第四章 分组

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

In [2]: import pandas as pd
```

一、分组模式及其对象

1. 分组的一般模式

分组操作在日常生活中使用极其广泛，例如：

- 依据 性别 分组，统计全国人口 寿命 的 平均值
- 依据 季节 分组，对每一个季节的 温度 进行 组内标准化
- 依据 班级 分组，筛选出组内 数学分数 的 平均值超过80分的班级

从上述的几个例子中不难看出，想要实现分组操作，必须明确三个要素：分组依据 、 数据来源 、 操作及其返回结果 。同时从充分性的角度来说，如果明确了这三方面，就能确定一个分组操作，从而分组代码的一般模式即：

```
df.groupby(分组依据)[数据来源].使用操作
```

例如第一个例子中的代码就应该如下：

```
df.groupby('Gender')['Longevity'].mean()
```

现在返回到学生体测的数据集上，如果想要按照性别统计身高中位数，就可以如下写出：

```
In [3]: df = pd.read_csv('data/learn_pandas.csv')

In [4]: df.groupby('Gender')['Height'].median()
Out[4]:
Gender
Female    159.6
Male      173.4
Name: Height, dtype: float64
```

2. 分组依据的本质

前面提到的若干例子都是以单一维度进行分组的，比如根据性别，如果现在需要根据多个维度进行分组，该如何做？事实上，只需在 `groupby` 中传入相应列名构成的列表即可。例如，现想根据学校和性别进行分组，统计身高的均值就可以如下写出：

```
In [5]: df.groupby(['School', 'Gender'])['Height'].mean()
Out[5]:
School                          Gender
Fudan University                Female    158.776923
                                Male      174.212500
Peking University               Female    158.666667
                                Male      172.030000
Shanghai Jiao Tong University   Female    159.122500
                                Male      176.760000
Tsinghua University             Female    159.753333
                                Male      171.638889
Name: Height, dtype: float64
```

目前为止，groupby的分组依据都是直接可以从列中按照名字获取的，那如果想要通过一定的复杂逻辑来分组，例如根据学生体重是否超过总体均值来分组，同样还是计算身高的均值。

首先应该先写出分组条件：

```
In [6]: condition = df.Weight > df.Weight.mean()
```

然后将其传入groupby中：

```
In [7]: df.groupby(condition)['Height'].mean()
Out[7]:
Weight
False    159.034646
True     172.705357
Name: Height, dtype: float64
```

> **练一练**
>
> 请根据上下四分位数分割，将体重分为high、normal、low三组，统计身高的均值。

从索引可以看出，其实最后产生的结果就是按照条件列表中元素的值（此处是 `True` 和 `False` ）来分组，下面用随机传入字母序列来验证这一想法：

```
In [8]: item = np.random.choice(list('abc'), df.shape[0])

In [9]: df.groupby(item)['Height'].mean()
Out[9]:
a    163.924242
b    162.928814
c    162.708621
Name: Height, dtype: float64
```

此处的索引就是原先item中的元素，如果传入多个序列进入 `groupby` ，那么最后分组的依据就是这两个序列对应行的唯一组合：

```
In [10]: df.groupby([condition, item])['Height'].mean()
Out[10]:
Weight
False   a    160.193617
        b    158.921951
        c    157.756410
True    a    173.152632
        b    172.055556
        c    172.873684
Name: Height, dtype: float64
```

由此可以看出，之前传入列名只是一种简便的记号，事实上等价于传入的是一个或多个列，最后分组的依据来自于数据来源组合的unique值，通过 `drop_duplicates` 就能知道具体的组类别：

```
In [11]: df[['School', 'Gender']].drop_duplicates()
Out[11]:
                          School  Gender
0   Shanghai Jiao Tong University  Female
1              Peking University    Male
2   Shanghai Jiao Tong University    Male
3                Fudan University  Female
4                Fudan University    Male
5             Tsinghua University  Female
9               Peking University  Female
16            Tsinghua University    Male

In [12]: df.groupby([df['School'], df['Gender']])['Height'].mean()
Out[12]:
School                         Gender
Fudan University               Female    158.776923
                               Male      174.212500
Peking University              Female    158.666667
                               Male      172.030000
Shanghai Jiao Tong University  Female    159.122500
                               Male      176.760000
Tsinghua University            Female    159.753333
                               Male      171.638889
Name: Height, dtype: float64
```

3. Groupby对象

能够注意到，最终具体做分组操作时，所调用的方法都来自于pandas中的 `groupby` 对象，这个对象上定义了许多方法，也具有一些方便的属性。

```
In [13]: gb = df.groupby(['School', 'Grade'])

In [14]: gb
Out[14]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001308CEB2048>
```

通过 ngroups 属性，可以访问分为了多少组：

```
In [15]: gb.ngroups
Out[15]: 16
```

通过 groups 属性，可以返回从 组名 映射到 组索引列表 的字典：

```
In [16]: res = gb.groups

In [17]: res.keys() # 字典的值由于是索引，元素个数过多，此处只展示字典的键
Out[17]: dict_keys([('Fudan University', 'Freshman'), ('Fudan University', 'Junior'),
('Fudan University', 'Senior'), ('Fudan University', 'Sophomore'), ('Peking University',
'Freshman'), ('Peking University', 'Junior'), ('Peking University', 'Senior'), ('Peking
University', 'Sophomore'), ('Shanghai Jiao Tong University', 'Freshman'), ('Shanghai Jiao
Tong University', 'Junior'), ('Shanghai Jiao Tong University', 'Senior'), ('Shanghai Jiao
Tong University', 'Sophomore'), ('Tsinghua University', 'Freshman'), ('Tsinghua
University', 'Junior'), ('Tsinghua University', 'Senior'), ('Tsinghua University',
'Sophomore')])
```

练一练

上一小节介绍了可以通过 drop_duplicates 得到具体的组类别，现请用 groups 属性完成类似的功能。

当 size 作为 DataFrame 的属性时，返回的是表长乘以表宽的大小，但在 groupby 对象上表示统计每个组的元素个数：

```
In [18]: gb.size()
Out[18]:
School                          Grade
Fudan University                Freshman      9
                                Junior       12
                                Senior       11
                                Sophomore     8
Peking University               Freshman     13
                                Junior        8
                                Senior        8
                                Sophomore     5
Shanghai Jiao Tong University   Freshman     13
                                Junior       17
                                Senior       22
                                Sophomore     5
Tsinghua University             Freshman     17
                                Junior       22
                                Senior       14
                                Sophomore    16
dtype: int64
```

通过 get_group 方法可以直接获取所在组对应的行，此时必须知道组的具体名字：

```
In [19]: gb.get_group(('Fudan University', 'Freshman')).iloc[:3, :3] # 展示一部分
Out[19]:
              School     Grade             Name
15  Fudan University  Freshman  Changqiang Yang
28  Fudan University  Freshman     Gaoqiang Qin
63  Fudan University  Freshman     Gaofeng Zhao
```

这里列出了2个属性和2个方法，而先前的 mean 、 median 都是 groupby 对象上的方法，这些函数和许多其他函数的操作具有高度相似性，将在之后的小节进行专门介绍。

4. 分组的三大操作

熟悉了一些分组的基本知识后，重新回到开头举的三个例子，可能会发现一些端倪，即这三种类型的分组返回数据的结果型态并不一样：

- 第一个例子中，每一个组返回一个标量值，可以是平均值、中位数、组容量 size 等
- 第二个例子中，做了原序列的标准化处理，也就是说每组返回的是一个 Series 类型

- 第三个例子中，既不是标量也不是序列，返回的整个组所在行的本身，即返回了 `DataFrame` 类型

由此，引申出分组的三大操作：聚合、变换和过滤，分别对应了三个例子的操作，下面就要分别介绍相应的 `agg` 、 `transform` 和 `filter` 函数及其操作。

二、聚合函数

1. 内置聚合函数

在介绍agg之前，首先要了解一些直接定义在groupby对象的聚合函数，因为它的速度基本都会经过内部的优化，使用功能时应当优先考虑。根据返回标量值的原则，包括如下函数：

`max/min/mean/median/count/all/any/idxmax/idxmin/mad/nunique/skew/quantile/sum/std/var/sem/size/prod` 。

```
In [20]: gb = df.groupby('Gender')['Height']

In [21]: gb.idxmin()
Out[21]:
Gender
Female    143
Male      199
Name: Height, dtype: int64

In [22]: gb.quantile(0.95)
Out[22]:
Gender
Female    166.8
Male      185.9
Name: Height, dtype: float64
```

> **练一练**
>
> 请查阅文档，明确 `all/any/mad/skew/sem/prod` 函数的含义。

这些聚合函数当传入的数据来源包含多个列时，将按照列进行迭代计算：

```
In [23]: gb = df.groupby('Gender')[['Height', 'Weight']]

In [24]: gb.max()
Out[24]:
        Height  Weight
Gender
Female   170.2    63.0
Male     193.9    97.0
```

2. agg方法

虽然在 `groupby` 对象上定义了许多方便的函数，但仍然有以下缺陷：

- 无法同时使用多个函数
- 无法对特定的列使用特定的聚合函数
- 无法使用自定义的聚合函数
- 无法直接对结果的列名在聚合前进行自定义命名

下面说明如何通过 `agg` 函数解决这四个缺陷：

【a】使用多个函数

当使用多个聚合函数时，需要用列表的形式把内置聚合函数的对应的字符串传入，先前提到的所有字符串都是合法的。

```
In [25]: gb.agg(['sum', 'idxmax', 'skew'])
Out[25]:
         Height                    Weight
            sum idxmax      skew     sum idxmax      skew
Gender
Female  21014.0     28 -0.219253  6469.0     28 -0.268482
Male     8854.9    193  0.437535  3947.0    193  0.075516
```

从结果看，此时的列索引为多级索引，第一层为数据源，第二层为使用的聚合方法，分别逐一对列使用聚合，因此结果为6列。

【b】对特定的列使用特定的聚合函数

对于方法和列的特殊对应，可以通过构造字典传入 agg 中实现，其中字典以列名为键，以聚合字符串或字符串列表为值。

```
In [26]: gb.agg({'Height':['mean','max'], 'Weight':'count'})
Out[26]:
           Height         Weight
             mean    max  count
Gender
Female  159.19697  170.2    135
Male    173.62549  193.9     54
```

> **练一练**
>
> 请使用【b】中的传入字典的方法完成【a】中等价的聚合任务。

【c】使用自定义函数

在 agg 中可以使用具体的自定义函数， 需要注意传入函数的参数是之前数据源中的列，逐列进行计算 。

下面分组计算身高和体重的极差：

```
In [27]: gb.agg(lambda x: x.mean()-x.min())
Out[27]:
          Height     Weight
Gender
Female  13.79697  13.918519
Male    17.92549  22.092593
```

> **练一练**
>
> 在 groupby 对象中可以使用 describe 方法进行统计信息汇总，请同时使用多个聚合函数，完成与该方法相同的功能。

由于传入的是序列，因此序列上的方法和属性都是可以在函数中使用的，只需保证返回值是标量即可。下面的例子是指，如果组的指标均值，超过该指标的总体均值，返回High，否则返回Low。

```
In [28]: def my_func(s):
   ....:     res = 'High'
   ....:     if s.mean() <= df[s.name].mean():
   ....:         res = 'Low'
   ....:     return res
   ....:

In [29]: gb.agg(my_func)
Out[29]:
       Height Weight
Gender
Female    Low    Low
Male     High   High
```

【d】聚合结果重命名

如果想要对结果进行重命名，只需要将上述函数的位置改写成元组，元组的第一个元素为新的名字，第二个位置为原来的函数，包括聚合字符串和自定义函数，现举若干例子说明：

```
In [30]: gb.agg([('range', lambda x: x.max()-x.min()), ('my_sum', 'sum')])
Out[30]:
       Height          Weight
        range   my_sum  range  my_sum
Gender
Female   24.8  21014.0   29.0  6469.0
Male     38.2   8854.9   46.0  3947.0
```

```
In [31]: gb.agg({'Height': [('my_func', my_func), 'sum'],
   ....:         'Weight': lambda x:x.max()})
   ....:
Out[31]:
        Height            Weight
       my_func      sum <lambda>
Gender
Female     Low  21014.0     63.0
Male      High   8854.9     97.0
```

另外需要注意，使用对一个或者多个列使用单个聚合的时候，重命名需要加方括号，否则就不知道是新的名字还是手误输错的内置函数字符串：

```
In [32]: gb.agg([('my_sum', 'sum')])
Out[32]:
         Height  Weight
         my_sum  my_sum
Gender
Female  21014.0  6469.0
Male     8854.9  3947.0
```

```
In [33]: gb.agg({'Height': [('my_func', my_func), 'sum'],
   ....:         'Weight': [('range', lambda x:x.max())]})
   ....:
Out[33]:
        Height          Weight
       my_func      sum  range
Gender
Female     Low  21014.0   63.0
Male      High   8854.9   97.0
```

三、变换和过滤

1. 变换函数与transform方法

变换函数的返回值为同长度的序列，最常用的内置变换函数是累计函数：

`cumcount/cumsum/cumprod/cummax/cummin`，它们的使用方式和聚合函数类似，只不过完成的是组内累计操作。此外在 `groupby` 对象上还定义了填充类和滑窗类的变换函数，这些函数的一般形式将会分别在第七章和第十章中讨论，此处略过。

```
In [34]: gb.cummax().head()
Out[34]:
   Height  Weight
0   158.9    46.0
1   166.5    70.0
2   188.9    89.0
3     NaN    46.0
4   188.9    89.0
```

> **练一练**
>
> 在 `groupby` 对象中， `rank` 方法也是一个实用的变换函数，请查阅它的功能并给出一个使用的例子。

当用自定义变换时需要使用 `transform` 方法，被调用的自定义函数， 其传入值为数据源的序列 ，与 `agg` 的传入类型是一致的，其最后的返回结果是行列索引与数据源一致的 `DataFrame` 。

现对身高和体重进行分组标准化，即减去组均值后除以组的标准差：

```
In [35]: gb.transform(lambda x: (x-x.mean())/x.std()).head()
Out[35]:
     Height    Weight
0 -0.058760 -0.354888
1 -1.010925 -0.367927
2  2.167063  1.892510
3       NaN -1.279789
4  0.053133  0.107955
```

> **练一练**
>
> 对于 `transform` 方法无法像 `agg` 一样，通过传入字典来对指定列使用特定的变换，如果需要在一次 `transform` 的调用中实现这种功能，请给出解决方案。

前面提到了 `transform` 只能返回同长度的序列，但事实上还可以返回一个标量，这会使得结果被广播到其所在的整个组，这种 标量广播 的技巧在特征工程中是非常常见的。例如，构造两列新特征来分别表示样本所在性别组的身高均值和体重均值：

```
In [36]: gb.transform('mean').head() # 传入返回标量的函数也是可以的
Out[36]:
      Height     Weight
0  159.19697  47.918519
1  173.62549  73.092593
2  173.62549  73.092593
3  159.19697  47.918519
4  173.62549  73.092593
```

2. 组索引与过滤

在上一章中介绍了索引的用法，那么索引和过滤有什么区别呢？

过滤在分组中是对于组的过滤，而索引是对于行的过滤，在第二章中的返回值，无论是布尔列表还是元素列表或者位置列表，本质上都是对于行的筛选，即如果筛选条件的则选入结果的表，否则不选入。

组过滤作为行过滤的推广，指的是如果对一个组的全体所在行进行统计的结果返回 `True` 则会被过滤，`False` 则该组会被选中，最后把所有未被过滤的组其对应的所在行拼接起来作为 `DataFrame` 返回。

在 `groupby` 对象中，定义了 `filter` 方法进行组的筛选，其中自定义函数的输入参数为数据源构成的 `DataFrame` 本身，在之前例子中定义的 `groupby` 对象中，传入的就是df[['Height', 'Weight']]，因此所有表方法和属性都可以在自定义函数中相应地使用，同时只需保证自定义函数的返回为布尔值即可。

例如，在原表中过滤容量大于100的组：

```
In [37]: gb.filter(lambda x: x.shape[0] > 100).head()
Out[37]:
   Height  Weight
0   158.9    46.0
3     NaN    41.0
5   158.0    51.0
6   162.5    52.0
7   161.9    50.0
```

> **练一练**
>
> 从概念上说，索引功能是组过滤功能的子集，请使用 `filter` 函数完成 `loc[.]` 的功能，这里假设 ” . “是元素列表。

四、跨列分组

1. apply的引入

之前几节介绍了三大分组操作，但事实上还有一种常见的分组场景，无法用前面介绍的任何一种方法处理，例如现在如下定义身体质量指数BMI：

$$\mathrm{BMI} = \frac{\mathrm{Weight}}{\mathrm{Height}^2}$$

其中体重和身高的单位分别为千克和米，需要分组计算组BMI的均值。

首先，这显然不是过滤操作，因此 `filter` 不符合要求；其次，返回的均值是标量而不是序列，因此 `transform` 不符合要求；最后，似乎使用 `agg` 函数能够处理，但是之前强调过聚合函数是逐列处理的，而不能够 多列数据同时处理 。由此，引出了 `apply` 函数来解决这一问题。

2. apply的使用

在设计上， `apply` 的自定义函数传入参数与 `filter` 完全一致，只不过后者只允许返回布尔值。现如下解决上述计算问题：

```
In [38]: def BMI(x):
    ....:     Height = x['Height']/100
    ....:     Weight = x['Weight']
    ....:     BMI_value = Weight/Height**2
    ....:     return BMI_value.mean()
    ....:

In [39]: gb.apply(BMI)
Out[39]:
Gender
Female    18.860930
Male      24.418396
dtype: float64
```

除了返回标量之外， `apply` 方法还可以返回一维 `Series` 和二维 `DataFrame` ，但它们产生的数据框维数和多级索引的层数应当如何变化？下面举三组例子就非常容易明白结果是如何生成的：

【a】标量情况：结果得到的是 `Series` ，索引与 `agg` 的结果一致

```
In [40]: gb = df.groupby(['Gender','Test_Number'])[['Height','Weight']]

In [41]: gb.apply(lambda x: 0)
Out[41]:
Gender  Test_Number
Female  1              0
        2              0
        3              0
Male    1              0
        2              0
        3              0
dtype: int64

In [42]: gb.apply(lambda x: [0, 0]) # 虽然是列表，但是作为返回值仍然看作标量
Out[42]:
Gender  Test_Number
Female  1              [0, 0]
        2              [0, 0]
        3              [0, 0]
Male    1              [0, 0]
        2              [0, 0]
        3              [0, 0]
dtype: object
```

【b】 `Series` 情况：得到的是 `DataFrame` ，行索引与标量情况一致，列索引为 `Series` 的索引

```
In [43]: gb.apply(lambda x: pd.Series([0,0],index=['a','b']))
Out[43]:
                     a  b
Gender Test_Number
Female 1             0  0
       2             0  0
       3             0  0
Male   1             0  0
       2             0  0
       3             0  0
```

> **练一练**
>
> 请尝试在 `apply` 传入的自定义函数中，根据组的某些特征返回相同长度但索引不同的 `Series` ，会报错吗？

【c】 `DataFrame` 情况：得到的是 `DataFrame` ，行索引最内层在每个组原先 `agg` 的结果索引上，再加一层返回的 `DataFrame` 行索引，同时分组结果 `DataFrame` 的列索引和返回的 `DataFrame` 列索引一致。

```
In [44]: gb.apply(lambda x: pd.DataFrame(np.ones((2,2)),
   ....:                                 index = ['a','b'],
   ....:                                 columns=pd.Index([('w','x'),('y','z')])))
   ....:
Out[44]:
                         w    y
                         x    z
Gender Test_Number
Female 1           a   1.0  1.0
                   b   1.0  1.0
       2           a   1.0  1.0
                   b   1.0  1.0
       3           a   1.0  1.0
                   b   1.0  1.0
Male   1           a   1.0  1.0
                   b   1.0  1.0
       2           a   1.0  1.0
                   b   1.0  1.0
       3           a   1.0  1.0
                   b   1.0  1.0
```

练一练

请尝试在 `apply` 传入的自定义函数中，根据组的某些特征返回相同大小但列索引不同的 `DataFrame`，会报错吗？如果只是行索引不同，会报错吗？

最后需要强调的是， `apply` 函数的灵活性是以牺牲一定性能为代价换得的，除非需要使用跨列处理的分组处理，否则应当使用其他专门设计的 `groupby` 对象方法，否则在性能上会存在较大的差距。同时，在使用聚合函数和变换函数时，也应当优先使用内置函数，它们经过了高度的性能优化，一般而言在速度上都会快于用自定义函数来实现。

练一练

在 `groupby` 对象中还定义了 `cov` 和 `corr` 函数，从概念上说也属于跨列的分组处理。请利用之前定义的 `gb` 对象，使用apply函数实现与 `gb.cov()` 同样的功能并比较它们的性能。