pandas data join, combine, reshape

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

层次化索引

Series 的层次化索引

In [2]:
```
data = pd.Series(np.random.randn(6),
                 index=[['a', 'a', 'b', 'b', 'c', 'c'],
                        [1, 2, 1, 3, 2, 3]])
data
```

Out[2]:
```
a  1    0.971509
   2    1.042921
b  1    0.316566
   3    1.730938
c  2   -0.457496
   3    1.183176
dtype: float64
```

In [3]:
```
data.index
```

Out[3]:
```
MultiIndex([('a', 1),
            ('a', 2),
            ('b', 1),
            ('b', 3),
            ('c', 2),
            ('c', 3)],
           )
```

切片

In [4]:
```
data['b']
```

Out[4]:
```
1    0.316566
3    1.730938
dtype: float64
```

In [5]:
```
data['b':'c']
```

Out[5]:
```
b  1    0.316566
   3    1.730938
c  2   -0.457496
   3    1.183176
dtype: float64
```

In [6]:
```
data.loc[['b', 'a']]
```

Out[6]:
```
b  1    0.316566
   3    1.730938
a  1    0.971509
   2    1.042921
dtype: float64
```

在“内层”中进行切片

In [7]:
```
data.loc[:, 2]
```

Out[7]:
```
a    1.042921
c   -0.457496
dtype: float64
```

series.unstack( ), frame.stack( ) : 数据重塑, Series → DataFrame, DataFrame →

Series

In [8]:

```
frame = data.unstack()
frame
```

Out[8]:

	1	2	3
a	0.971509	1.042921	NaN
b	0.316566	NaN	1.730938
c	NaN	-0.457496	1.183176

In [9]:

```
frame.stack()
```

Out[9]:

```
a  1    0.971509
   2    1.042921
b  1    0.316566
   3    1.730938
c  2   -0.457496
   3    1.183176
dtype: float64
```

DataFrame 的层次化索引

In [10]:

```
frame = pd.DataFrame(np.arange(12).reshape((4, 3)),
                     index=[['a', 'a', 'b', 'b'],
                            [1, 2, 1, 2]],
                     columns=[['Ohio', 'Ohio', 'Colorado'],
                              ['Green', 'Red', 'Green']])
frame
```

Out[10]:

		Ohio		Colorado
		Green	**Red**	**Green**
a	**1**	0	1	2
	2	3	4	5
b	**1**	6	7	8
	2	9	10	11

names 属性

In [11]:

```
frame.index.names = ['key1', 'key2']
frame.columns.names = ['state', 'color']
frame
```

Out[11]:

	state	Ohio		Colorado
	color	**Green**	**Red**	**Green**
key1	**key2**			
a	**1**	0	1	2
	2	3	4	5
b	**1**	6	7	8
	2	9	10	11

切片

In [12]:
```
frame['Ohio'], frame['Ohio']['Green']
```

Out[12]:
```
(color      Green  Red
 key1 key2
 a    1         0    1
      2         3    4
 b    1         6    7
      2         9   10,
 key1  key2
 a     1       0
       2       3
 b     1       6
       2       9
 Name: Green, dtype: int32)
```

frame.set_index(keys, drop), frame.reset_index(level) : 将 DataFrame 的列转为索引

In [13]:
```
frame2 = frame.reset_index()
frame2
```

Out[13]:

state	key1	key2	Ohio		Colorado
color			Green	Red	Green
0	a	1	0	1	2
1	a	2	3	4	5
2	b	1	6	7	8
3	b	2	9	10	11

In [14]:
```
frame2.set_index(['key1', 'key2'])
```

Out[14]:

	state	Ohio		Colorado
	color	Green	Red	Green
key1	key2			
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

In [15]:
```
frame2.set_index(['key1', 'key2'], drop=False)
```

Out[15]:

	state	key1	key2	Ohio		Colorado
	color			Green	Red	Green
key1	key2					
a	1	a	1	0	1	2
	2	a	2	3	4	5
b	1	b	1	6	7	8
	2	b	2	9	10	11

frame.swaplevel(level1, level2), frame.sort_index(level) : 重排与分级排序

In [16]:

```
frame
```

Out[16]:

```
            state         Ohio  Colorado
            color  Green  Red      Green
key1  key2
   a     1         0      1          2
         2         3      4          5
   b     1         6      7          8
         2         9     10         11
```

In [17]:

```
frame.sort_index(level=1)
```

Out[17]:

```
            state         Ohio  Colorado
            color  Green  Red      Green
key1  key2
   a     1         0      1          2
   b     1         6      7          8
   a     2         3      4          5
   b     2         9     10         11
```

In [18]:

```
frame.swaplevel('key1', 'key2')
```

Out[18]:

```
            state         Ohio  Colorado
            color  Green  Red      Green
key2  key1
   1     a         0      1          2
   2     a         3      4          5
   1     b         6      7          8
   2     b         9     10         11
```

In [19]:

```
frame.swaplevel('key1', 'key2').sort_index(level='key2')
```

Out[19]:

```
            state         Ohio  Colorado
            color  Green  Red      Green
key2  key1
   1     a         0      1          2
         b         6      7          8
   2     a         3      4          5
         b         9     10         11
```

frame.groupby(axis, level).sum( ) : 汇总统计

In [20]:

```
frame.groupby(level='key2').sum()
```

Out[20]:

```
state         Ohio  Colorado
color  Green  Red      Green
key2
   1       6    8         10
   2      12   14         16
```

In [21]:
```
frame.groupby(axis=1, level='color').sum()
```

Out[21]:

	color	Green	Red
key1	key2		
a	1	2	1
	2	8	4
b	1	14	7
	2	20	10

合并数据

pd.merge(left, right, on, how, suffixes)

> *on* : 根据指定列进行合并, 默认为重叠列
> *how* : 连接效果, 默认为 ‘inner’, 其他方式还有 ‘outer’, ‘left’, ‘right’
> *suffixes* : 控制重复列的列名

In [22]:
```
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                    'data1': range(7)})
df2 = pd.DataFrame({'key': ['a', 'b', 'd'],
                    'data2': range(3)})
df1, df2
```

Out[22]:
```
(  key  data1
 0   b      0
 1   b      1
 2   a      2
 3   c      3
 4   a      4
 5   a      5
 6   b      6,
   key  data2
 0   a      0
 1   b      1
 2   d      2)
```

on : 控制连接的列

In [23]:
```
pd.merge(df1, df2)
# 并没有指明要用哪个列进行连接, 如果没有指定, merge 就会将重叠列的列名当做键
```

Out[23]:

	key	data1	data2
0	b	0	1
1	b	1	1
2	b	6	1
3	a	2	0
4	a	4	0
5	a	5	0

In [24]:
```
pd.merge(df1, df2, on = 'key')
```

Out[24]:

	key	data1	data2
0	b	0	1
1	b	1	1
2	b	6	1
3	a	2	0
4	a	4	0
5	a	5	0

根据多个列进行合并

In [25]:
```
left = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
                     'key2': ['one', 'two', 'one'],
                     'lval': [1, 2, 3]})
right = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
                      'key2': ['one', 'one', 'one', 'two'],
                      'rval': [4, 5, 6, 7]})
left, right
```

Out[25]:
```
(  key1 key2  lval
 0  foo  one     1
 1  foo  two     2
 2  bar  one     3,
   key1 key2  rval
 0  foo  one     4
 1  foo  one     5
 2  bar  one     6
 3  bar  two     7)
```

In [26]:
```
pd.merge(left, right, on=['key1', 'key2'], how='outer')
```

Out[26]:

	key1	key2	lval	rval
0	foo	one	1.0	4.0
1	foo	one	1.0	5.0
2	foo	two	2.0	NaN
3	bar	one	3.0	6.0
4	bar	two	NaN	7.0

suffixes : 控制合并后重复列的命名

In [27]:
```
pd.merge(left, right, on='key1')
```

Out[27]:

	key1	key2_x	lval	key2_y	rval
0	foo	one	1	one	4
1	foo	one	1	one	5
2	foo	two	2	one	4
3	foo	two	2	one	5
4	bar	one	3	one	6
5	bar	one	3	two	7

In [28]:
```
pd.merge(left, right, on='key1', suffixes=('_left', '_right'))
```

Out[28]:

	key1	key2_left	lval	key2_right	rval
0	foo	one	1	one	4
1	foo	one	1	one	5
2	foo	two	2	one	4
3	foo	two	2	one	5
4	bar	one	3	one	6
5	bar	one	3	two	7

pd.merge(left, right, left_on, right_on, how, suffixes)：分别根据左右指定列进行合并

In [29]:

```
df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                    'data1': range(7)})
df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'],
                    'data2': range(3)})
df3, df4
```

Out[29]:

```
(  lkey  data1
 0    b      0
 1    b      1
 2    a      2
 3    c      3
 4    a      4
 5    a      5
 6    b      6,
   rkey  data2
 0    a      0
 1    b      1
 2    d      2)
```

In [30]:

```
pd.merge(df3, df4, left_on='lkey', right_on='rkey')
```

Out[30]:

	lkey	data1	rkey	data2
0	b	0	b	1
1	b	1	b	1
2	b	6	b	1
3	a	2	a	0
4	a	4	a	0
5	a	5	a	0

In [31]:

```
pd.merge(df3, df4, left_on='lkey', right_on='rkey', how = 'outer')
## 出现了 c 和 d
```

Out[31]:

	lkey	data1	rkey	data2
0	b	0.0	b	1.0
1	b	1.0	b	1.0
2	b	6.0	b	1.0
3	a	2.0	a	0.0
4	a	4.0	a	0.0
5	a	5.0	a	0.0
6	c	3.0	NaN	NaN
7	NaN	NaN	d	2.0

In [32]:

```
pd.merge(df3, df4, left_on='lkey', right_on='rkey', how = 'left')
# 只有 c
```

Out[32]:

	lkey	data1	rkey	data2
0	b	0	b	1.0
1	b	1	b	1.0
2	a	2	a	0.0
3	c	3	NaN	NaN
4	a	4	a	0.0
5	a	5	a	0.0
6	b	6	b	1.0

pd.merge(left, right, left_index, right_index, how, suffixes)：分别根据左右索引进行合并

In [33]:
```
left1 = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'],
                      'value': range(6)})
right1 =  pd.DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])
left1, right1
```

Out[33]:
```
(  key  value
 0   a      0
 1   b      1
 2   a      2
 3   a      3
 4   b      4
 5   c      5,
    group_val
 a        3.5
 b        7.0)
```

In [34]:
```
pd.merge(left1, right1, left_index=True, right_index=True, how='outer')
```

Out[34]:

	key	value	group_val
0	a	0.0	NaN
1	b	1.0	NaN
2	a	2.0	NaN
3	a	3.0	NaN
4	b	4.0	NaN
5	c	5.0	NaN
a	NaN	NaN	3.5
b	NaN	NaN	7.0

In [35]:
```
pd.merge(left1, right1, left_on='key', right_index=True, how='outer')
```

Out[35]:

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0
5	c	5	NaN

层次化索引数据的合并：索引的合并默认是多键合并

In [36]:
```
lefth = pd.DataFrame({'key1': ['Ohio', 'Ohio', 'Ohio',
                               'Nevada', 'Nevada'],
                      'key2': [2000, 2001, 2002, 2001, 2002],
                      'data': np.arange(5.)})
righth = pd.DataFrame(np.arange(12).reshape((6, 2)),
                      index=[['Nevada', 'Nevada', 'Ohio', 'Ohio','Ohio', 'Ohio'],
                             [2001, 2000, 2000, 2000, 2001, 2002]],
                      columns=['event1', 'event2'])
lefth, righth
```

Out[36]:
```
(     key1  key2  data
 0    Ohio  2000   0.0
 1    Ohio  2001   1.0
 2    Ohio  2002   2.0
 3  Nevada  2001   3.0
 4  Nevada  2002   4.0,
              event1  event2
 Nevada 2001       0       1
        2000       2       3
 Ohio   2000       4       5
        2000       6       7
        2001       8       9
        2002      10      11)
```

In [37]:

```
pd.merge(lefth, righth, left_on=['key1', 'key2'], right_index=True, how='outer')
# righth 索引的合并默认是多键合并, 所以 lefth 必须以列表的形式指明多个列
```

Out[37]:

	key1	key2	data	event1	event2
0	Ohio	2000	0.0	4.0	5.0
0	Ohio	2000	0.0	6.0	7.0
1	Ohio	2001	1.0	8.0	9.0
2	Ohio	2002	2.0	10.0	11.0
3	Nevada	2001	3.0	0.0	1.0
4	Nevada	2002	4.0	NaN	NaN
4	Nevada	2000	NaN	2.0	3.0

left.join(right, how, on) : 更方便地实现按索引合并, 但要求没有重叠的列

> *on* : 合并时, left 按 on 指定的列合并, right 按索引进行合并
> *right* :

In [38]:

```
left2 = pd.DataFrame([[1., 2.], [3., 4.], [5., 6.]],
                     index=['a', 'c', 'e'],
                     columns=['Ohio', 'Nevada'])
right2 = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [13, 14]],
                      index=['b', 'c', 'd', 'e'],
                      columns=['Missouri', 'Alabama'])
left2, right2
```

Out[38]:

```
(   Ohio  Nevada
 a   1.0     2.0
 c   3.0     4.0
 e   5.0     6.0,
    Missouri  Alabama
 b       7.0      8.0
 c       9.0     10.0
 d      11.0     12.0
 e      13.0     14.0)
```

In [39]:

```
left2.join(right2)
```

Out[39]:

	Ohio	Nevada	Missouri	Alabama
a	1.0	2.0	NaN	NaN
c	3.0	4.0	9.0	10.0
e	5.0	6.0	13.0	14.0

In [40]:

```
left1, right1
```

Out[40]:

```
(  key  value
 0   a      0
 1   b      1
 2   a      2
 3   a      3
 4   b      4
 5   c      5,
    group_val
 a        3.5
 b        7.0)
```

In [41]:

```
left1.join(right1, on='key')
```

Out[41]:

	key	value	group_val
0	a	0	3.5
1	b	1	7.0
2	a	2	3.5
3	a	3	3.5
4	b	4	7.0
5	c	5	NaN

np.concatenate(arrs, axis) : NumPy 的轴向连接

In [42]:

```
arr = np.arange(12).reshape((3, 4))
arr
```

Out[42]:

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [43]:

```
np.concatenate([arr, arr], axis=1)
```

Out[43]:

```
array([[ 0,  1,  2,  3,  0,  1,  2,  3],
       [ 4,  5,  6,  7,  4,  5,  6,  7],
       [ 8,  9, 10, 11,  8,  9, 10, 11]])
```

pd.concat(objs, axis, join, keys, ignore_index) : pandas 的轴向连接, 可以将值和索引连在一起

In [44]:

```
s1 = pd.Series([0, 1], index=['a', 'b'])
s2 = pd.Series([2, 3, 4], index=['a', 'b', 'e'])
s3 = pd.Series([5, 6], index=['a', 'b'])
s1, s2, s3
```

Out[44]:

```
(a    0
 b    1
 dtype: int64,
 a    2
 b    3
 e    4
 dtype: int64,
 a    5
 b    6
 dtype: int64)
```

In [45]:

```
pd.concat([s1, s2, s3])
```

Out[45]:

```
a    0
b    1
a    2
b    3
e    4
a    5
b    6
dtype: int64
```

In [46]:

```
pd.concat([s1, s2, s3], axis=1)
```

Out[46]:

	0	1	2
a	0.0	2	5.0
b	1.0	3	6.0
e	NaN	4	NaN

join : 默认为 'outer'

In [47]:
```
pd.concat([s1, s2, s3], axis=1, join='inner')
```

Out[47]:

	0	1	2
a	0	2	5
b	1	3	6

In [48]:
```
pd.concat([s1, s2, s3], axis=1, join='outer')
```

Out[48]:

	0	1	2
a	0.0	2	5.0
b	1.0	3	6.0
e	NaN	4	NaN

keys :

沿着 axis=0 对 series 进行合并, 在连接轴创建一个层次化索引

In [49]:
```
pd.concat([s1, s1, s3], keys=['one','two', 'three'])
```

Out[49]:
```
one    a    0
       b    1
two    a    0
       b    1
three  a    5
       b    6
dtype: int64
```

沿着 axis=1 对 series 进行合并, keys 为列头

In [50]:
```
pd.concat([s1, s1, s3], axis=1, keys=['one','two', 'three'])
```

Out[50]:

	one	two	three
a	0	0	5
b	1	1	6

同样的逻辑适用于 DataFrame

In [51]:
```
df1 = pd.DataFrame(np.arange(6).reshape(3, 2), index=['a', 'b', 'c'],
                   columns=['one', 'two'])
df2 = pd.DataFrame(5 + np.arange(4).reshape(2, 2), index=['a', 'c'],
                   columns=['three', 'four'])
df1, df2
```

Out[51]:
```
(   one  two
 a    0    1
 b    2    3
 c    4    5,
    three  four
 a      5     6
 c      7     8)
```

In [52]:
```
pd.concat([df1, df2], axis=1, keys=['level1', 'level2'])
```

Out[52]:

	level1		level2	
	one	two	three	four
a	0	1	5.0	6.0
b	2	3	NaN	NaN
c	4	5	7.0	8.0

如果传入的 objs 不是列表而是一个字典, 则字典的键就会被当做 keys 的值

In [53]:
```
pd.concat({'level1': df1, 'level2': df2}, axis=1)
```

Out[53]:

	level1		level2	
	one	**two**	**three**	**four**
a	0	1	5.0	6.0
b	2	3	NaN	NaN
c	4	5	7.0	8.0

ignore_index : 不保留连接轴上的索引, 而是产生一组新索引

In [54]:
```
df1 = pd.DataFrame(np.random.randn(3, 4), columns=['a', 'b', 'c', 'd'])
df2 = pd.DataFrame(np.random.randn(2, 3), columns=['b', 'd', 'a'])
df1, df2
```

Out[54]:
```
(          a         b         c         d
 0 -0.384417 -0.394028  0.098734  0.325203
 1  2.274706 -1.128029 -2.126771  0.412944
 2  1.957194  1.825529  0.009230 -0.479239,
           b         d         a
 0 -0.757975 -1.809006 -0.396801
 1  0.112866  2.343804  0.815144)
```

In [55]:
```
pd.concat([df1, df2], ignore_index=False)
```

Out[55]:

	a	b	c	d
0	-0.384417	-0.394028	0.098734	0.325203
1	2.274706	-1.128029	-2.126771	0.412944
2	1.957194	1.825529	0.009230	-0.479239
0	-0.396801	-0.757975	NaN	-1.809006
1	0.815144	0.112866	NaN	2.343804

In [56]:
```
pd.concat([df1, df2], ignore_index=True)
```

Out[56]:

	a	b	c	d
0	-0.384417	-0.394028	0.098734	0.325203
1	2.274706	-1.128029	-2.126771	0.412944
2	1.957194	1.825529	0.009230	-0.479239
3	-0.396801	-0.757975	NaN	-1.809006
4	0.815144	0.112866	NaN	2.343804

np.where(condition, arr1, arr2) : Numpy 合并索引全部或部分重叠的数据的方法

In [57]:

```
a = pd.Series([np.nan, 2.5, np.nan, 3.5, 4.5, np.nan],
              index=['f', 'e', 'd', 'c', 'b', 'a'])
b = pd.Series(np.arange(len(a), dtype=np.float64),
              index=['f', 'e', 'd', 'c', 'b', 'a'])
b[-1] = np.nan
a, b
```

Out[57]:

```
(f    NaN
 e    2.5
 d    NaN
 c    3.5
 b    4.5
 a    NaN
 dtype: float64,
 f    0.0
 e    1.0
 d    2.0
 c    3.0
 b    4.0
 a    NaN
 dtype: float64)
```

In [58]:

```
np.where(pd.notnull(a), a, b)
```

Out[58]:

```
array([0. , 2.5, 2. , 3.5, 4.5, nan])
```

obj1.combine_first(obj2) : pandas 合并索引全部或部分重叠的数据的方法

In [59]:

```
a.combine_first(b)
# 将 b 覆盖到 a 上，值已经存在的部分不填充，值为缺失的部分填充
```

Out[59]:

```
f    0.0
e    2.5
d    2.0
c    3.5
b    4.5
a    NaN
dtype: float64
```

In [60]:

```
df1 = pd.DataFrame({'a': [1., np.nan, 5., np.nan],
                    'b': [np.nan, 2., np.nan, 6.],
                    'c': range(2, 18, 4)})
df2 = pd.DataFrame({'a': [5., 4., np.nan, 3., 7.],
                    'b': [np.nan, 3., 4., 6., 8.]})
df1, df2
```

Out[60]:

```
(     a    b   c
 0  1.0  NaN   2
 1  NaN  2.0   6
 2  5.0  NaN  10
 3  NaN  6.0  14,
      a    b
 0  5.0  NaN
 1  4.0  3.0
 2  NaN  4.0
 3  3.0  6.0
 4  7.0  8.0)
```

In [61]:

```
df1.combine_first(df2)
# 将 df2 覆盖到 df1 上，值已经存在的部分不填充，值为缺失的部分填充
```

Out[61]:

	a	b	c
0	1.0	NaN	2.0
1	4.0	2.0	6.0
2	5.0	4.0	10.0
3	3.0	6.0	14.0
4	7.0	8.0	NaN

数据重塑和轴向旋转

进行 unstack 或 stack 时, 返回的结果中, 作为 **旋转轴（即 *level* ）** 的级别将成为 **最低级别**

frame.stack(level) : 将横轴旋转为竖轴, DataFrame → Series

In [62]:
```
frame = pd.DataFrame(np.arange(6).reshape((2, 3)),
                     index=pd.Index(['Ohio', 'Colorado'], name='state'),
                     columns=pd.Index(['one', 'two', 'three'],
                     name='number'))
frame
```

Out[62]:

number	one	two	three
state			
Ohio	0	1	2
Colorado	3	4	5

In [63]:
```
series = frame.stack()
series
```

Out[63]:
```
state     number
Ohio      one       0
          two       1
          three     2
Colorado  one       3
          two       4
          three     5
dtype: int32
```

level : 进行 unstack 或 stack 时, 返回的结果中, 作为 **旋转轴（即 *level* ）** 的级别将成为 **最低级别**

In [64]:
```
df1 = pd.DataFrame(
                  {'left': series, 'right': series + 5},
                  columns = pd.Index(['left', 'right'], name='side')
                 )
df1
```

Out[64]:

	side	left	right
state	number		
Ohio	one	0	5
	two	1	6
	three	2	7
Colorado	one	3	8
	two	4	9
	three	5	10

In [65]:
```
df2 = df1.unstack('state')
df2
```

Out[65]:

side	left		right	
state	Ohio	Colorado	Ohio	Colorado
number				
one	0	3	5	8
two	1	4	6	9
three	2	5	7	10

In [66]:
```
df3 = df2.stack('side')
df3
```

Out[66]:

	state	Colorado	Ohio
number	**side**		
one	**left**	3	0
	right	8	5
two	**left**	4	1
	right	9	6
three	**left**	5	2
	right	10	7

series.unstack(level) : 将竖轴旋转为横轴, Series → DataFrame

In [67]:
```
series
```

Out[67]:
```
state     number
Ohio      one       0
          two       1
          three     2
Colorado  one       3
          two       4
          three     5
dtype: int32
```

In [68]:
```
series.unstack()
```

Out[68]:

number	one	two	three
state			
Ohio	0	1	2
Colorado	3	4	5

In [69]:
```
series.unstack(level=0)
```

Out[69]:

state	Ohio	Colorado
number		
one	0	3
two	1	4
three	2	5

In [70]:
```
series.unstack(level='state')
```

Out[70]:

state	Ohio	Colorado
number		
one	0	3
two	1	4
three	2	5

如果不是所有的索引都能在各分组中找到的话, 则 unstack 可能会引入缺失值

In [71]:
```
s1 = pd.Series([0, 1, 2, 3], index=['a', 'b', 'c', 'd'])
s2 = pd.Series([4, 5, 6], index=['c', 'd', 'e'])
s = pd.concat([s1, s2], keys=['one', 'two'])
s
```

Out[71]:
```
one  a    0
     b    1
     c    2
     d    3
two  c    4
     d    5
     e    6
dtype: int64
```

In [72]:
```
df = s.unstack()
# two 分组中找不到 a, b 这两个索引
df
```

Out[72]:

	a	b	c	d	e
one	0.0	1.0	2.0	3.0	NaN
two	NaN	NaN	4.0	5.0	6.0

stack 会默认过滤掉缺失值

In [73]:
```
df.stack()
```

Out[73]:
```
one  a    0.0
     b    1.0
     c    2.0
     d    3.0
two  c    4.0
     d    5.0
     e    6.0
dtype: float64
```

可以利用 ***dropna*** 控制是否过滤缺失值

In [74]:
```
df.stack(dropna = False)
```

Out[74]:
```
one  a    0.0
     b    1.0
     c    2.0
     d    3.0
     e    NaN
two  a    NaN
     b    NaN
     c    4.0
     d    5.0
     e    6.0
dtype: float64
```

frame.pivot(index, columns, values)：将长格式旋转为宽格式

index 和 columns 分别用作行和列索引, values 用于填充的数据列（可选）

In [75]:

```
data = pd.read_csv('pydata-book-2nd-edition/examples/macrodata.csv')
periods = pd.PeriodIndex(year=data.year, quarter=data.quarter,
                         name='date')
columns = pd.Index(['realgdp', 'infl', 'unemp'], name='item')
data = data.reindex(columns=columns)
data.index = periods.to_timestamp('D', 'end')
ldata = data.stack().reset_index().rename(columns={0: 'value'})
ldata
```

Out[75]:

	date	item	value
0	1959-03-31 23:59:59.999999999	realgdp	2710.349
1	1959-03-31 23:59:59.999999999	infl	0.000
2	1959-03-31 23:59:59.999999999	unemp	5.800
3	1959-06-30 23:59:59.999999999	realgdp	2778.801
4	1959-06-30 23:59:59.999999999	infl	2.340
...	...	...	...
604	2009-06-30 23:59:59.999999999	infl	3.370
605	2009-06-30 23:59:59.999999999	unemp	9.200
606	2009-09-30 23:59:59.999999999	realgdp	12990.341
607	2009-09-30 23:59:59.999999999	infl	3.560
608	2009-09-30 23:59:59.999999999	unemp	9.600

609 rows × 3 columns

In [76]:

```
ldata.pivot('date', 'item', 'value')
# 以 date 为索引（去除共同项），将 item 进行拆分旋转，将 value 填到对应的位置中
```

Out[76]:

item	infl	realgdp	unemp
date			
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2
...	...	...	...
2008-09-30 23:59:59.999999999	-3.16	13324.600	6.0
2008-12-31 23:59:59.999999999	-8.79	13141.920	6.9
2009-03-31 23:59:59.999999999	0.94	12925.410	8.1
2009-06-30 23:59:59.999999999	3.37	12901.504	9.2
2009-09-30 23:59:59.999999999	3.56	12990.341	9.6

203 rows × 3 columns

假设有两个需要同时重塑的数据列

In [77]:

```
ldata['value2'] = np.random.randn(len(ldata))
ldata
```

Out[77]:

	date	item	value	value2
0	1959-03-31 23:59:59.999999999	realgdp	2710.349	0.176200
1	1959-03-31 23:59:59.999999999	infl	0.000	-0.194493
2	1959-03-31 23:59:59.999999999	unemp	5.800	-0.363329
3	1959-06-30 23:59:59.999999999	realgdp	2778.801	-1.510863
4	1959-06-30 23:59:59.999999999	infl	2.340	0.144886
...	...	...	...	...
604	2009-06-30 23:59:59.999999999	infl	3.370	-0.936350
605	2009-06-30 23:59:59.999999999	unemp	9.200	1.452993
606	2009-09-30 23:59:59.999999999	realgdp	12990.341	-0.137653
607	2009-09-30 23:59:59.999999999	infl	3.560	-0.423567
608	2009-09-30 23:59:59.999999999	unemp	9.600	0.086320

609 rows × 4 columns

In [78]:

```
ldata.pivot('date', 'item')
```

Out[78]:

		value			value2	
item	**infl**	**realgdp**	**unemp**	**infl**	**realgdp**	**unemp**
date						
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8	-0.194493	0.176200	-0.363329
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1	0.144886	-1.510863	0.619092
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3	-1.519818	1.400742	-0.669686
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6	-0.027059	1.053187	-1.069302
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2	-1.157763	-1.174653	-0.562152
...	...	...	...	...	...	...
2008-09-30 23:59:59.999999999	-3.16	13324.600	6.0	0.632464	-0.447070	1.267958
2008-12-31 23:59:59.999999999	-8.79	13141.920	6.9	-1.030839	-0.359484	0.606553
2009-03-31 23:59:59.999999999	0.94	12925.410	8.1	-1.447697	0.711061	-0.873898
2009-06-30 23:59:59.999999999	3.37	12901.504	9.2	-0.936350	0.395591	1.452993
2009-09-30 23:59:59.999999999	3.56	12990.341	9.6	-0.423567	-0.137653	0.086320

203 rows × 6 columns

In [79]:

```
ldata.pivot('date', 'item', ['value', 'value2'])
```

Out[79]:

	value			value2		
item	**infl**	**realgdp**	**unemp**	**infl**	**realgdp**	**unemp**
date						
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8	-0.194493	0.176200	-0.363329
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1	0.144886	-1.510863	0.619092
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3	-1.519818	1.400742	-0.669686
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6	-0.027059	1.053187	-1.069302
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2	-1.157763	-1.174653	-0.562152
...	...	...	...	...	...	...
2008-09-30 23:59:59.999999999	-3.16	13324.600	6.0	0.632464	-0.447070	1.267958
2008-12-31 23:59:59.999999999	-8.79	13141.920	6.9	-1.030839	-0.359484	0.606553
2009-03-31 23:59:59.999999999	0.94	12925.410	8.1	-1.447697	0.711061	-0.873898
2009-06-30 23:59:59.999999999	3.37	12901.504	9.2	-0.936350	0.395591	1.452993
2009-09-30 23:59:59.999999999	3.56	12990.341	9.6	-0.423567	-0.137653	0.086320

203 rows × 6 columns

pivot 其实就是用 set_index 创建层次化索引, 再用 unstack 重塑

In [80]:

```
ldata.set_index(['date', 'item']).unstack('item')
```

Out[80]:

	value			value2		
item	**infl**	**realgdp**	**unemp**	**infl**	**realgdp**	**unemp**
date						
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8	-0.194493	0.176200	-0.363329
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1	0.144886	-1.510863	0.619092
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3	-1.519818	1.400742	-0.669686
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6	-0.027059	1.053187	-1.069302
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2	-1.157763	-1.174653	-0.562152
...	...	...	...	...	...	...
2008-09-30 23:59:59.999999999	-3.16	13324.600	6.0	0.632464	-0.447070	1.267958
2008-12-31 23:59:59.999999999	-8.79	13141.920	6.9	-1.030839	-0.359484	0.606553
2009-03-31 23:59:59.999999999	0.94	12925.410	8.1	-1.447697	0.711061	-0.873898
2009-06-30 23:59:59.999999999	3.37	12901.504	9.2	-0.936350	0.395591	1.452993
2009-09-30 23:59:59.999999999	3.56	12990.341	9.6	-0.423567	-0.137653	0.086320

203 rows × 6 columns

pd.melt(frame, id_vars, value_vars) : 将宽格式旋转为长格式

id_vars 为分组指标, value_vars 为合并到一起的列

In [81]:

```
df = pd.DataFrame({'key': ['foo', 'bar', 'baz'],
                   'A': [1, 2, 3],
                   'B': [4, 5, 6],
                   'C': [7, 8, 9]})
df
```

Out[81]:

	key	A	B	C
0	foo	1	4	7
1	bar	2	5	8
2	baz	3	6	9

In [82]:
```
pd.melt(df, ['key'], ['A', 'B'])
```

Out[82]:
```
   key  variable  value
0  foo         A      1
1  bar         A      2
2  baz         A      3
3  foo         B      4
4  bar         B      5
5  baz         B      6
```

In [83]:
```
pd.melt(df, ['A', 'B'], ['C', 'key'])
```

Out[83]:
```
   A  B  variable  value
0  1  4         C      7
1  2  5         C      8
2  3  6         C      9
3  1  4       key    foo
4  2  5       key    bar
5  3  6       key    baz
```