数据分析 3 Pandas

pandas是专门为处理表格和混杂数据设计的，而NumPy更适合处理统一的数值数组数据。

```
In [1]: import pandas as pd
```

pandas的数据结构介绍

两个主要数据结构：Series和DataFrame

Series

Series是一种类似于一维数组的对象，它由一组数据（各种NumPy数据类型）以及一组与之相关的数据标签（即索引）组成

```
In [11]: obj = pd.Series([4, 7, -5, 3])

In [12]: obj
Out[12]:
0    4
1    7
2   -5
3    3
dtype: int64
```

```
In [13]: obj.values
Out[13]: array([ 4,  7, -5,  3])

In [14]: obj.index  /# like range(4)/
Out[14]: RangeIndex(start=0, stop=4, step=1)
```

创建Series带有一个可以对各个数据点进行标记的索引

```
In [15]: obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])

In [16]: obj2
Out[16]:
d    4
b    7
a   -5
c    3
dtype: int64

In [17]: obj2.index
Out[17]: Index(['d', 'b', 'a', 'c'], dtype='object')
```

通过索引的方式选取Series中的单个或一组值

```
In [18]: obj2['a']
Out[18]: -5

In [19]: obj2['d'] = 6

In [20]: obj2[['c', 'a', 'd']]
Out[20]:
c    3
a   -5
d    6
dtype: int64
```

```
In [21]: obj2[obj2 > 0]
Out[21]:
d    6
b    7
c    3
dtype: int64

In [22]: obj2 * 2
```

```
Out[22]:
d    12
b    14
a   -10
c     6
dtype: int64
```

Series看成是一个定长的有序字典

```
In [24]: 'b' in obj2
Out[24]: True

In [25]: 'e' in obj2
Out[25]: False
```

直接通过这个字典来创建Series

```
In [26]: sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}

In [27]: obj3 = pd.Series(sdata)

In [28]: obj3
Out[28]:
Ohio      35000
Oregon    16000
Texas     71000
Utah       5000
dtype: int64
```

传入排好序的字典的键以改变顺序

```
In [29]: states = ['California', 'Ohio', 'Oregon', 'Texas']

In [30]: obj4 = pd.Series(sdata, index=states)

In [31]: obj4
```

```
Out[31]:
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

NaN（即“非数字”（not a number），在pandas中，它用于表示缺失或NA值）

pandas的isnull和notnull函数可用于检测缺失数据

```
In [32]: pd.isnull(obj4)
Out[32]:
California     True
Ohio          False
Oregon        False
Texas         False
dtype: bool

In [33]: pd.notnull(obj4)
Out[33]:
California    False
Ohio           True
Oregon         True
Texas          True
dtype: bool
```

Series最重要的一个功能是，它会根据运算的索引标签自动对齐数据

```
In [35]: obj3
Out[35]:
Ohio      35000
Oregon    16000
Texas     71000
Utah       5000
dtype: int64
```

```
In [36]: obj4
Out[36]:
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64

In [37]: obj3 + obj4
Out[37]:
California         NaN
Ohio           70000.0
Oregon         32000.0
Texas         142000.0
Utah               NaN
dtype: float64
```

Series对象本身及其索引都有一个name属性

```
In [38]: obj4.name = 'population'

In [39]: obj4.index.name = 'state'

In [40]: obj4
Out[40]:
state
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
Name: population, dtype: float64
```

Series的索引可以通过赋值的方式就地修改

```
In [41]: obj
Out[41]:
0    4
```

```
1    7
2   -5
3    3
dtype: int64

In [42]: obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']

In [43]: obj
Out[43]:
Bob      4
Steve    7
Jeff    -5
Ryan     3
dtype: int64
```

DataFrame

DataFrame是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔值等），DataFrame既有行索引也有列索引。

建DataFrame, 传入一个由等长列表或NumPy数组组成的字典

```
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)
```

特别大的DataFrame，head方法会选取前五行

```
In [46]: frame.head()
Out[46]:
   pop   state  year
0  1.5    Ohio  2000
1  1.7    Ohio  2001
2  3.6    Ohio  2002
3  2.4  Nevada  2001
```

```
4  2.9  Nevada  2002
```

如果指定了列序列，则DataFrame的列就会按照指定顺序进行排列

```
In [47]: pd.DataFrame(data, columns=['year', 'state', 'pop'])
Out[47]:
   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
```

```
In [48]: frame2 = pd.DataFrame(data, columns=['year', 'state', 'pop', 'debt'],
   ....:                       index=['one', 'two', 'three', 'four',
   ....:                              'five', 'six'])

In [49]: frame2
Out[49]:
       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  NaN

In [50]: frame2.columns
Out[50]: Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

将DataFrame的列获取为一个Series

```
In [51]: frame2['state']
Out[51]:
```

```
one        Ohio
two        Ohio
three      Ohio
four     Nevada
five     Nevada
six      Nevada
Name: state, dtype: object

In [52]: frame2.year
Out[52]:
one      2000
two      2001
three    2002
four     2001
five     2002
six      2003
Name: year, dtype: int64
```

行也可以通过位置或名称的方式进行获取

```
In [53]: frame2.loc['three']
Out[53]:
year     2002
state    Ohio
pop       3.6
debt      NaN
Name: three, dtype: object
```

以给那个空的"debt"列赋上一个标量值或一组值

```
In [54]: frame2['debt'] = 16.5

In [55]: frame2
Out[55]:
       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 [56]: frame2['debt'] = np.arange(6.)

In [57]: frame2
Out[57]:
       year   state  pop  debt
one    2000    Ohio  1.5   0.0
two    2001    Ohio  1.7   1.0
three  2002    Ohio  3.6   2.0
four   2001  Nevada  2.4   3.0
five   2002  Nevada  2.9   4.0
six    2003  Nevada  3.2   5.0
```

如果赋值的是一个Series，就会精确匹配DataFrame的索引，所有的空位都将被填上缺失值

```
In [58]: val = pd.Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])

In [59]: frame2['debt'] = val

In [60]: frame2
Out[60]:
       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
```

为不存在的列赋值会创建出一个新列。关键字del用于删除列。

```
In [61]: frame2['eastern'] = frame2.state == 'Ohio'
```

```
In [62]: frame2
Out[62]:
       year   state  pop  debt  eastern
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

# 删除列
del frame2['eastern']
```

如果嵌套字典传给DataFrame，pandas就会被解释为：外层字典的键作为列，内层键则作为行索引

```
In [65]: pop = {'Nevada': {2001: 2.4, 2002: 2.9},
....:        'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}

In [66]: frame3 = pd.DataFrame(pop)

In [67]: frame3
Out[67]:
      Nevada  Ohio
2000     NaN   1.5
2001     2.4   1.7
2002     2.9   3.6
```

对DataFrame进行转置（交换行和列）

```
In [68]: frame3.T
Out[68]:
         2000  2001  2002
Nevada   NaN   2.4   2.9
Ohio     1.5   1.7   3.6
```

设置了DataFrame的index和columns的name属性，则这些信息也会被显示出来

```
In [72]: frame3.index.name = 'year'; frame3.columns.name = 'state'

In [73]: frame3
Out[73]:
state  Nevada  Ohio
year
2000      NaN   1.5
2001      2.4   1.7
2002      2.9   3.6
```

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

```
In [74]: frame3.values
Out[74]:
array([[ nan,  1.5],
       [ 2.4,  1.7],
       [ 2.9,  3.6]])
```

如果DataFrame各列的数据类型不同，则值数组的dtype就会选用能兼容所有列的数据类型：

```
In [75]: frame2.values
Out[75]:
array([[2000, 'Ohio', 1.5, nan],
       [2001, 'Ohio', 1.7, -1.2],
       [2002, 'Ohio', 3.6, nan],
       [2001, 'Nevada', 2.4, -1.5],
       [2002, 'Nevada', 2.9, -1.7],
       [2003, 'Nevada', 3.2, nan]], dtype=object)
```

索引对象

```
In [76]: obj = pd.Series(range(3), index=['a', 'b', 'c'])

In [77]: index = obj.index

In [78]: index
Out[78]: Index(['a', 'b', 'c'], dtype='object')

In [79]: index[1:]
Out[79]: Index(['b', 'c'], dtype='object')
```

Index对象是不可变的，因此用户不能对其进行修改

```
index[1] = ‘d’  # TypeError
```

不可变可以使Index对象在多个数据结构之间安全共享

```
In [80]: labels = pd.Index(np.arange(3))

In [81]: labels
Out[81]: Int64Index([0, 1, 2], dtype='int64')

In [82]: obj2 = pd.Series([1.5, -2.5, 0], index=labels)

In [83]: obj2
Out[83]:
0    1.5
1   -2.5
2    0.0
dtype: float64

In [84]: obj2.index is labels
Out[84]: True
```

基本功能

pandas对象的一个重要方法是reindex，其作用是创建一个新对象，它的数据符合新的索引。

```
In [91]: obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])

In [92]: obj
Out[92]:
d    4.5
b    7.2
a   -5.3
c    3.6
dtype: float64

In [93]: obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])

In [94]: obj2
Out[94]:
a   -5.3
b    7.2
c    3.6
d    4.5
e    NaN
dtype: float64
```

时间序列这样的有序数据，重新索引时可能需要做一些插值处理。method选项即可达到此目的，例如，使用ffill可以实现前向值填充

```
In [95]: obj3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])

In [96]: obj3
Out[96]:
0      blue
2    purple
4    yellow
dtype: object

In [97]: obj3.reindex(range(6), method='ffill')
Out[97]:
```

```
0      blue
1      blue
2    purple
3    purple
4    yellow
5    yellow
dtype: object
```

reindex可以修改（行）索引和列。只传递一个序列时，会重新索引结果的行

```
In [98]: frame = pd.DataFrame(np.arange(9).reshape((3, 3)),
   ....:                      index=['a', 'c', 'd'],
   ....:                      columns=['Ohio', 'Texas', 'California'])

In [99]: frame
Out[99]:
   Ohio  Texas  California
a     0      1           2
c     3      4           5
d     6      7           8

In [100]: frame2 = frame.reindex(['a', 'b', 'c', 'd'])

In [101]: frame2
Out[101]:
   Ohio  Texas  California
a   0.0    1.0         2.0
b   NaN    NaN         NaN
c   3.0    4.0         5.0
d   6.0    7.0         8.0
```

列可以用columns关键字重新索引

```
In [102]: states = ['Texas', 'Utah', 'California']

In [103]: frame.reindex(columns=states)
Out[103]:
```

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

丢弃指定轴上的项

drop方法返回的是一个在指定轴上删除了指定值的新对象

```
In [105]: obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])

In [106]: obj
Out[106]:
a    0.0
b    1.0
c    2.0
d    3.0
e    4.0
dtype: float64

In [107]: new_obj = obj.drop('c')

In [108]: new_obj
Out[108]:
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64

In [109]: obj.drop(['d', 'c'])
Out[109]:
a    0.0
b    1.0
e    4.0
dtype: float64
```

对于DataFrame，可以删除任意轴上的索引值

```
In [110]: data = pd.DataFrame(np.arange(16).reshape((4, 4)),
   .....:                      index=['Ohio', 'Colorado', 'Utah', 'New York'],
   .....:                      columns=['one', 'two', 'three', 'four'])

In [111]: data
Out[111]:
          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 [112]: data.drop(['Colorado', 'Ohio'])
Out[112]:
          one  two  three  four
Utah        8    9     10    11
New York   12   13     14    15
```

通过传递axis=1或axis='columns'可以删除列的值

```
In [113]: data.drop('two', axis=1)
Out[113]:
          one  three  four
Ohio        0      2     3
Colorado    4      6     7
Utah        8     10    11
New York   12     14    15

In [114]: data.drop(['two', 'four'], axis='columns')
Out[114]:
          one  three
Ohio        0      2
Colorado    4      6
```

```
Utah        8     10
New York   12     14
```

索引、选取和过滤

```
In [117]: obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])

In [118]: obj
Out[118]:
a    0.0
b    1.0
c    2.0
d    3.0
dtype: float64

In [119]: obj['b']
Out[119]: 1.0

In [120]: obj[1]
Out[120]: 1.0

In [121]: obj[2:4]
Out[121]:
c    2.0
d    3.0
dtype: float64

In [122]: obj[['b', 'a', 'd']]
Out[122]:
b    1.0
a    0.0
d    3.0
dtype: float64

In [123]: obj[[1, 3]]
Out[123]:
b    1.0
```

```
d    3.0
dtype: float64

In [124]: obj[obj < 2]
Out[124]:
a    0.0
b    1.0
dtype: float64
```

利用标签的切片运算与普通的Python切片运算不同，其末端是包含的

```
In [125]: obj['b':'c']
Out[125]:
b    1.0
c    2.0
dtype: float64
```

```
In [126]: obj['b':'c'] = 5

In [127]: obj
Out[127]:
a    0.0
b    5.0
c    5.0
d    3.0
dtype: float64
```

用一个值或序列对DataFrame进行索引其实就是获取一个或多个列

```
In [128]: data = pd.DataFrame(np.arange(16).reshape((4, 4)),
   .....:                      index=['Ohio', 'Colorado', 'Utah', 'New York'],
   .....:                      columns=['one', 'two', 'three', 'four'])

In [129]: data
Out[129]:
```

```
          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 [130]: data['two']
Out[130]:
Ohio         1
Colorado     5
Utah         9
New York    13
Name: two, dtype: int64

In [131]: data[['three', 'one']]
Out[131]:
          three  one
Ohio          2    0
Colorado      6    4
Utah         10    8
New York     14   12
```

```
In [132]: data[:2]
Out[132]:
          one  two  three  four
Ohio        0    1      2     3
Colorado    4    5      6     7

In [133]: data[data['three'] > 5]
Out[133]:
          one  two  three  four
Colorado    4    5      6     7
Utah        8    9     10    11
New York   12   13     14    15
```

```
In [134]: data < 5
Out[134]:
            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

In [135]: data[data < 5] = 0

In [136]: data
Out[136]:
          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
```

用loc和iloc进行选取

使用轴标签（loc）或整数索引（iloc），从DataFrame选择行和列的子集。

```
In [137]: data.loc['Colorado', ['two', 'three']]
Out[137]:
two      5
three    6
Name: Colorado, dtype: int64
```

用iloc和整数进行选取

```
In [138]: data.iloc[2, [3, 0, 1]]
Out[138]:
four    11
one      8
two      9
Name: Utah, dtype: int64
```

可以对不同索引的对象进行算术运算

```
In [139]: data.iloc[2]
Out[139]:
one       8
two       9
three    10
four     11
Name: Utah, dtype: int64

In [140]: data.iloc[[1, 2], [3, 0, 1]]
Out[140]:
          four  one  two
Colorado     7    0    5
Utah        11    8    9
```

```
In [141]: data.loc[:'Utah', 'two']
Out[141]:
Ohio        0
Colorado    5
Utah        9
Name: two, dtype: int64

In [142]: data.iloc[:, :3][data.three > 5]
Out[142]:
          one  two  three
Colorado    0    5      6
Utah        8    9     10
New York   12   13     14
```

算术运算和数据对齐

可以对不同索引的对象进行算术运算

```
In [150]: s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])

In [151]: s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],
```

```
   .....:                 index=['a', 'c', 'e', 'f', 'g'])

In [152]: s1
Out[152]:
a    7.3
c   -2.5
d    3.4
e    1.5
dtype: float64

In [153]: s2
Out[153]:
a   -2.1
c    3.6
e   -1.5
f    4.0
g    3.1
dtype: float64
```

```
In [154]: s1 + s2
Out[154]:
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
dtype: float64
```

对于DataFrame，对齐操作会同时发生在行和列上

```
In [155]: df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)),
columns=list('bcd'),
   .....:                     index=['Ohio', 'Texas', 'Colorado'])

In [156]: df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)),
```

```
columns=list('bde'),
   .....:                     index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [157]: df1
Out[157]:
            b    c    d
Ohio      0.0  1.0  2.0
Texas     3.0  4.0  5.0
Colorado  6.0  7.0  8.0

In [158]: df2
Out[158]:
          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
```

```
In [159]: df1 + df2
Out[159]:
            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
```

如果DataFrame对象相加，没有共用的列或行标签，结果都会是空

```
In [160]: df1 = pd.DataFrame({'A': [1, 2]})

In [161]: df2 = pd.DataFrame({'B': [3, 4]})

In [162]: df1
Out[162]:
   A
```

```
0  1
1  2

In [163]: df2
Out[163]:
   B
0  3
1  4

In [164]: df1 - df2
Out[164]:
    A   B
0 NaN NaN
1 NaN NaN
```

在算术方法中填充值

当一个对象中某个轴标签在另一个对象中找不到时填充一个特殊值（比如0）

```

In [165]: df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),
   .....:                     columns=list('abcd'))

In [166]: df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)),
   .....:                     columns=list('abcde'))

In [167]: df2.loc[1, 'b'] = np.nan

In [168]: df1
Out[168]:
     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 [169]: df2
Out[169]:
      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
```

相加时，没有重叠的位置就会产生NA值

```
In [170]: df1 + df2
Out[170]:
      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
```

```
In [171]: df1.add(df2, fill_value=0)
Out[171]:
      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
```

DataFrame和Series之间的运算

```
In [179]: frame = pd.DataFrame(np.arange(12.).reshape((4, 3)),
   .....:                       columns=list('bde'),
   .....:                       index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [180]: series = frame.iloc[0]

In [181]: frame
Out[181]:
          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

In [182]: series
Out[182]:
b    0.0
d    1.0
e    2.0
Name: Utah, dtype: float64

In [183]: frame - series
Out[183]:
          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
```

如果某个索引值在DataFrame的列或Series的索引中找不到，则参与运算的两个对象就会被重新索引以形成并集

```
In [184]: series2 = pd.Series(range(3), index=['b', 'e', 'f'])

In [185]: frame + series2
Out[185]:
          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
```

```
In [186]: series3 = frame['d']
```

```
In [187]: frame
Out[187]:
          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

In [188]: series3
Out[188]:
Utah       1.0
Ohio       4.0
Texas      7.0
Oregon    10.0
Name: d, dtype: float64

In [189]: frame.sub(series3, axis='index')
Out[189]:
          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
```

函数应用和映射

```
In [190]: frame = pd.DataFrame(np.random.randn(4, 3), columns=list('bde'),
   .....:                       index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [191]: frame
Out[191]:
               b         d         e
Utah   -0.204708  0.478943 -0.519439
Ohio   -0.555730  1.965781  1.393406
Texas   0.092908  0.281746  0.769023
Oregon  1.246435  1.007189 -1.296221
```

```
In [192]: np.abs(frame)
Out[192]:
               b         d         e
Utah    0.204708  0.478943  0.519439
Ohio    0.555730  1.965781  1.393406
Texas   0.092908  0.281746  0.769023
Oregon  1.246435  1.007189  1.296221
```

应用到每列

```
In [193]: f = lambda x: x.max() - x.min()

In [194]: frame.apply(f)
Out[194]:
b    1.802165
d    1.684034
e    2.689627
dtype: float64
```

传递axis='columns'到apply，这个函数会在每行执行

```
In [195]: frame.apply(f, axis='columns')
Out[195]:
Utah      0.998382
Ohio      2.521511
Texas     0.676115
Oregon    2.542656
dtype: float64
```

```
In [196]: def f(x):
   .....:     return pd.Series([x.min(), x.max()], index=['min', 'max'])

In [197]: frame.apply(f)
Out[197]:
```

```
            b         d         e
min -0.555730  0.281746 -1.296221
max  1.246435  1.965781  1.393406
```

得到frame中各个浮点值的格式化字符串，使用applymap即可

```
In [198]: format = lambda x: '%.2f' % x

In [199]: frame.applymap(format)
Out[199]:
            b     d      e
Utah    -0.20  0.48  -0.52
Ohio    -0.56  1.97   1.39
Texas    0.09  0.28   0.77
Oregon   1.25  1.01  -1.30
```

Series有一个用于应用元素级函数的map方法

```
In [200]: frame['e'].map(format)
Out[200]:
Utah      -0.52
Ohio       1.39
Texas      0.77
Oregon    -1.30
Name: e, dtype: object
```

排序和排名

要对行或列索引进行排序（按字典顺序），可使用sort_index方法，它将返回一个已排序的新对象

```
In [201]: obj = pd.Series(range(4), index=['d', 'a', 'b', 'c'])

In [202]: obj.sort_index()
Out[202]:
```

```
a    1
b    2
c    3
d    0
dtype: int64
```

DataFrame，可以根据任意一个轴上的索引进行排序

```
In [203]: frame = pd.DataFrame(np.arange(8).reshape((2, 4)),
   .....:                      index=['three', 'one'],
   .....:                      columns=['d', 'a', 'b', 'c'])

In [445]: frame
Out[445]:
       d  a  b  c
three  0  1  2  3
one    4  5  6  7

In [204]: frame.sort_index()
Out[204]:
       d  a  b  c
one    4  5  6  7
three  0  1  2  3

In [205]: frame.sort_index(axis=1)
Out[205]:
       a  b  c  d
three  1  2  3  0
one    5  6  7  4
```

降序排序

```
In [206]: frame.sort_index(axis=1, ascending=False)
Out[206]:
       d  c  b  a
three  0  3  2  1
one    4  7  6  5
```

```
```

按值对Series进行排序，可使用其sort_values方法

```
In [207]: obj = pd.Series([4, 7, -3, 2])

In [208]: obj.sort_values()
Out[208]:
2   -3
3    2
0    4
1    7
dtype: int64
```

排序时，任何缺失值默认都会被放到Series的末尾

```
In [209]: obj = pd.Series([4, np.nan, 7, np.nan, -3, 2])

In [210]: obj.sort_values()
Out[210]:
4   -3.0
5    2.0
0    4.0
2    7.0
1    NaN
3    NaN
dtype: float64
```

排序一个DataFrame时，根据一个或多个列中的值进行排序。将一个或多个列的名字传递给sort_values的by选项即可

```
In [211]: frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})

In [212]: frame
Out[212]:
   a  b
```

```
0  0  4
1  1  7
2  0 -3
3  1  2

In [213]: frame.sort_values(by='b')
Out[213]:
   a  b
2  0 -3
3  1  2
0  0  4
1  1  7
```

```
In [214]: frame.sort_values(by=['a', 'b'])
Out[214]:
   a  b
2  0 -3
0  0  4
3  1  2
1  1  7
```

rank为各组分配一个平均排名

```
In [215]: obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
In [216]: obj.rank()
Out[216]:
0    6.5
1    1.0
2    6.5
3    4.5
4    3.0
5    2.0
6    4.5
dtype: float64
```

根据值在原数据中出现的顺序给出排名

```
In [217]: obj.rank(method='first')
Out[217]:
0    6.0
1    1.0
2    7.0
3    4.0
4    3.0
5    2.0
6    5.0
dtype: float64
```

也可以按降序进行排名

```
In [217]: obj.rank(ascending=False, method='first')
Out[217]:
0    1.0
1    7.0
2    2.0
3    3.0
4    5.0
5    6.0
6    4.0
dtype: float64
```

DataFrame可以在行或列上计算排名

```
In [219]: frame = pd.DataFrame({'b': [4.3, 7, -3, 2], 'a': [0, 1, 0, 1],
   .....:                       'c': [-2, 5, 8, -2.5]})

In [220]: frame
Out[220]:
   a    b    c
0  0  4.3 -2.0
1  1  7.0  5.0
2  0 -3.0  8.0
```

```
3  1  2.0 -2.5

In [221]: frame.rank(axis='columns')
Out[221]:
     a    b    c
0  2.0  3.0  1.0
1  1.0  3.0  2.0
2  2.0  1.0  3.0
3  2.0  3.0  1.0
```

带有重复标签的轴索引

带有重复索引值的Series

```
In [222]: obj = pd.Series(range(5), index=['a', 'a', 'b', 'b', 'c'])

In [223]: obj
Out[223]:
a    0
a    1
b    2
b    3
c    4
dtype: int64

In [224]: obj.index.is_unique
Out[224]: False
```

如果某个索引对应多个值，则返回一个Series；而对应单个值的，则返回一个标量值

```
In [225]: obj['a']
Out[225]:
a    0
a    1
dtype: int64

In [226]: obj['c']
```

```
Out[226]: 4
```

```
In [227]: df = pd.DataFrame(np.random.randn(4, 3), index=['a', 'a', 'b', 'b'])

In [228]: df
Out[228]:
          0         1         2
a  0.274992  0.228913  1.352917
a  0.886429 -2.001637 -0.371843
b  1.669025 -0.438570 -0.539741
b  0.476985  3.248944 -1.021228

In [229]: df.loc['b']
Out[229]:
          0         1         2
b  1.669025 -0.438570 -0.539741
b  0.476985  3.248944 -1.021228
```

汇总和计算描述统计

pandas对象拥有一组常用的数学和统计方法。它们大部分都属于约简和汇总统计，用于从Series中提取单个值（如sum或mean）或从DataFrame的行或列中提取一个Series。

```
In [230]: 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'])

In [231]: df
Out[231]:
    one  two
a  1.40  NaN
b  7.10 -4.5
c   NaN  NaN
d  0.75 -1.3
```

```
```

调用DataFrame的sum方法将会返回一个含有列的和的Series

```
In [232]: df.sum()
Out[232]:
one    9.25
two   -5.80
dtype: float64
```

传入axis='columns'或axis=1将会按行进行求和运算

```
In [233]: df.sum(axis=1)
Out[233]:
a    1.40
b    2.60
c     NaN
d   -0.55
```

NA值会自动被排除，除非整个切片（这里指的是行或列）都是NA。通过skipna选项可以禁用该功能：

```
In [234]: df.mean(axis='columns', skipna=False)
Out[234]:
a      NaN
b    1.300
c      NaN
d   -0.275
dtype: float64
```

唯一值、值计数以及成员资格

unique，它可以得到Series中的唯一值数组

```
In [251]: obj = pd.Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])
```

```
In [252]: uniques = obj.unique()

In [253]: uniques
Out[253]: array(['c', 'a', 'd', 'b'], dtype=object)
```

value_counts用于计算一个Series中各值出现的频率

```
In [254]: obj.value_counts()
Out[254]:
c    3
a    3
b    2
d    1
dtype: int64
```

isin用于判断矢量化集合的成员资格

```
In [256]: obj
Out[256]:
0    c
1    a
2    d
3    a
4    a
5    b
6    b
7    c
8    c
dtype: object

In [257]: mask = obj.isin(['b', 'c'])

In [258]: mask
Out[258]:
0     True
1    False
2    False
```

```
3    False
4    False
5     True
6     True
7     True
8     True
dtype: bool

In [259]: obj[mask]
Out[259]:
0    c
5    b
6    b
7    c
8    c
dtype: object
```

结果中的行标签是所有列的唯一值。后面的频率值是每个列中这些值的相应计数

```

In [263]: data = pd.DataFrame({'Qu1': [1, 3, 4, 3, 4],
   .....:                      'Qu2': [2, 3, 1, 2, 3],
   .....:                      'Qu3': [1, 5, 2, 4, 4]})

In [264]: data
Out[264]:
   Qu1  Qu2  Qu3
0    1    2    1
1    3    3    5
2    4    1    2
3    3    2    4
4    4    3    4

In [265]: result = data.apply(pd.value_counts).fillna(0)

In [266]: result
Out[266]:
   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
```