Ch 8 Data Wrangling Join Combine and Reshape 数据规整 连接 联合 与重塑

In [311]:
```
import pandas as pd
import numpy as np
```

8.1 分层索引：Hierarchical Indexing

In [312]:
```
#分层索引是pandas重要特性，允许在一个轴向上拥有多个索引层级。
#分层索引提供了一种在更低维度处理更高维度数据的方式。
#先创建一个series，以列表的列表作为索引：

data = pd.Series(np.random.randn(9),
                 index = [['a', 'a', 'a', 'b', 'b', 'c', 'c', 'd', 'd'],
                          [1, 2, 3, 1, 3, 1, 2, 2, 3]])
data
# a  1   -1.305418
#    2    1.776525
#    3   -0.201306
# b  1   -0.670462
#    3   -0.233106
# c  1   -0.837201
#    2   -0.001711
# d  2   -0.729868
#    3    1.570326
# dtype: float64

#此处看到的是以multiIndex作为索引的series的美化视图。索引中的间隙，表示直接使用上面的标签。
data.index
# MultiIndex([('a', 1),
#             ('a', 2),
#             ('a', 3),
#             ('b', 1),
#             ('b', 3),
#             ('c', 1),
#             ('c', 2),
#             ('d', 2),
#             ('d', 3)],
#            )
```

Out[312]:
```
MultiIndex([('a', 1),
            ('a', 2),
            ('a', 3),
            ('b', 1),
            ('b', 3),
            ('c', 1),
            ('c', 2),
            ('d', 2),
            ('d', 3)],
           )
```

In [313]:
```
#通过分层索引对象，也可以成为部分索引，允许简洁地选择出数据的子集：
data['b']
# 1   -0.973907
# 3   -0.593977
# dtype: float64
data['b':'c']
# b  1   -0.973907
#    3   -0.593977
# c  1    0.636276
#    2    1.477976
# dtype: float64
data.loc[['b','d']]
# b  1   -0.973907
#    3   -0.593977
# d  2   -1.099617
#    3    0.175689
# dtype: float64

#在内部层级中选择也可以：
data.loc[:, 2]     #这从DataFrame对象 data 中选择所有行的第2列数据。？
# a   -0.680452
# c    1.477976
# d   -1.099617
# dtype: float64
```

Out[313]:
```
a    0.499375
c   -1.348917
d   -0.849413
dtype: float64
```

In [314]:
```
#unstack：unstack() 是一个用于将多层索引的数据进行重塑的方法。
#如果DataFrame对象具有层次化的列索引（MultiIndex），则可以使用 unstack() 方法将其中一个或多个层次的列索引转换为行索引，
#从而得到一个更加扁平化的表格形式。

#分层索引在重塑数据和数组透视表等分组操作中很重要。
#可以用unstack方法，在data frame中重新排列：
data.unstack()
# 1 2   3
# a -1.815016   -0.680452   0.069912
# b -0.973907   NaN -0.593977
# c 0.636276    1.477976    NaN
# d NaN -1.099617   0.175689

#unstack的反操作是stack：
data.unstack().stack()
# a  1   -1.815016
#    2   -0.680452
#    3    0.069912
# b  1   -0.973907
#    3   -0.593977
# c  1    0.636276
#    2    1.477976
# d  2   -1.099617
#    3    0.175689
# dtype: float64
```

Out[314]:
```
a  1    0.635871
   2    0.499375
   3   -1.427378
b  1    1.509858
   3    0.674879
c  1   -1.093609
   2   -1.348917
d  2   -0.849413
   3    1.870791
dtype: float64
```

In [315]:
```
#在data frame中，每个轴都可以拥有分层索引:
#先创建一个4*3的表，纵轴定为a,b，每个字母下设12，横轴定为Ohio, Colorado, Ohio下设Green, Red
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
#   Ohio    Colorado
# Green Red Green
# a 1   0   1   2
# 2 3   4   5
# b 1   6   7   8
# 2 9   10  11
```

Out[315]:

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

In [316]:
```
#分层的层级可以有名称（字符串或python对象）。如果层级有名称，名称会在控制台输出中显示。

#将a,b;1,2命名为key; 将ohio, colorado; green, red, green命名为state, color
frame.index.names = ['key1', 'key2']
frame.columns.names = ['state', 'color']
frame
#   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

#通过部分列索引，可以选出列中的组:
frame['Ohio']
#   color   Green   Red
# key1  key2
# a 1   0   1
# 2 3   4
# b 1   6   7
# 2 9   10

#一个multiIndex的对象可以使用自身的构造函数创建并复用。带有层级的data frame的列可以这样创建:
#MultiIndex.from_arrays([['Ohio', 'Ohio', 'Colorado'],['Green', 'Red', 'Green']], names = ['state', 'color'])
```

Out[316]:

	color	Green	Red
key1	key2		
a	1	0	1
	2	3	4
b	1	6	7
	2	9	10

8.1.1 重排序和层次排序 Reordering and Sorting Levels

In [317]:
```
#需要重新排列轴上的层级顺序，或按照特定层级的值，对数据排序。
#swaplevel接收两个层级序号或层级名称。返回一个进行了层级变更的新对象。

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

#sort_index只能在单一层级上，对数据排序。
#进行层级变换时，使用sort_index使得结果按照层级进行字典排序也常见：
frame.sort_index(level = 1)                               #表示根据索引的第一层级进行排序。
#   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

frame.swaplevel(0, 1).sort_index(level = 0)           #交换DataFrame对象 frame 的两个层级的顺序，并在第一个层级上进行排序。
# # 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
```

Out[317]:

	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

8.1.2 按层级进行汇总统计 Summary Statistics by Level

In [318]:
```
#dataframe和series中很多描述性和汇总性统计有一个level选项，通过level选项可以指定再某个特定的轴上进行聚合。
#可以按照层级在行或列聚合：

frame.sum(level = 'key2')   #以key2为groupby，内部进行sum
# state Ohio    Colorado
# color Green   Red Green
# key2
# 1 6    8   10
# 2 12   14  16

frame.sum(level = 'color', axis = 1)  #以color为group by，内部进行sum
# color Green   Red
# key1   key2
# a 1    2   1
# 2 8    4
# b 1    14  7
# 2 20   10
```

```
C:\Users\miran\AppData\Local\Temp\ipykernel_76136\805996806.py:4: FutureWarning: Using the level keyword in DataFrame and Series aggregations is deprecated and will be removed in a future version. Use groupby instead. df.sum(level=1) should use df.groupby(level=1).sum().
  frame.sum(level = 'key2')   #以key2为groupby，内部进行sum
C:\Users\miran\AppData\Local\Temp\ipykernel_76136\805996806.py:11: FutureWarning: Using the level keyword in DataFrame and Series aggregations is deprecated and will be removed in a future version. Use groupby instead. df.sum(level=1) should use df.groupby(level=1).sum().
  frame.sum(level = 'color', axis = 1)  #以color为group by，内部进行sum
```

Out[318]:

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

8.1.3 使用data frame的列进行索引 Indexing with a DataFrame's columns

In [319]:
```
#当想将行索引移动到data frame中。

frame = pd.DataFrame({'a': range(7), 'b': range(7, 0, -1),
                      'c': ['one', 'one', 'one', 'two', 'two', 'two', 'two'],
                      'd': [0, 1, 2, 0, 1, 2, 3]})


frame
# a b   c   d
# 0 0   7   one 0
# 1 1   6   one 1
# 2 2   5   one 2
# 3 3   4   two 0
# 4 4   3   two 1
# 5 5   2   two 2
# 6 6   1   two 3

#data frame的set_index会生成一个新的data frame，新df使用一个或多个列作为索引：
frame2 = frame.set_index(['c', 'd'])                            #把c,d 从列名，变为index行名
frame2
#       a   b
# c d
# one   0   0   7
# 1 1   6
# 2 2   5
# two   0   3   4
# 1 4   3
# 2 5   2
# 3 6   1

#默认情况，这些列会从df中移除，也可以将其留在data frame中：
frame.set_index(['c', 'd'], drop = False)                  #纵列的cd仍然保留，也出现在行名中
# a b   c   d
# c d
# one   0   0   7   one 0
# 1 1   6   one 1
# 2 2   5   one 2
# two   0   3   4   two 0
# 1 4   3   two 1
# 2 5   2   two 2
# 3 6   1   two 3

#reset_index是set_index 反操作，分层索引的行会被返回移动到原来的列中：
frame2.reset_index()
# c d   a   b
# 0 one 0   0   7
# 1 one 1   1   6
# 2 one 2   2   5
# 3 two 0   3   4
# 4 two 1   4   3
# 5 two 2   5   2
# 6 two 3   6   1
```

Out[319]:

	c	d	a	b
0	one	0	0	7
1	one	1	1	6
2	one	2	2	5
3	two	0	3	4
4	two	1	4	3
5	two	2	5	2
6	two	3	6	1

8.2 联合与合并数据库 Combining and Merging Datasets

8.2.1 数据库风格的data frame连接 Database-Style DataFrame Joins

In [320]:

```
#合并或连接操作通过一个或多个键连接行，来联合数据集。
#pandas中的merge用于将各种join应用于数据

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
# key   data1
# 0 b   0
# 1 b   1
# 2 a   2
# 3 c   3
# 4 a   4
# 5 a   5
# 6 b   6
df2
# key   data2
# 0 a   0
# 1 b   1
# 2 d   2

#这是多对一的例子：df1的数据有多个行的标签为a,b；而df2在key列每个值仅有一行
#调用merge处理获得的对象：找到两个DataFrame中匹配的列（或索引）并将它们对应的行合并在一起
#因为没有指定再哪一列上进行连接，如果连接的键信息没有指定，merge会自动将重叠列名作为连接的键。
pd.merge(df1, df2)
# 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

#如果每个对象的列名是不同的，可以分别制定列名：
df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                    'data1': range(7)})
df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'],
                    'data2': range(3)})
pd.merge(df3, df4, left_on = 'lkey', right_on = 'rkey')  #左边DataFrame的 'lkey' 列和右边DataFrame的 'rkey' 列上进行匹配
# 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

#因为结果中缺少'c', 'd'的值，以及相关的数据。
#默认情况下，merge做的是内连接：inner join；其余选项为'left', 'right', 'outer'
pd.merge(df1, df2, how = 'outer')
#   key data1   data2
# 0 b   0.0 1.0
# 1 b   1.0 1.0
# 2 b   6.0 1.0
# 3 a   2.0 0.0
# 4 a   4.0 0.0
# 5 a   5.0 0.0
# 6 c   3.0 NaN
# 7 d   NaN 2.0
```

Out[320]:

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

In [321]:
```
#多对多的合并，有明确的行为:

df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                   'data1': range(6)})
df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],
                   'data2': range(5)})
df1
# key   data1
# 0 b   0
# 1 b   1
# 2 a   2
# 3 c   3
# 4 a   4
# 5 b   5
df2
# key   data2
# 0 a   0
# 1 b   1
# 2 a   2
# 3 b   3
# 4 d   4

pd.merge(df1, df2, on = 'key', how = 'left')
# key   data1   data2
# 0 b   0   1.0
# 1 b   0   3.0
# 2 b   1   1.0
# 3 b   1   3.0
# 4 a   2   0.0
# 5 a   2   2.0
# 6 c   3   NaN
# 7 a   4   0.0
# 8 a   4   2.0
# 9 b   5   1.0
# 10    b   5   3.0

#多对多连接是行的笛卡尔积，由于在左边的data frame中有3个'b'行
#在右边有两行，因此在结果中有6个'b'行

pd.merge(df1, df2, how = 'inner')
#   key data1   data2
# 0 b   0   1
# 1 b   0   3
# 2 b   1   1
# 3 b   1   3
# 4 b   5   1
# 5 b   5   3
# 6 a   2   0
# 7 a   2   2
# 8 a   4   0
# 9 a   4   2

#on = ['key1', 'key2']: 多个键进行合并时，传入一个列名的列表:
df3 = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
                     'key2': ['one', 'two', 'one'],
                     'lval': [1, 2, 3]})

df4 = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
                      'key2': ['one', 'one', 'one', 'two'],
                      'rval': [4, 5, 6, 7]})
df3
#   key1    key2    lval
# 0 foo one 1
# 1 foo two 2
# 2 bar one 3
df4
# key1  key2    rval
# 0 foo one 4
# 1 foo one 5
# 2 bar one 6
# 3 bar two 7

pd.merge(df3, df4, on = ['key1', 'key2'], how = 'outer')
# # 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
```

Out[321]:

	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

In [322]:

```
#merge有个suffixes后缀，用于在左右两边data frame对象的重叠列名后添加字符串:
#eg. key2_x ->key2_left key2_y ->key2_right

pd.merge(df3, df4, on = 'key1')
#   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

pd.merge(df3, df4, on = 'key1', suffixes = ('_left', '_right'))
# 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
```

Out[322]:

	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

In [323]:

```
#merge函数
# left
# right
# how
# on
# left_on
# right_on
# left_index
# right_index
# sort
# suffixes
# copy
# indicator
```

8.2.2 根据索引合并 Merging on Index

In [324]:

```
#left_index = True or right_index = True来表示索引需要用来作为合并的键:
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
#   key value
# 0 a   0
# 1 b   1
# 2 a   2
# 3 a   3
# 4 b   4
# 5 c   5
right1
# # group_val
# # a   3.5
# # b   7.0

#类似于join的左边是key,右边是index
# left_on='key' 和 right_index=True: 根据左侧DataFrame中的 'key' 列和右侧DataFrame的索引index进行匹配, 将匹配的行合并在一起
pd.merge(left1, right1, left_on = 'key', right_index = True)
#   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

#由于默认的合并方法是连接键相交, 可以使用外连接进行合并:
pd.merge(left1, right1, left_on = 'key', right_index = True, how = 'outer')
# 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
```

Out[324]:

	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

In [325]:

```
#多层索引数据情况下，索引上连接是一个隐式的多键合并：
left = pd.DataFrame({'key1': ['Ohio','Ohio', 'Ohio', 'Nevada', 'Nevada'],
                     'key2': [2000, 2001, 2002, 2001, 2002],
                     'data': np.arange(5.)})
lefth
#   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
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'])
righth
# event1    event2
# Nevada    2001    0   1
# 2000  2   3
# Ohio  2000    4   5
# 2000  6   7
# 2001  8   9
# 2002  10  11

#join多个key：以列表的方式，指明合并所需多个列（使用how = 'outer'处理重复的索引值）：
pd.merge(lefth, righth, left_on = ['key1','key2'], right_index = True, how = 'outer')
# 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
```

Out[325]:

	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

In [326]:
```
#使用两边的索引进行合并也可以：
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
# Ohio  Nevada
# a 1.0 2.0
# c 3.0 4.0
# e 5.0 6.0
right2
# Missouri  Alabama
# b 7.0 8.0
# c 9.0 10.0
# d 11.0    12.0
# e 13.0    14.0

pd.merge(left2, right2, how = 'outer', left_index = True, right_index = True)
#   Ohio    Nevada  Missouri    Alabama
# a 1.0 2.0 NaN NaN
# b NaN NaN 7.0 8.0
# c 3.0 4.0 9.0 10.0
# d NaN NaN 11.0    12.0
# e 5.0 6.0 13.0    14.0

#data frame有一个方便的join实例方法，用于按照用于合并多个索引相同或相似，但没有重叠列的data frame.
left2.join(right2, how = 'outer')
# Ohio  Nevada  Missouri    Alabama
# a 1.0 2.0 NaN NaN
# b NaN NaN 7.0 8.0
# c 3.0 4.0 9.0 10.0
# d NaN NaN 11.0    12.0
# e 5.0 6.0 13.0    14.0
```

Out[326]:

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

In [327]:
```
#data frame的join方法，进行连接键上的左连接，完全保留左边data frame的行索引。
left1.join(right1, on = 'key')
#   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

#对于一些简单索引-索引合并，可以向join方法传入一个Data frame的列表，此法可以替代concat函数方法：
another = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [16., 17.]],
                      index = ['a', 'c', 'e', 'f'],
                      columns = ['New York', 'Oregon'])

another
# New York  Oregon
# a 7.0 8.0
# c 9.0 10.0
# e 11.0    12.0
# f 16.0    17.0


#将三个DataFrame对象 left2、right2 和 another 进行连接操作。
#left2 是左侧的DataFrame，right2 和 another 是右侧的DataFrame。
#默认使用左连接 (left join) ，即保留左侧DataFrame中的所有行，并将右侧DataFrame中匹配的行进行连接。
#连接后的结果将包含左侧DataFrame的所有列和右侧DataFrame的列。

left2.join([right2, another])
#   Ohio    Nevada  Missouri    Alabama New York    Oregon
# a 1.0 2.0 NaN NaN 7.0 8.0
# c 3.0 4.0 9.0 10.0    9.0 10.0
# e 5.0 6.0 13.0    14.0    11.0    12.0

#加入how使其全连接。
left2.join([right2, another], how = 'outer')
#   Ohio    Nevada  Missouri    Alabama New York    Oregon
# a 1.0 2.0 NaN NaN 7.0 8.0
# c 3.0 4.0 9.0 10.0    9.0 10.0
# e 5.0 6.0 13.0    14.0    11.0    12.0
# b NaN NaN 7.0 8.0 NaN NaN
# d NaN NaN 11.0    12.0    NaN NaN
# f NaN NaN NaN NaN 16.0    17.0
```

Out[327]:

	Ohio	Nevada	Missouri	Alabama	New York	Oregon
a	1.0	2.0	NaN	NaN	7.0	8.0
c	3.0	4.0	9.0	10.0	9.0	10.0
e	5.0	6.0	13.0	14.0	11.0	12.0
b	NaN	NaN	7.0	8.0	NaN	NaN
d	NaN	NaN	11.0	12.0	NaN	NaN
f	NaN	NaN	NaN	NaN	16.0	17.0

8.2.3 沿轴向连接 Concatenating Along an Axis

In [328]:

```
#numpy的concatenate函数可以在numpy数组上实现功能:

arr = np.arange(12).reshape((3,4))
arr
# array([[ 0,  1,  2,  3],
#        [ 4,  5,  6,  7],
#        [ 8,  9, 10, 11]])

#把两个array连在一起
np.concatenate([arr, arr], axis = 1)
# 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]])

#在series和data frame等pandas对象的上下文中，使用标记的轴，可以进一步泛化数组连接。
#pandas的concat函数提供了一种一致性的方式来解决。假设有3个索引不存在重叠的series。
s1 = pd.Series([0, 1], index = ['a', 'b'])
s2 = pd.Series([2, 3, 4], index = ['c', 'd', 'e'])
s3 = pd.Series([5, 6], index = ['f', 'g'])

#列表中的对象调用concat方法会将值和索引粘在一起:
pd.concat([s1, s2, s3])
# a    0
# b    1
# c    2
# d    3
# e    4
# f    5
# g    6
# dtype: int64

#默认情况下，concat方法是沿着Axis = 0的轴向生效的，生成另一个series。如果传递axis = 1, 返回的结果是一个data frame。
pd.concat([s1, s2, s3], axis = 1)
# 0 1   2
# a 0.0 NaN NaN
# b 1.0 NaN NaN
# c NaN 2.0 NaN
# d NaN 3.0 NaN
# e NaN 4.0 NaN
# f NaN NaN 5.0
# g NaN NaN 6.0

#在这个案例中另一个轴向上没有重叠，可以看到排序后的索引合集（'outer' join外连接）。也可以传入join = 'inner':
s4 = pd.concat([s1, s3])
s4
# a    0
# b    1
# f    5
# g    6
# dtype: int64
pd.concat([s1, s4], axis = 1)  #出df
# 0 1
# a 0.0 0
# b 1.0 1
# f NaN 5
# g NaN 6
pd.concat([s1, s4], axis = 1, join = 'inner') #出df
# 0 1
# a 0   0
# b 1   1
```

Out[328]:

	0	1
a	0	0
b	1	1

In [329]:
```
#可以使用reindex指定用于连接其他轴向的轴:
#pd.concat([s1, s4], axis = 1, join_axes = [['a', 'c', 'b', 'e']])     #书中的写法join_axes已被弃用。可以尝试使用 reindex 方法;
pd.concat([s1, s4], axis=1).reindex(['a', 'c', 'b', 'e'])

#拼接在一起的各部分无法在结果中区分是一个问题。如果想在连接轴向上，创建一个多层索引，可以使用keys参数来实现:
result = pd.concat([s1, s1, s3], keys = ['one', 'two', 'three'])
result
# one    a    0
#        b    1
# two    a    0
#        b    1
# three  f    5
#        g    6
# dtype: int64

result.unstack()
# a b   f   g
# one   0.0 1.0 NaN NaN
# two   0.0 1.0 NaN NaN
# three NaN NaN 5.0 6.0

#沿着轴向axis = 1连接series, keys成为data frame的列头:
pd.concat([s1, s2, s3], axis = 1, keys = ['one', 'two', 'three'])
# one   two three
# a 0.0 NaN NaN
# b 1.0 NaN NaN
# c NaN 2.0 NaN
# d NaN 3.0 NaN
# e NaN 4.0 NaN
# f NaN NaN 5.0
# g NaN NaN 6.0

#相同逻辑拓展到data frame对象:
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
# one   two
# a 0   1
# b 2   3
# c 4   5
df2
# three four
# a 5   6
# c 7   8

pd.concat([df1, df2], axis = 1, keys = ['level1', 'level2'])
# 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

#给df加轴标签:
pd.concat({'level1': df1, 'level2':df2}, axis =1)
# 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

#names生成轴层级:
pd.concat([df1, df2], axis = 1, keys = ['level1', 'level2'],
          names = ['upper', 'lower'])

#最后考虑的是行索引不包含任何相关数据的data frame:

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
# a b   c   d
# 0 -1.420429   -0.764390   2.550669    0.229916
# 1 0.916529    0.350413    -0.156132   -0.201702
# 2 -0.215096   -0.091527   -1.516001   0.082754
df2
# b d   a
# 0 0.419201    -2.630736   -1.722084
# 1 -0.142221   -0.640445   -1.123706

#在这个示例中，传入ignore_index = True
pd.concat([df1, df2], ignore_index = False)
# a b   c   d
```

```
# 0 1.536883    0.838211    -0.025507   -0.215849
# 1 0.056875    -1.279308   -0.571636   1.111044
# 2 -0.676328   -1.273652   2.598412    -1.196276
# 0 -1.016712   -1.313193   NaN -0.067425
# 1 0.180768    0.656747    NaN -0.731031

pd.concat([df1, df2], ignore_index = True)
# a b   c   d
# 0 -0.161185   0.637754    -1.165464   1.061758
# 1 -0.312984   -0.811002   -0.989989   -0.202072
# 2 1.127441    -0.930106   1.279880    -1.656696
# 3 -0.181884   0.180735    NaN 1.210358
# 4 -0.806664   0.157353    NaN -0.782372
```

Out[329]:

	a	b	c	d
0	-0.214617	-0.922113	-0.660571	0.265220
1	-0.000708	0.079102	0.336035	-0.162124
2	1.008081	1.806378	-1.608740	1.806727
3	0.478874	-0.159042	NaN	-0.524383
4	0.081622	0.880130	NaN	0.464158

In [330]:

```
#concat函数的参数
# objs：需要连接的pandas对象列表或字典
# axis：连接的轴向，默认是0
# join：'inner' 或 'outer'
# join_axes：指定其他 n-1 轴的特定索引，可以替代内/外连接的逻辑
# keys：与要连接的对象关联的值，沿着连接轴行成分层索引
# levels：键值传递，用于指定多层索引的层级
# names：用于多层次索引的层级名称
# verify_integrity：检查连接对象中的新轴是否重复
# ignore_index：不沿着连接轴保留索引，而产生一段新的索引（长度为total_length）
```

8.2.4 联合重叠数据 Combining Data with Overlap

In [331]:

```
a = pd.Series([np.nan, 2.5, 0.0, 3.5, 4.5, np.nan],
              index = ['f', 'e', 'd', 'c', 'b', 'a'])
b = pd.Series([0., np.nan, 2., np.nan, np.nan, 5.],
              index = ['a', 'b', 'c', 'd', 'e', 'f'])
a
# f     NaN
# e     2.5
# d     0.0
# c     3.5
# b     4.5
# a     NaN
# dtype: float64
b
# a     0.0
# b     NaN
# c     2.0
# d     NaN
# e     NaN
# f     5.0
# dtype: float64
np.where(pd.isnull(a), b, a)
# array([0. , 2.5, 0. , 3.5, 4.5, 5. ])

#np.where(pd.isnull(a), b, a) 这行代码的含义是，当数组 a 中的元素为缺失值（NaN）时，将其替换为数组 b 中对应位置的值，否则保持原始值。
```

Out[331]: array([0. , 2.5, 0. , 3.5, 4.5, 5.])

8.3 重塑和透视 Reshaping and Pivoting

In [332]:

```
#重排列表格型数据，称为重塑或透视。
```

8.3.1 使用多层索引进行重塑 Reshaping with Hierarchical Indexing

In [333]:
```
#多层索引在data frame中提供了一种一致性方式用于重排列数据。
#statck #堆叠
#unstack #拆堆

#考虑一个带有字符串数组作为行和列索引的小型data frame:
data = pd.DataFrame(np.arange(6).reshape((2,3)),
                    index = pd.Index(['Ohio', 'Colorado'], name = 'state'),
                    columns = pd.Index(['one', 'two', 'three'],
                    name = 'number'))
data
# number    one two three
# state
# Ohio  0   1   2
# Colorado  3   4   5

result = data.stack()
result
# state     number
# Ohio      one       0
#           two       1
#           three     2
# Colorado  one       3
#           two       4
#           three     5
# dtype: int32

#在这份数据上使用stack方法会将列透视到行，产生一个新的series:
result = data.stack()
result
# state     number
# Ohio      one       0
#           two       1
#           three     2
# Colorado  one       3
#           two       4
#           three     5
# dtype: int32

#从一个多层索引序列中，可以使用unstack方法数据重排列后，放入一个data frame中:
result.unstack()
# number    one two three
# state
# Ohio  0   1   2
# Colorado  3   4   5

#对结果进行行列转置或重塑操作，将索引为0的级别转换为列，并重新组织数据框的结构。
#unstack就是把行转换为列:
result.unstack(0)                                  #最内层是已拆堆的，可以传入一个层级序号或名称来拆分一个不同的层级:
# state Ohio    Colorado
# number
# one   0   3
# two   1   4
# three 2   5
```

Out[333]:

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

In [334]:
```
result.unstack('state')
# state Ohio    Colorado
# number
# one    0    3
# two    1    4
# three 2    5
```

Out[334]:

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

In [335]:
```
#如果层级中的所有值未包含于每个子分组中，拆分可能会引入缺失值：
s1 = pd.Series([0, 1, 2, 3], index = ['a', 'b', 'c', 'd'])
s2 = pd.Series([4, 5, 6], index = ['c', 'd', 'e'])
data2 = pd.concat([s1, s2], keys = ['one', 'two'])
# data2
# one  a    0
#      b    1
#      c    2
#      d    3
# two  c    4
#      d    5
#      e    6
# dtype: int64

data2.unstack()
# a b   c   d   e
# one   0.0 1.0 2.0 3.0 NaN
# two   NaN NaN 4.0 5.0 6.0

data2.unstack().stack()
# 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

data2.unstack().stack(dropna = False)
# 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
```

Out[335]:
```
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
```

In [336]:
```
#当在data frame中拆堆时，被拆堆的层级会变为结果中最低的层级：

df = pd.DataFrame({'left': result, 'right': result + 5},
                  columns = pd.Index(['left','right'], name = 'side'))
df
# 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
df.unstack('state')
# side  left    right
# state Ohio    Colorado    Ohio    Colorado
# number
# one   0   3   5   8
# two   1   4   6   9
# three 2   5   7   10

#在调用stack方法时，可以指明需要堆叠的轴向名称：
df.unstack('state').stack('side')
# 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
```

Out[336]:

	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

8.3.2 将“长”透视为“宽” Pivoting “Long” to “Wide” Format

In [337]:

```
#将数据库和csv中存储多时间序列的方式就是所谓的长格式或堆叠格式。
#现在做少量时间序列规整和其他数据清洗操作:

data = pd.read_csv('C:/Users/miran/lpthw/macrodata.csv')
data.head()
# year  quarter realgdp realcons    realinv realgovt    realdpi cpi m1  tbilrate    unemp   pop infl    realint
# 0 1959    1   2710.349    1707.4  286.898 470.045 1886.9  28.98   139.7   2.82    5.8 177.146 0.00    0.00
# 1 1959    2   2778.801    1733.7  310.859 481.301 1919.7  29.15   141.7   3.08    5.1 177.830 2.34    0.74
# 2 1959    3   2775.488    1751.8  289.226 491.260 1916.4  29.35   140.5   3.82    5.3 178.657 2.74    1.09
# 3 1959    4   2785.204    1753.7  299.356 484.052 1931.3  29.37   140.0   4.33    5.6 179.386 0.27    4.06
# 4 1960    1   2847.699    1770.5  331.722 462.199 1955.5  29.54   139.6   3.50    5.2 180.007 2.31    1.19

#PeriodIndex将year和quarter进行联合，生成了一种时间间隔类型:
#将列名:realgdp, infl, unemp PIVOT为了横向的行
#FutureWarning: In a future version of pandas all arguments of DataFrame.pivot will be keyword-only.【? 】

periods = pd.PeriodIndex(year = data.year, quarter = data.quarter, name = 'date')
columns = pd.Index(['realgdp', 'infl', 'unemp'], name = 'item')
data.index = periods.to_timestamp('D', 'end')
ldata = data.stack().reset_index().rename(columns = {0: 'value'})

ldata[:10]
# date  level_1 value
# 0 1959-03-31 23:59:59.999999999   year    1959.000
# 1 1959-03-31 23:59:59.999999999   quarter 1.000
# 2 1959-03-31 23:59:59.999999999   realgdp 2710.349
# 3 1959-03-31 23:59:59.999999999   realcons    1707.400
# 4 1959-03-31 23:59:59.999999999   realinv 286.898
# 5 1959-03-31 23:59:59.999999999   realgovt    470.045
# 6 1959-03-31 23:59:59.999999999   realdpi 1886.900
# 7 1959-03-31 23:59:59.999999999   cpi 28.980
# 8 1959-03-31 23:59:59.999999999   m1  139.700
# 9 1959-03-31 23:59:59.999999999   tbilrate    2.820
```

Out[337]:

	date	level_1	value
0	1959-03-31 23:59:59.999999999	year	1959.000
1	1959-03-31 23:59:59.999999999	quarter	1.000
2	1959-03-31 23:59:59.999999999	realgdp	2710.349
3	1959-03-31 23:59:59.999999999	realcons	1707.400
4	1959-03-31 23:59:59.999999999	realinv	286.898
5	1959-03-31 23:59:59.999999999	realgovt	470.045
6	1959-03-31 23:59:59.999999999	realdpi	1886.900
7	1959-03-31 23:59:59.999999999	cpi	28.980
8	1959-03-31 23:59:59.999999999	m1	139.700
9	1959-03-31 23:59:59.999999999	tbilrate	2.820

In [338]:

```
#这种数据即所谓的多时间序列的长格式，或称为具有两个或更多个键的其他观测数据。
#表中的每一行表示一个时间点上的单个观测值。
pivoted = ldata.pivot('date', 'level_1', 'value')
pivoted

#传递的前2个值，是分别用作行和列索引的列，然后是可选的数值列以填充data frame.
#假设有2个数值列，想同时进行重塑：
ldata['value2'] = np.random.randn(len(ldata))
ldata[:10]
# date  level_1 value   value2
# 0 1959-03-31 23:59:59.999999999   year    1959.000    0.812289
# 1 1959-03-31 23:59:59.999999999   quarter 1.000   -0.125265
# 2 1959-03-31 23:59:59.999999999   realgdp 2710.349    0.719325
# 3 1959-03-31 23:59:59.999999999   realcons    1707.400    1.206542
# 4 1959-03-31 23:59:59.999999999   realinv 286.898 0.330828
# 5 1959-03-31 23:59:59.999999999   realgovt    470.045 -0.070886
# 6 1959-03-31 23:59:59.999999999   realdpi 1886.900    1.539682
# 7 1959-03-31 23:59:59.999999999   cpi 28.980  1.185263
# 8 1959-03-31 23:59:59.999999999   m1  139.700 -0.123213
# 9 1959-03-31 23:59:59.999999999   tbilrate    2.820   -0.020939

#如果遗漏最后一个参数，会得到一个含有多层列的data frame:
pivoted = ldata.pivot('date', 'level_1')
pivoted[:5]
#   value   ... value2
# level_1   cpi infl    m1  pop quarter realcons    realdpi realgdp realgovt    realint ... quarter realcons    realdpi real
# date
# 1959-03-31 23:59:59.999999999 28.98   0.00    139.7   177.146 1.0 1707.4  1886.9  2710.349    470.045 0.00    ... -1.66484
# 1959-06-30 23:59:59.999999999 29.15   2.34    141.7   177.830 2.0 1733.7  1919.7  2778.801    481.301 0.74    ... -1.93054
# 1959-09-30 23:59:59.999999999 29.35   2.74    140.5   178.657 3.0 1751.8  1916.4  2775.488    491.260 1.09    ... 2.521226
# 1959-12-31 23:59:59.999999999 29.37   0.27    140.0   179.386 4.0 1753.7  1931.3  2785.204    484.052 4.06    ... 0.044842
# 1960-03-31 23:59:59.999999999 29.54   2.31    139.6   180.007 1.0 1770.5  1955.5  2847.699    462.199 1.19    ... 0.819116
# 5 rows × 28 columns
```

```
C:\Users\miran\AppData\Local\Temp\ipykernel_76136\591368657.py:3: FutureWarning: In a future version of pandas all argument
s of DataFrame.pivot will be keyword-only.
  pivoted = ldata.pivot('date', 'level_1', 'value')
C:\Users\miran\AppData\Local\Temp\ipykernel_76136\591368657.py:23: FutureWarning: In a future version of pandas all argumen
ts of DataFrame.pivot will be keyword-only.
  pivoted = ldata.pivot('date', 'level_1')
```

Out[338]:

	value										...	value2				
level_1	cpi	infl	m1	pop	quarter	realcons	realdpi	realgdp	realgovt	realint	...	quarter	realcons	realdpi	realgdp	realg
date																
1959-03-31 23:59:59.999999999	28.98	0.00	139.7	177.146	1.0	1707.4	1886.9	2710.349	470.045	0.00	...	0.700132	-0.410415	0.378186	-0.617162	0.28
1959-06-30 23:59:59.999999999	29.15	2.34	141.7	177.830	2.0	1733.7	1919.7	2778.801	481.301	0.74	...	-0.985043	-1.255267	0.689742	-1.130173	-1.11
1959-09-30 23:59:59.999999999	29.35	2.74	140.5	178.657	3.0	1751.8	1916.4	2775.488	491.260	1.09	...	-0.435432	0.885687	-0.156177	0.967569	-0.77
1959-12-31 23:59:59.999999999	29.37	0.27	140.0	179.386	4.0	1753.7	1931.3	2785.204	484.052	4.06	...	0.319396	-1.220671	-0.095110	-0.428982	0.97
1960-03-31 23:59:59.999999999	29.54	2.31	139.6	180.007	1.0	1770.5	1955.5	2847.699	462.199	1.19	...	0.270213	0.630176	0.753995	-0.320868	-0.62

5 rows × 28 columns

In [339]:
```
pivoted['value'][:5]
# level_1   cpi infl    m1  pop quarter realcons    realdpi realgdp realgovt    realint realinv tbilrate    unemp   year
# date
# 1959-03-31 23:59:59.999999999 28.98   0.00    139.7   177.146 1.0 1707.4  1886.9  2710.349    470.045 0.00    286.898 2.82
# 1959-06-30 23:59:59.999999999 29.15   2.34    141.7   177.830 2.0 1733.7  1919.7  2778.801    481.301 0.74    310.859 3.08
# 1959-09-30 23:59:59.999999999 29.35   2.74    140.5   178.657 3.0 1751.8  1916.4  2775.488    491.260 1.09    289.226 3.82
# 1959-12-31 23:59:59.999999999 29.37   0.27    140.0   179.386 4.0 1753.7  1931.3  2785.204    484.052 4.06    299.356 4.33
# 1960-03-31 23:59:59.999999999 29.54   2.31    139.6   180.007 1.0 1770.5  1955.5  2847.699    462.199 1.19    331.722 3.50

#pivot方法等价于使用set_index创建分层索引，然后调用unstack:
unstacked = ldata.set_index(['date', 'level_1']).unstack('level_1')
unstacked[:7]
#   value   ... value2
# level_1   cpi infl    m1  pop quarter realcons    realdpi realgdp realgovt    realint ... quarter realcons    realdpi real
# date
# 1959-03-31 23:59:59.999999999 28.98   0.00    139.7   177.146 1.0 1707.4  1886.9  2710.349    470.045 0.00    ... -0.08970
# 1959-06-30 23:59:59.999999999 29.15   2.34    141.7   177.830 2.0 1733.7  1919.7  2778.801    481.301 0.74    ... -0.56767
# 1959-09-30 23:59:59.999999999 29.35   2.74    140.5   178.657 3.0 1751.8  1916.4  2775.488    491.260 1.09    ... -1.08149
# 1959-12-31 23:59:59.999999999 29.37   0.27    140.0   179.386 4.0 1753.7  1931.3  2785.204    484.052 4.06    ... -0.74608
# 1960-03-31 23:59:59.999999999 29.54   2.31    139.6   180.007 1.0 1770.5  1955.5  2847.699    462.199 1.19    ... -0.10508
# 1960-06-30 23:59:59.999999999 29.55   0.14    140.2   180.671 2.0 1792.9  1966.1  2834.390    460.400 2.55    ... 2.047081
# 1960-09-30 23:59:59.999999999 29.75   2.70    140.9   181.528 3.0 1785.8  1967.8  2839.022    474.676 -0.34   ... 0.251499
# 7 rows × 28 columns
```

Out[339]:

	value										...	value2				
level_1	**cpi**	**infl**	**m1**	**pop**	**quarter**	**realcons**	**realdpi**	**realgdp**	**realgovt**	**realint**	**...**	**quarter**	**realcons**	**realdpi**	**realgdp**	**realg**
date																
1959-03-31 23:59:59.999999999	28.98	0.00	139.7	177.146	1.0	1707.4	1886.9	2710.349	470.045	0.00	...	0.700132	-0.410415	0.378186	-0.617162	0.28
1959-06-30 23:59:59.999999999	29.15	2.34	141.7	177.830	2.0	1733.7	1919.7	2778.801	481.301	0.74	...	-0.985043	-1.255267	0.689742	-1.130173	-1.11
1959-09-30 23:59:59.999999999	29.35	2.74	140.5	178.657	3.0	1751.8	1916.4	2775.488	491.260	1.09	...	-0.435432	0.885687	-0.156177	0.967569	-0.77
1959-12-31 23:59:59.999999999	29.37	0.27	140.0	179.386	4.0	1753.7	1931.3	2785.204	484.052	4.06	...	0.319396	-1.220671	-0.095110	-0.428982	0.97
1960-03-31 23:59:59.999999999	29.54	2.31	139.6	180.007	1.0	1770.5	1955.5	2847.699	462.199	1.19	...	0.270213	0.630176	0.753995	-0.320868	-0.62
1960-06-30 23:59:59.999999999	29.55	0.14	140.2	180.671	2.0	1792.9	1966.1	2834.390	460.400	2.55	...	0.113213	-0.669722	0.967928	-1.184810	-0.31
1960-09-30 23:59:59.999999999	29.75	2.70	140.9	181.528	3.0	1785.8	1967.8	2839.022	474.676	-0.34	...	1.346063	-1.151493	-0.751452	-0.525326	-1.02

7 rows × 28 columns

8.3.3 将“宽”透视为“长” 使用多层索引进行重塑 Pivoting “Wide” to “Long” Format

In [340]:
```
#在data frame中，pivot方法的反操作是pandas.melt。
#它将多列合并成一列，产生一个新的data frame，长度比输入更长。

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

#key列可以作为分组指标，其他列均为数据值。当使用pandas.melt时，必须指明那些列是分组指标。
#让我们使用key作为唯一的分组指标：
melted = pd.melt(df, ['key'])
melted
# 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
# 6 foo C   7
# 7 bar C   8
# 8 baz C   9
```

Out[340]:

	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
6	foo	C	7
7	bar	C	8
8	baz	C	9

In [341]:
```
# pivot方法，可以将数据重塑回原先的布局：
reshaped = melted.pivot('key', 'variable', 'value')
reshaped
# variable  A   B   C
# key
# bar   2   5   8
# baz   3   6   9
# foo   1   4   7

# 由于pivot的结果根据作为行标签的列生成了索引，可能会想要使用reset_index来将数据回移一列：
# 在表的最前面一列加上index 0-2
reshaped.reset_index()
# variable  key A   B   C
# 0 bar 2   5   8
# 1 baz 3   6   9
# 2 foo 1   4   7

#也可以指定列的子集，作为值列：
pd.melt(df, id_vars = ['key'], value_vars = ['A', 'B'])

#pandas.melt也可以无需分组指标：
pd.melt(df, value_vars = ['A', 'B', 'C'])
#   variable    value
# 0 A   1
# 1 A   2
# 2 A   3
# 3 B   4
# 4 B   5
# 5 B   6
# 6 C   7
# 7 C   8
# 8 C   9
pd.melt(df, value_vars = ['key', 'A', 'B'])
# variable  value
# 0 key foo
# 1 key bar
# 2 key baz
# 3 A   1
# 4 A   2
# 5 A   3
# 6 B   4
# 7 B   5
# 8 B   6
```

```
C:\Users\miran\AppData\Local\Temp\ipykernel_76136\1311024960.py:2: FutureWarning: In a future version of pandas all argumen
ts of DataFrame.pivot will be keyword-only.
  reshaped = melted.pivot('key', 'variable', 'value')
```

Out[341]:

	variable	value
0	key	foo
1	key	bar
2	key	baz
3	A	1
4	A	2
5	A	3
6	B	4
7	B	5
8	B	6

8.4 小结 Conclusion

In [342]:
```
#pandas基础知识V; 数据导入, 清洗, 重组 -> matplotlib可视化
```