983416314 2	
df.sum()	A 2.811925 B 2.389442 dtype: float64	0 0.265267 1 0.504399 2 0.799085 3 0.599887 4 0.432046 dtype: float64
df.mean(axis='columns')		
df.describe()		

7、Pandas统计与排序

(2)排序 分为对索引排序 sort_index 和对 值排序 sort_values

- numpy排序操作为sort(axis=0 或 1)

<table>
<tr><th>函数名称</th><th colspan="3">说明</th></tr>
<tr><td>obj = pd.Series(range(4), index=['d','a','b','c'])
obj.sort_index()</td><td>d 0
a 1
b 2
c 3
dtype: int64</td><td colspan="2">a 1
b 2
c 3
d 0
dtype: int64</td></tr>
<tr><td></td><td colspan="3"></td></tr>
<tr><td>frame = pd.DataFrame(np.arange(8).reshape((2,4)),
index=['three','one'], columns=['d','a','b','c'])</td><td rowspan="2">d a b c
three 0 1 2 3
one 4 5 6 7</td><td rowspan="2">d a b c
one 4 5 6 7
three 0 1 2 3</td><td rowspan="2">a b c d
three 1 2 3 0
one 5 6 7 4</td></tr>
<tr><td>frame.sort_index()
frame.sort_index(axis=1)

#frame.sort_index(axis=1, ascending=False)
#降序</td></tr>
</table>

7、Pandas统计与排序

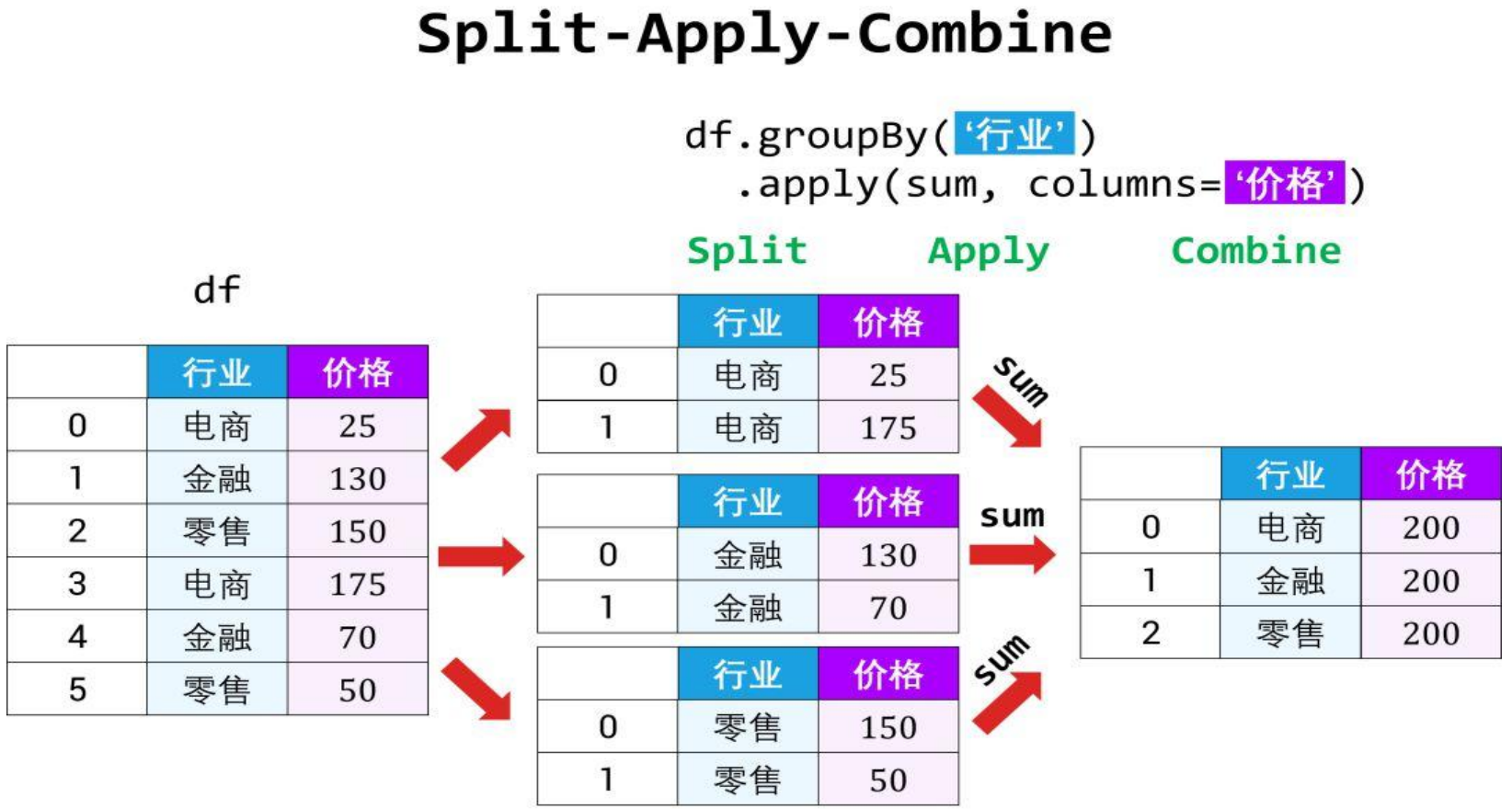
(2)排序 分为对索引排序 sort_index 和对 值排序 sort_values

- Series两个特殊函数， unique()统计所有去重后的值 value_counts()统计去重后值，分别出现的次数

<table>
<tr><th>函数名称</th><th colspan="3">说明</th></tr>
<tr><td>obj = pd.Series(range(4), index=['d','a','b','c'])
obj.sort_values(ascending=False)</td><td>d 0
a 1
b 2
c 3
dtype: int64</td><td colspan="2">c 3
b 2
a 1
d 0
dtype: int64</td></tr>
<tr><td>frame = pd.DataFrame(np.arange(8).reshape((2,4)),
index=['three','one'], columns=['d','a','b','c'])
frame.sort_values(by='a',ascending=False)
#frame.sort_values(by=['a','b'],ascending=False)
frame.sort_values(by='one',axis=1,ascending=Fals
e)</td><td>d a b c
three 0 1 2 3
one 4 5 6 7</td><td>d a b c
one 4 5 6 7
three 0 1 2 3</td><td>c b a d
three 3 2 1 0
one 7 6 5 4</td></tr>
</table>

8、Pandas分组统计

局部分组统计——groupby



8、Pandas分组统计

局部分组统计——groupby

- 基本处理步骤为按指定的键分割成若干组、
- 对每个组应用统计函数处理
- 将每一组的结果合并组合输出

函数名称	说明
df=pd.DataFrame({'key':['A','B','C','A','B','C'],'data':range(6)},columns=['key','data'])	key data 0 A 0 1 B 1 2 C 2 3 A 3 4 B 4 5 C 5
df.groupby('key').sum()	data key A 3 B 5 C 7

8、Pandas分组统计-数据透视表

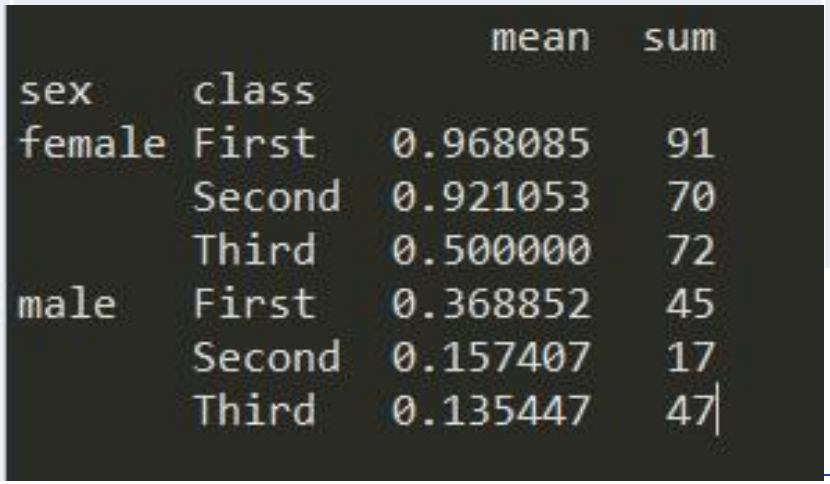
泰坦尼克数据集

- Survived => 获救情况（1为获救，0为未获救）
- Pclass => 乘客等级(1/2/3等舱位)
- Name => 乘客姓名
- Sex => 性别
- Age => 年龄
- SibSp => 堂兄弟/妹个数
- Parch => 父母与小孩个数
- Ticket => 船票信息
- Fare => 票价
- Cabin => 客舱
- Embarked => 登船港口

]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

8、Pandas分组统计-数据透视表

函数名称	说明
import numpy as np import pandas as pd import seaborn as sns titan=sns.load_dataset('titanic')	survived pclass sex age ... deck embark_town alive alone 0 0 3 male 22.0 ... NaN Southampton no False 1 1 1 emale 38.0 ... C Cherbourg yes False 2 1 3 female 26.0 ... NaN Southampton yes True 3 1 1 female 35.0 ... C Southampton yes False 4 0 3 male 35.0 ... NaN Southampton no True
titan.groupby('sex')[['survived']].mean()	survived sex female 0.742038 male 0.188908
titan.groupby(['sex','class'])['survived'].mean()	sex class female First 0.968085 Second 0.921053 Third 0.500000 male First 0.368852 Second 0.157407 Third 0.135447 Name: survived, dtype: float64
titan.groupby(['sex','class'])['survived'].aggregate(['mean','sum']) #stack() unstack()	

8、Pandas分组统计-数据透视表

函数名称	说明
titan.pivot_table('survived',index='sex',columns='class')	class First Second Third sex female 0.968085 0.921053 0.500000 male 0.368852 0.157407 0.135447
titan.pivot_table('survived',index='sex',columns='class',aggfunc='sum')	class First Second Third sex female 91 70 72 male 45 17 47
titan.pivot_table(index='sex',columns='class',aggfunc={'survived':'sum','age':'mean'})	age survived class First Second Third First Second Third sex female 34.611765 28.722973 21.750000 91 70 72 male 41.281386 30.740707 26.507589 45 17 47

8、Pandas分组统计-数据透视表

pandas可视化：Pandas数据可直接调用绘图函数

- pandas中的数据可视化已经可以满足我们大部分的要求了，也就省下了我们很多自己使用 如 matplotlib 来数据可视化的工作。

函数名称	说明
‘line’ ‘bar’ or ‘barh’ for bar plots ‘hist’ for histogram ‘box’ for boxplot ‘area’ for area plots ‘scatter’ for scatter plots ‘pie’ for pie plots	可以添加其它参数：x= ，y= ，title、label、figsize、color、xsticks、ysticks
suv=titan.groupby(['sex','class'])['survived'].mean() suv.plot(kind='bar')