In [1]:
```
import numpy as np
import pandas as pd
pd.__version__
```

Out[1]:
```
'1.2.4'
```

一、查找~排序

In [2]:
```
header=pd.Index(['体力值','种族值','战斗力'],name='属性')
list1=['男','男','男','男','女','女','女']
list2=['百里守约','项羽','猪八戒','蔡徐坤','上官婉儿','小乔','蔡文姬']
arr=np.random.randint(1,100,[7,3])
index_col=pd.MultiIndex.from_arrays([list1,list2],names=['性别','姓名'])
df=pd.DataFrame(data=arr,index=index_col,columns=header)
arr,df
```

Out[2]:
```
(array([[84, 21, 19],
        [42,  3, 72],
        [43, 90, 78],
        [29, 62, 24],
        [88, 15, 86],
        [17, 63, 79],
        [76, 25, 33]]),
 属性        体力值  种族值  战斗力
 性别 姓名
 男  百里守约   84   21   19
    项羽      42    3   72
    猪八戒     43   90   78
    蔡徐坤     29   62   24
 女  上官婉儿    88   15   86
    小乔      17   63   79
    蔡文姬     76   25   33)
```

(1).通过布尔DataFrame筛选数据

- 先分别看看布尔型的ndarray，DataFrame长什么样的

In [3]:
```
arr>50                  ##这是一个布尔ndarray
```

Out[3]:
```
array([[ True, False, False],
       [False, False,  True],
       [False,  True,  True],
       [False,  True, False],
       [ True, False,  True],
       [False,  True,  True],
       [ True, False, False]])
```

In [4]:
```
df>50                   ##这是一个布尔DataFrame
```

Out[4]:

属性		体力值	种族值	战斗力
性别	姓名			
男	百里守约	True	False	False
	项羽	False	False	True
	猪八戒	False	True	True
	蔡徐坤	False	True	False
女	上官婉儿	True	False	True
	小乔	False	True	True
	蔡文姬	True	False	False

In [5]:
```
df["战斗力"]>40             ##这是一个布尔Series
```

Out[5]:
```
性别  姓名
男   百里守约    False
    项羽       True
    猪八戒      True
    蔡徐坤     False
女   上官婉儿     True
    小乔       True
    蔡文姬     False
Name: 战斗力, dtype: bool
```

- 把布尔DataFrame作为参数传入[ ]中，会将False的地方的值变成NaN
- 由于NaN是float浮点型的某个特殊值，所以存储数据的intblock会变成floatblock

In [6]:
```
df[df>50]               ##每一列的block会从intblock变成floatblock
```

Out[6]:

性别	姓名	属性	体力值	种族值	战斗力
男	百里守约		84.0	NaN	NaN
	项羽		NaN	NaN	72.0
	猪八戒		NaN	90.0	78.0
	蔡徐坤		NaN	62.0	NaN
女	上官婉儿		88.0	NaN	86.0
	小乔		NaN	63.0	79.0
	蔡文姬		76.0	NaN	NaN

- 把布尔Series作为参数传入[ ]中，会把False的行删除，只保留True的行

In [7]:
```
df[df['战斗力']>40]
```

Out[7]:

性别	姓名	属性	体力值	种族值	战斗力
男	项羽		42	3	72
	猪八戒		43	90	78
女	上官婉儿		88	15	86
	小乔		17	63	79

- 实际上面传入bool序列即可，本质[ ]中的参数是个序列

In [8]:
```
(df["战斗力"]>40).values
```

Out[8]:
```
array([False,  True,  True, False,  True,  True, False])
```

In [9]:
```
df[ [True, False,  True, False, False, False, False] ]
```

Out[9]:

性别	姓名	属性	体力值	种族值	战斗力
男	百里守约		84	21	19
	猪八戒		43	90	78

In [10]:
```
df[(df["战斗力"]>40).values]
```

Out[10]:

性别	姓名	属性	体力值	种族值	战斗力
男	项羽		42	3	72
	猪八戒		43	90	78
女	上官婉儿		88	15	86
	小乔		17	63	79

- 通过.loc[ ]实现上面的数据筛选

In [11]:
```
df.loc[df["战斗力"]>40,:]
```

Out[11]:

性别	姓名	属性	体力值	种族值	战斗力
男	项羽		42	3	72
	猪八戒		43	90	78
女	上官婉儿		88	15	86
	小乔		17	63	79

In [12]:
```
df.loc[df['战斗力']>20,'体力值']      ##将战斗力大于20的人的体力值输出
```

Out[12]:
```
性别  姓名
男   项羽       42
    猪八戒      43
    蔡徐坤      29
女   上官婉儿     88
    小乔       17
    蔡文姬      76
Name: 体力值, dtype: int32
```

- 上面都是通过战斗力一列进行数据筛选的例子，下面给出通过行的条件进行筛选的原理，因为用的不多，所以不编例子了

分析：.loc[]第一个参数实际是一个控制行的bool列表，有多少行这个bool列表就需要多长，同理第二个参数需要相应列数的bool列表控制列

In [13]:
```
df.loc[[True, False,  False, False, False, False, True],:]   ##取第0行和最后一行
```

Out[13]:

性别	属性 姓名	体力值	种族值	战斗力
男	百里守约	84	21	19
女	蔡文姬	76	25	33

In [14]:
```
df.loc[:,[True,False,True]]   ##取第0列和最后一列
```

Out[14]:

性别	属性 姓名	体力值	战斗力
男	百里守约	84	19
	项羽	42	72
	猪八戒	43	78
	蔡徐坤	29	24
女	上官婉儿	88	86
	小乔	17	79
	蔡文姬	76	33

（2）.query()方法查找

注：这里条件筛选查找，全都是针对每列数据的条件

In [15]:
```
df[df['战斗力']>30]
```

Out[15]:

性别	属性 姓名	体力值	种族值	战斗力
男	项羽	42	3	72
	猪八戒	43	90	78
女	上官婉儿	88	15	86
	小乔	17	63	79
	蔡文姬	76	25	33

In [16]:
```
df.query("战斗力>30")
```

Out[16]:

性别	属性 姓名	体力值	种族值	战斗力
男	项羽	42	3	72
	猪八戒	43	90	78
女	上官婉儿	88	15	86
	小乔	17	63	79
	蔡文姬	76	25	33

In [17]:
```
df.query("战斗力>30 and 体力值<80")
```

Out[17]:

性别	姓名	属性	体力值	种族值	战斗力
男	项羽		42	3	72
	猪八戒		43	90	78
女	小乔		17	63	79
	蔡文姬		76	25	33

（3）.sort_index()对指定轴的标签排序

参数：

- axis：指定沿着那个轴排序。如果为0/'index'，则对沿着0轴，对行label排序；如果为1/'columns'，则沿着 1轴对列label排序。
- level：一个整数、label、整数列表、label list或者None。对于多级索引，它指定在哪一级上排序。
- ascending：一个布尔值，如果为True，则升序排序；如果是False，则降序排序。
- inplace：一个布尔值，如果为True，则原地修改。如果为False，则返回排好序的新对象
- kind：一个字符串，指定排序算法。可以为'quicksort'/'mergesort'/'heapsort'。注意只有归并排序是稳定排序的
- na_position：一个字符串，值为'first'/'last'，指示：将NaN排在最开始还是最末尾。
- sort_remaining：一个布尔值。如果为True，则当多级索引排序中，指定level的索引排序完毕后，对剩下level的索引也排序。

In [18]:
```
df2 = pd.DataFrame({'c1':[1,3,2,4],'c2':[11,14,13,12]},index=['a','c','b','d'])
df2
```

Out[18]:

	c1	c2
a	1	11
c	3	14
b	2	13
d	4	12

In [19]:
```
df2.sort_index()  ##默认对行标签进行升序排序
```

Out[19]:

	c1	c2
a	1	11
b	2	13
c	3	14
d	4	12

In [20]:
```
df2.sort_index(ascending=False)  ##设置ascending使行标签降序排序
```

Out[20]:

	c1	c2
d	4	12
c	3	14
b	2	13
a	1	11

In [21]:
```
df2.sort_index(axis=1,ascending=False) ##行标签axis=0，列标签axis=1
```

Out[21]:

	c2	c1
a	11	1
c	14	3
b	13	2
d	12	4

（4）.sort_values()对_data的ndarray进行排序

参数：

- by：一个字符串或者字符串的列表，指定希望对那些label对应的列或者行的元素进行排序。对于DataFrame，必须指定该参数。而Series不能指定该参数。

如果是一个字符串列表，则排在前面的label的优先级较高。它指定了用于比较的字段

- axis：指定沿着那个轴排序。如果为0/'index'，则沿着0轴排序（此时by指定列label，根据该列的各元素大小，重排列各行）；如果为1/'columns'，则沿着 1轴排序（此时by指定行label，根据该行的各元素大小，重排列各列）。
- ascending：一个布尔值，如果为True，则升序排序；如果是False，则降序排序。

- inplace：一个布尔值，如果为True，则原地修改。如果为False，则返回排好序的新对象
- kind:一个字符串，指定排序算法。可以为'quicksort'/'mergesort'/'heapsort'。注意只有归并排序是稳定排序的
- na_position：一个字符串，值为'first'/'last'，指示：将NaN排在最开始还是最末尾。

In [22]:
```
df3 = pd.DataFrame({'a':[1,3,21,4],'c':[11,14,130,12]},index=['a','c','b','d'])
df3
```

Out[22]:

	a	c
a	1	11
c	3	14
b	21	130
d	4	12

In [23]:
```
df3.sort_values(by='a',axis=0)  ##对a列排序，同行数据跟着动
```

Out[23]:

	a	c
a	1	11
c	3	14
d	4	12
b	21	130

In [24]:
```
df3.sort_values(by='a',axis=1)  ##对a行排序，同列数据跟着动
```

Out[24]:

	a	c
a	1	11
c	3	14
b	21	130
d	4	12

二、插入~删除~修改

In [25]:
```
df4 = pd.DataFrame({'a':[1,3,21],'c':[11,14,130]},index=['a','c','b'])   ##以下代码必须从这里依次往下按顺序运行，否则因为内存中数据
df4
```

Out[25]:

	a	c
a	1	11
c	3	14
b	21	130

（1）.使用.loc[ ]在尾部插入一行或一列

In [26]:
```
df4.loc[:,'b']=[2,4,6]          ##注意，如果标签b填在第一个参数位置，由于行标签已经存在b了，此时会变成修改值的操作
df4
```

Out[26]:

	a	c	b
a	1	11	2
c	3	14	4
b	21	130	6

In [27]:
```
df4.loc['insert_col',:]=[33,44,55]    ##插入一行
df4
```

Out[27]:

	a	c	b
a	1.0	11.0	2.0
c	3.0	14.0	4.0
b	21.0	130.0	6.0
insert_col	33.0	44.0	55.0

（2）.使用.insert(loc, column, value,allow_duplicates = False)插入一列

参数

- loc 必要字段，int类型数据，表示插入新列的列位置，原来在该位置的列将向右移。
- column 必要字段，插入新列的列名。
- value 必要字段，新列插入的值。如果仅提供一个值，将为所有行设置相同的值。可以是int，string，float等，甚至可以是series /值列表。
- allow_duplicates 布尔值，用于检查是否存在具有相同名称的列。默认为False，不允许与已有的列名重复。

In [28]:
```
df4.insert(loc=0,column='insert',value=[11,11,11,11])
df4
```

Out[28]:

	insert	a	c	b
a	11	1.0	11.0	2.0
c	11	3.0	14.0	4.0
b	11	21.0	130.0	6.0
insert_col	11	33.0	44.0	55.0

（5）.使用.drop去除一行或一列

参数：

- labels：单个label或者一个label序列，代表要被丢弃的label
- axis：一个整数，或者轴的名字。默认为 0 轴
- level：一个整数或者level名字，用于MultiIndex。因为可能在多个level上都有同名的label。
- inplace：一个布尔值。如果为True，则原地修改并且返回None
- errors：可以为'ignore'/'raise'

In [29]:
```
df4.drop(labels='insert',axis=1,inplace=True)
df4
```

Out[29]:

	a	c	b
a	1.0	11.0	2.0
c	3.0	14.0	4.0
b	21.0	130.0	6.0
insert_col	33.0	44.0	55.0

In [30]:
```
df4.drop(labels='insert_col',axis=0,inplace=True)
df4
```

Out[30]:

	a	c	b
a	1.0	11.0	2.0
c	3.0	14.0	4.0
b	21.0	130.0	6.0

（6）.使用.loc[ ]赋值的方法修改值

In [31]:
```
df4.loc['a',:]=df4.loc['a',:]*2+1
df4
```

Out[31]:

	a	c	b
a	3.0	23.0	5.0
c	3.0	14.0	4.0
b	21.0	130.0	6.0

(7). 使用.replace()批量替换某些值

In [32]:
```
df5=pd.DataFrame({'k1':['one'] * 3 + ['two'] * 4,'k2':[1, 1, 2, 3, 3, 4, 4]})
df5
```

Out[32]:

	k1	k2
0	one	1
1	one	1
2	one	2
3	two	3
4	two	3
5	two	4
6	two	4

In [33]:
```
df5.replace(1, 5)     # 把所有1替换为5
```

Out[33]:

	k1	k2
0	one	5
1	one	5
2	one	2
3	two	3
4	two	3
5	two	4
6	two	4

In [34]:
```
df2.replace([1,2],[5,6])     # 1->5, 2->6
```

Out[34]:

	c1	c2
a	5	11
c	3	14
b	6	13
d	4	12

In [35]:
```
df5.replace({1:5, 'one':'ONE'})  # 使用字典指定替换关系
```

Out[35]:

	k1	k2
0	ONE	5
1	ONE	5
2	ONE	2
3	two	3
4	two	3
5	two	4
6	two	4