In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

1 缺失值的处理

1.1 缺失值的判断

In [2]:
```
import numpy as np
import pandas as pd
from pandas import Series,DataFrame
```

In [3]:
```
data1=[[1, 2, None],[4, 5, np.nan],[7, 8, 9]] #二维列表创建
d1=DataFrame(data1,index=['a','b','c'],columns=['one','two','three'])
d1
```

Out[3]:

	one	two	three
a	1	2	NaN
b	4	5	NaN
c	7	8	9.0

In [4]:
```
d1.isnull()
```

Out[4]:

	one	two	three
a	False	False	True
b	False	False	True
c	False	False	False

In [5]:
```
1 d1.notnull()
```

Out[5]:

	one	two	three
a	True	True	False
b	True	True	False
c	True	True	True

In [6]:
```
1 d1
```

Out[6]:

	one	two	three
a	1	2	NaN
b	4	5	NaN
c	7	8	9.0

In [7]:
```
1 d1.isnull().sum()  #每列缺失值统计    按列求和,以列为单位数据类型一样
```

Out[7]:
```
one      0
two      0
three    2
dtype: int64
```

In [8]:
```
1 d1.isnull().sum().sum() #总的多少个
```

Out[8]:
```
2
```

In [9]:
```
1 d1.isnull().sum(axis=1)  #每行缺失值统计
```

Out[9]:
```
a    1
b    1
c    0
dtype: int64
```

1.2 缺失值的删除

In [12]:
```
df = pd.DataFrame(np.random.rand(4, 5),index = ['a', 'b', 'c', 'd'],columns = ['value1','value2','value3','value4','value5'])
df.iloc[0:3,1]=np.nan
df.iloc[2:3,2]=np.nan
df.iloc[2:3,:]=np.nan
df  #先产生有缺失值的数据
```

Out[12]:

	value1	value2	value3	value4	value5
a	0.005566	NaN	0.974658	0.843130	0.985263
b	0.677798	NaN	0.915878	0.114531	0.959331
c	NaN	NaN	NaN	NaN	NaN
d	0.462880	0.579225	0.979482	0.665500	0.133233

In [13]:
```
df.dropna(axis=0)       #删除有缺失的行,只要某行有缺失值就都会被删除
```

Out[13]:

	value1	value2	value3	value4	value5
d	0.46288	0.579225	0.979482	0.6655	0.133233

In [14]:
```
df.dropna(axis=1) #删除有缺失的列,只要某列里有缺失值就都会被删除
```

Out[14]:

a
b
c
d

In [18]:
```
df.loc['d','value2']=np.nan
df   #随便生成一些数据
```

Out[18]:

	value1	value2	value3	value4	value5
a	0.005566	NaN	0.974658	0.843130	0.985263
b	0.677798	NaN	0.915878	0.114531	0.959331
c	NaN	NaN	NaN	NaN	NaN
d	0.462880	NaN	0.979482	0.665500	0.133233

In [19]:
```
df.dropna(how='all',axis=0) #删除的是行，全部为nan才删除
#how这个参数确定删除的方式,how='all'代表某一行或某一列全部为缺失值的时候才会把该行或该列删除
#how='any'代表某一行或某一列只要有缺失值,则该行或该列就都会被删除
#如果how这个参数不写,默认how='any'
```

Out[19]:

	value1	value2	value3	value4	value5
a	0.005566	NaN	0.974658	0.843130	0.985263
b	0.677798	NaN	0.915878	0.114531	0.959331
d	0.462880	NaN	0.979482	0.665500	0.133233

In [24]:
```
df.dropna(how='any',axis=0) #删除的是行，只要有nan就删除
```

Out[24]:

	value1	value2	value3	value4	value5

In [25]:
```
1 df.dropna(how='all',axis=1) #删除的是列，全部为nan才删除
```

Out[25]:

	value1	value3	value4	value5
a	0.005566	0.974658	0.843130	0.985263
b	0.677798	0.915878	0.114531	0.959331
c	NaN	NaN	NaN	NaN
d	0.462880	0.979482	0.665500	0.133233

In [26]:
```
1 df.dropna(how='any',axis=1) #删除的是列，只要为nan就删除
```

Out[26]:

a
b
c
d

In [27]:
```
1 df
```

Out[27]:

	value1	value2	value3	value4	value5
a	0.005566	NaN	0.974658	0.843130	0.985263
b	0.677798	NaN	0.915878	0.114531	0.959331
c	NaN	NaN	NaN	NaN	NaN
d	0.462880	NaN	0.979482	0.665500	0.133233

In [28]:
```
df.dropna(subset=['value3'],axis=0)   #删除value3列有缺失的行
#subset锁定某一列,对列中元素进行缺失值的判断,,但最终删除的是有缺失值的行
```

Out[28]:

	value1	value2	value3	value4	value5
a	0.005566	NaN	0.974658	0.843130	0.985263
b	0.677798	NaN	0.915878	0.114531	0.959331
d	0.462880	NaN	0.979482	0.665500	0.133233

In [29]:
```
df.dropna(subset=['value3'],axis=0,inplace=True) #原地删除
```

In [30]:
```
df
```

Out[30]:

	value1	value2	value3	value4	value5
a	0.005566	NaN	0.974658	0.843130	0.985263
b	0.677798	NaN	0.915878	0.114531	0.959331
d	0.462880	NaN	0.979482	0.665500	0.133233

1.3 缺失值的填充

In [45]:
```
a = [[1, 2, 2],[3,None,6],[3, 7, None],[5,None,7]]
df = DataFrame(a,index=['a','b','c','d'],columns=['one','two','three'])
df
```

Out[45]:

	one	two	three
a	1	2.0	2.0
b	3	NaN	6.0
c	3	7.0	NaN
d	5	NaN	7.0

In [34]:
```
df.fillna(0)      #用常数进行填充,括号里的参数是几就用几进行填充
```

Out[34]:

	one	two	three
a	1	2.0	2.0
b	3	0.0	6.0
c	3	7.0	0.0
d	5	0.0	7.0

In [46]:
```
df
```

Out[46]:

	one	two	three
a	1	2.0	2.0
b	3	NaN	6.0
c	3	7.0	NaN
d	5	NaN	7.0

In [36]:
```
df.fillna({'two':0,'three':100})    #用不同的值填充不同的列    ,是键值对所以写成字典的形式,用大括号
```

Out[36]:

	one	two	three
a	1	2.0	2.0
b	3	0.0	6.0
c	3	7.0	100.0
d	5	0.0	7.0

In [37]:
```
df
```

Out[37]:

	one	two	three
a	1	2.0	2.0
b	3	NaN	6.0
c	3	7.0	NaN
d	5	NaN	7.0

In [38]:
```
df.fillna(method='ffill') #前向填充，用同一列的前一行的那个数进行填充，默认axis=0,  forward
```

Out[38]:

	one	two	three
a	1	2.0	2.0
b	3	2.0	6.0
c	3	7.0	6.0
d	5	7.0	7.0

In [39]:
```
1 df.fillna(method='bfill') #后向填充  back  ,用同一列的后一行的那个数进行填充
```

Out[39]:

	one	two	three
a	1	2.0	2.0
b	3	7.0	6.0
c	3	7.0	7.0
d	5	NaN	7.0

In [47]:
```
1 df
```

Out[47]:

	one	two	three
a	1	2.0	2.0
b	3	NaN	6.0
c	3	7.0	NaN
d	5	NaN	7.0

In [41]:
```
1 df.fillna(df.mean()) #用所对应列的均值填充，按照列进行 max，min，sum ， mode(众数) ,median(中位数)
```

Out[41]:

	one	two	three
a	1	2.0	2.0
b	3	4.5	6.0
c	3	7.0	5.0
d	5	4.5	7.0

In [ ]:
```
1 #连续型数据用均值或中位数填充，如果是正态分布这种数据用均值填充,左偏分布或者右偏分布这种数据用中位数填充
2 #离散型数据用众数填充,看出现次数最多的
```

In [48]:
```
df.fillna(df.median())
```

Out[48]:

	one	two	three
a	1	2.0	2.0
b	3	4.5	6.0
c	3	7.0	6.0
d	5	4.5	7.0

In [49]:
```
df.mode()
#用众数填充的时候,当出现多个众数时,需要指定用哪一行的众数来填充
#需要先将想要用的众数行转换成序列,然后再填充
```

Out[49]:

	one	two	three
0	3.0	2.0	2.0
1	NaN	7.0	6.0
2	NaN	NaN	7.0

In [51]:
```
df.mode().iloc[0]
#将第一行的众数提取出来成为序列
```

Out[51]:
```
one      3.0
two      2.0
three    2.0
Name: 0, dtype: float64
```

In [52]:
```
df.fillna(df.mode().iloc[0])  #用提取出来的众数进行填充
```

Out[52]:

	one	two	three
a	1	2.0	2.0
b	3	2.0	6.0
c	3	7.0	2.0
d	5	2.0	7.0

2 重复值的处理

```
#重复值的处理两种:1.判断  2.删除
```

In [53]:
```
#重复值
df = pd.DataFrame(np.random.rand(2, 5),index = ['a', 'b'],columns = ['value1','value2','value3','value4','value5'])
df.loc['e']=[1,2,3,4,5]
df.loc['f']=[1,2,3,4,5]
df.loc['g']=[1,2,2,3,3]
df  #生成数据
```

Out[53]:

	value1	value2	value3	value4	value5
a	0.561918	0.515297	0.254660	0.944588	0.259432
b	0.445586	0.287430	0.519986	0.459388	0.502899
e	1.000000	2.000000	3.000000	4.000000	5.000000
f	1.000000	2.000000	3.000000	4.000000	5.000000
g	1.000000	2.000000	2.000000	3.000000	3.000000

2.1 重复值的判断

In [54]:
```
df.duplicated() #判断是否有重复，所有重复
#判断这一行和前面所有行是否整行重复,如果重复则为Ture,不重复则为False
#(是指这一行整行所有值和前面所有行是否整行完全一样,如果整行完全一样显示True,否则为False)
#无法做到去判断哪几行是重复的
```

Out[54]:
```
a    False
b    False
e    False
f     True
g    False
dtype: bool
```

In [55]:
```
df.duplicated().sum()  #有多少行是重复的
```

Out[55]:
```
1
```

In [56]:
```
df
```

Out[56]:

	value1	value2	value3	value4	value5
a	0.561918	0.515297	0.254660	0.944588	0.259432
b	0.445586	0.287430	0.519986	0.459388	0.502899
e	1.000000	2.000000	3.000000	4.000000	5.000000
f	1.000000	2.000000	3.000000	4.000000	5.000000
g	1.000000	2.000000	2.000000	3.000000	3.000000

In [58]:
```
df.duplicated(subset=['value1','value2']) #指定某个列重复则为重复
#指定列里面的某一行值一样,判断为在这两列下的这两行算重复
```

Out[58]:
```
a    False
b    False
e    False
f     True
g     True
dtype: bool
```

2.2 重复值的删除

In [59]:
```
1 df
```

Out[59]:

	value1	value2	value3	value4	value5
a	0.561918	0.515297	0.254660	0.944588	0.259432
b	0.445586	0.287430	0.519986	0.459388	0.502899
e	1.000000	2.000000	3.000000	4.000000	5.000000
f	1.000000	2.000000	3.000000	4.000000	5.000000
g	1.000000	2.000000	2.000000	3.000000	3.000000

In [60]:
```
1 df.drop_duplicates(keep='first') #删除行，这一行所有列的值与前面某一行所有列的值相同认为是重复
2 #对于完全一样的行(即重复行)保留一行即可,其余行可以都删除,而Keep是指定保留所有重复行里面的第一行还是最后一行
3 #(keep='first')代表保留所有重复行里面的第一行
4 #(keep='last')代表保留所有重复行里面的最后一行
```

Out[60]:

	value1	value2	value3	value4	value5
a	0.561918	0.515297	0.254660	0.944588	0.259432
b	0.445586	0.287430	0.519986	0.459388	0.502899
e	1.000000	2.000000	3.000000	4.000000	5.000000
g	1.000000	2.000000	2.000000	3.000000	3.000000

In [61]:
```
1 df.drop_duplicates(keep='last') #删除行，这一行所有列的值与前面某一行所有列的值相同认为是重复
```

Out[61]:

	value1	value2	value3	value4	value5
a	0.561918	0.515297	0.254660	0.944588	0.259432
b	0.445586	0.287430	0.519986	0.459388	0.502899
f	1.000000	2.000000	3.000000	4.000000	5.000000
g	1.000000	2.000000	2.000000	3.000000	3.000000

In [62]:
```
1 df
```

Out[62]:

	value1	value2	value3	value4	value5
a	0.561918	0.515297	0.254660	0.944588	0.259432
b	0.445586	0.287430	0.519986	0.459388	0.502899
e	1.000000	2.000000	3.000000	4.000000	5.000000
f	1.000000	2.000000	3.000000	4.000000	5.000000
g	1.000000	2.000000	2.000000	3.000000	3.000000

In [66]:
```
1 df.drop_duplicates(subset=['value1'],keep='first')  #value1和value2相同则去重复
2 #指定列里面的某一行值和其他行值一样,判断为在该列下的这两行算重复
3 #指定列里面的判断为重复行的,可以只保留其中一行,其他可删除,不用考虑其他列
4 # subset=['value1'] 指定列      keep='first'确定重复行里面保留第一行
```

Out[66]:

	value1	value2	value3	value4	value5
a	0.561918	0.515297	0.254660	0.944588	0.259432
b	0.445586	0.287430	0.519986	0.459388	0.502899
e	1.000000	2.000000	3.000000	4.000000	5.000000

In [67]:
```
1 df
```

Out[67]:

	value1	value2	value3	value4	value5
a	0.561918	0.515297	0.254660	0.944588	0.259432
b	0.445586	0.287430	0.519986	0.459388	0.502899
e	1.000000	2.000000	3.000000	4.000000	5.000000
f	1.000000	2.000000	3.000000	4.000000	5.000000
g	1.000000	2.000000	2.000000	3.000000	3.000000

In [68]:
```
1 df[df.duplicated()] #筛选出重复的行
```

Out[68]:

	value1	value2	value3	value4	value5
f	1.0	2.0	3.0	4.0	5.0

In [ ]:
```
1
```

3 数据的替换

In [71]:
```
df = pd.DataFrame([
    ['小ming', 18],
    ['小强', 19],
    ['小丽', np.NaN],
    ['小花', np.NaN]
],columns=['name', 'age'], index=['a','b','c','d'])
df
```

Out[71]:

	name	age
a	小ming	18.0
b	小强	19.0
c	小丽	NaN
d	小花	NaN

In [72]:
```
df.replace('小ming','小明',inplace=True)  #inplace为True时, 原地替换
#第一个参数是原名字,第二参数是替换后的名字,第三个参数为True则原地替换,为False则新生成操作
```

In [73]:
```
df
```

Out[73]:

	name	age
a	小明	18.0
b	小强	19.0
c	小丽	NaN
d	小花	NaN

In [77]:
```
df.replace('小明','ming',inplace=True)
df
```

Out[77]:

	name	age
a	ming	18.0
b	小强	19.0
c	小丽	NaN
d	小花	NaN

In [63]:
```
df=pd.read_clipboard()#从粘贴板读取数据
df
```

Out[63]:

	排行	名称	相关链接	10日涨跌幅	10日主力净流入	10日超大单净流入	10日大单净流入
0	1	电池	股吧详情	--	17.64亿	19.75亿	-2.10亿
1	2	汽车整车	股吧详情	--	9.33亿	16.74亿	-7.41亿
2	3	钢铁行业	股吧详情	--	5.41亿	7.23亿	-1.82亿
3	4	光伏设备	股吧详情	--	5.12亿	10.89亿	-5.77亿
4	5	小金属	股吧详情	--	4.91亿	21.42亿	-16.51亿

In [65]:
```
df.replace('--',np.nan,inplace=True)
```

In [66]:
```
1 df
```

Out[66]:

	排行	名称	相关链接	10日涨跌幅	10日主力净流入	10日超大单净流入	10日大单净流入
0	1	电池	股吧详情	NaN	17.64亿	19.75亿	-2.10亿
1	2	汽车整车	股吧详情	NaN	9.33亿	16.74亿	-7.41亿
2	3	钢铁行业	股吧详情	NaN	5.41亿	7.23亿	-1.82亿
3	4	光伏设备	股吧详情	NaN	5.12亿	10.89亿	-5.77亿
4	5	小金属	股吧详情	NaN	4.91亿	21.42亿	-16.51亿

4 算术运算和对齐

In [78]:
```
1 #序列的算术运算，索引自动对齐
2 s1=Series([3,4,5,6],index=['a','b','c','d']) #指定索引
3 s2=Series([4,5,6,3],index=['a','d','c','b']) #指定索引
4 s1+s2
```

Out[78]:
```
a     7
b     7
c    11
d    11
dtype: int64
```

4.1 dataframe之间进行运算

In [101]:
```
#dataframe的运算，索引自动对齐，行索引对齐，列索引对齐，不足的部分引入缺失值
df1 = DataFrame([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]],
    index=['a','c','b'],columns=['one','two','three'])

df2 = DataFrame([
    [1, 2, 3, 5],
    [4, 5, 6, 7],
    [7, 8, 9, 9],
    [7, 8, 1, 4]],
    index=['a','b','c','d'],columns=['one','two','three','four'])
df1+df2  #直接相加
```

Out[101]:

	four	one	three	two
a	NaN	2.0	6.0	4.0
b	NaN	11.0	15.0	13.0
c	NaN	11.0	15.0	13.0
d	NaN	NaN	NaN	NaN

In [102]:
```
df1.add(df2,fill_value=0) #调用方法,对有缺失的数据进行填充，填充完再进行相加
```

Out[102]:

	four	one	three	two
a	5.0	2.0	6.0	4.0
b	7.0	11.0	15.0	13.0
c	9.0	11.0	15.0	13.0
d	4.0	7.0	1.0	8.0

4.2 datafram和数字运算

In [81]:
```
1 df1
```

Out[81]:

	one	two	three
a	1	2	3
c	4	5	6
b	7	8	9

In [82]:
```
1 df1+10  #每一个元素都进行操作
```

Out[82]:

	one	two	three
a	11	12	13
c	14	15	16
b	17	18	19

4.3 映射：apply,map,applymap

In [83]:
```
1 df1
```

Out[83]:

	one	two	three
a	1	2	3
c	4	5	6
b	7	8	9

4.3.1 dataframe：apply

In [84]:
```
#apply
df1.apply(lambda x:x+10 )       #apply,针对行和列操作，可以对dataframe，可以对序列
#lambda 映射列,,,lambda x表示第x列,看的是整个全列.   x+10 表示这一列的所有元素全部进行+10操作
#x是一个临时变量,可以依次取值
```

Out[84]:

	one	two	three
a	11	12	13
c	14	15	16
b	17	18	19

In [85]:
```
def f(x):
    return x+10
df1.apply(f)    #标准的函数书写格式   x表示的是某一整列,是一个临时变量
```

Out[85]:

	one	two	three
a	11	12	13
c	14	15	16
b	17	18	19

In [91]:
```
def f(x):
    return x+10
df1.applymap(f)     #也可以用applymap,这时x就代表了元素
```

Out[91]:

	one	two	three
a	11	12	13
c	14	15	16
b	17	18	19

4.3.2 序列：apply，map

In [86]:
```
df1['one']
```

Out[86]:
```
a    1
c    4
b    7
Name: one, dtype: int64
```

In [87]:
```
df1['one'].apply(lambda x:x+10)
```

Out[87]:
```
a    11
c    14
b    17
Name: one, dtype: int64
```

In [88]:
```
df1['one'].map(lambda x:x+10)    #对序列中的单个元素进行操作
#也就是说这里的x代表的是元素本身,比如1,4,7这样的元素
```

Out[88]:
```
a    11
c    14
b    17
Name: one, dtype: int64
```

In [ ]:
```

```

4.3.3 dataframe：applymap

In [89]:
```
df1
```

Out[89]:

	one	two	three
a	1	2	3
c	4	5	6
b	7	8	9

In [90]:
```
#df1中小于5的元素加10
def f(x):
    if x<5:
        return x+10
    else:
        return x
df1.applymap(f)       #x代表的是元素
```

Out[90]:

	one	two	three
a	11	12	13
c	14	5	6
b	7	8	9

In [ ]:
```
apply：针对行和列操作，可以对dataframe，可以对序列
map：对序列中的单个元素进行操作，
applymap：对dataframe中的单个元素进行操作
```

In [92]:
```
df1
```

Out[92]:

	one	two	three
a	1	2	3
c	4	5	6
b	7	8	9

In [94]:
```
df1.apply(np.mean,axis=0)    #np.mean,std,sum,max  numpy系统带的一些函数
```

Out[94]:
```
one      4.0
two      5.0
three    6.0
dtype: float64
```

In [93]:
```
df1.applymap(np.mean)
#map是对单个元素进行运算的,
#这个写法代表对单个元素求平均,一个元素的平均值还是它自己本身,没有意义,所以apply和map是有区别的
```

Out[93]:

	one	two	three
a	1.0	2.0	3.0
c	4.0	5.0	6.0
b	7.0	8.0	9.0

In [95]:
```
df1.apply(np.mean,axis=1) #需要特别指定
```

Out[95]:
```
a    2.0
c    5.0
b    8.0
dtype: float64
```

In [103]:
```
df1
```

Out[103]:

	one	two	three
a	1	2	3
c	4	5	6
b	7	8	9

In [104]:
```
for i in range(3):   #指行
    for j in range(3):  #指列
        df1.iloc[i,j]+=1
df1
```

Out[104]:

	one	two	three
a	2	3	4
c	5	6	7
b	8	9	10

In [107]:
```
df=pd.DataFrame(np.arange(25).reshape(5,5))
df
```

Out[107]:

	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14
3	15	16	17	18	19
4	20	21	22	23	24

In [108]:
```
for i in range(4):  #对角线上的元素想要进行操作,可以用for循环,但是比较麻烦,不常用,不建议
    df.iloc[i,i]=df.iloc[i,i]+10
    df.iloc[i,i+1]=df.iloc[i,i+1]+10
df
```

Out[108]:

	0	1	2	3	4
0	10	11	2	3	4
1	5	16	17	8	9
2	10	11	22	23	14
3	15	16	17	28	29
4	20	21	22	23	24

In [111]:
```
#例子
df=pd.read_clipboard() #从粘贴板读取数据
df
```

Out[111]:

	排行	名称	相关链接	今日涨跌幅	今日主力净流入	今日超大单净流入	今日大单净流入
0	1	包钢股份	股吧详情	8.46%	9.30亿	11.12亿	-1.82亿
1	2	隆基股份	股吧详情	1.43%	5.98亿	5.68亿	3066.40万
2	3	贵州茅台	股吧详情	1.90%	5.21亿	8.79亿	-3.58亿
3	4	舍得酒业	股吧详情	4.68%	3.70亿	3.20亿	5004.48万
4	5	汇川技术	股吧详情	4.50%	3.28亿	4.03亿	-7497.96万

In [ ]:
```
#for循环实现
```

In [112]:
```
for i in range(5):
    df.loc[i,'今日涨跌幅']=np.float64(df.loc[i,'今日涨跌幅'].strip('%'))/100
```

In [113]:
```
1 df
```

Out[113]:

	排行	名称	相关链接	今日涨跌幅	今日主力净流入	今日超大单净流入	今日大单净流入
0	1	包钢股份	股吧详情	0.0846	9.30亿	11.12亿	-1.82亿
1	2	隆基股份	股吧详情	0.0143	5.98亿	5.68亿	3066.40万
2	3	贵州茅台	股吧详情	0.019	5.21亿	8.79亿	-3.58亿
3	4	舍得酒业	股吧详情	0.0468	3.70亿	3.20亿	5004.48万
4	5	汇川技术	股吧详情	0.045	3.28亿	4.03亿	-7497.96万

In [103]:
```
1 s='8.08%'
2 np.float64(s.strip('%'))/100
```

Out[103]: 0.0808

In [100]:
```
1 df['今日涨跌幅']=df['今日涨跌幅'].map(lambda x:np.float64(x.strip('%'))/100)
```

In [101]:
```
1 df
```

Out[101]:

	排行	名称	相关链接	今日涨跌幅	今日主力净流入	今日超大单净流入	今日大单净流入
0	1	包钢股份	股吧详情	0.0808	9.26亿	11.05亿	-1.79亿
1	2	隆基股份	股吧详情	0.0167	6.08亿	5.71亿	3745.27万
2	3	贵州茅台	股吧详情	0.0188	5.23亿	8.86亿	-3.62亿
3	4	舍得酒业	股吧详情	0.0448	3.69亿	3.22亿	4689.22万
4	5	汇川技术	股吧详情	0.0465	3.25亿	3.98亿	-7340.82万

In [115]:
```
df=pd.read_clipboard() #从粘贴板读取数据
df
```

Out[115]:

	排行	名称	相关链接	今日涨跌幅	今日主力净流入	今日超大单净流入	今日大单净流入
0	1	包钢股份	股吧详情	8.46%	9.30亿	11.12亿	-1.82亿
1	2	隆基股份	股吧详情	1.43%	5.98亿	5.68亿	3066.40万
2	3	贵州茅台	股吧详情	1.90%	5.21亿	8.79亿	-3.58亿
3	4	舍得酒业	股吧详情	4.68%	3.70亿	3.20亿	5004.48万
4	5	汇川技术	股吧详情	4.50%	3.28亿	4.03亿	-7497.96万

In [116]:
```
# 大于5%+0.1
#小于5%-0.1
def f(x):
    x1=np.float64(x.strip('%'))/100
    if x1>0.05:
        return x1+0.1
    else:
        return x1-0.1
df['今日涨跌幅'].map(f)
```

Out[116]:
```
0    0.1846
1   -0.0857
2   -0.0810
3   -0.0532
4   -0.0550
Name: 今日涨跌幅, dtype: float64
```

In [ ]:
```

```

5 排序

5.1 按照索引排序

In [109]:
```
d1=pd.DataFrame([[2,4,1,5],[3,1,4,5],[5,1,4,2]],columns=['b','a','d','c'],index=['one','two','four'])
d1
```

Out[109]:

	b	a	d	c
one	2	4	1	5
two	3	1	4	5
four	5	1	4	2

In [110]:
```
d1.sort_index(axis=0,ascending=True,inplace=True) #默认,升序
#sort_index 按索引排序 ,  ascending定义排序规则
#ascending=True按升序排序(一般不写这个参数默认按升序排)    ascending=False按降序排序
```

In [111]:
```
d1
```

Out[111]:

	b	a	d	c
four	5	1	4	2
one	2	4	1	5
two	3	1	4	5

In [112]:
```
d1.sort_index(axis=0,ascending=False) #降序
```

Out[112]:

	b	a	d	c
two	3	1	4	5
one	2	4	1	5
four	5	1	4	2

In [113]:
```
1 d1
```

Out[113]:

	b	a	d	c
four	5	1	4	2
one	2	4	1	5
two	3	1	4	5

In [114]:
```
1 d1.sort_index(axis=1,ascending=True) #沿着1轴排序
```

Out[114]:

	a	b	c	d
four	1	5	2	4
one	4	2	5	1
two	1	3	5	4

5.2 按照值排序

In [115]:
```
1 d1
```

Out[115]:

	b	a	d	c
four	5	1	4	2
one	2	4	1	5
two	3	1	4	5

In [116]:
```
d1.sort_values(by='b',ascending=True) #默认按照列排序
#里面有一个参数是axis,不写默认按照列的值进行排序
#看b列的值的大小,按照从小到大的顺序对行进行重新排序
```

Out[116]:

	b	a	d	c
one	2	4	1	5
two	3	1	4	5
four	5	1	4	2

In [117]:
```
d1
```

Out[117]:

	b	a	d	c
four	5	1	4	2
one	2	4	1	5
two	3	1	4	5

In [118]:
```
d1.sort_values(by='two',ascending=True,axis=1,inplace=True) #按照行排序, 需要指定轴
#按照two行的值的大小,从小到大对列进行重新排序,需要写上axis=1
```

In [120]:
```
d1
```

Out[120]:

	a	b	d	c
four	1	5	4	2
one	4	2	1	5
two	1	3	4	5

5.3 排序案例

5.3.1 按照值排序

In [121]:
```
df=pd.read_csv('taobao_data.csv')
df   #先读取数据
```

0	新款中老年女装春装雪纺打底衫妈妈装夏装中袖宽松上衣中年人t恤	99.0	16647	夏奈凤凰旗舰店	江苏
1	中老年女装清凉两件套妈妈装夏装大码短袖T恤上衣雪纺衫裙裤套装	286.0	14045	夏洛特的文艺	上海
2	母亲节衣服夏季妈妈装夏装套装短袖中年人40-50岁中老年女装T恤	298.0	13458	云新旗舰店	江苏
3	母亲节衣服中老年人春装女40岁50中年妈妈装套装夏装奶奶装两件套	279.0	13340	韶妃旗舰店	浙江
4	中老年女装春夏装裤大码 中年妇女40-50岁妈妈装夏装套装七分裤	59.0	12939	千百奈旗舰店	江苏
...	...	...	...	...	...
95	母亲节中老年女装运动服套装中年妈妈春装外套40岁50衣服2017新款	368.0	4041	欧芮嘉旗舰店	湖北
96	母亲节衣服中年妈妈装夏装短袖套装30中老年女装春装雪纺衫40岁50	181.0	4023	浅恋旗舰店	浙江
97	母亲节妈妈装夏装套装女40-50岁夏季衣服两件套中老年春装连衣裙	195.0	4000	若澜锦蒂旗舰店	浙江
98	母亲节衣服夏季中老年女装夏装短袖套装雪纺衫T恤妈妈装两件套	498.0	3968	蕴涵旗舰店	江苏
99	中老年女装春装t恤纱袖针织衫40-50岁妈妈装七分袖上衣夏装打底衫	688.0	3956	潮流前线9170	浙江

100 rows × 5 columns

In [122]:
```
1 df.info()  #查看数据类型
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   宝贝      100 non-null    object
 1   价格      100 non-null    float64
 2   成交量     100 non-null    int64
 3   卖家      100 non-null    object
 4   位置      100 non-null    object
dtypes: float64(1), int64(1), object(3)
memory usage: 4.0+ KB
```

In [123]:
```
1 df.sort_values(by='成交量',ascending=False) [:3]
2 #看一下成交量最多的宝贝
```

Out[123]:

	宝贝	价格	成交量	卖家	位置
0	新款中老年女装春装雪纺打底衫妈妈装夏装中袖宽松上衣中年人t恤	99.0	16647	夏奈凤凰旗舰店	江苏
1	中老年女装清凉两件套妈妈装夏装大码短袖T恤上衣雪纺衫裙裤套装	286.0	14045	夏洛特的文艺	上海
2	母亲节衣服夏季妈妈装夏装套装短袖中年人40-50岁中老年女装T恤	298.0	13458	云新旗舰店	江苏

In [124]:
```
1 df['成交量'].nlargest()
```

Out[124]:
```
0    16647
1    14045
2    13458
3    13340
4    12939
Name: 成交量, dtype: int64
```

In [ ]:
```
1 df.sort_values(by='成交量',ascending=False,inplace=True)
2 df[:3]  #原地操作需要调用这个表执行后续的操作
```

In [ ]:
```
df=pd.read_csv('taobao_data.txt')
df.head(3)
```

5.3.2 设置索引再排序

In [125]:
```
df.set_index('位置').sort_index()
```

Out[125]:

位置	宝贝	价格	成交量	卖家
上海	中老年女装夏装套装加肥加大码T恤上衣妈妈装时尚短袖夏季两件套	298.0	5325	简港旗舰店
上海	中老年女装清凉两件套妈妈装夏装大码短袖T恤上衣雪纺衫裙裤套装	286.0	14045	夏洛特的文艺
上海	中老年人女装套装妈妈装夏装大码奶奶装40-50岁60短袖T恤70两件套	29.0	4752	佳福妈妈商城
上海	母亲节衣服夏季中老年女装夏装套装上衣40-50岁妈妈装T恤衫两件套	198.0	7466	简港旗舰店
上海	母亲节中老年女装夏装短袖40-50岁雪纺衫大码妈妈装T恤宽上衣套装	49.0	4164	金良国际
...	...	...	...	...
湖北	母亲节妈妈夏装套装中老年雪纺衫中年女春装阔腿裤两件套40岁衣服	318.0	8580	薇诗琪旗舰店
湖北	母亲节衣服妈妈夏装雪纺连衣裙40岁50中年短袖中老年女装夏季裙子	306.0	4138	sonmellny尚曼妮旗舰店
湖北	中老年女夏装上衣短袖雪纺衬衫40-50岁大码胖妈妈装T恤母亲节衣服	178.0	4111	凯利娜格旗舰店
湖北	母亲节中老年女装运动服套装中年妈妈春装外套40岁50衣服2017新款	368.0	4041	欧芮嘉旗舰店
湖北	母亲节上衣服夏季中年妈妈春装雪纺衫夏装短袖中老年女装大码外套	256.0	5005	乔芙丽旗舰店

100 rows × 4 columns

5.3.3 多重排序

In [126]:
```
1 df.sort_values(by=['位置','价格'],ascending=[True,False]).set_index(['位置','价格'])
```

Out[126]:

位置	价格	宝贝	成交量	卖家
上海	298.0	中老年女装夏装套装加肥加大码T恤上衣妈妈装时尚短袖夏季两件套	5325	简港旗舰店
	286.0	中老年女装清凉两件套妈妈装夏装大码短袖T恤上衣雪纺衫裙裤套装	14045	夏洛特的文艺
	198.0	母亲节衣服夏季中老年女装夏装套装上衣40-50岁妈妈装T恤衫两件套	7466	简港旗舰店
	198.0	中老年女装夏装连衣裙中年雪纺上衣妈妈装中长款40-50岁大码裙子	5304	婆家娘家商城
	189.0	中老年女装夏装套装圆领上衣裤子夏季中年妈妈装短袖T恤两件套	11632	简港旗舰店
...	...	...	...	...
湖北	306.0	母亲节衣服妈妈夏装雪纺连衣裙40岁50中年短袖中老年女装夏季裙子	4138	sonmellny尚曼妮旗舰店
	256.0	母亲节上衣服夏季中年妈妈春装雪纺衫夏装短袖中老年女装大码外套	5005	乔芙丽旗舰店
	199.0	妈妈装春夏装T恤宽松雪纺衬衫40-50岁中老年女装大码中袖上衣套装	12398	千百萌旗舰店
	178.0	中老年女夏装上衣短袖雪纺衬衫40-50岁大码胖妈妈装T恤母亲节衣服	4111	凯利娜格旗舰店
	158.0	妈妈装春夏装雪纺衫大码中老年女装上衣中长袖t恤中年人打底衬衫	5001	千香旗舰店

100 rows × 3 columns

6 统计和描述

In [129]:
```
1 data=pd.read_csv('iris.csv') #读
2 data.head()
```

Out[129]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [130]:
```
1 data.iloc[:,:4].sum()     #mean.max.min var std  都是基于非空的数(如果有空值,会被忽略掉)
2 #数据类型为object的求和没有意义,所以一般也不进行求和
```

Out[130]:
```
sepal_length    876.5
sepal_width     458.6
petal_length    563.7
petal_width     179.9
dtype: float64
```

In [131]:
```
1 data.iloc[:,:4].mean()    #求平均
```

Out[131]:
```
sepal_length    5.843333
sepal_width     3.057333
petal_length    3.758000
petal_width     1.199333
dtype: float64
```

In [132]:
```
1 data.iloc[:,:4].sum(axis=1)  #分别求每一行的和
```

Out[132]:
```
0      10.2
1       9.5
2       9.4
3       9.4
4      10.2
       ...
145    17.2
146    15.7
147    16.7
148    17.3
149    15.8
Length: 150, dtype: float64
```

In [133]:
```
1 data.iloc[:,:4]
```

Out[133]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [134]:
```
1 data.count()  #计算每列的非空的数量  ， 计数
```

Out[134]:
```
sepal_length    150
sepal_width     150
petal_length    150
petal_width     150
species         150
dtype: int64
```

In [135]:
```
1 data['species'].unique()  #取值 ， 查看species有哪些个类别
```

Out[135]:
```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

In [136]:
```
1 data['species'].nunique()  #取值数量  ， 对species的类别进行计数,看species的类别有多少个
```

Out[136]:
```
3
```

In [137]:
```
1 data['species'].value_counts()  #分别查看species的每个类别的数量有多少
```

Out[137]:
```
setosa        50
versicolor    50
virginica     50
Name: species, dtype: int64
```

In [138]:
```
1 data.describe()     #查看 数值型统计描述，基于非空的
```

Out[138]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [141]:
```
1 data.describe(include='object')     #查看数据类型为object的数据统计描述信息
```

Out[141]:

	species
count	150
unique	3
top	setosa
freq	50

In [142]:
```
1 data.describe(include='O')  #查看数据类型为object的数据统计描述信息,,引号中是大写的欧
```

Out[142]:

	species
count	150
unique	3
top	setosa
freq	50

In [139]:
```
1 data.describe(include='all')  #查看所有数据类型的数据统计描述
```

Out[139]:

	sepal_length	sepal_width	petal_length	petal_width	species
count	150.000000	150.000000	150.000000	150.000000	150
unique	NaN	NaN	NaN	NaN	3
top	NaN	NaN	NaN	NaN	setosa
freq	NaN	NaN	NaN	NaN	50
mean	5.843333	3.057333	3.758000	1.199333	NaN
std	0.828066	0.435866	1.765298	0.762238	NaN
min	4.300000	2.000000	1.000000	0.100000	NaN
25%	5.100000	2.800000	1.600000	0.300000	NaN
50%	5.800000	3.000000	4.350000	1.300000	NaN
75%	6.400000	3.300000	5.100000	1.800000	NaN
max	7.900000	4.400000	6.900000	2.500000	NaN

7 数据的合并和重塑

7.1 拼接

In [143]:
```
df1 = DataFrame(np.ones((2,4))*2, index=['one','two'],columns=['a','b','c','d'])
df2 = DataFrame(np.ones((3,3))*1, index=['one','two','three'],columns=['b','d','e'])
df1
```

Out[143]:

	a	b	c	d
one	2.0	2.0	2.0	2.0
two	2.0	2.0	2.0	2.0

In [144]:
```
df2
```

Out[144]:

	b	d	e
one	1.0	1.0	1.0
two	1.0	1.0	1.0
three	1.0	1.0	1.0

7.1.1 concat

In [145]:
```
pd.concat([df1,df2],axis=1,join='outer')  #横着拼接，外连接
```

Out[145]:

	a	b	c	d	b	d	e
one	2.0	2.0	2.0	2.0	1.0	1.0	1.0
two	2.0	2.0	2.0	2.0	1.0	1.0	1.0
three	NaN	NaN	NaN	NaN	1.0	1.0	1.0

In [146]:
```
1 pd.concat([df1,df2],axis=1,join='inner')  #横着拼接，内连接  共有的行索引进行拼接
```

Out[146]:

	a	b	c	d	b	d	e
one	2.0	2.0	2.0	2.0	1.0	1.0	1.0
two	2.0	2.0	2.0	2.0	1.0	1.0	1.0

In [147]:
```
1 pd.concat([df1,df2],axis=0,join='outer')  #纵向拼接，外连接
```

Out[147]:

	a	b	c	d	e
one	2.0	2.0	2.0	2.0	NaN
two	2.0	2.0	2.0	2.0	NaN
one	NaN	1.0	NaN	1.0	1.0
two	NaN	1.0	NaN	1.0	1.0
three	NaN	1.0	NaN	1.0	1.0

In [148]:
```
1 pd.concat([df1,df2],axis=0,join='inner')  #纵向拼接，内连接，  共有的列索引进行拼接
```

Out[148]:

	b	d
one	2.0	2.0
two	2.0	2.0
one	1.0	1.0
two	1.0	1.0
three	1.0	1.0

In [149]:
```
pd.concat([df1,df2],axis=0,join='inner',ignore_index=True)  #重新设置索引
```

Out[149]:

	b	d
0	2.0	2.0
1	2.0	2.0
2	1.0	1.0
3	1.0	1.0
4	1.0	1.0

7.1.2 append

In [150]:
```
#append,只有纵向
df1=df1.append(df2,ignore_index=True) #新生成操作，外连接操作  ,生成0轴新的索引
df1
```

Out[150]:

	a	b	c	d	e
0	2.0	2.0	2.0	2.0	NaN
1	2.0	2.0	2.0	2.0	NaN
2	NaN	1.0	NaN	1.0	1.0
3	NaN	1.0	NaN	1.0	1.0
4	NaN	1.0	NaN	1.0	1.0

In [151]:
```
L=[1,2,3]
L.append(4) #原地操作
L
```

Out[151]: [1, 2, 3, 4]

7.2 主键合并

7.2.1 单个key

In [152]:
```
left = pd.DataFrame({'姓名':['张三','李四','王五', '老六'],
                     '数学':[90,89,92,70],
                     '物理':[90,86,85,70]})
right = pd.DataFrame({'姓名':['张三','李四','老六','王五'],
                      '英语':[89,90,94,78],
                      '语文':[80,90,70,60]})
left
```

Out[152]:

	姓名	数学	物理
0	张三	90	90
1	李四	89	86
2	王五	92	85
3	老六	70	70

In [153]:
```
right
```

Out[153]:

	姓名	英语	语文
0	张三	89	80
1	李四	90	90
2	老六	94	70
3	王五	78	60

In [ ]:
```
#方法1 把姓名设置为索引，然后cancat
```

In [154]:
```
1 pd.concat([left.set_index('姓名'),right.set_index('姓名')],axis=1)
```

Out[154]:

姓名	数学	物理	英语	语文
张三	90	90	89	80
李四	89	86	90	90
王五	92	85	78	60
老六	70	70	94	70

In [155]:
```
1 #方法2
2 pd.merge(left,right,on='姓名')
```

Out[155]:

	姓名	数学	物理	英语	语文
0	张三	90	90	89	80
1	李四	89	86	90	90
2	王五	92	85	78	60
3	老六	70	70	94	70

In [156]:
```
left = pd.DataFrame({'姓名':['张四','李四','王五', '老六'],
                     '数学':[90,89,92,70],
                     '物理':[90,86,85,70]})
right = pd.DataFrame({'姓名':['张三','李四','老六','王五'],
                      '英语':[89,90,94,78],
                      '语文':[80,90,70,60]})
pd.merge(left,right,on='姓名')  #默认以内连接的方式拼接
```

Out[156]:

	姓名	数学	物理	英语	语文
0	李四	89	86	90	90
1	王五	92	85	78	60
2	老六	70	70	94	70

In [157]:
```
pd.merge(left,right,on='姓名',how='outer')  #外连接
```

Out[157]:

	姓名	数学	物理	英语	语文
0	张四	90.0	90.0	NaN	NaN
1	李四	89.0	86.0	90.0	90.0
2	王五	92.0	85.0	78.0	60.0
3	老六	70.0	70.0	94.0	70.0
4	张三	NaN	NaN	89.0	80.0

In [158]:
```
1 pd.merge(left,right,on='姓名',how='left') #以左边为准  左连接
```

Out[158]:

	姓名	数学	物理	英语	语文
0	张四	90	90	NaN	NaN
1	李四	89	86	90.0	90.0
2	王五	92	85	78.0	60.0
3	老六	70	70	94.0	70.0

In [159]:
```
1 pd.merge(left,right,on='姓名',how='right')#右连接
```

Out[159]:

	姓名	数学	物理	英语	语文
0	张三	NaN	NaN	89	80
1	李四	89.0	86.0	90	90
2	老六	70.0	70.0	94	70
3	王五	92.0	85.0	78	60

7.2.2 两个key

In [160]:
```
left = pd.DataFrame({'姓名':['张三','李四','李四','王五','老六'],
                     '学号':[201,202,207,203,204],
                         '数学':[90,89,92,70,90],
                         '物理':[90,86,85,70,90]})
right = pd.DataFrame({'姓名':['张三','李四','李四','老六','王五'],
                      '学号':[201,207,202,204,203],
                          '英语':[89,90,94,78,90],
                          '语文':[80,90,70,60,90]})
left
```

Out[160]:

	姓名	学号	数学	物理
0	张三	201	90	90
1	李四	202	89	86
2	李四	207	92	85
3	王五	203	70	70
4	老六	204	90	90

In [161]:
```
right
```

Out[161]:

	姓名	学号	英语	语文
0	张三	201	89	80
1	李四	207	90	90
2	李四	202	94	70
3	老六	204	78	60
4	王五	203	90	90

In [162]:
```
1 pd.merge(left,right,on=['姓名','学号'],how='outer')
```

Out[162]:

	姓名	学号	数学	物理	英语	语文
0	张三	201	90	90	89	80
1	李四	202	89	86	94	70
2	李四	207	92	85	90	90
3	王五	203	70	70	90	90
4	老六	204	90	90	78	60

7.2.3 一对一，一对多，多对多问题

In [163]:
```
1 left = pd.DataFrame({'姓名':['张三','李四','王五', '老六'],
2                       '数学':[90,89,92,70],
3                       '物理':[90,86,85,70]})
4 right = pd.DataFrame({'姓名':['张三','李四','老六','王五'],
5                       '英语':[89,90,94,78],
6                       '语文':[80,90,70,60]})
7 left
```

Out[163]:

	姓名	数学	物理
0	张三	90	90
1	李四	89	86
2	王五	92	85
3	老六	70	70

In [164]:
```
1 right
```

Out[164]:

	姓名	英语	语文
0	张三	89	80
1	李四	90	90
2	老六	94	70
3	王五	78	60

In [165]:
```
1 pd.merge(left,right,on='姓名')   #1对1
```

Out[165]:

	姓名	数学	物理	英语	语文
0	张三	90	90	89	80
1	李四	89	86	90	90
2	王五	92	85	78	60
3	老六	70	70	94	70

In [166]:
```
left = pd.DataFrame({'姓名':['张三','李四','王五', '老六'],
                     '数学':[90,89,92,70],
                     '物理':[90,86,85,70]})
right = pd.DataFrame({'姓名':['张三','张三','老六','王五'],
                      '英语':[89,90,94,78],
                      '语文':[80,90,70,60]})
left
```

Out[166]:

	姓名	数学	物理
0	张三	90	90
1	李四	89	86
2	王五	92	85
3	老六	70	70

In [167]:
```
right
```

Out[167]:

	姓名	英语	语文
0	张三	89	80
1	张三	90	90
2	老六	94	70
3	王五	78	60

In [168]:
```
pd.merge(left,right,on='姓名',how='outer')    #1对多  这个时候就需要看一下数据是否有问题或者拼接过程有问题
```

Out[168]:

	姓名	数学	物理	英语	语文
0	张三	90	90	89.0	80.0
1	张三	90	90	90.0	90.0
2	李四	89	86	NaN	NaN
3	王五	92	85	78.0	60.0
4	老六	70	70	94.0	70.0

In [169]:
```
left = pd.DataFrame({'姓名':['张三','张三','王五', '老六'],
                                '数学':[90,89,92,70],
                                '物理':[90,86,85,70]})
right = pd.DataFrame({'姓名':['张三','张三','老六','王五'],
                                 '英语':[89,90,94,78],
                                 '语文':[80,90,70,60]})
left
```

Out[169]:

	姓名	数学	物理
0	张三	90	90
1	张三	89	86
2	王五	92	85
3	老六	70	70

In [170]:
```
1 right
```

Out[170]:

	姓名	英语	语文
0	张三	89	80
1	张三	90	90
2	老六	94	70
3	王五	78	60

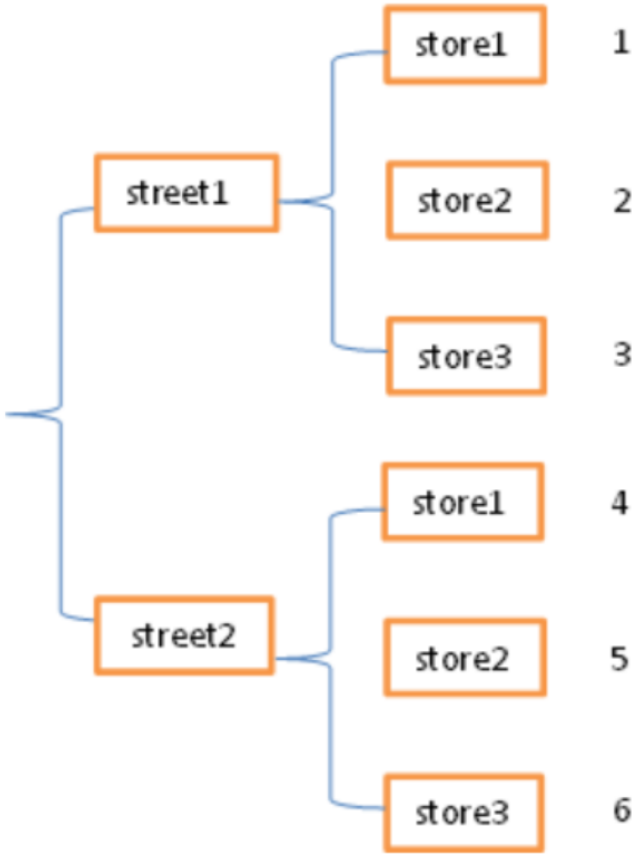
In [171]:
```
1 pd.merge(left,right,on='姓名',how='outer')    #多对多    这个时候就需要看一下数据是否有问题或者拼接过程有问题
```

Out[171]:

	姓名	数学	物理	英语	语文
0	张三	90	90	89	80
1	张三	90	90	90	90
2	张三	89	86	89	80
3	张三	89	86	90	90
4	王五	92	85	78	60
5	老六	70	70	94	70

7.3 重塑

```
1 重塑也可以叫行列变换,比如下面脑图(堆叠的形式)转换成表格的形式
```



	store1	store2	store3
street1	1	2	3
street2	4	5	6

In [172]:
```
df = DataFrame({'水果':['苹果','梨','草莓'],
                '数量':[3,4,5],
                '价格':[4,5,6]})
df
```

Out[172]:

	水果	数量	价格
0	苹果	3	4
1	梨	4	5
2	草莓	5	6

In [173]:
```
df1=df.stack()  #先进行一个 堆叠
df1
```

Out[173]:
```
0  水果     苹果
   数量       3
   价格       4
1  水果      梨
   数量       4
   价格       5
2  水果     草莓
   数量       5
   价格       6
dtype: object
```

In [174]:
```
df1.unstack()  #再做一个 展开
```

Out[174]:

	水果	数量	价格
0	苹果	3	4
1	梨	4	5
2	草莓	5	6

In [175]:
```
df = DataFrame({'ID':['101','101','102','102','103','103'],
                '物品':['小刀','笔','水','面包','笔','饼干'],
                '数量':[4,3,5,4,6,5]})
df  #商场购物数据
```

Out[175]:

	ID	物品	数量
0	101	小刀	4
1	101	笔	3
2	102	水	5
3	102	面包	4
4	103	笔	6
5	103	饼干	5

In [177]:
```
df.set_index(['ID','物品'])
```

Out[177]:

		数量
ID	**物品**	
101	**小刀**	4
	笔	3
102	**水**	5
	面包	4
103	**笔**	6
	饼干	5

In [178]:
```
1 df.set_index(['ID','物品']).unstack()   #重塑
```

Out[178]:

	数量				
物品	小刀	水	笔	面包	饼干
ID					
101	4.0	NaN	3.0	NaN	NaN
102	NaN	5.0	NaN	4.0	NaN
103	NaN	NaN	6.0	NaN	5.0

In [179]:
```
1 df
```

Out[179]:

	ID	物品	数量
0	101	小刀	4
1	101	笔	3
2	102	水	5
3	102	面包	4
4	103	笔	6
5	103	饼干	5

In [181]:
```
1 df.pivot?
```

In [180]:
```
df.pivot('ID','物品','数量')  #pivot(index=None, columns=None, values=None) 轴向旋转
#第一个参数是纵轴,第二个参数是横轴,第三个参数是值
#在这个里面pivot('ID','物品','数量')其实是pivot(index='ID', columns='物品', values='数量')
```

Out[180]:

物品	小刀	水	笔	面包	饼干
ID					
101	4.0	NaN	3.0	NaN	NaN
102	NaN	5.0	NaN	4.0	NaN
103	NaN	NaN	6.0	NaN	5.0

8 分组运算

8.1 groupby

In [182]:
```
df = DataFrame({'类别':['水果','水果','水果','蔬菜','蔬菜','肉类','肉类'],
                '产地':['美国','中国','中国','中国','新西兰','新西兰','美国'],
                '种类':['苹果','梨','草莓','番茄','黄瓜','羊肉','牛肉'],
               '数量':[5,5,9,3,2,10,8],
               '价格':[5,5,10,3,3,13,20]})
df
```

Out[182]:

	类别	产地	种类	数量	价格
0	水果	美国	苹果	5	5
1	水果	中国	梨	5	5
2	水果	中国	草莓	9	10
3	蔬菜	中国	番茄	3	3
4	蔬菜	新西兰	黄瓜	2	3
5	肉类	新西兰	羊肉	10	13
6	肉类	美国	牛肉	8	20

In [183]:
```
df[['类别','价格']].groupby('类别').mean()  #std,var,min,max,sum
```

Out[183]:

	价格
类别	
水果	6.666667
肉类	16.500000
蔬菜	3.000000

In [184]:
```
1 df[['类别','数量']].groupby('类别').sum()    #std,var,min,max,sum
```

Out[184]:

类别	数量
水果	19
肉类	18
蔬菜	5

In [185]:
```
1 df
```

Out[185]:

	类别	产地	种类	数量	价格
0	水果	美国	苹果	5	5
1	水果	中国	梨	5	5
2	水果	中国	草莓	9	10
3	蔬菜	中国	番茄	3	3
4	蔬菜	新西兰	黄瓜	2	3
5	肉类	新西兰	羊肉	10	13
6	肉类	美国	牛肉	8	20

In [186]:
```
1 df[['类别','价格','产地']].groupby(['产地','类别']).mean() #std,var,min,max,sum
```

Out[186]:

		价格
产地	类别	
中国	水果	7.5
	蔬菜	3.0
新西兰	肉类	13.0
	蔬菜	3.0
美国	水果	5.0
	肉类	20.0

In [187]:
```
1 df[['类别','数量','产地']].groupby(['产地','类别']).count() #统计数量
```

Out[187]:

		数量
产地	类别	
中国	水果	2
	蔬菜	1
新西兰	肉类	1
	蔬菜	1
美国	水果	1
	肉类	1

8.1.1 淘宝数据分析

In [205]:
```
df = pd.read_csv('taobao_data.csv')
df.head(2)
```

Out[205]:

	宝贝	价格	成交量	卖家	位置
0	新款中老年女装春装雪纺打底衫妈妈装夏装中袖宽松上衣中年人t恤	99.0	16647	夏奈凤凰旗舰店	江苏
1	中老年女装清凉两件套妈妈装夏装大码短袖T恤上衣雪纺衫裙裤套装	286.0	14045	夏洛特的文艺	上海

In [189]:
```
df[['卖家','价格','位置']].groupby(['位置']).mean()
```

Out[189]:

	价格
位置	
上海	161.200000
北京	150.000000
广东	326.000000
江苏	223.611364
河北	152.000000
河南	119.000000
浙江	290.428571
湖北	254.714286

In [190]:
```
1 df[['卖家','价格','位置']].groupby(['位置','卖家']).sum()
```

Out[190]:

		价格
位置	**卖家**	
上海	**xudong158**	99.0
	佳福妈妈商城	29.0
	夏洛特的文艺	286.0
	妃莲慕旗舰店	266.0
	婆家娘家商城	198.0
...	**...**	...
湖北	**凯利娜格旗舰店**	178.0
	千百萌旗舰店	199.0
	千香旗舰店	158.0
	欧芮嘉旗舰店	368.0
	薇诗琪旗舰店	318.0

85 rows × 1 columns

In [191]:
```
1 df[['卖家','成交量','位置']].groupby(['位置','卖家']).sum()
```

Out[191]:

		成交量
位置	**卖家**	
上海	**xudong158**	4572
	佳福妈妈商城	4752
	夏洛特的文艺	14045
	妃莲慕旗舰店	10755
	婆家娘家商城	5304
...	**...**	...
湖北	**凯利娜格旗舰店**	4111
	千百萌旗舰店	12398
	千香旗舰店	5001
	欧芮嘉旗舰店	4041
	薇诗琪旗舰店	8580

85 rows × 1 columns

In [192]:
```
1 df[['卖家','成交量','宝贝']].groupby(['卖家','宝贝']).sum()
```

Out[192]:

		成交量
卖家	宝贝	
bobolove987	新母亲节妈妈夏装短袖 中老年T恤衫宽松牛奶丝上衣圆领印花短袖	4261
ceo放牛	妈妈夏装两件套母亲节衣服老人上衣60-70岁人夏季中老年女装套装	11655
hi大脚丫	中老年女装绵绸连衣裙40-50岁中年妈妈装短袖大码修身碎花裙子夏	4460
kewang5188	中老年人女装套装夏装妈妈装套装60-70-80岁老奶奶短袖上衣七分裤	5348
loueddssd倍艾旗舰店	中老年女裤夏装薄款中年松紧高腰九分裤老年人宽松大码妈妈裤子女	7675
...	...	...
韶妃旗舰店	母亲节衣服中老年人春装女40岁50中年妈妈装套装夏装奶奶装两件套	13340
香颜旗舰店	中老年女装夏装套装妈妈装两件套雪纺衫中年春装T恤母亲节上衣服	5908
	母亲节妈妈装夏装真丝T恤中老年女装短袖上衣服中年女桑蚕丝衬衫	4314
	香颜妈妈装春装连衣裙中老年女装夏装两件套雪纺衫中年套装裙子	4249
高康洁	中老年女装妈妈装夏装中年两件套装40-50岁印花套装老年人夏装t恤	6795

100 rows × 1 columns

8.1.1.1 哪个地方的成交量最多

In [195]:
```
1 df.groupby('位置')['成交量'].sum().sort_values(ascending=False)
```

Out[195]:
```
位置
江苏    309360
浙江    161826
上海     68015
湖北     43274
北京     27116
河北     18152
河南      5986
广东      5164
Name: 成交量, dtype: int64
```

In [196]:
```
1 df.groupby('位置')['成交量'].sum().nlargest(3)
```

Out[196]:
```
位置
江苏    309360
浙江    161826
上海     68015
Name: 成交量, dtype: int64
```

In [214]:
```
1 df[['位置','成交量']].groupby('位置').sum().sort_values(by='成交量',ascending=False)
```

Out[214]:

位置	成交量
江苏	309360
浙江	161826
上海	68015
湖北	43274
北京	27116
河北	18152
河南	5986
广东	5164

8.1.1.2 哪个卖家成交量最多

In [197]:
```
1 df.groupby('卖家')['成交量'].sum().nlargest(3)
```

Out[197]:
```
卖家
简港旗舰店      24423
朵莹旗舰店      21957
蒲洛妃旗舰店     21683
Name: 成交量, dtype: int64
```

8.1.1.3 哪个商家的成交额最多

In [206]:
```
1 df.insert(3,'成交额',df['价格']*df['成交量'])
```

In [207]:
```
1 df
```

Out[207]:

	宝贝	价格	成交量	成交额	卖家	位置
0	新款中老年女装春装雪纺打底衫妈妈装夏装中袖宽松上衣中年人t恤	99.0	16647	1648053.0	夏奈凤凰旗舰店	江苏
1	中老年女装清凉两件套妈妈装夏装大码短袖T恤上衣雪纺衫裙裤套装	286.0	14045	4016870.0	夏洛特的文艺	上海
2	母亲节衣服夏季妈妈装夏装套装短袖中年人40-50岁中老年女装T恤	298.0	13458	4010484.0	云新旗舰店	江苏
3	母亲节衣服中老年人春装女40岁50中年妈妈装套装夏装奶奶装两件套	279.0	13340	3721860.0	韶妃旗舰店	浙江
4	中老年女装春夏装裤大码 中年妇女40-50岁妈妈装夏装套装七分裤	59.0	12939	763401.0	千百奈旗舰店	江苏
...	...	...	...	...	...	...
95	母亲节中老年女装运动服套装中年妈妈春装外套40岁50衣服2017新款	368.0	4041	1487088.0	欧芮嘉旗舰店	湖北
96	母亲节衣服中年妈妈装夏装短袖套装30中老年女装春装雪纺衫40岁50	181.0	4023	728163.0	浅恋旗舰店	浙江
97	母亲节妈妈装夏装套装女40-50岁夏季衣服两件套中老年春装连衣裙	195.0	4000	780000.0	若澜锦蒂旗舰店	浙江
98	母亲节衣服夏季中老年女装夏装短袖套装雪纺衫T恤妈妈装两件套	498.0	3968	1976064.0	蕴涵旗舰店	江苏

In [208]:
```
1 df.groupby('卖家')['成交额'].sum().nlargest(3)
```

Out[208]:
```
卖家
潮流前线9170     6935336.0
朵莹旗舰店       5772296.0
简港旗舰店       5263566.0
Name: 成交额, dtype: float64
```

8.1.1.4 各个位置成交额最多的商家（前3家）

In [213]:
```
def s(group):
    return group.nlargest(3,columns='成交额')
df[['位置','成交额','卖家']].groupby(['位置']).apply(s).droplevel(1).drop('位置',axis=1)
```

Out[213]:

位置	成交额	卖家
上海	4016870.0	夏洛特的文艺
上海	2198448.0	简港旗舰店
上海	1586850.0	简港旗舰店
北京	1296000.0	妈妈装工厂店1988
北京	875112.0	凯飞服饰1717
北京	615480.0	hi大脚丫
广东	1683464.0	安静式风格
江苏	4010484.0	云新旗舰店
江苏	3481056.0	zxtvszml
江苏	3473190.0	ceo放牛
河北	1221264.0	loueddssd倍艾旗舰店
河北	990075.0	loueddssd倍艾旗舰店
河北	555861.0	艾尔梦艺娜旗舰店
河南	712334.0	z昊铭么么哒666
浙江	4213608.0	潮流前线9170
浙江	3721860.0	韶妃旗舰店
浙江	3627506.0	简狐旗舰店
湖北	2728440.0	薇诗琪旗舰店
湖北	2467202.0	千百萌旗舰店
湖北	1487088.0	欧芮嘉旗舰店

8.1.2 分组填充

In [215]:
```
#分组填充
df = pd.DataFrame([
    ['1','小明', 18,160],
    ['2','小强', 19,170],
    ['2','小丽', 20,170],
    ['2','小花', np.NaN,np.NaN],
    ['1','小张', np.NaN,np.NaN]
],columns=['年级','name', 'age','height'], index=['a','b','c','d','e'])
df
```

Out[215]:

	年级	name	age	height
a	1	小明	18.0	160.0
b	2	小强	19.0	170.0
c	2	小丽	20.0	170.0
d	2	小花	NaN	NaN
e	1	小张	NaN	NaN

In [216]:
```
df['age']=df[['年级','age']].groupby('年级').apply(lambda x:x.fillna(x.mean()))['age'].droplevel(0)
```

In [217]:
```
1 df
```

Out[217]:

	年级	name	age	height
a	1	小明	18.0	160.0
b	2	小强	19.0	170.0
c	2	小丽	20.0	170.0
d	2	小花	19.5	NaN
e	1	小张	18.0	NaN

8.2 数据聚合

In [218]:
```
df = DataFrame({'类别':['水果','水果','水果','蔬菜','蔬菜','肉类','肉类'],
                '产地':['美国','中国','中国','中国','新西兰','新西兰','美国'],
                '种类':['苹果','梨','草莓','番茄','黄瓜','羊肉','牛肉'],
               '数量':[5,5,9,3,2,10,8],
               '价格':[5,5,10,3,3,13,20]})
df
```

Out[218]:

	类别	产地	种类	数量	价格
0	水果	美国	苹果	5	5
1	水果	中国	梨	5	5
2	水果	中国	草莓	9	10
3	蔬菜	中国	番茄	3	3
4	蔬菜	新西兰	黄瓜	2	3
5	肉类	新西兰	羊肉	10	13
6	肉类	美国	牛肉	8	20

8.2.1 传入标准函数

In [219]:
```
df.groupby('类别').agg(np.sum)#传入标准函数 std, var, min,max
```

Out[219]:

	数量	价格
类别		
水果	19	20
肉类	18	33
蔬菜	5	6

In [ ]:
```

```

8.2.2 不同的列不同的聚合函数

In [221]:
```
mapping={'数量':np.sum,'价格':np.mean} # 传入标准函数，不同的列传入不同的函数
df.groupby('类别').agg(mapping)     #用字典的形式来表示不同的列做不通过的运算
#这个代码是对数量进行求和,对价格进行求平均,按照类别分组
```

Out[221]:

类别	数量	价格
水果	19	6.666667
肉类	18	16.500000
蔬菜	5	3.000000

8.2.3 自定义函数

In [222]:
```
df
```

Out[222]:

	类别	产地	种类	数量	价格
0	水果	美国	苹果	5	5
1	水果	中国	梨	5	5
2	水果	中国	草莓	9	10
3	蔬菜	中国	番茄	3	3
4	蔬菜	新西兰	黄瓜	2	3
5	肉类	新西兰	羊肉	10	13
6	肉类	美国	牛肉	8	20

In [223]:
```
#峰峰值 （求极差）
def ptp(x):
    return x.max()-x.min()

df[['类别','价格']].groupby(['类别']).agg([ptp]) #自定义函数
```

Out[223]:

	价格
	ptp
类别	
水果	5
肉类	7
蔬菜	0

8.2.4 调用多个聚合函数

In [224]:
```
df[['类别','价格']].groupby(['类别']).agg([np.mean,np.max,np.min,ptp])  #可以调用多个聚合函数
```

Out[224]:

	价格			
	mean	amax	amin	ptp
类别				
水果	6.666667	10	5	5
肉类	16.500000	20	13	7
蔬菜	3.000000	3	3	0

8.3 透视表和交叉表

8.3.1 透视表

In [230]:
```
1 df
```

Out[230]:

	类别	产地	种类	数量	价格
0	水果	美国	苹果	5	5
1	水果	中国	梨	5	5
2	水果	中国	草莓	9	10
3	蔬菜	中国	番茄	3	3
4	蔬菜	新西兰	黄瓜	2	3
5	肉类	新西兰	羊肉	10	13
6	肉类	美国	牛肉	8	20

In [231]:
```
1 df[['类别','价格']].pivot_table(index=['类别']) #先对类别groupby，再对价格求mean(默认)
```

Out[231]:

	价格
类别	
水果	6.666667
肉类	16.500000
蔬菜	3.000000

In [227]:
```
df.pivot_table(index=['产地','类别']) #先对产地再对类别groupby，再求mean
#没有提前做df[['类别','价格']].,就是没有进行选择,只是用pivot_table(index=['产地','类别'])进行了分组,
#默认对价格和数量都进行求平均
```

Out[227]:

		价格	数量
产地	**类别**		
中国	**水果**	7.5	7
	蔬菜	3.0	3
新西兰	**肉类**	13.0	10
	蔬菜	3.0	2
美国	**水果**	5.0	5
	肉类	20.0	8

In [232]:
```
df.pivot_table(index=['产地','类别'],aggfunc=np.sum)  #使用定义的聚合函数
```

Out[232]:

		价格	数量
产地	**类别**		
中国	**水果**	15	14
	蔬菜	3	3
新西兰	**肉类**	13	10
	蔬菜	3	2
美国	**水果**	5	5
	肉类	20	8

In [233]:
```
df.pivot_table(index=['类别'],aggfunc=ptp)  #自定义函数
#下面是一个警告,产地和种类没有办法进行计算,所以最好是计算哪些量就先选出来,然后再进行计算
#没有问题的写法如下
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\groupby\generic.py:303: FutureWarning: Dropping invalid columns in SeriesGroupBy.agg is deprecated. In a future version, a TypeError will be raised. Before calling .agg, select only columns which should be valid for the aggregating function.
  results[key] = self.aggregate(func)
```

Out[233]:

类别	价格	数量
水果	5	4
肉类	7	2
蔬菜	0	1

In [235]:
```
df[['类别','价格','数量']].pivot_table(index=['类别'],aggfunc=ptp)  #自定义函数
```

Out[235]:

类别	价格	数量
水果	5	4
肉类	7	2
蔬菜	0	1

8.3.2 交叉表

In [236]:
```
1 df
```

Out[236]:

	类别	产地	种类	数量	价格
0	水果	美国	苹果	5	5
1	水果	中国	梨	5	5
2	水果	中国	草莓	9	10
3	蔬菜	中国	番茄	3	3
4	蔬菜	新西兰	黄瓜	2	3
5	肉类	新西兰	羊肉	10	13
6	肉类	美国	牛肉	8	20

In [237]:
```
1 pd.crosstab(df['产地'],df['类别'])  #统计的是分组的频率
2 #crosstab(,)里面的参数换了位置的话,轴会换位置
```

Out[237]:

类别	水果	肉类	蔬菜
产地			
中国	2	0	1
新西兰	0	1	1
美国	1	1	0

In [239]:
```
1 pd.crosstab(df['类别'],df['产地'])
```

Out[239]:

产地	中国	新西兰	美国
类别			
水果	2	0	1
肉类	0	1	1
蔬菜	1	1	0

In [238]:
```
1 df[['产地','类别','种类']].groupby(['产地','类别']).count().unstack()
```

Out[238]:

	种类		
类别	水果	肉类	蔬菜
产地			
中国	2.0	NaN	1.0
新西兰	NaN	1.0	1.0
美国	1.0	1.0	NaN

In [ ]:
```
1
```