对数据集进行分组并对各组应用一个函数（无论是聚合还是转换），通常是数据分析工作中的重要环节。在将数据集加载、融合、准备好之后，通常就是计算分组统计或生成透视表。pandas提供了一个灵活高效的gruopby功能，它使你能以一种自然的方式对数据集进行切片、切块、摘要等操作。

关系型数据库和SQL（Structured Query Language，结构化查询语言）能够如此流行的原因之一就是其能够方便地对数据进行连接、过滤、转换和聚合。但是，像SQL这样的查询语言所能执行的分组运算的种类很有限。在本章中你将会看到，由于Python和pandas强大的表达能力，我们可以执行复杂得多的分组运算（利用任何可以接受pandas对象或NumPy数组的函数）。在本章中，你将会学到：

- 使用一个或多个键（形式可以是函数、数组或DataFrame列名）分割pandas对象。
- 计算分组的概述统计，比如数量、平均值或标准差，或是用户定义的函数。
- 应用组内转换或其他运算，如规格化、线性回归、排名或选取子集等。
- 计算透视表或交叉表。
- 执行分位数分析以及其它统计分组分析。

笔记：对时间序列数据的聚合（groupby的特殊用法之一）也称作重采样（resampling），本书将在第11章中单独对其进行讲解。

10.1 GroupBy机制

Hadley Wickham（许多热门R语言包的作者）创造了一个用于表示分组运算的术语“split-apply-combine”（拆分－应用－合并）。第一个阶段，pandas对象（无论是Series、DataFrame还是其他的）中的数据会根据你所提供的一个或多个键被拆分（split）为多组。拆分操作是在对象的特定轴上执行的。例如，DataFrame可以在其行（axis=0）或列（axis=1）上进行分组。然后，将一个函数应用（apply）到各个分组并产生一个新值。最后，所有这些函数的执行结果会被合并（combine）到最终的结果对象中。结果对象的形式一般取决于数据上所执行的操作。图10-1大致说明了一个简单的分组聚合过程。

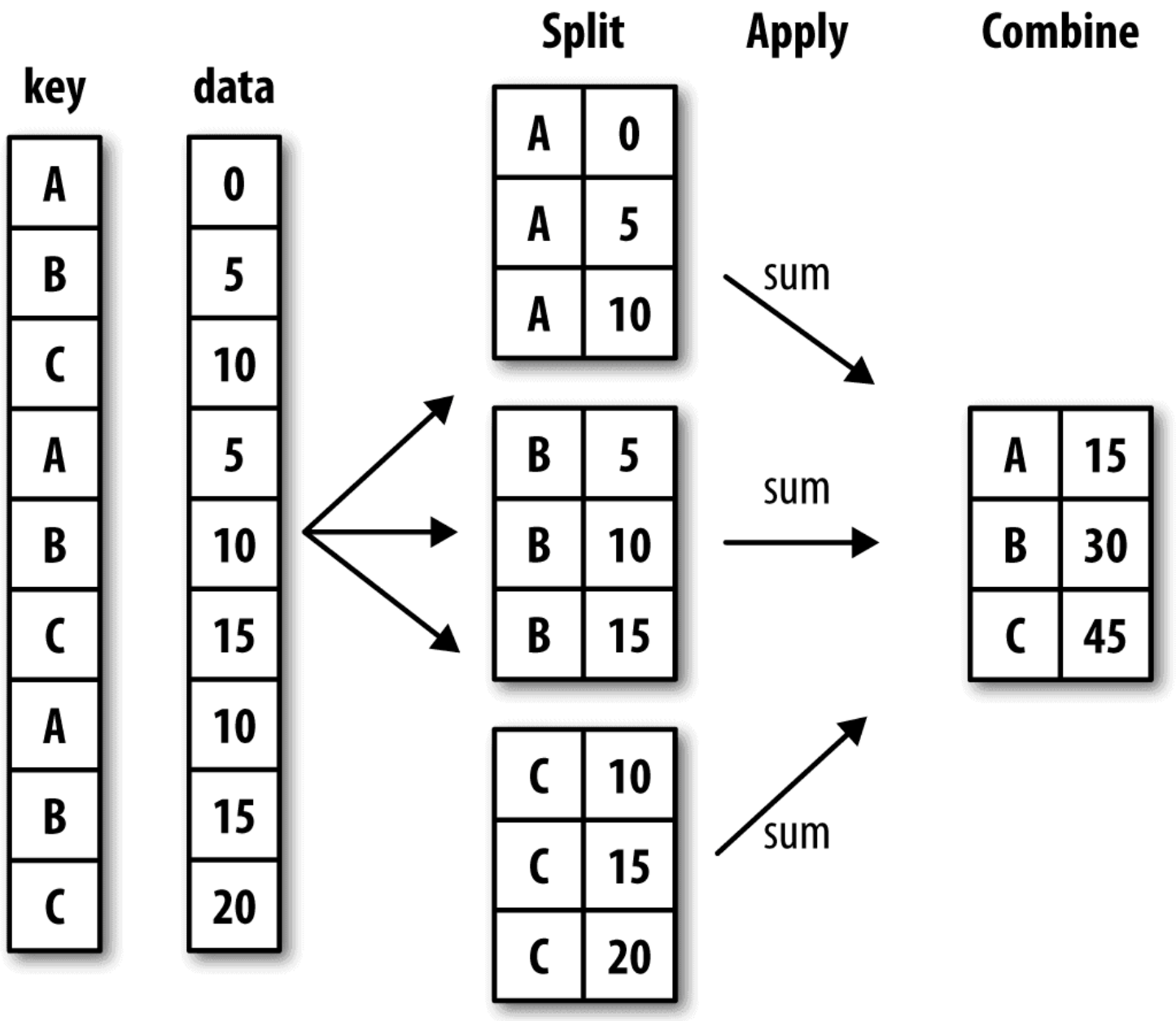

图10-1 分组聚合演示

分组键可以有多种形式，且类型不必相同：

- 列表或数组，其长度与待分组的轴一样。
- 表示DataFrame某个列名的值。
- 字典或Series，给出待分组轴上的值与分组名之间的对应关系。
- 函数，用于处理轴索引或索引中的各个标签。

注意，后三种都只是快捷方式而已，其最终目的仍然是产生一组用于拆分对象的值。如果觉得这些东西看起来很抽象，不用担心，我将在本章中给出大量有关于此的示例。首先来看看下面这个非常简单的表格型数据集（以DataFrame的形式）：

```
In [10]: df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
   ....:                    'key2' : ['one', 'two', 'one', 'two', 'one'],
   ....:                    'data1' : np.random.randn(5),
   ....:                    'data2' : np.random.randn(5)})

In [11]: df
Out[11]:
      data1     data2 key1 key2
0 -0.204708  1.393406    a  one
1  0.478943  0.092908    a  two
2 -0.519439  0.281746    b  one
3 -0.555730  0.769023    b  two
4  1.965781  1.246435    a  one
```

假设你想要按key1进行分组，并计算data1列的平均值。实现该功能的方式有很多，而我们这里要用的是：访问data1，并根据key1调用groupby：

```
In [12]: grouped = df['data1'].groupby(df['key1'])

In [13]: grouped
Out[13]: <pandas.core.groupby.SeriesGroupBy object at 0x7faa31537390>
```

变量grouped是一个GroupBy对象。它实际上还没有进行任何计算，只是含有一些有关分组键df['key1']的中间数据而已。换句话说，该对象已经有了接下来对各分组执行运算所需的一切信息。例如，我们可以调用GroupBy的mean方法来计算分组平均值：

```
In [14]: grouped.mean()
Out[14]:
key1
a    0.746672
b   -0.537585
Name: data1, dtype: float64
```

稍后我将详细讲解.mean()的调用过程。这里最重要的是，数据（Series）根据分组键进行了聚合，产生了一个新的Series，其索引为key1列中的唯一值。之所以结果中索引的名称为key1，是因为原始DataFrame的列df['key1']就叫这个名字。

如果我们一次传入多个数组的列表，就会得到不同的结果：

```
In [15]: means = df['data1'].groupby([df['key1'], df['key2']]).mean()

In [16]: means
Out[16]:
key1  key2
a     one     0.880536
      two     0.478943
b     one    -0.519439
      two    -0.555730
Name: data1, dtype: float64
```

这里，我通过两个键对数据进行了分组，得到的Series具有一个层次化索引（由唯一的键对组成）：

```
In [17]: means.unstack()
Out[17]:
key2       one       two
key1
a     0.880536  0.478943
b    -0.519439 -0.555730
```

在这个例子中，分组键均为Series。实际上，分组键可以是任何长度适当的数组：

```
In [18]: states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])

In [19]: years = np.array([2005, 2005, 2006, 2005, 2006])

In [20]: df['data1'].groupby([states, years]).mean()
Out[20]:
California  2005    0.478943
            2006   -0.519439
Ohio        2005   -0.380219
            2006    1.965781
Name: data1, dtype: float64
```

通常，分组信息就位于相同的要处理DataFrame中。这里，你还可以将列名（可以是字符串、数字或其他Python对象）用作分组键：

```
In [21]: df.groupby('key1').mean()
Out[21]:
         data1     data2
key1
a     0.746672  0.910916
b    -0.537585  0.525384

In [22]: df.groupby(['key1', 'key2']).mean()
Out[22]:
              data1     data2
key1 key2
a    one   0.880536  1.319920
     two   0.478943  0.092908
b    one  -0.519439  0.281746
     two  -0.555730  0.769023
```

你可能已经注意到了，第一个例子在执行df.groupby('key1').mean()时，结果中没有key2列。这是因为df['key2']不是数值数据（俗称"麻烦列"），所以被从结果中排除了。默认情况下，所有数值列都会被聚合，虽然有时可能会被过滤为一个子集，稍后就会碰到。

无论你准备拿groupby做什么，都有可能会用到GroupBy的size方法，它可以返回一个含有分组大小的Series：

```
In [23]: df.groupby(['key1', 'key2']).size()
Out[23]:
key1  key2
a     one     2
      two     1
b     one     1
      two     1
dtype: int64
```

注意，任何分组关键词中的缺失值，都会被从结果中除去。

对**分**组进**行迭代**

GroupBy对象支持迭代，可以产生一组二元元组（由分组名和数据块组成）。看下面的例子：

```
In [24]: for name, group in df.groupby('key1'):
   ....:     print(name)
   ....:     print(group)
   ....:
a
      data1     data2 key1 key2
0 -0.204708  1.393406    a  one
1  0.478943  0.092908    a  two
4  1.965781  1.246435    a  one
b
      data1     data2 key1 key2
2 -0.519439  0.281746    b  one
3 -0.555730  0.769023    b  two
```

对于多重键的情况，元组的第一个元素将会是由键值组成的元组：

```
In [25]: for (k1, k2), group in df.groupby(['key1', 'key2']):
   ....:     print((k1, k2))
   ....:     print(group)
   ....:
('a', 'one')
      data1     data2 key1 key2
0 -0.204708  1.393406    a  one
4  1.965781  1.246435    a  one
('a', 'two')
      data1     data2 key1 key2
1  0.478943  0.092908    a  two
('b', 'one')
      data1     data2 key1 key2
2 -0.519439  0.281746    b  one
('b', 'two')
     data1     data2 key1 key2
3 -0.55573  0.769023    b  two
```

当然，你可以对这些数据片段做任何操作。有一个你可能会觉得有用的运算：将这些数据片段做成一个字典：

```
In [26]: pieces = dict(list(df.groupby('key1')))

In [27]: pieces['b']
Out[27]:
      data1     data2 key1 key2
2 -0.519439  0.281746    b  one
3 -0.555730  0.769023    b  two
```

groupby默认是在axis=0上进行分组的，通过设置也可以在其他任何轴上进行分组。拿上面例子中的df来说，我们可以根据dtype对列进行分组：

```
In [28]: df.dtypes
Out[28]:
data1    float64
data2    float64
key1      object
key2      object
dtype: object

In [29]: grouped = df.groupby(df.dtypes, axis=1)
```

可以如下打印分组：

```
In [30]: for dtype, group in grouped:
   ....:     print(dtype)
   ....:     print(group)
   ....:
float64
      data1     data2
0 -0.204708  1.393406
1  0.478943  0.092908
2 -0.519439  0.281746
3 -0.555730  0.769023
4  1.965781  1.246435
object
  key1 key2
0    a  one
1    a  two
2    b  one
3    b  two
4    a  one
```

选**取一列或列的子集**

对于由DataFrame产生的GroupBy对象，如果用一个（单个字符串）或一组（字符串数组）列名对其进行索引，就能实现选取部分列进行聚合的目的。也就是说：

```
df.groupby('key1')['data1']
df.groupby('key1')[['data2']]
```

是以下代码的语法糖：

```
df['data1'].groupby(df['key1'])
df[['data2']].groupby(df['key1'])
```

尤其对于大数据集，很可能只需要对部分列进行聚合。例如，在前面那个数据集中，如果只需计算data2列的平均值并以DataFrame形式得到结果，可以这样写：

```
In [31]: df.groupby(['key1', 'key2'])[['data2']].mean()
Out[31]:
              data2
key1 key2
a    one   1.319920
     two   0.092908
b    one   0.281746
     two   0.769023
```

这种索引操作所返回的对象是一个已分组的DataFrame（如果传入的是列表或数组）或已分组的Series（如果传入的是标量形式的单个列名）：

```
In [32]: s_grouped = df.groupby(['key1', 'key2'])['data2']

In [33]: s_grouped
Out[33]: <pandas.core.groupby.SeriesGroupBy object at 0x7faa30c78da0>

In [34]: s_grouped.mean()
Out[34]:
key1  key2
a     one     1.319920
      two     0.092908
b     one     0.281746
      two     0.769023
Name: data2, dtype: float64
```

##通过字典或Series进行分组 除数组以外，分组信息还可以其他形式存在。来看另一个示例DataFrame：

```
In [35]: people = pd.DataFrame(np.random.randn(5, 5),
   ....:                       columns=['a', 'b', 'c', 'd', 'e'],
   ....:                       index=['Joe', 'Steve', 'Wes', 'Jim', 'Travis'])

In [36]: people.iloc[2:3, [1, 2]] = np.nan # Add a few NA values

In [37]: people
Out[37]:
              a         b         c         d         e
Joe     1.007189 -1.296221  0.274992  0.228913  1.352917
Steve   0.886429 -2.001637 -0.371843  1.669025 -0.438570
Wes    -0.539741       NaN       NaN -1.021228 -0.577087
Jim     0.124121  0.302614  0.523772  0.000940  1.343810
Travis -0.713544 -0.831154 -2.370232 -1.860761 -0.860757
```

现在，假设已知列的分组关系，并希望根据分组计算列的和：

```
In [38]: mapping = {'a': 'red', 'b': 'red', 'c': 'blue',
   ....:            'd': 'blue', 'e': 'red', 'f' : 'orange'}
```

现在，你可以将这个字典传给groupby，来构造数组，但我们可以直接传递字典（我包含了键“f”来强调，存在未使用的分组键是可以的）：

```
In [39]: by_column = people.groupby(mapping, axis=1)

In [40]: by_column.sum()
Out[40]:
            blue       red
Joe     0.503905  1.063885
Steve   1.297183 -1.553778
Wes    -1.021228 -1.116829
Jim     0.524712  1.770545
Travis -4.230992 -2.405455
```

Series也有同样的功能，它可以被看做一个固定大小的映射：

```
In [41]: map_series = pd.Series(mapping)

In [42]: map_series
Out[42]:
a       red
b       red
c      blue
d      blue
e       red
f    orange
dtype: object

In [43]: people.groupby(map_series, axis=1).count()
Out[43]:
       blue  red
Joe       2    3
Steve     2    3
Wes       1    2
Jim       2    3
Travis    2    3
```

##通过函数进行分组 比起使用字典或Series，使用Python函数是一种更原生的方法定义分组映射。任何被当做分组键的函数都会在各个索引值上被调用一次，其返回值就会被用作分组名称。具体点说，以上一小节的示例DataFrame为例，其索引值为人的名字。你可以计算一个字符串长度的数组，更简单的方法是传入len函数：

```
In [44]: people.groupby(len).sum()
Out[44]:
          a         b         c         d         e
3  0.591569 -0.993608  0.798764 -0.791374  2.119639
5  0.886429 -2.001637 -0.371843  1.669025 -0.438570
6 -0.713544 -0.831154 -2.370232 -1.860761 -0.860757
```

将函数跟数组、列表、字典、Series混合使用也不是问题，因为任何东西在内部都会被转换为数组：

```
In [45]: key_list = ['one', 'one', 'one', 'two', 'two']

In [46]: people.groupby([len, key_list]).min()
Out[46]:
              a         b         c         d         e
3 one -0.539741 -1.296221  0.274992 -1.021228 -0.577087
  two  0.124121  0.302614  0.523772  0.000940  1.343810
5 one  0.886429 -2.001637 -0.371843  1.669025 -0.438570
6 two -0.713544 -0.831154 -2.370232 -1.860761 -0.860757
```

根据索引级别分组

层次化索引数据集最方便的地方就在于它能够根据轴索引的一个级别进行聚合：

```
In [47]: columns = pd.MultiIndex.from_arrays([['US', 'US', 'US', 'JP', 'JP'],
   ....:                                      [1, 3, 5, 1, 3]],
   ....:                                      names=['cty', 'tenor'])

In [48]: hier_df = pd.DataFrame(np.random.randn(4, 5), columns=columns)

In [49]: hier_df
Out[49]:
cty          US                            JP
tenor         1         3         5         1         3
0      0.560145 -1.265934  0.119827 -1.063512  0.332883
1     -2.359419 -0.199543 -1.541996 -0.970736 -1.307030
```

```
2      0.286350  0.377984 -0.753887  0.331286  1.349742
3      0.069877  0.246674 -0.011862  1.004812  1.327195
```

要根据级别分组，使用level关键字传递级别序号或名字：

```
In [50]: hier_df.groupby(level='cty', axis=1).count()
Out[50]:
cty  JP  US
0     2   3
1     2   3
2     2   3
3     2   3
```

10.2 数据聚合

聚合指的是任何能够从数组产生标量值的数据转换过程。之前的例子已经用过一些，比如mean、count、min以及sum等。你可能想知道在GroupBy对象上调用mean()时究竟发生了什么。许多常见的聚合运算（如表10-1所示）都有进行优化。然而，除了这些方法，你还可以使用其它的。

函数名	说明
count	分组中非NA值的数量
sum	非NA值的和
mean	非NA值的平均值
median	非NA值的算术中位数
std、var	无偏（分母为n - 1）标准差和方差
min、max	非NA值的最小值和最大值
prod	非NA值的积
first、last	第一个和最后一个非NA值

表10-1 经过优化的groupby方法

你可以使用自己发明的聚合运算，还可以调用分组对象上已经定义好的任何方法。例如，quantile可以计算Series或DataFrame列的样本分位数。

虽然quantile并没有明确地实现于GroupBy，但它是一个Series方法，所以这里是能用的。实际上，GroupBy会高效地对Series进行切片，然后对各片调用piece.quantile(0.9)，最后将这些结果组装成最终结果：

```
In [51]: df
Out[51]:
      data1     data2 key1 key2
0 -0.204708  1.393406    a  one
1  0.478943  0.092908    a  two
2 -0.519439  0.281746    b  one
3 -0.555730  0.769023    b  two
4  1.965781  1.246435    a  one

In [52]: grouped = df.groupby('key1')

In [53]: grouped['data1'].quantile(0.9)
Out[53]:
key1
a    1.668413
b   -0.523068
Name: data1, dtype: float64
```

如果要使用你自己的聚合函数，只需将其传入aggregate或agg方法即可：

```
In [54]: def peak_to_peak(arr):
   ....:     return arr.max() - arr.min()
In [55]: grouped.agg(peak_to_peak)
Out[55]:
         data1     data2
key1
a     2.170488  1.300498
b     0.036292  0.487276
```

你可能注意到注意，有些方法（如describe）也是可以用在这里的，即使严格来讲，它们并非聚合运算：

```
In [56]: grouped.describe()
Out[56]:
     data1                                                              \
     count      mean       std       min       25%       50%       75%
key1
a      3.0  0.746672  1.109736 -0.204708  0.137118  0.478943  1.222362
b      2.0 -0.537585  0.025662 -0.555730 -0.546657 -0.537585 -0.528512
             data2                                                      \
max count      mean       std       min       25%       50%
key1
a     1.965781   3.0  0.910916  0.712217  0.092908  0.669671  1.246435
b    -0.519439   2.0  0.525384  0.344556  0.281746  0.403565  0.525384

           75%       max
key1
a     1.319920  1.393406
b     0.647203  0.769023
```

在后面的10.3节，我将详细说明这到底是怎么回事。

> 笔记：自定义聚合函数要比表10-1中那些经过优化的函数慢得多。这是因为在构造中间分组数据块时存在非常大的开销（函数调用、数据重排等）。

面向列的多函数应用

回到前面小费的例子。使用read_csv导入数据之后，我们添加了一个小费百分比的列tip_pct：

```
In [57]: tips = pd.read_csv('examples/tips.csv')

# Add tip percentage of total bill
```

```
In [58]: tips['tip_pct'] = tips['tip'] / tips['total_bill']

In [59]: tips[:6]
Out[59]:
   total_bill   tip smoker  day    time  size   tip_pct
0       16.99  1.01     No  Sun  Dinner     2  0.059447
1       10.34  1.66     No  Sun  Dinner     3  0.160542
2       21.01  3.50     No  Sun  Dinner     3  0.166587
3       23.68  3.31     No  Sun  Dinner     2  0.139780
4       24.59  3.61     No  Sun  Dinner     4  0.146808
5       25.29  4.71     No  Sun  Dinner     4  0.186240
```

你已经看到，对Series或DataFrame列的聚合运算其实就是使用aggregate（使用自定义函数）或调用诸如mean、std之类的方法。然而，你可能希望对不同的列使用不同的聚合函数，或一次应用多个函数。其实这也好办，我将通过一些示例来进行讲解。首先，我根据天和smoker对tips进行分组：

```
In [60]: grouped = tips.groupby(['day', 'smoker'])
```

注意，对于表10-1中的那些描述统计，可以将函数名以字符串的形式传入：

```
In [61]: grouped_pct = grouped['tip_pct']

In [62]: grouped_pct.agg('mean')
Out[62]:
day   smoker
Fri   No        0.151650
      Yes       0.174783
Sat   No        0.158048
      Yes       0.147906
Sun   No        0.160113
      Yes       0.187250
Thur  No        0.160298
      Yes       0.163863
Name: tip_pct, dtype: float64
```

如果传入一组函数或函数名，得到的DataFrame的列就会以相应的函数命名：

```
In [63]: grouped_pct.agg(['mean', 'std', peak_to_peak])
Out[63]:
                mean       std  peak_to_peak
day  smoker
Fri  No      0.151650  0.028123      0.067349
     Yes     0.174783  0.051293      0.159925
Sat  No      0.158048  0.039767      0.235193
     Yes     0.147906  0.061375      0.290095
Sun  No      0.160113  0.042347      0.193226
     Yes     0.187250  0.154134      0.644685
Thur No      0.160298  0.038774      0.193350
     Yes     0.163863  0.039389      0.151240
```

这里，我们传递了一组聚合函数进行聚合，独立对数据分组进行评估。

你并非一定要接受GroupBy自动给出的那些列名，特别是lambda函数，它们的名称是‘’，这样的辨识度就很低了（通过函数的__name__属性看看就知道了）。因此，如果传入的是一个由(name,function)元组组成的列表，则各元组的第一个元素就会被用作DataFrame的列名（可以将这种二元元组列表看做一个有序映射）：

```
In [64]: grouped_pct.agg([('foo', 'mean'), ('bar', np.std)])
Out[64]:
                 foo       bar
day  smoker
Fri  No      0.151650  0.028123
     Yes     0.174783  0.051293
Sat  No      0.158048  0.039767
     Yes     0.147906  0.061375
Sun  No      0.160113  0.042347
     Yes     0.187250  0.154134
Thur No      0.160298  0.038774
     Yes     0.163863  0.039389
```

对于DataFrame，你还有更多选择，你可以定义一组应用于全部列的一组函数，或不同的列应用不同的函数。假设我们想要对tip_pct和total_bill列计算三个统计信息：

```
In [65]: functions = ['count', 'mean', 'max']

In [66]: result = grouped['tip_pct', 'total_bill'].agg(functions)

In [67]: result
Out[67]:
             tip_pct                     total_bill
               count      mean       max      count       mean    max
day  smoker
Fri  No            4  0.151650  0.187735          4  18.420000  22.75
     Yes          15  0.174783  0.263480         15  16.813333  40.17
Sat  No           45  0.158048  0.291990         45  19.661778  48.33
     Yes          42  0.147906  0.325733         42  21.276667  50.81
Sun  No           57  0.160113  0.252672         57  20.506667  48.17
     Yes          19  0.187250  0.710345         19  24.120000  45.35
Thur No           45  0.160298  0.266312         45  17.113111  41.19
     Yes          17  0.163863  0.241255         17  19.190588  43.11
```

如你所见，结果DataFrame拥有层次化的列，这相当于分别对各列进行聚合，然后用concat将结果组装到一起，使用列名用作keys参数：

```
In [68]: result['tip_pct']
Out[68]:
             count      mean       max
day  smoker
Fri  No          4  0.151650  0.187735
     Yes        15  0.174783  0.263480
Sat  No         45  0.158048  0.291990
     Yes        42  0.147906  0.325733
Sun  No         57  0.160113  0.252672
     Yes        19  0.187250  0.710345
Thur No         45  0.160298  0.266312
     Yes        17  0.163863  0.241255
```

跟前面一样，这里也可以传入带有自定义名称的一组元组：

```
In [69]: ftuples = [('Durchschnitt', 'mean'),('Abweichung', np.var)]

In [70]: grouped['tip_pct', 'total_bill'].agg(ftuples)
Out[70]:
               tip_pct              total_bill
```

```
            Durchschnitt Abweichung Durchschnitt  Abweichung
day  smoker
Fri  No         0.151650   0.000791    18.420000   25.596333
     Yes        0.174783   0.002631    16.813333   82.562438
Sat  No         0.158048   0.001581    19.661778   79.908965
     Yes        0.147906   0.003767    21.276667  101.387535
Sun  No         0.160113   0.001793    20.506667   66.099980
     Yes        0.187250   0.023757    24.120000  109.046044
Thur No         0.160298   0.001503    17.113111   59.625081
     Yes        0.163863   0.001551    19.190588   69.808518
```

现在，假设你想要对一个列或不同的列应用不同的函数。具体的办法是向agg传入一个从列名映射到函数的字典：

```
In [71]: grouped.agg({'tip' : np.max, 'size' : 'sum'})
Out[71]:
              tip  size
day  smoker
Fri  No       3.50     9
     Yes      4.73    31
Sat  No       9.00   115
     Yes     10.00   104
Sun  No       6.00   167
     Yes      6.50    49
Thur No       6.70   112
     Yes      5.00    40

In [72]: grouped.agg({'tip_pct' : ['min', 'max', 'mean', 'std'],
   ....:              'size' : 'sum'})
Out[72]:
             tip_pct                               size
                 min       max      mean       std  sum
day  smoker
Fri  No      0.120385  0.187735  0.151650  0.028123    9
     Yes     0.103555  0.263480  0.174783  0.051293   31
Sat  No      0.056797  0.291990  0.158048  0.039767  115
     Yes     0.035638  0.325733  0.147906  0.061375  104
Sun  No      0.059447  0.252672  0.160113  0.042347  167
     Yes     0.065660  0.710345  0.187250  0.154134   49
Thur No      0.072961  0.266312  0.160298  0.038774  112
     Yes     0.090014  0.241255  0.163863  0.039389   40
```

只有将多个函数应用到至少一列时，DataFrame才会拥有层次化的列。

以“没有行索引”的形式返回聚合数据

到目前为止，所有示例中的聚合数据都有由唯一的分组键组成的索引（可能还是层次化的）。由于并不总是需要如此，所以你可以向groupby传入as_index=False以禁用该功能：

```
In [73]: tips.groupby(['day', 'smoker'], as_index=False).mean()
Out[73]:
    day smoker  total_bill       tip      size   tip_pct
0   Fri     No   18.420000  2.812500  2.250000  0.151650
1   Fri    Yes   16.813333  2.714000  2.066667  0.174783
2   Sat     No   19.661778  3.102889  2.555556  0.158048
3   Sat    Yes   21.276667  2.875476  2.476190  0.147906
4   Sun     No   20.506667  3.167895  2.929825  0.160113
5   Sun    Yes   24.120000  3.516842  2.578947  0.187250
6  Thur     No   17.113111  2.673778  2.488889  0.160298
7  Thur    Yes   19.190588  3.030000  2.352941  0.163863
```

当然，对结果调用reset_index也能得到这种形式的结果。使用as_index=False方法可以避免一些不必要的计算。

10.3 apply：一般性的“拆分 – 应用 – 合并”

最通用的GroupBy方法是apply，本节剩余部分将重点讲解它。如图10-2所示，apply会将待处理的对象拆分成多个片段，然后对各片段调用传入的函数，最后尝试将各片段组合到一起。

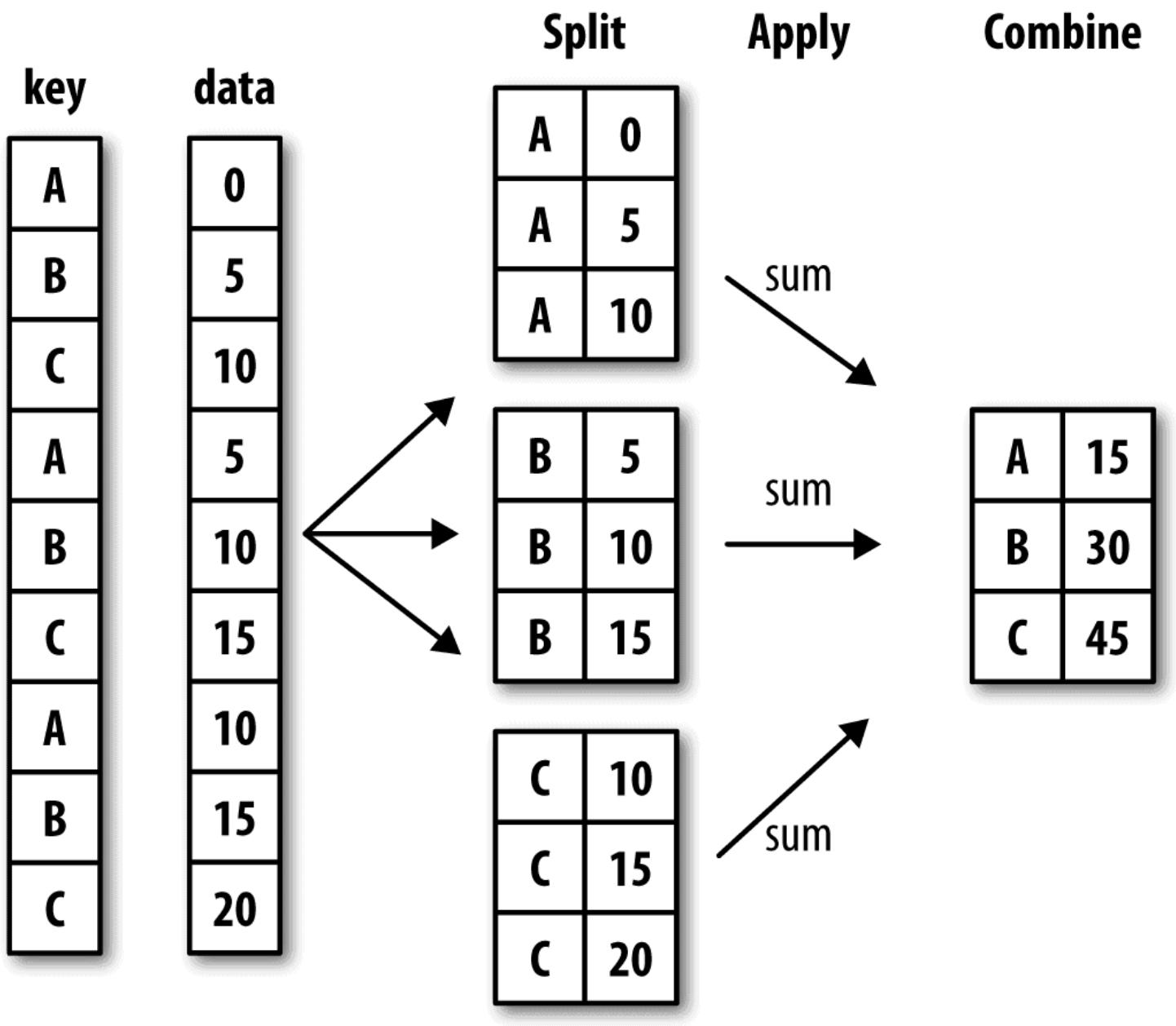

图10-2 分组聚合示例

回到之前那个小费数据集，假设你想要根据分组选出最高的5个tip_pct值。首先，编写一个选取指定列具有最大值的行的函数：

```
In [74]: def top(df, n=5, column='tip_pct'):
   ....:     return df.sort_values(by=column)[-n:]

In [75]: top(tips, n=6)
Out[75]:
     total_bill   tip smoker  day    time  size   tip_pct
109       14.31  4.00    Yes  Sat  Dinner     2  0.279525
183       23.17  6.50    Yes  Sun  Dinner     4  0.280535
232       11.61  3.39     No  Sat  Dinner     2  0.291990
67         3.07  1.00    Yes  Sat  Dinner     1  0.325733
178        9.60  4.00    Yes  Sun  Dinner     2  0.416667
172        7.25  5.15    Yes  Sun  Dinner     2  0.710345
```

现在，如果对smoker分组并用该函数调用apply，就会得到：

```
In [76]: tips.groupby('smoker').apply(top)
Out[76]:
            total_bill   tip smoker   day    time  size   tip_pct
smoker
No     88        24.71  5.85     No  Thur   Lunch     2  0.236746
       185       20.69  5.00     No   Sun  Dinner     5  0.241663
       51        10.29  2.60     No   Sun  Dinner     2  0.252672
       149        7.51  2.00     No  Thur   Lunch     2  0.266312
       232       11.61  3.39     No   Sat  Dinner     2  0.291990
Yes    109       14.31  4.00    Yes   Sat  Dinner     2  0.279525
       183       23.17  6.50    Yes   Sun  Dinner     4  0.280535
       67         3.07  1.00    Yes   Sat  Dinner     1  0.325733
       178        9.60  4.00    Yes   Sun  Dinner     2  0.416667
       172        7.25  5.15    Yes   Sun  Dinner     2  0.710345
```

这里发生了什么？top函数在DataFrame的各个片段上调用，然后结果由pandas.concat组装到一起，并以分组名称进行了标记。于是，最终结果就有了一个层次化索引，其内层索引值来自原DataFrame。

如果传给apply的函数能够接受其他参数或关键字，则可以将这些内容放在函数名后面一并传入：

```
In [77]: tips.groupby(['smoker', 'day']).apply(top, n=1, column='total_bill')
Out[77]:
                 total_bill    tip smoker   day    time  size   tip_pct
```

```
smoker day
No     Fri  94        22.75   3.25     No   Fri  Dinner     2  0.142857
       Sat  212       48.33   9.00     No   Sat  Dinner     4  0.186220
       Sun  156       48.17   5.00     No   Sun  Dinner     6  0.103799
       Thur 142       41.19   5.00     No  Thur   Lunch     5  0.121389
Yes    Fri  95        40.17   4.73    Yes   Fri  Dinner     4  0.117750
       Sat  170       50.81  10.00    Yes   Sat  Dinner     3  0.196812
       Sun  182       45.35   3.50    Yes   Sun  Dinner     3  0.077178
       Thur 197       43.11   5.00    Yes  Thur   Lunch     4  0.115982
```

笔记：除这些基本用法之外，能否充分发挥apply的威力很大程度上取决于你的创造力。传入的那个函数能做什么全由你说了算，它只需返回一个pandas对象或标量值即可。本章后续部分的示例主要用于讲解如何利用groupby解决各种各样的问题。

可能你已经想起来了，之前我在GroupBy对象上调用过describe：

```
In [78]: result = tips.groupby('smoker')['tip_pct'].describe()

In [79]: result
Out[79]:
        count      mean       std       min       25%       50%       75%  \
smoker
No      151.0  0.159328  0.039910  0.056797  0.136906  0.155625  0.185014
Yes      93.0  0.163196  0.085119  0.035638  0.106771  0.153846  0.195059
             max
smoker

No      0.291990
Yes     0.710345

In [80]: result.unstack('smoker')
Out[80]:
        smoker
count   No        151.000000
        Yes        93.000000
mean    No          0.159328
        Yes         0.163196
std     No          0.039910
        Yes         0.085119
min     No          0.056797
        Yes         0.035638
25%     No          0.136906
        Yes         0.106771
50%     No          0.155625
        Yes         0.153846
75%     No          0.185014
        Yes         0.195059
max     No          0.291990
        Yes         0.710345
dtype: float64
```

在GroupBy中，当你调用诸如describe之类的方法时，实际上只是应用了下面两条代码的快捷方式而已：

```
f = lambda x: x.describe()
grouped.apply(f)
```

禁止分组键

从上面的例子中可以看出，分组键会跟原始对象的索引共同构成结果对象中的层次化索引。将group_keys=False传入groupby即可禁止该效果：

```
In [81]: tips.groupby('smoker', group_keys=False).apply(top)
Out[81]:
     total_bill   tip smoker   day    time  size   tip_pct
88        24.71  5.85     No  Thur   Lunch     2  0.236746
185       20.69  5.00     No   Sun  Dinner     5  0.241663
51        10.29  2.60     No   Sun  Dinner     2  0.252672
149        7.51  2.00     No  Thur   Lunch     2  0.266312
232       11.61  3.39     No   Sat  Dinner     2  0.291990
109       14.31  4.00    Yes   Sat  Dinner     2  0.279525
183       23.17  6.50    Yes   Sun  Dinner     4  0.280535
67         3.07  1.00    Yes   Sat  Dinner     1  0.325733
178        9.60  4.00    Yes   Sun  Dinner     2  0.416667
172        7.25  5.15    Yes   Sun  Dinner     2  0.710345
```

分位数和桶分析

我曾在第8章中讲过，pandas有一些能根据指定面元或样本分位数将数据拆分成多块的工具（比如cut和qcut）。将这些函数跟groupby结合起来，就能非常轻松地实现对数据集的桶（bucket）或分位数（quantile）分析了。以下面这个简单的随机数据集为例，我们利用cut将其装入长度相等的桶中：

```
In [82]: frame = pd.DataFrame({'data1': np.random.randn(1000),
   ....:                       'data2': np.random.randn(1000)})

In [83]: quartiles = pd.cut(frame.data1, 4)

In [84]: quartiles[:10]
Out[84]:
0     (-1.23, 0.489]
1    (-2.956, -1.23]
2     (-1.23, 0.489]
3     (0.489, 2.208]
4     (-1.23, 0.489]
5     (0.489, 2.208]
6     (-1.23, 0.489]
7     (-1.23, 0.489]
8     (0.489, 2.208]
9     (0.489, 2.208]
Name: data1, dtype: category
Categories (4, interval[float64]): [(-2.956, -1.23] < (-1.23, 0.489] < (0.489, 2.
208] < (2.208, 3.928]]
```

由cut返回的Categorical对象可直接传递到groupby。因此，我们可以像下面这样对data2列做一些统计计算：

```
In [85]: def get_stats(group):
   ....:     return {'min': group.min(), 'max': group.max(),
   ....:             'count': group.count(), 'mean': group.mean()}

In [86]: grouped = frame.data2.groupby(quartiles)

In [87]: grouped.apply(get_stats).unstack()
Out[87]:
                count       max      mean       min
```

```
data1
(-2.956, -1.23]   95.0  1.670835 -0.039521 -3.399312
(-1.23, 0.489]   598.0  3.260383 -0.002051 -2.989741
(0.489, 2.208]   297.0  2.954439  0.081822 -3.745356
(2.208, 3.928]    10.0  1.765640  0.024750 -1.929776
```

这些都是长度相等的桶。要根据样本分位数得到大小相等的桶，使用qcut即可。传入labels=False即可只获取分位数的编号：

```
# Return quantile numbers
In [88]: grouping = pd.qcut(frame.data1, 10, labels=False)

In [89]: grouped = frame.data2.groupby(grouping)

In [90]: grouped.apply(get_stats).unstack()
Out[90]:
       count       max      mean       min
data1
0      100.0  1.670835 -0.049902 -3.399312
1      100.0  2.628441  0.030989 -1.950098
2      100.0  2.527939 -0.067179 -2.925113
3      100.0  3.260383  0.065713 -2.315555
4      100.0  2.074345 -0.111653 -2.047939
5      100.0  2.184810  0.052130 -2.989741
6      100.0  2.458842 -0.021489 -2.223506
7      100.0  2.954439 -0.026459 -3.056990
8      100.0  2.735527  0.103406 -3.745356
9      100.0  2.377020  0.220122 -2.064111
```

我们会在第12章详细讲解pandas的Categorical类型。

示例：用特定于分组的值填充缺失值

对于缺失数据的清理工作，有时你会用dropna将其替换掉，而有时则可能会希望用一个固定值或由数据集本身所衍生出来的值去填充NA值。这时就得使用fillna这个工具了。在下面这个例子中，我用平均值去填充NA值：

```
In [91]: s = pd.Series(np.random.randn(6))

In [92]: s[::2] = np.nan

In [93]: s
Out[93]:
0         NaN
1   -0.125921
2         NaN
3   -0.884475
4         NaN
5    0.227290
dtype: float64

In [94]: s.fillna(s.mean())
Out[94]:
0   -0.261035
1   -0.125921
2   -0.261035
3   -0.884475
4   -0.261035
5    0.227290
dtype: float64
```

假设你需要对不同的分组填充不同的值。一种方法是将数据分组，并使用apply和一个能够对各数据块调用fillna的函数即可。下面是一些有关美国几个州的示例数据，这些州又被分为东部和西部：

```
In [95]: states = ['Ohio', 'New York', 'Vermont', 'Florida',
   ....:           'Oregon', 'Nevada', 'California', 'Idaho']

In [96]: group_key = ['East'] * 4 + ['West'] * 4

In [97]: data = pd.Series(np.random.randn(8), index=states)

In [98]: data
Out[98]:
Ohio          0.922264
New York     -2.153545
Vermont      -0.365757
Florida      -0.375842
Oregon        0.329939
Nevada        0.981994
California    1.105913
Idaho        -1.613716
dtype: float64
```

['East'] * 4产生了一个列表，包括了['East']中元素的四个拷贝。将这些列表串联起来。

将一些值设为缺失：

```
In [99]: data[['Vermont', 'Nevada', 'Idaho']] = np.nan

In [100]: data
Out[100]:
Ohio          0.922264
New York     -2.153545
Vermont            NaN
Florida      -0.375842
Oregon        0.329939
Nevada             NaN
California    1.105913
Idaho              NaN
dtype: float64

In [101]: data.groupby(group_key).mean()
Out[101]:
East   -0.535707
West    0.717926
dtype: float64
```

我们可以用分组平均值去填充NA值:

```
In [102]: fill_mean = lambda g: g.fillna(g.mean())

In [103]: data.groupby(group_key).apply(fill_mean)
Out[103]:
Ohio          0.922264
```

```
New York     -2.153545
Vermont      -0.535707
Florida      -0.375842
Oregon        0.329939
Nevada        0.717926
California    1.105913
Idaho         0.717926
dtype: float64
```

另外，也可以在代码中预定义各组的填充值。由于分组具有一个name属性，所以我们可以拿来用一下：

```
In [104]: fill_values = {'East': 0.5, 'West': -1}

In [105]: fill_func = lambda g: g.fillna(fill_values[g.name])

In [106]: data.groupby(group_key).apply(fill_func)
Out[106]:
Ohio          0.922264
New York     -2.153545
Vermont       0.500000
Florida      -0.375842
Oregon        0.329939
Nevada       -1.000000
California    1.105913
Idaho        -1.000000
dtype: float64
```

示例：随机采样和排列

假设你想要从一个大数据集中随机抽取（进行替换或不替换）样本以进行蒙特卡罗模拟（Monte Carlo simulation）或其他分析工作。“抽取”的方式有很多，这里使用的方法是对Series使用sample方法：

```
# Hearts, Spades, Clubs, Diamonds
suits = ['H', 'S', 'C', 'D']
card_val = (list(range(1, 11)) + [10] * 3) * 4
base_names = ['A'] + list(range(2, 11)) + ['J', 'K', 'Q']
cards = []
for suit in ['H', 'S', 'C', 'D']:
    cards.extend(str(num) + suit for num in base_names)

deck = pd.Series(card_val, index=cards)
```

现在我有了一个长度为52的Series，其索引包括牌名，值则是21点或其他游戏中用于计分的点数（为了简单起见，我当A的点数为1）：

```
In [108]: deck[:13]
Out[108]:
AH      1
2H      2
3H      3
4H      4
5H      5
6H      6
7H      7
8H      8
9H      9
10H    10
JH     10
KH     10
QH     10
dtype: int64
```

现在，根据我上面所讲的，从整副牌中抽出5张，代码如下：

```
In [109]: def draw(deck, n=5):
   .....:     return deck.sample(n)

In [110]: draw(deck)
Out[110]:
AD     1
8C     8
5H     5
KC    10
2C     2
dtype: int64
```

假设你想要从每种花色中随机抽取两张牌。由于花色是牌名的最后一个字符，所以我们可以据此进行分组，并使用apply：

```
In [111]: get_suit = lambda card: card[-1] # last letter is suit

In [112]: deck.groupby(get_suit).apply(draw, n=2)
Out[112]:
C  2C     2
   3C     3
D  KD    10
   8D     8
H  KH    10
   3H     3
S  2S     2
   4S     4
dtype: int64
```

或者，也可以这样写：

```
In [113]: deck.groupby(get_suit, group_keys=False).apply(draw, n=2)
Out[113]:
KC    10
JC    10
AD     1
5D     5
5H     5
6H     6
7S     7
KS    10
dtype: int64
```

示例：分组加权平均数和相关系数

根据groupby的“拆分－应用－合并”范式，可以进行DataFrame的列与列之间或两个Series之间的运算（比如分组加权平均）。以下面这个数据集为例，它含有分组键、值以及一些权重值：

```
In [114]: df = pd.DataFrame({'category': ['a', 'a', 'a', 'a',
   .....:                                  'b', 'b', 'b', 'b'],
   .....:                    'data': np.random.randn(8),
   .....:                    'weights': np.random.rand(8)})

In [115]: df
Out[115]:
  category      data   weights
0        a  1.561587  0.957515
1        a  1.219984  0.347267
2        a -0.482239  0.581362
3        a  0.315667  0.217091
4        b -0.047852  0.894406
5        b -0.454145  0.918564
6        b -0.556774  0.277825
7        b  0.253321  0.955905
```

然后可以利用category计算分组加权平均数：

```
In [116]: grouped = df.groupby('category')

In [117]: get_wavg = lambda g: np.average(g['data'], weights=g['weights'])

In [118]: grouped.apply(get_wavg)
Out[118]:
category
a    0.811643
b   -0.122262
dtype: float64
```

另一个例子，考虑一个来自Yahoo!Finance的数据集，其中含有几只股票和标准普尔500指数（符号SPX）的收盘价：

```
In [119]: close_px = pd.read_csv('examples/stock_px_2.csv', parse_dates=True,
   .....:                        index_col=0)

In [120]: close_px.info()
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2214 entries, 2003-01-02 to 2011-10-14
Data columns (total 4 columns):
AAPL    2214 non-null float64
MSFT    2214 non-null float64
XOM     2214 non-null float64
SPX     2214 non-null float64
dtypes: float64(4)
memory usage: 86.5 KB

In [121]: close_px[-4:]
Out[121]:
              AAPL   MSFT    XOM      SPX
2011-10-11  400.29  27.00  76.27  1195.54
2011-10-12  402.19  26.96  77.16  1207.25
2011-10-13  408.43  27.18  76.37  1203.66
2011-10-14  422.00  27.27  78.11  1224.58
```

来做一个比较有趣的任务：计算一个由日收益率（通过百分数变化计算）与SPX之间的年度相关系数组成的DataFrame。下面是一个实现办法，我们先创建一个函数，用它计算每列和SPX列的成对相关系数：

```
In [122]: spx_corr = lambda x: x.corrwith(x['SPX'])
```

接下来，我们使用pct_change计算close_px的百分比变化：

```
In [123]: rets = close_px.pct_change().dropna()
```

最后，我们用年对百分比变化进行分组，可以用一个一行的函数，从每行的标签返回每个datetime标签的year属性：

```
In [124]: get_year = lambda x: x.year

In [125]: by_year = rets.groupby(get_year)

In [126]: by_year.apply(spx_corr)
Out[126]:
          AAPL      MSFT       XOM  SPX
2003  0.541124  0.745174  0.661265  1.0
2004  0.374283  0.588531  0.557742  1.0
2005  0.467540  0.562374  0.631010  1.0
2006  0.428267  0.406126  0.518514  1.0
2007  0.508118  0.658770  0.786264  1.0
2008  0.681434  0.804626  0.828303  1.0
2009  0.707103  0.654902  0.797921  1.0
2010  0.710105  0.730118  0.839057  1.0
2011  0.691931  0.800996  0.859975  1.0
```

当然，你还可以计算列与列之间的相关系数。这里，我们计算Apple和Microsoft的年相关系数：

```
In [127]: by_year.apply(lambda g: g['AAPL'].corr(g['MSFT']))
Out[127]:
2003    0.480868
2004    0.259024
2005    0.300093
2006    0.161735
2007    0.417738
2008    0.611901
2009    0.432738
2010    0.571946
2011    0.581987
dtype: float64
```

示例：组级别的线性回归

顺着上一个例子继续，你可以用groupby执行更为复杂的分组统计分析，只要函数返回的是pandas对象或标量值即可。例如，我可以定义下面这个regress函数（利用statsmodels计量经济学库）对各数据块执行普通最小二乘法（Ordinary Least Squares，OLS）回归：

```
import statsmodels.api as sm
def regress(data, yvar, xvars):
    Y = data[yvar]
    X = data[xvars]
    X['intercept'] = 1.
    result = sm.OLS(Y, X).fit()
    return result.params
```

现在，为了按年计算AAPL对SPX收益率的线性回归，执行：

```
In [129]: by_year.apply(regress, 'AAPL', ['SPX'])
Out[129]:
          SPX  intercept
2003  1.195406   0.000710
2004  1.363463   0.004201
2005  1.766415   0.003246
2006  1.645496   0.000080
2007  1.198761   0.003438
2008  0.968016  -0.001110
2009  0.879103   0.002954
2010  1.052608   0.001261
2011  0.806605   0.001514
```

10.4 透视表和交叉表

透视表（pivot table）是各种电子表格程序和其他数据分析软件中一种常见的数据汇总工具。它根据一个或多个键对数据进行聚合，并根据行和列上的分组键将数据分配到各个矩形区域中。在Python和pandas中，可以通过本章所介绍的groupby功能以及（能够利用层次化索引的）重塑运算制作透视表。DataFrame有一个pivot_table方法，此外还有一个顶级的pandas.pivot_table函数。除能为groupby提供便利之外，pivot_table还可以添加分项小计，也叫做margins。

回到小费数据集，假设我想要根据day和smoker计算分组平均数（pivot_table的默认聚合类型），并将day和smoker放到行上：

```
In [130]: tips.pivot_table(index=['day', 'smoker'])
Out[130]:
                size       tip   tip_pct  total_bill
day  smoker
Fri  No      2.250000  2.812500  0.151650   18.420000
     Yes     2.066667  2.714000  0.174783   16.813333
Sat  No      2.555556  3.102889  0.158048   19.661778
     Yes     2.476190  2.875476  0.147906   21.276667
Sun  No      2.929825  3.167895  0.160113   20.506667
     Yes     2.578947  3.516842  0.187250   24.120000
Thur No      2.488889  2.673778  0.160298   17.113111
     Yes     2.352941  3.030000  0.163863   19.190588
```

可以用groupby直接来做。现在，假设我们只想聚合tip_pct和size，而且想根据time进行分组。我将smoker放到列上，把day放到行上：

```
In [131]: tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
   .....:                  columns='smoker')
Out[131]:
                size             tip_pct
smoker            No       Yes        No       Yes
time   day
Dinner Fri   2.000000  2.222222  0.139622  0.165347
       Sat   2.555556  2.476190  0.158048  0.147906
       Sun   2.929825  2.578947  0.160113  0.187250
       Thur  2.000000       NaN  0.159744       NaN
Lunch  Fri   3.000000  1.833333  0.187735  0.188937
       Thur  2.500000  2.352941  0.160311  0.163863
```

还可以对这个表作进一步的处理，传入margins=True添加分项小计。这将会添加标签为All的行和列，其值对应于单个等级中所有数据的分组统计：

```
In [132]: tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
   .....:                  columns='smoker', margins=True)
Out[132]:
                size                     tip_pct
smoker            No       Yes       All        No       Yes       All
time   day
Dinner Fri   2.000000  2.222222  2.166667  0.139622  0.165347  0.158916
       Sat   2.555556  2.476190  2.517241  0.158048  0.147906  0.153152
       Sun   2.929825  2.578947  2.842105  0.160113  0.187250  0.166897
       Thur  2.000000       NaN  2.000000  0.159744       NaN  0.159744
Lunch  Fri   3.000000  1.833333  2.000000  0.187735  0.188937  0.188765
       Thur  2.500000  2.352941  2.459016  0.160311  0.163863  0.161301
All          2.668874  2.408602  2.569672  0.159328  0.163196  0.160803
```

这里，All值为平均数：不单独考虑烟民与非烟民（All列），不单独考虑行分组两个级别中的任何单项（All行）。

要使用其他的聚合函数，将其传给aggfunc即可。例如，使用count或len可以得到有关分组大小的交叉表（计数或频率）：

```
In [133]: tips.pivot_table('tip_pct', index=['time', 'smoker'], columns='day',
   .....:                  aggfunc=len, margins=True)
Out[133]:
day             Fri   Sat   Sun  Thur    All
time   smoker
Dinner No       3.0  45.0  57.0   1.0  106.0
       Yes      9.0  42.0  19.0   NaN   70.0
Lunch  No       1.0   NaN   NaN  44.0   45.0
       Yes      6.0   NaN   NaN  17.0   23.0
All            19.0  87.0  76.0  62.0  244.0
```

如果存在空的组合（也就是NA），你可能会希望设置一个fill_value：

```
In [134]: tips.pivot_table('tip_pct', index=['time', 'size', 'smoker'],
   .....:                  columns='day', aggfunc='mean', fill_value=0)
Out[134]:
day                     Fri       Sat       Sun      Thur
time   size smoker
Dinner 1    No      0.000000  0.137931  0.000000  0.000000
            Yes     0.000000  0.325733  0.000000  0.000000
       2    No      0.139622  0.162705  0.168859  0.159744
            Yes     0.171297  0.148668  0.207893  0.000000
       3    No      0.000000  0.154661  0.152663  0.000000
            Yes     0.000000  0.144995  0.152660  0.000000
       4    No      0.000000  0.150096  0.148143  0.000000
            Yes     0.117750  0.124515  0.193370  0.000000
       5    No      0.000000  0.000000  0.206928  0.000000
Yes     0.000000  0.106572  0.065660  0.000000
...                      ...       ...       ...       ...
Lunch  1    No      0.000000  0.000000  0.000000  0.181728
            Yes     0.223776  0.000000  0.000000  0.000000
       2    No      0.000000  0.000000  0.000000  0.166005
            Yes     0.181969  0.000000  0.000000  0.158843
       3    No      0.187735  0.000000  0.000000  0.084246
            Yes     0.000000  0.000000  0.000000  0.204952
       4    No      0.000000  0.000000  0.000000  0.138919
            Yes     0.000000  0.000000  0.000000  0.155410
```

```
       5    No      0.000000  0.000000  0.000000  0.121389
       6    No      0.000000  0.000000  0.000000  0.173706
[21 rows x 4 columns]
```

pivot_table的参数说明请参见表10-2。

函数名	说明
values	待聚合的列的名称。默认聚合所有数值列
index	用于分组的列名或其他分组键，出现在结果透视表的行
columns	用于分组的列名或其他分组键，出现在结果透视表的列
aggfunc	聚合函数或函数列表，默认为 mean。可以是任何对 groupby 有效的函数
fill_value	用于替换结果表中的缺失值
dropna	如果为 True，不添加条目都为 NA 的列
margins	添加行/列小计和总计，默认为 False

表10-2 pivot_table的选项

交叉表：crosstab

交叉表（cross-tabulation，简称crosstab）是一种用于计算分组频率的特殊透视表。看下面的例子：

```
In [138]: data
Out[138]:
   Sample Nationality    Handedness
0       1         USA  Right-handed
1       2       Japan   Left-handed
2       3         USA  Right-handed
3       4       Japan  Right-handed
4       5       Japan   Left-handed
5       6       Japan  Right-handed
6       7         USA  Right-handed
7       8         USA   Left-handed
8       9       Japan  Right-handed
9      10         USA  Right-handed
```

作为调查分析的一部分，我们可能想要根据国籍和用手习惯对这段数据进行统计汇总。虽然可以用pivot_table实现该功能，但是pandas.crosstab函数会更方便：

```
In [139]: pd.crosstab(data.Nationality, data.Handedness, margins=True)
Out[139]:
Handedness   Left-handed  Right-handed  All
Nationality
Japan                  2             3    5
USA                    1             4    5
All                    3             7   10
```

crosstab的前两个参数可以是数组或Series，或是数组列表。就像小费数据：

```
In [140]: pd.crosstab([tips.time, tips.day], tips.smoker, margins=True)
Out[140]:
smoker        No  Yes  All
time   day
Dinner Fri     3    9   12
       Sat    45   42   87
       Sun    57   19   76
       Thur    1    0    1
Lunch  Fri     1    6    7
       Thur   44   17   61
All          151   93  244
```

10.5 总结

掌握pandas数据分组工具既有助于数据清理，也有助于建模或统计分析工作。在第14章，我们会看几个例子，对真实数据使用groupby。

在下一章，我们将关注时间序列数据。