Ch5 Pandas

5.1 Pandas数据结构介绍：Series和Data Frame

In [493]:
```
import pandas as pd
from pandas import Series, DataFrame
import numpy as np
```

5.1.1 Series

In [494]:
```
#Series（系列）是Pandas库中的一种数据结构，是一种一维的数组型对象，包含了一个值序列，
#它是长度固定且有序的字典


# 创建一个Series对象
data = [10, 20, 30, 40, 50]
index = ['A', 'B', 'C', 'D', 'E']
series = pd.Series(data, index)
print(series)
# A    10
# B    20
# C    30
# D    40
# E    50
# dtype: int64

#最简单的序列可以仅由一个数组行成：
obj = pd.Series([4, 7, -5, 3])
obj

# 0    4
# 1    7
# 2   -5
# 3    3
# dtype: int64
```

```
A    10
B    20
C    30
D    40
E    50
dtype: int64
```

Out[494]:
```
0    4
1    7
2   -5
3    3
dtype: int64
```

In [495]:
```
#索引index在左边，值value在右边。
#默认生成的索引，是从0到N-1。
#通过values属性，和index属性，分别获得Series对象的值和索引。

obj.index                                    #obj.index与range(4)类似
#RangeIndex(start=0, stop=4, step=1)
obj.values
#array([ 4,  7, -5,  3], dtype=int64)
```

Out[495]:
```
array([ 4,  7, -5,  3], dtype=int64)
```

In [496]:
```
#创建一个索引序列，用标签标识每个数据点：

obj2 = pd.Series([4, 7, -5, 3], index = ['d', 'b','a','c'])
obj2
# d    4
# b    7
# a   -5
# c    3
# dtype: int64
obj2.index
#Index(['d', 'b', 'a', 'c'], dtype='object')

#与Numpy数组相比，可以从数据中选择数据时使用标签来进行索引：
obj2['a']
#-5

obj2['d']
#4

#修改元素：
obj2['d'] = 6
obj2[['c','a','d']]
# c    3
# a   -5
# d    6
# dtype: int64
obj2
# d    6
# b    7
# a   -5
# c    3
# dtype: int64
```

Out[496]:
```
d    6
b    7
a   -5
c    3
dtype: int64
```

In [497]:
```
#使用numpy的函数，使用布尔值数组进行过滤，与标量相乘，或是应用数学函数，这些操作将保存

obj2[obj2 > 0]          #只要series中的正值

obj2 * 2
# d    12
# b    14
# a   -10
# c     6
# dtype: int64

np.exp(obj2)           #np.exp计算指数函数值：e(2.71828)的x次幂
# d     403.428793
# b    1096.633158
# a       0.006738
# c      20.085537
# dtype: float64
```

Out[497]:
```
d     403.428793
b    1096.633158
a       0.006738
c      20.085537
dtype: float64
```

In [498]:
```
#另一个角度理解Series， 它是长度固定且有序的字典。因为它将索引值和数据值按位置配对。

'b' in obj2
#True
'2' in obj2
#False

#如果已有数据包含在字典中，可以用字典生成一个Series：

#list和dictionary转Series
# 1)list转series系列：series = pd.Series(data, index)
# 2)字典转series系列：series = pd.Series(dictionary)

sdata = {'Ohio':35000, 'Texas':71000, 'Oregon':16000, 'Utah':5000}
obj3 = pd.Series(sdata)
obj3
# Ohio      35000
# Texas     71000
# Oregon    16000
# Utah       5000
# dtype: int64
```

Out[498]:
```
Ohio      35000
Texas     71000
Oregon    16000
Utah       5000
dtype: int64
```

In [499]:
```
# 把字典传递给Series构造函数，产生的Series的索引，将是排序号的字典键。
# 可以将字典键按照所想顺序传递给构造函数，从而使生成的Series的索引顺序符合预期:

states = ['California', 'Ohio', 'Oregon', 'Texas']                #key重新排序
obj4 = pd.Series(sdata, index = states)
obj4                                                              #之前没有赋值的
                                                                  #之前赋值过的'

# California        NaN
# Ohio          35000.0
# Oregon        16000.0
# Texas         71000.0
# dtype: float64
```

Out[499]:
```
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

In [500]:
```
# pandas中使用isnull和notnull函数 检查缺失数据:
# pd.isnull(series)
# pd.notnull(series)
# series.isnull()

pd.isnull(obj4)
# California     True
# Ohio          False
# Oregon        False
# Texas         False
# dtype: bool

pd.notnull(obj4)
# California    False
# Ohio           True
# Oregon         True
# Texas          True
# dtype: bool

#isnull和notnull也是Series的实例方法:
obj4.isnull()
# California     True
# Ohio          False
# Oregon        False
# Texas         False
# dtype: bool
```

Out[500]:
```
California     True
Ohio          False
Oregon        False
Texas         False
dtype: bool
```

In [501]:
```
#数学操作中自动对齐索引是Series的一个非常有用的特性
#即虽然key不是完全相同在同一行，仍然可以自动匹配key后value值进行操作
#与join相似

obj3
# Ohio      35000
# Texas     71000
# Oregon    16000
# Utah       5000
# dtype: int64

obj4
# California        NaN
# Ohio          35000.0
# Oregon        16000.0
# Texas         71000.0
# dtype: float64

obj3 + obj4
# California         NaN
# Ohio           70000.0
# Oregon         32000.0
# Texas         142000.0
# Utah               NaN
# dtype: float64
```

Out[501]:
```
California         NaN
Ohio           70000.0
Oregon         32000.0
Texas         142000.0
Utah               NaN
dtype: float64
```

In [502]:
```
#Series对象自身和索引，都有name属性，此特性与pandas其他功能集成在一起：
#给series的index赋名

obj4.name = 'population'
obj4.index.name = 'state'
obj4
# state
# California        NaN
# Ohio          35000.0
# Oregon        16000.0
# Texas         71000.0
# Name: population, dtype: float64
```

Out[502]:
```
state
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
Name: population, dtype: float64
```

In [503]:
```
#Series的索引可以通过按位置赋值的方式进行改变：

obj
# 0    4
# 1    7
# 2   -5
# 3    3
# dtype: int64

#给series命名index
obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']

obj
# Bob      4
# Steve    7
# Jeff    -5
# Ryan     3
# dtype: int64
```

Out[503]:
```
Bob      4
Steve    7
Jeff    -5
Ryan     3
dtype: int64
```

5.1.2 DataFrame

In [504]:
```
#DataFrame表示的矩阵(matrix)的数据表，包含已排序的列集合，每一列可以是不同值类型（数
#被视为一个共享相同索引的series的字典，既有行索引，也有列索引。
#在dataframe中，数据被存储为一个以上的二维块，而不是列表、字典或其他一维数组的集合。

data = {'state':['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
        'year':[2000, 2001, 2002, 2001, 2002, 2003],
        'pop':[1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
frame = pd.DataFrame(data)
frame
# state year    pop
# 0 Ohio    2000    1.5
# 1 Ohio    2001    1.7
# 2 Ohio    2002    3.6
# 3 Nevada  2001    2.4
# 4 Nevada  2002    2.9
# 5 Nevada  2003    3.2

#frame共享index 0,1,2; 但每列是一个series, 而且是不同值的。
```

Out[504]:

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

In [505]:
```
#如果传的列不包含在字典中，将会在结果中出现缺失值。

frame2 = pd.DataFrame(data, columns = ['year', 'state', 'pop'])
frame2
#   year    state   pop
# 0 2000    Ohio    1.5
# 1 2001    Ohio    1.7
# 2 2002    Ohio    3.6
# 3 2001    Nevada  2.4
# 4 2002    Nevada  2.9
# 5 2003    Nevada  3.2
frame2.columns
# Index(['year', 'state', 'pop'], dtype='object')

#新传一列debt
frame2 = pd.DataFrame(data, columns = ['year','state','pop','debt'],
                            index = ['one', 'two', 'three', 'four','five','six
frame2
# year  state   pop debt
# one   2000    Ohio    1.5 NaN
# two   2001    Ohio    1.7 NaN
# three 2002    Ohio    3.6 NaN
# four  2001    Nevada  2.4 NaN
# five  2002    Nevada  2.9 NaN
# six   2003    Nevada  3.2 Na

frame2.columns
#Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

Out[505]:
```
Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

In [506]:
```
#DataFrame中的一列，可以按字典型标记或属性，检索为Series:
#截取一列: frame['column'] = frame.column (column是字符串，要加'')
#活用tab键，补全列名
#返回series与原dataframe有相同索引，且series的name属性，也会被合理设置。

frame2['state']
# one        Ohio
# two        Ohio
# three      Ohio
# four     Nevada
# five     Nevada
# six      Nevada
# Name: state, dtype: object

frame2.year
# one      2000
# two      2001
# three    2002
# four     2001
# five     2002
# six      2003
# Name: year, dtype: int64
```

Out[506]:
```
one      2000
two      2001
three    2002
four     2001
five     2002
six      2003
Name: year, dtype: int64
```

In [507]:
```
#截取一行：行也可以通过位置，或特殊属性loc进行选取:
#frame.loc[index]

frame2.loc['three']
# year     2002
# state    Ohio
# pop       3.6
# debt      NaN
# Name: three, dtype: object
```

Out[507]:
```
year     2002
state    Ohio
pop       3.6
debt      NaN
Name: three, dtype: object
```

In [508]:
```
#可修改列：赋值给空值的debt列

frame2['debt'] = 16.5
frame2
#   year    state   pop debt
# one   2000    Ohio    1.5 16.5
# two   2001    Ohio    1.7 16.5
# three 2002    Ohio    3.6 16.5
# four  2001    Nevada  2.4 16.5
# five  2002    Nevada  2.9 16.5
# six   2003    Nevada  3.2 16.5
```

Out[508]:

	year	state	pop	debt
one	2000	Ohio	1.5	16.5
two	2001	Ohio	1.7	16.5
three	2002	Ohio	3.6	16.5
four	2001	Nevada	2.4	16.5
five	2002	Nevada	2.9	16.5
six	2003	Nevada	3.2	16.5

In [509]:
```
#将列表或数组赋值给一列时，值的产股必须和dataframe的长度匹配。
#将series赋值给一列时，series的索引，将会按照dataframe的索引重新排列，并在空缺地方填

val = pd.Series([-1.2, -1.5, -1.7], index = ['two', 'four', 'five'])
frame2['debt'] = val
frame2

#   year    state   pop debt
# one   2000    Ohio    1.5 NaN
# two   2001    Ohio    1.7 -1.2
# three 2002    Ohio    3.6 NaN
# four  2001    Nevada  2.4 -1.5
# five  2002    Nevada  2.9 -1.7
# six   2003    Nevada  3.2 NaN
```

Out[509]:

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	-1.2
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	-1.5
five	2002	Nevada	2.9	-1.7
six	2003	Nevada	3.2	NaN

In [510]:
```
#如果赋值不存在，会生成一个新的列。
#del可以像在字典中那样，对dataframe删除列。

#del例子中，首先增加一列，这一列是布尔值，判断条件式state列是否为'ohio':
frame2['estern'] = frame2.state == 'Ohio'
frame2
#   year    state   pop debt    estern
# one   2000    Ohio    1.5 NaN True
# two   2001    Ohio    1.7 -1.2    True
# three 2002    Ohio    3.6 NaN True
# four  2001    Nevada  2.4 -1.5    False
# five  2002    Nevada  2.9 -1.7    False
# six   2003    Nevada  3.2 NaN False
```

Out[510]:

	year	state	pop	debt	estern
one	2000	Ohio	1.5	NaN	True
two	2001	Ohio	1.7	-1.2	True
three	2002	Ohio	3.6	NaN	True
four	2001	Nevada	2.4	-1.5	False
five	2002	Nevada	2.9	-1.7	False
six	2003	Nevada	3.2	NaN	False

In [511]:
```
#del关键字可以像在字典中，对dataframe删除列。

#先增加一列，判断条件是state列是否为'ohio'

frame2['eastern'] = frame2.state == 'Ohio'
frame2
#   year    state   pop debt    estern
# one   2000    Ohio    1.5 NaN True
# two   2001    Ohio    1.7 -1.2    True
# three 2002    Ohio    3.6 NaN True
# four  2001    Nevada  2.4 -1.5    False
# five  2002    Nevada  2.9 -1.7    False
# six   2003    Nevada  3.2 NaN False

#del可以用于移除之前新建的列:
del frame2['eastern']
frame2.columns
#Index(['year', 'state', 'pop', 'debt', 'estern'], dtype='object')          #
```

Out[511]:
```
Index(['year', 'state', 'pop', 'debt', 'estern'], dtype='object')
```

In [512]:
```
#包含字典的嵌套字典

pop = {'Nevada':{2001:2.4, 2002:2.9},
       'Ohio':{2000:1.5, 2001:1.7, 2002:3.6}}
frame3 = pd.DataFrame(pop)
frame3                                                    #在index空余的行，赋值为NaN

#   Nevada  Ohio
# 2001  2.4 1.7
# 2002  2.9 3.6
# 2000  NaN 1.5

#frame.T：横纵轴转置操作
frame3.T
#   2001    2002    2000
# Nevada    2.4 2.9 NaN
# Ohio  1.7 3.6 1.5

#内部字典的键被联合、排序后形成了结果的索引。
#如果已经显式指明索引的话，内部字典的键将不会被排序。


pd.DataFrame(pop, index = [2001, 2002, 2003])        #03年没有输入值，所以为空；0
#   Nevada  Ohio
# 2001  2.4 1.7
# 2002  2.9 3.6
# 2003  NaN NaN
```

Out[512]:

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2003	NaN	NaN

In [513]:
```
#包含Series的字典，也可以用于构造DataFrame：

frame3
# Nevada    Ohio
# 2001  2.4 1.7
# 2002  2.9 3.6
# 2000  NaN 1.5

pdata = {'Ohio': frame3['Ohio'][:-1],                    #[:-1]是指从头到倒
         'Nevada': frame3['Nevada'][:2]}                 #[:2]是指[0,1]个
pdata
# {'Ohio': 2001    1.7
#  2002    3.6
#  Name: Ohio, dtype: float64,
#  'Nevada': 2001    2.4
#  2002    2.9
#  Name: Nevada, dtype: float64}
pd.DataFrame(pdata)
#   Ohio    Nevada
# 2001  1.7 2.4
# 2002  3.6 2.9
```

Out[513]:

	Ohio	Nevada
2001	1.7	2.4
2002	3.6	2.9

In [514]:
```
#如果dataframe的索引和列拥有name属性，则name属性也会被显示：

frame3.index.name = 'year'; frame3.columns.name = 'state'
frame3
# state Nevada  Ohio
# year
# 2001  2.4 1.7
# 2002  2.9 3.6
# 2000  NaN 1.5
```

Out[514]:

state	Nevada	Ohio
year		
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

In [515]:
```
#和Series相似，dataframe的values属性，会包含在dataframe中的数据，以二维ndarray的形

frame3.values
# array([[2.4, 1.7],
#        [2.9, 3.6],
#        [nan, 1.5]])

#如果dataframe的列是不同的dtypes，则values会自动选择适合所有列的类型：
frame2.values
# array([[2000, 'Ohio', 1.5, nan, True],
#        [2001, 'Ohio', 1.7, -1.2, True],
#        [2002, 'Ohio', 3.6, nan, True],
#        [2001, 'Nevada', 2.4, -1.5, False],
#        [2002, 'Nevada', 2.9, -1.7, False],
#        [2003, 'Nevada', 3.2, nan, False]], dtype=object)
```

Out[515]:
```
array([[2000, 'Ohio', 1.5, nan, True],
       [2001, 'Ohio', 1.7, -1.2, True],
       [2002, 'Ohio', 3.6, nan, True],
       [2001, 'Nevada', 2.4, -1.5, False],
       [2002, 'Nevada', 2.9, -1.7, False],
       [2003, 'Nevada', 3.2, nan, False]], dtype=object)
```

5.1.3索引对象

In [516]:
```
#pandas的索引对象，用于存储轴标签和其他元数据的。
#在构造Series或DataFrame时，所使用的任意数组或标签序列，都可以在内部转换为索引对象。

obj = pd.Series(range(3), index = ['a','b','c'])
index = obj.index
index
#Index(['a', 'b', 'c'], dtype='object')
index[1:]
#Index(['b', 'c'], dtype='object'
```

Out[516]:
```
Index(['b', 'c'], dtype='object')
```

In [517]:
```
#索引对象不可变，用户无法修改索引对象：
#index[1] = 'd'                                    #Index does not support mutable

#不变性是的在多种数据结构中，分享索引对象更安全：
labels = pd.Index(np.arange(3))
labels
#Int64Index([0, 1, 2], dtype='int64')

obj2 = pd.Series([1.5, -2.5, 0], index = labels)
obj2
# 0    1.5
# 1   -2.5
# 2    0.0
# dtype: float64

obj2.index is labels
#True
```

Out[517]:
```
True
```

In [518]:
```
#索引对象也想一个固定大小的集合：
frame3

# state Nevada  Ohio
# year
# 2001  2.4 1.7
# 2002  2.9 3.6
# 2000  NaN 1.5

frame3.columns
#Index(['Nevada', 'Ohio'], dtype='object', name='state')

'Ohio' in frame3.columns
#True
2003 in frame3.index
#False
```

Out[518]:
```
False
```

In [519]:
```
#Pandas索引对象可以包含重复标签：

dup_labels = pd.Index(['foo','foo','bar','bar'])
dup_labels
#Index(['foo', 'foo', 'bar', 'bar'], dtype='object')
```

Out[519]:
```
Index(['foo', 'foo', 'bar', 'bar'], dtype='object')
```

In [520]:
```
#根据重复标签进行筛选，会选取所有重复标签对应的数据：

#append：将额外的索引对象粘贴到原索引后，产生一个新的索引
#difference：两个索引的差集
#intersection：索引交集
#union：索引并集
#isin：计算每一个值，是否在传值容器中的布尔数组
#delete：将位置i删除，并产生新的索引
#drop：根据传参删除指定索引值，并产生新的索引
#insert：在位置i插入元素，并产生新的索引
#is_monotonic：如果索引递增，返回True
#is_unique：如果索引唯一，返回True
#unique：计算索引的唯一值序列
```

5.2 基本功能

5.2.1 重建索引

In [521]:
```
#reindex是pandas对象的重要方法，该方法用于创建一个符合新索引的新对象。
obj = pd.Series([4.5, 7.2, -5.3, 3.6], index = ['d', 'b', 'a', 'c'])
obj
# d    4.5
# b    7.2
# a   -5.3
# c    3.6
# dtype: float64

#Series调用reindex方法时，会将数据按照新的索引进行排列，如果某个索引值之前不存在，则
obj2 = obj.reindex(['a','b','c','d','e'])
obj2
# a   -5.3
# b    7.2
# c    3.6
# d    4.5
# e    NaN
# dtype: float64

#对于顺序数据，比如时间序列，在重建索引时，可能会需要进行插值或填值。
obj3 = pd.Series(['blue', 'purple', 'yellow'], index = [0, 2, 4])
obj3
# 0      blue
# 2    purple
# 4    yellow
# dtype: object

obj3.reindex(range(6), method = 'ffill')
# 0      blue
# 1      blue
# 2    purple
# 3    purple
# 4    yellow
# 5    yellow
# dtype: object

#在data frame中，reindex可以改变行索引，列索引，也可以同时改变两者。
#当进传入一个序列时，结果中的行，会重建索引：

frame = pd.DataFrame(np.arange(9).reshape((3,3)),
                     index = ['a', 'b', 'd'],
                     columns = ['Ohio', 'Texas', 'California'])
frame
# Ohio  Texas   California
# a 0   1   2
# b 3   4   5
# d 6   7   8

frame2
# year  state   pop debt    estern
# one   2000    Ohio    1.5 NaN True
# two   2001    Ohio    1.7 -1.2    True
# three 2002    Ohio    3.6 NaN True
# four  2001    Nevada  2.4 -1.5    False
# five  2002    Nevada  2.9 -1.7    False
# six   2003    Nevada  3.2 NaN False
```

```
#列可以使用columns关键字重建索引:
states = ['Texas', 'Utah', 'California']
frame.reindex(columns=states)

# Texas Utah    California
# a 1   NaN 2
# b 4   NaN 5
# d 7   NaN 8

#loc进行更为间接的标签索引
#frame.loc[['a','b','c','d'], states]
```

Out[521]:

	Texas	Utah	California
a	1	NaN	2
b	4	NaN	5
d	7	NaN	8

5.2.2 轴向上删除条目

In [522]:
```
#如果已经有索引数组或不含条目的列表，在轴向上，删除一个或更多的条目就非常容易，但这样
#drop方法会返回一个含有指示值，或轴向上删除值的新对象:
obj = pd.Series(np.arange(5.), index = ['a', 'b', 'c', 'd', 'e'])
obj
# a    0.0
# b    1.0
# c    2.0
# d    3.0
# e    4.0
# dtype: float64
new_obj = obj.drop('c')
new_obj
# a    0.0
# b    1.0
# d    3.0
# e    4.0
# dtype: float64
obj.drop(['d', 'c'])
# a    0.0
# b    1.0
# e    4.0
# dtype: float64
```

Out[522]:
```
a    0.0
b    1.0
e    4.0
dtype: float64
```

In [523]:
```
#在dataframe中，索引值可以从轴向上删除。
data = pd.DataFrame(np.arange(16).reshape((4,4)),
                    index = ['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns = ['one', 'two', 'three', 'four'])
data
#   one two three   four
# Ohio  0   1   2   3
# Colorado  4   5   6   7
# Utah  8   9   10  11
# New York  12  13  14  15
```

Out[523]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

In [524]:
```
#在调用drop时，使用标签序列，会根据行标签，删除值（轴0:可不写 or 1）：
#data.drop(['name', axis = 0 or 1 or 'columns'])

data.drop(['Colorado','Ohio'])
#data.drop(['Colorado','Ohio'], axis = 0)

#   one two three   four
# Utah  8   9   10  11
# New York  12  13  14  15

data.drop('two', axis = 1)
# #     one three   four
# # Ohio    0   2   3
# # Colorado    4   6   7
# # Utah    8   10  11
# # New York    12  14  15

data.drop(['two','four'], axis = 'columns')
#   one three
# Ohio  0   2
# Colorado  4   6
# Utah  8   10
# New York  12  14

#drop会修改Series或data frame的尺寸或形状，这些方法直接操作原对象，而不返回新对象：
#inplace 会清除被删除的数据。
obj = pd.Series([4.5, 7.2, -5.3, 3.6], index = ['d', 'b', 'a', 'c'])
obj

obj.drop('c', inplace = True)
obj
# a    0.0
# b    1.0
# d    3.0
# e    4.0
# dtype: float64
```

Out[524]:
```
d    4.5
b    7.2
a   -5.3
dtype: float64
```

5.2.3 索引，选择，过滤

In [525]:

```
#Series的索引与Numpy数组索引的功能类似，只不过Series的索引值可以不仅仅是整数。
obj = pd.Series(np.arange(4.), index = ['a', 'b', 'c', 'd'])
obj
# a    0.0
# b    1.0
# c    2.0
# d    3.0
# dtype: float64

obj['b']
#1.0

obj[1]
#1.0

obj[2:4]                    #[2:4]是2,3个元素
# c    2.0
# d    3.0
# dtype: float64

obj[['b','a','d']]
# b    1.0
# a    0.0
# d    3.0
# dtype: float64

obj[[1,3]]                  #obj[[1,3]]是索引位置为1和3的元素。
# b    1.0
# d    3.0
# dtype: float64

obj[obj < 2]                #将obj value 小于2的挑出来。
# a    0.0
# b    1.0
# dtype: float64

#普通python切片不包含尾部，Series切片与之不同:
obj['b':'c']

# b    1.0
# c    2.0
# dtype: float64

#使用这些方法设值时，会修改series相应的部分。
#obj['key1':'key2'] = a

obj['b':'c'] = 5
obj
# a    0.0
# b    5.0
# c    5.0
# d    3.0
```

```
# dtype: float64
```

Out[525]:
```
a    0.0
b    5.0
c    5.0
d    3.0
dtype: float64
```

In [526]:
```
#使用单个值或序列，可以从dataframe中索引出一个或多个列：

data = pd.DataFrame(np.arange(16).reshape((4,4)),
                    index = ['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns = ['one', 'two', 'three', 'four'])
data
# one    two three    four
# Ohio   0    1    2    3
# Colorado   4    5    6    7
# Utah   8    9    10   11
# New York   12   13   14   15

data['two']                              #data['column_name'] 切列
# Ohio          1
# Colorado      5
# Utah          9
# New York     13
# Name: two, dtype: int32

data[['three','one']]                    #data['column_name1', 'column_name2'] 切
# three one
# Ohio   2    0
# Colorado   6    4
# Utah   10   8
# New York   14   12

#索引方式也有特殊案例。可以根据一个布尔值数组切片或选择数据：
data[:2]                                 #data[:n] 切前n-1行
#    one two three    four
# Ohio   0    1    2    3
# Colorado   4    5    6    7

data[data['three'] > 5]                  #data[data['column_name'] > a] 切某列值大
#    one two three    four
# Colorado   4    5    6    7
# Utah   8    9    10   11
# New York   12   13   14   15

data < 5                                 #data < 5, 结果为True False
#    one two three    four
# Ohio   True     True     True     True
# Colorado   True     False    False    False
# Utah   False    False    False    False
# New York   False    False    False    False

data[data < 5] = 0                       #若data < 5则赋值为0
data
#    one two three    four
# Ohio   0    0    0    0
# Colorado   0    5    6    7
# Utah   8    9    10   11
# New York   12   13   14   15
```

Out[526]:

	one	two	three	four
Ohio	0	0	0	0
Colorado	0	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

5.2.3.1 使用loc和iloc选择数据

In [527]:
```
#5.2.3.1 使用loc和iloc选择数据

#介绍特殊的索引符号loc和iloc
#允许使用轴标签loc或整数标签iloc以numpy语法，从Data frame中选出数组的行和列的子集

data.loc['Colorado', ['two','three']]                    #data.loc['row_name',[
# two      5
# three    6
# Name: Colorado, dtype: int32

data.iloc[2, [3, 0, 1]]
# four    11
# one      8
# two      9
# Name: Utah, dtype: int32

data.iloc[2]
# one       8
# two       9
# three    10
# four     11
# Name: Utah, dtype: int32

data.iloc[[1,2],[3,0,1]]
# four  one two
# Colorado  7   0   5
# Utah  11  8   9

#除了单个标签或标签列表外，索引功能还可以用于切片：

data.loc[:'Utah','two']                                  #data.loc[:'row','column']:
# Ohio        0
# Colorado    5
# Utah        9
# Name: two, dtype: int32

data.iloc[:, :3][data.three > 5]                        #data.iloc[:,:3][data.column

# one   two three
# Colorado  0   5   6
# Utah  8   9   10
# New York  12  13  14
```

Out[527]:

	one	two	three
Colorado	0	5	6
Utah	8	9	10
New York	12	13	14

In [528]:
```
#dataframe索引选项


#df[val]：#从dataframe选择单列或列序列，特殊情况的便利，布尔数组（过滤行），切片（切
#df.loc[val]：选择行
#df.loc[:,val]：选择列或多列
#df.loc[val1, val2]：同时选择行和列的一部分
#df.iloc[where]：根据整数为止选择单行或多行
#df.iloc[where_i, where_j]：根据整数为止选择行和列
#df.at[label_i, label_j]：根据行列整数为止选择单个标量值
#df.iat[i, j]：根据行、列整数为止选择单个标量值
#reindex方法：通过标签选择行或列
#get_value, set_value：根据行和列的标签设值单个值
```

5.2.4 整数索引

In [529]:
```
ser = pd.Series(np.arange(3.))
ser
# 0    0.0
# 1    1.0
# 2    2.0
# dtype: float64

ser2 = pd.Series(np.arange(3.), index = ['a','b','c'])
ser2
# a    0.0
# b    1.0
# c    2.0
# dtype: float64

ser2[-1]
#2.0

ser[:1]
# 0    0.0
# dtype: float64

ser.loc[:1]
# 0    0.0
# 1    1.0
# dtype: float64

ser.iloc[:1]
# 0    0.0
# dtype: float64
```

Out[529]:
```
0    0.0
dtype: float64
```

5.2.5 算数和数据对齐

In [530]:
```
s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index = ['a','c','d','e'])
s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],
               index = ['a','c','e','f','g'])
s1
# a    7.3
# c   -2.5
# d    3.4
# e    1.5
# dtype: float64
s2
# a   -2.1
# c    3.6
# e   -1.5
# f    4.0
# g    3.1
# dtype: float64
s1 + s2
# a    5.2
# c    1.1
# d    NaN
# e    0.0
# f    NaN
# g    NaN
```

Out[530]:
```
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
dtype: float64
```

In [531]:
```
#在没有交叠的标签位置上，内部数据对齐会产生缺失值。缺失值会在后续算数操作上产生影响。
df1 = pd.DataFrame(np.arange(9.).reshape((3,3)), columns = list('bcd'),
                   index = ['Ohio', 'Texas', 'Colorado'])
df2 = pd.DataFrame(np.arange(12.).reshape((4,3)), columns = list('bde'),
                   index = ['Utah', 'Ohio', 'Texas', 'Oregon'])
df1
# b c   d
# Ohio  0.0 1.0 2.0
# Texas 3.0 4.0 5.0
# Colorado  6.0 7.0 8.0
df2
#   b   d   e
# Utah  0.0 1.0 2.0
# Ohio  3.0 4.0 5.0
# Texas 6.0 7.0 8.0
# Oregon    9.0 10.0    11.0
df1 + df2                                   #由于c列和e列，并不是2个data frame共有的
# b c   d   e
# Colorado  NaN NaN NaN NaN
# Ohio  3.0 NaN 6.0 NaN
# Oregon    NaN NaN NaN NaN
# Texas 9.0 NaN 12.0    NaN
# Utah  NaN NaN NaN NaN
df1 = pd.DataFrame({'A': [1, 2]})       #建立一个仅有1,2的data frame
df2 = pd.DataFrame({'B': [3, 4]})       #建立一个仅有3,4的data frame
df1
#   A
# 0 1
# 1 2
df2
#   B
# 0 3
# 1 4
df1 - df2                                   #将两个行或列完全不同的data frame对象相加,
#   A   B
# 0 NaN NaN
# 1 NaN NaN
```

Out[531]:

	A	B
0	NaN	NaN
1	NaN	NaN

5.2.5.1 使用填充值的算数方法

In [532]:
```
#在两个不同的索引化，对象之间进行算数操作时，可以使用特殊填充值；
#当轴标签在一个对象中存在，在另一个对象中不存在时，将缺失值填充为0：

df1 = pd.DataFrame(np.arange(12.).reshape((3,4)),
                   columns = list('abcd'))
df2 = pd.DataFrame(np.arange(20.).reshape((4,5)),
                   columns = list('abcde'))
df2.loc[1, 'b'] = np.nan                        #把df2，第一行，第b列设为nan

df1
#   a   b   c   d
# 0 0.0 1.0 2.0 3.0
# 1 4.0 5.0 6.0 7.0
# 2 8.0 9.0 10.0    11.0
df2
# a b   c   d   e
# 0 0.0 1.0 2.0 3.0 4.0
# 1 5.0 NaN 7.0 8.0 9.0
# 2 10.0    11.0    12.0    13.0    14.0
# 3 15.0    16.0    17.0    18.0    19.0

#将df添加到一起，会导致在一些不重叠的位置出现na值：
df1 + df2                                               #另一个data frame没有值的，相加
# a b   c   d   e
# 0 0.0 2.0 4.0 6.0 NaN
# 1 9.0 NaN 13.0    15.0    NaN
# 2 18.0    20.0    22.0    24.0    NaN
# 3 NaN NaN NaN NaN NaN

#在df1上使用Add方法，可将df2和一个fill_value作为参数传入：
df1.add(df2, fill_value = 0)
# a b   c   d   e
# 0 0.0 2.0 4.0 6.0 4.0
# 1 9.0 5.0 13.0    15.0    9.0
# 2 18.0    20.0    22.0    24.0    14.0
# 3 15.0    16.0    17.0    18.0    19.0

#两个语句效果等价：
1 / df1
#   a   b   c   d
# 0 inf 1.000000    0.500000    0.333333
# 1 0.250   0.200000    0.166667    0.142857
# 2 0.125   0.111111    0.100000    0.090909

df1.rdiv(1)
# a b   c   d
# 0 inf 1.000000    0.500000    0.333333
# 1 0.250   0.200000    0.166667    0.142857
# 2 0.125   0.111111    0.100000    0.090909

#当对Series或data frame重建索引时，也可以指定一个不同的填充值：
df1.reindex(columns = df2.columns, fill_value = 0)
df1
#   a   b   c   d   e
# 0 0.0 1.0 2.0 3.0 0
# 1 4.0 5.0 6.0 7.0 0
```

```
# 2 8.0 9.0 10.0    11.0    0
```

Out[532]:

	a	b	c	d
0	0.0	1.0	2.0	3.0
1	4.0	5.0	6.0	7.0
2	8.0	9.0	10.0	11.0

In [533]:

```
#灵活算术法:
# add, radd +
# sub, rsub -
# div, rdiv /
# floordiv, rfloordiv //
# mul, rmul *
# pow, rpow **
```

5.2.5.2 data frame和series操作

In [534]:
```
arr = np.arange(12.).reshape((3,4))
arr
# array([[ 0.,  1.,  2.,  3.],
#        [ 4.,  5.,  6.,  7.],
#        [ 8.,  9., 10., 11.]])
arr[0]                                          #arr[n]: 前n行
#array([0., 1., 2., 3.])

#当从arr - arr[0]，减法在每行进行操作，广播机制。
arr - arr[0]
# array([[0., 0., 0., 0.],
#        [4., 4., 4., 4.],
#        [8., 8., 8., 8.]])

frame = pd.DataFrame(np.arange(12.).reshape((4,3)),
                     columns = list('bde'),
                     index = ['Utah', 'Ohio', 'Texas', 'Oregon'])
series = frame.iloc[0]
series
# b    0.0
# d    1.0
# e    2.0
# Name: Utah, dtype: float64
frame
#    b   d   e
# Utah  0.0 1.0 2.0
# Ohio  3.0 4.0 5.0
# Texas 6.0 7.0 8.0
# Oregon    9.0 10.0    11.0

#dataframe和series的数学操作中，将series的索引和dataframe的列进行匹配，广播到各行：
frame - series
#    b   d   e
# Utah  0.0 0.0 0.0
# Ohio  3.0 3.0 3.0
# Texas 6.0 6.0 6.0
# Oregon    9.0 9.0 9.0

#如果索引值，不在data frame列中，也不在series索引中，则对象会重建索引并形成联合：
series2 = pd.Series(range(3), index = ['b', 'e', 'f'])
frame + series2
#    b   d   e   f
# Utah  0.0 NaN 3.0 NaN
# Ohio  3.0 NaN 6.0 NaN
# Texas 6.0 NaN 9.0 NaN
# Oregon    9.0 NaN 12.0    NaN

#如果想改为列上广播，在行上匹配，可以使用算数方法中的一种。
series3 = frame['d']
frame
#    b   d   e
# Utah  0.0 1.0 2.0
# Ohio  3.0 4.0 5.0
# Texas 6.0 7.0 8.0
# Oregon    9.0 10.0    11.0
series3
# Utah       1.0
```

```
# Ohio       4.0
# Texas      7.0
# Oregon    10.0
# Name: d, dtype: float64
frame.sub(series3, axis = 'index')
#    b    d    e
# Utah   -1.0    0.0 1.0
# Ohio   -1.0    0.0 1.0
# Texas -1.0    0.0 1.0
# Oregon    -1.0    0.0 1.0

#传递的axis值，用于匹配轴的。需要在data frame的行索引上，对行匹配(axis = 'index'或
```

Out[534]:

	b	d	e
Utah	-1.0	0.0	1.0
Ohio	-1.0	0.0	1.0
Texas	-1.0	0.0	1.0
Oregon	-1.0	0.0	1.0

5.2.6 函数应用和映射

In [535]:
```
frame = pd.DataFrame(np.random.randn(4,3), columns = list('bde'),
                     index = ['Utah', 'Ohio', 'Texas', 'Oregon'])
frame
#np.abs(frame)：frame中全为正值
np.abs(frame)

#将函数应用到一行或一列的一维数组上。apply可以实现。
f = lambda x: x.max() - x.min()
frame.apply(f)

#传递axis = 'columns'给apply函数，函数将会被每行调用一次：
frame.apply(f, axis = 'columns')
# Utah      2.321128
# Ohio      2.966694
# Texas     1.538849
# Oregon    2.282300
# dtype: float64
```

Out[535]:
```
Utah      1.741286
Ohio      1.653590
Texas     2.805567
Oregon    0.869071
dtype: float64
```

In [536]:
```
#定义了一个函数f(x)，该函数接受一个Series对象x作为参数。
#在这个例子中，f(x)的作用是计算给定Series对象x的最小值和最大值，
#并返回一个包含最小值和最大值的新的Series对象
#apply()方法会将每一列作为一个Series对象传递给函数f进行处理。

def f(x):
    return pd.Series([x.min(), x.max()], index = ['min','max'])

frame.apply(f)
# b d   e
# min   -0.894319   -1.656942   -1.071672
# max   0.921426    1.374991    1.103413

format = lambda x: '%.2f' % x                                   #lambda函数将输入
frame.applymap(format)
# b d   e
# Utah  -0.67   1.37    0.87
# Ohio  -0.89   -1.66   1.10
# Texas 0.72    0.21    0.22
# Oregon    0.92    0.46    -1.07

#使用applymap作为函数名，是因为series有map方法，可以将一个逐元素函数，应用到series上
frame['e'].map(format)
# Utah       0.87
# Ohio       1.10
# Texas      0.22
# Oregon    -1.07
# Name: e, dtype: object
```

Out[536]:
```
Utah      -1.34
Ohio       0.71
Texas     -2.27
Oregon    -0.55
Name: e, dtype: object
```

5.2.7 排序和排名

In [537]:
```
obj = pd.Series(range(4), index = ['d','a','b','c'])
obj.sort_index
# <bound method Series.sort_index of d    0
# a    1
# b    2
# c    3
# dtype: int64>

#在data frame中，可以在各个轴上，按索引排序:
frame = pd.DataFrame(np.arange(8).reshape((2,4)),
                     index = ['three', 'one'],
                     columns = ['d','a','b','c'])
frame.sort_index
# d  a  b  c
# three  0  1  2  3
# one    4  5  6  7
frame.sort_index(axis=1)
#    a   b   c   d
# three 1   2   3   0
# one   5   6   7   4

#数据index默认会升序排序，但也可以降序排序:
frame.sort_index(axis=1,ascending=False)
#    d   c   b   a
# three 0   3   2   1
# one   4   7   6   5

#根据series的值排序，使用sort_values方法:
obj = pd.Series([4, 7, -3, 2])
obj.sort_values()
# 2   -3
# 3    2
# 0    4
# 1    7
# dtype: int64

#默认情况下，所有缺失值，都会排序至series的尾部:
obj = pd.Series([4, np.nan, 7, np.nan, -3, 2])
obj.sort_values()
# 4   -3.0
# 5    2.0
# 0    4.0
# 2    7.0
# 1    NaN
# 3    NaN
# dtype: float64

#当对data frame排序时，可以使用一列或多列作为排序键。
frame = pd.DataFrame({'b':[4, 7, -3, 2], 'a':[0, 1, 0, 1]})
frame
#    b   a
# 0 4   0
# 1 7   1
# 2 -3  0
# 3 2   1
frame.sort_values(by = 'b')                                    #根据b列的
#    b   a
```

```
# 2 -3  0
# 3 2   1
# 0 4   0
# 1 7   1
frame.sort_values(by = ['a', 'b'])                                    #先根据a列
#   b   a
# 2 -3  0
# 0 4   0
# 3 2   1
# 1 7   1

#排名是指对数组从1到有效数据点总数，分配名次的操作。
#series和data frame的rank方法是实现排名的方法
#在默认情况下，rank通过将平均排名分配到每个组来打破平级关系。
obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
obj.rank
# <bound method NDFrame.rank of 0    7
# 1   -5
# 2    7
# 3    4
# 4    2
# 5    0
# 6    4
# dtype: int64>
obj.rank(method = 'first')
# 0    6.0
# 1    1.0
# 2    7.0
# 3    4.0
# 4    3.0
# 5    2.0
# 6    5.0
# dtype: float64

#上面的例子中，对条目0和2设置的名次为6和7，而不是之前的平均排名6.5，是因为在数据中标
obj.rank(ascending = False, method = 'max')
# 0    2.0
# 1    7.0
# 2    2.0
# 3    4.0
# 4    5.0
# 5    6.0
# 6    4.0
# dtype: float64

#Data Frame可以对行或列计算排名：
frame = pd.DataFrame({'b':[4.3, 7, -3, 2], 'a':[0, 1, 0, 1], 'c':[-2, 5, 8, -2
frame
#   b   a   c
# 0 4.3 0   -2.0
# 1 7.0 1   5.0
# 2 -3.0    0   8.0
# 3 2.0 1   -2.5
frame.rank(axis = 'columns')                                  #给每行，自己进行排序
#   b   a   c
# 0 3.0 2.0 1.0
# 1 3.0 1.0 2.0
```

```
# 2 1.0 2.0 3.0
# 3 3.0 2.0 1.0
```

Out[537]:

	b	a	c
0	3.0	2.0	1.0
1	3.0	1.0	2.0
2	1.0	2.0	3.0
3	3.0	2.0	1.0

In [538]:
```
#排名中的平级关系

#average：在每个组中分配平均排名
#min：整个组使用最小排名
#max：整个组使用最大排名
#first：按照值在数据中出现的次序分配排名
#dense：类似于method='min'，但组间排名总是增加1，而不是一个组中的相等元素的数量
```

5.2.8 含有重复标签的轴索引

In [539]:
```
obj = pd.Series(range(5), index = ['a','a','b','b','c'])
obj
# a    0
# a    1
# b    2
# b    3
# c    4
# dtype: int64

#obj.index.is_unique：标签是否唯一
obj.index.is_unique
#False
```

Out[539]: False

In [540]:
```
#根据一个标签索引多个条目会返回一个序列，而单个条目会返回标量值：

obj['a']
# a    0
# a    1
# dtype: int64
obj['c']
#4

#相同逻辑拓展到dataframe中索引：
df = pd.DataFrame(np.random.randn(4,3), index = ['a', 'a', 'b', 'b'])
df
# 0 1   2
# a 0.838824    -1.144286   0.991359
# a -0.969174   -1.567550   -2.222181
# b 1.322482    0.060120    -0.414145
# b -0.705280   0.509646    1.098486
df.loc['b']
# 0 1   2
# b 0.564160    -0.865147   1.249547
# b 0.768409    1.237698    0.891844
```

Out[540]:

	0	1	2
b	0.134276	-0.048467	-0.781009
b	-1.921011	0.546589	-0.762945

5.3 描述性统计的概述与计算

In [541]:
```
#pandas从data frame中，抽取一个series或一系列值的单个值。比numpy相比，内建了处理缺失

df = pd.DataFrame([[1.4, np.nan], [7.1, -4.5],
                   [np.nan, np.nan], [0.75, -1.3]],
                    index = ['a', 'b', 'c', 'd'],
                    columns = ['one', 'two'])
df
#   one two
# a 1.40    NaN
# b 7.10    -4.5
# c NaN NaN
# d 0.75    -1.3

#调用data frame的sum方法，返回一个包含列上加和的series:
df.sum()
# one    9.25
# two   -5.80
# dtype: float64

#传入axis = 'columns' 或 axis = 1，则会将一行上各个列的值相加:
df.sum(axis = 'columns')
# a    1.40
# b    2.60
# c    0.00
# d   -0.55
# dtype: float64

#除非整个切片上都是na，否则na值是被自动排除的。
#可以通过禁用skipna实现不排除na值。
#df.mean是求该行/列的平均值。；skipna = False：只要有na值就skip该行/列。
#axis = 列，则运算的是每行的平均值
df.mean(axis = 'columns', skipna = False)
# a      NaN
# b    1.300
# c      NaN
# d   -0.275
# dtype: float64

#归约方法可选参数:
# axis：归约轴，0
# skipna：排除缺失值，默认为True
# level：如果轴是多层索引的，该参数可以缩减分组层级

#idxmin(), idxmax()，返回的是间接统计信息，比如最小值或最大值的索引值:
df.idxmax()
# one    b
# two    d
# dtype: object

#df.cumsum()：每列，上面所有行，加和综合
df.cumsum()
# one   two
# a 1.40    NaN
# b 8.50    -4.5
# c NaN NaN
# d 9.25    -5.8
```

```
#df.describe(): 一次性产生多个汇总统计:
df.describe()
# one   two
# count 3.000000    2.000000
# mean  3.083333    -2.900000
# std   3.493685    2.262742
# min   0.750000    -4.500000
# 25%   1.075000    -3.700000
# 50%   1.400000    -2.900000
# 75%   4.250000    -2.100000
# max   7.100000    -1.300000

#对于非数值型数据，describe产生另一种汇总统计:

obj = pd.Series(['a','a','b','c'] * 4)
obj
# 0     a
# 1     a
# 2     b
# 3     c
# 4     a
# 5     a
# 6     b
# 7     c
# 8     a
# 9     a
# 10    b
# 11    c
# 12    a
# 13    a
# 14    b
# 15    c
# dtype: object

obj.describe()
# count     16
# unique     3
# top        a
# freq       8
# dtype: object
```

Out[541]:
```
count     16
unique     3
top        a
freq       8
dtype: object
```

In [542]:
```
#描述性统计和汇总统计：

# count：非NA值的个数
# describe：计算series或data frame各列的汇总统计集合
# min, max：计算最小/大值的索引位置
# argmin, argmax：计算最小/大值的索引标签
# idxmin, idxmax：计算最小/大值，所在的索引标签
# quantile：计算样本的从0到1的分位数
# sum：加和
# mean：均值
# median：中位数（50%分位数）
# mad：平均值的平均绝对偏差
# prod：所有值的积
# var：值的样本方差
# std：值的样本标准差
# skew：样本偏度（第三时刻）值
# kurt：样本偏度（第四时刻）值
# cumsum：累计值
# cummin, cummax：累计值的最小值或最大值
# cumprod：值的累计积
# diff：计算第一个算数差值（对时间序列有用）
# pct_change：计算百分比
```

5.3.1 相关性和协方差

In [543]:
```
#使用附加库获取包含股价和交易量的data frame：
#conda install pandas-datareader
```

In [544]:
```
# import pandas_datareader.data as web

# all_data = {ticker: web.get_data_yahoo(ticker)
#             for ticker in ['AAPL','IBM','MSFT','GOOG']}
# price = pd.DataFrame({ticker: data['Adj Close']
#                       for ticker, data in all_data.items()})
# volume = pd.DataFrame({ticker: data['Volume']
#                        for ticker, data in all_data.items()})
```

In [545]:
```
# returns = price.pct_change()
# returns.tail()

#series的corr方法计算的事2个series重叠的，非NA的，按索引对齐的值的相关性。
# returns['MSFT'].corr(returns['IBM'])
# returns['MSFT'].cov(returns['IBM'])

#可以使用根伟间接的语法获取数据：
# returns.MSFT.corr(returns.IBM)

#data frame的corr和cov，会分别以data frame的形式，返回相关性和协方差矩阵：
# returns.corr()
# returns.cov()

#使用data frame的corrwith方法，可以计算data frame中的行或列，与另一个序列或data fr
#传入一个series时，会返回一个含有为每列计算相关性值的series：
# returns.corrwith(returns.IBM)
# returns.corrwith(volume)
```

5.3.2 唯一值、计数和成员属性

In [546]:
```
obj = pd.Series(['c','a','d','a','a','b','b','c','c'])

#obj.unique(): 有几个不同的值
uniques = obj.unique()
uniques
#array(['c', 'a', 'd', 'b'], dtype=object)

#obj.value_counts: 算每个index有多少个:
obj.value_counts
pd.value_counts(obj.values, sort = False)
# c    3
# a    3
# d    1
# b    2
# dtype: int64

obj
# 0    c
# 1    a
# 2    d
# 3    a
# 4    a
# 5    b
# 6    b
# 7    c
# 8    c
# dtype: object

#isin(): isin执行向量化的成员属性检查，可以将数据集以series或data frame一列的形式,
mask = obj.isin(['b','c'])
mask
# 0     True
# 1    False
# 2    False
# 3    False
# 4    False
# 5     True
# 6     True
# 7     True
# 8     True
# dtype: bool

#obj[mask]: 找出所有isin的元素
obj[mask]
# 0    c
# 5    b
# 6    b
# 7    c
# 8    c
# dtype: object

#与isin相关的index.get_indexer方法，提供一个索引数组，将可能非唯一值数组，转换为另一
to_match = pd.Series(['c','a','b','b','c','a'])
unique_vals = pd.Series(['c','b','a'])                    #key定义为c[0],b
pd.Index(unique_vals).get_indexer(to_match)               #找到每个value对
#array([0, 2, 1, 1, 0, 2], dtype=int64)
```

Out[546]:
```
array([0, 2, 1, 1, 0, 2], dtype=int64)
```

In [547]:
```
#唯一值，计数，集合成员：

# isin：计算表征series中每个值，是否包含于传入序列的布尔值数组
# match：计算数组中每个值的整数索引，形成一个唯一值数组。助于对齐和join。
# unique：计算series值中的唯一值数组，按照观察顺序返回
# value_counts：返回一个series，索引是唯一值序列，值是计数个数，按照个数降序排序
```

In [548]:
```
#data frame多个相关列的直方图：

data = pd.DataFrame({'Qu1': [1, 3, 4, 3, 4],
                     'Qu2': [2, 3, 1, 2, 3],
                     'Qu3': [1, 5, 2, 4, 4]})
data
# Qu1   Qu2 Qu3
# 0 1   2   1
# 1 3   3   5
# 2 4   1   2
# 3 3   2   4
# 4 4   3   4

#行标签是所有列中出现的不同值，数值是这些不同值，在每个列中出现的次数。
#1在Qu1/2/3中，各出现1次，2在Qu1出现0次，在Qu2出现2次，在Qu3出现3次。
result = data.apply(pd.value_counts).fillna(0)
result
#    Qu1 Qu2 Qu3
# 1 1.0 1.0 1.0
# 2 0.0 2.0 1.0
# 3 2.0 2.0 0.0
# 4 2.0 0.0 2.0
# 5 0.0 0.0 1.0
```

Out[548]:

	Qu1	Qu2	Qu3
1	1.0	1.0	1.0
2	0.0	2.0	1.0
3	2.0	2.0	0.0
4	2.0	0.0	2.0
5	0.0	0.0	1.0

In [549]:
```
#相关性（correlation）和协方差（covariance）是用于衡量两个变量之间关系的统计量，但它

# 相关性（correlation）是协方差的标准化版本，它衡量的是两个变量之间线性关系的强度和方
# 相关性为1表示两个变量完全正相关，相关性为-1表示两个变量完全负相关，相关性为0表示两个
# 相关性的取值范围使得它更易于比较不同数据集之间的关系强度，并且不受数据量纲的影响。

# 协方差（covariance）是一种度量两个变量之间线性关系强度和方向的统计量。它衡量的是两个
# 协方差的取值范围没有限制，可以为正值、负值或零。
# 如果协方差为正值，则表示两个变量呈正向关系，即一个变量增加时另一个变量也增加；
# 如果协方差为负值，则表示两个变量呈负向关系，即一个变量增加时另一个变量减小；
# 如果协方差为零，则表示两个变量之间没有线性关系。
```

In [550]:
```
#Array（数组）和List（列表）是在Python中用于存储和操作数据的两种常见数据结构，它们有

#数据类型：
#Array是NumPy库中的数据结构，可以存储同类型的元素，提供了高效的数值计算和向量化操作。
#List是Python内置的数据结构，可以存储不同类型的元素，具有更灵活的操作。

#内存占用和性能：
#Array通常比List占用更少的内存空间，并且具有更高的执行效率。
#Array使用连续的内存块存储数据，允许进行快速的数值计算和向量化操作。
#List是由指针组成的动态数组，存储的是对象的引用，对于大规模数值计算较为低效。

#操作和功能：
#Array提供了丰富的数学和科学函数，支持矩阵操作、向量化运算、广播等功能，适用于数值计算
#List提供了更多的通用操作和方法，如增加、删除、索引等，适用于常规的数据存储和处理。

#可变性：
#Array的大小和形状通常是固定的，一旦创建后不易修改。
#List是可变的，可以动态地添加、删除或修改元素。

#Array适用于数值计算和科学计算领域，提供了高效的数值操作和向量化计算能力。
#List适用于通用的数据存储和处理，具有更灵活的操作和可变性。

# 创建一个NumPy数组：arr = np.array([1, 2, 3, 4, 5])
# 创建一个Python列表：lst = [1, 2, 3, 4, 5]
```