< 组合数据集：Merge 和 Join | 目录 | 数据透视表 >

Open in Colab

Aggregation and Grouping

聚合与分组

> An essential piece of analysis of large data is efficient summarization: computing aggregations like `sum()`, `mean()`, `median()`, `min()`, and `max()`, in which a single number gives insight into the nature of a potentially large dataset. In this section, we'll explore aggregations in Pandas, from simple operations akin to what we've seen on NumPy arrays, to more sophisticated operations based on the concept of a `groupby`.

对于一个大数据集进行分析的关键部分是使用有效的概括：对数据集进行 `sum()`、`mean()`、`median()`、`min()` 和 `max()` 聚合运算，这些运算的结果就可能可以给出大数据集的一些内在特征。在本节中，我们会探讨Pandas中的聚合，从我们已经在NumPy数组中进行过的那些简单的操作，直到基于分组 `groupby` 概念进行的更复杂的操作。

> For convenience, we'll use the same `display` magic function that we've seen in previous sections:

方便起见，我们还是使用与前两节同样的 `display` 类来展示多个数据集：

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

class display(object):
    """Display HTML representation of multiple objects"""
    template = """<div style="float: left; padding: 10px;">
    <p style='font-family:"Courier New", Courier, monospace'>{0}</p>{1}
    </div>"""
    def __init__(self, *args):
        self.args = args

    def _repr_html_(self):
        return '\n'.join(self.template.format(a, eval(a)._repr_html_())
                         for a in self.args)

    def __repr__(self):
        return '\n\n'.join(a + '\n' + repr(eval(a))
                           for a in self.args)
```

Planets Data

行星数据

> Here we will use the Planets dataset, available via the Seaborn package (see Visualization With Seaborn). It gives information on planets that astronomers have discovered around other stars (known as *extrasolar planets* or *exoplanets* for short). It can be downloaded with a simple Seaborn command:

这里我们会使用Seaborn包提供的行星数据（参见使用Seaborn进行可视化）。这个数据集提供了天文学家发现的其他恒星的行星的数据（被称为太阳系外行星）。数据集可以简单的使用一个Seaborn命令来下载：

In [2]:
```python
import seaborn as sns
planets = sns.load_dataset('planets')
planets.shape
```

Out[2]:
```
(1035, 6)
```

In [3]:
```python
planets.head()
```

Out[3]:

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009

> This has some details on the 1,000+ extrasolar planets discovered up to 2014.

直到2014年已经有超过1000个太阳系外行星的数据。

Simple Aggregation in Pandas

在Pandas中进行简单聚合

> Earlier, we explored some of the data aggregations available for NumPy arrays ("Aggregations: Min, Max, and Everything In Between"). As with a one-dimensional NumPy array, for a Pandas `Series` the aggregates return a single value:

上一章中，我们已经介绍了NumPy数组的数据聚合操作（聚合：Min, Max, 以及其他）。正如一维NumPy数组，Pandas的 `Series` 的聚合结果是一个标量：

In [4]:
```python
rng = np.random.RandomState(42)
ser = pd.Series(rng.rand(5))
ser
```

Out[4]:
```
0    0.374540
1    0.950714
2    0.731994
3    0.598658
4    0.156019
dtype: float64
```

In [5]:
```python
ser.sum()
```

Out[5]:
```
2.811925491708157
```

In [6]:
```python
ser.mean()
```

Out[6]:
```
0.5623850983416314
```

> For a `DataFrame`, by default the aggregates return results within each column:

对于 `DataFrame` 来说，默认情况下是每个列进行聚合的结果：

In [7]:
```python
df = pd.DataFrame({'A': rng.rand(5),
                   'B': rng.rand(5)})
df
```

Out[7]:

	A	B
0	0.155995	0.020584
1	0.058084	0.969910
2	0.866176	0.832443
3	0.601115	0.212339
4	0.708073	0.181825

In [8]:
```python
df.mean()
```

Out[8]:
```
A    0.477888
B    0.443420
dtype: float64
```

> By specifying the `axis` argument, you can instead aggregate within each row:

通过指定 `axis` 参数，可以为每一行进行聚合操作：

In [9]:
```python
df.mean(axis='columns')
```

Out[9]:
```
0    0.088290
1    0.513997
2    0.849309
3    0.406727
4    0.444949
dtype: float64
```

> Pandas `Series` and `DataFrame` s include all of the common aggregates mentioned in Aggregations: Min, Max, and Everything In Between; in addition, there is a convenience method `describe()` that computes several common aggregates for each column and returns the result. Let's use this on the Planets data, for now dropping rows with missing values:

Pandas的 `Series` 和 `DataFrame` 包括了所有我们在聚合：Min, Max, 以及其他中介绍过的通用聚合操作；而且Pandas还提供了很方便的 `describe()` 可以用来对每个列计算这些通用的聚合结果。让我们在行星数据集上使用这个函数，暂时先移除含有空值的行：

In [10]:
```python
planets.dropna().describe()
```

Out[10]:

	number	orbital_period	mass	distance	year
count	498.00000	498.000000	498.000000	498.000000	498.000000
mean	1.73494	835.778671	2.509320	52.068213	2007.377510
std	1.17572	1469.128259	3.636274	46.596041	4.167284
min	1.00000	1.328300	0.003600	1.350000	1989.000000
25%	1.00000	38.272250	0.212500	24.497500	2005.000000
50%	1.00000	357.000000	1.245000	39.940000	2009.000000
75%	2.00000	999.600000	2.867500	59.332500	2011.000000
max	6.00000	17337.500000	25.000000	354.000000	2014.000000

> This can be a useful way to begin understanding the overall properties of a dataset. For example, we see in the `year` column that although exoplanets were discovered as far back as 1989, half of all known expolanets were not discovered until 2010 or after. This is largely thanks to the *Kepler* mission, which is a space-based telescope specifically designed for finding eclipsing planets around other stars.

对于开始理解数据集的整体情况来说，这是一个非常有用的方法。例如，在发现年份 `year` 列上，结果显示，虽然第一颗太阳系外行星是1989年发现的，但是一半的行星直到2010年以后才被发现的。这多亏了*开普勒Kepler*计划，它是一个太空望远镜，专门设计用来寻找其他恒星的椭圆轨道行星的。

> The following table summarizes some other built-in Pandas aggregations:

下表概括了Pandas内建的聚合操作：

聚合函数	描述
`count()`	元素个数
`first()`, `last()`	第一个和最后一个元素
`mean()`, `median()`	平均值和中位数
`min()`, `max()`	最小和最大值
`std()`, `var()`	标准差和方差
`mad()`	平均绝对离差
`prod()`	所有元素的乘积
`sum()`	所有元素的总和

> These are all methods of `DataFrame` and `Series` objects.

它们都是 `DataFrame` 和 `Series` 对象的方法。

> To go deeper into the data, however, simple aggregates are often not enough. The next level of data summarization is the `groupby` operation, which allows you to quickly and efficiently compute aggregates on subsets of data.

然而要深入了解数据，简单的聚合经常是不够的。`groupby` 操作为我们提供更高层次的概括功能，通过它能很快速和有效地计算子数据集的聚合数据。

GroupBy: Split, Apply, Combine

分组：拆分、应用、组合

> Simple aggregations can give you a flavor of your dataset, but often we would prefer to aggregate conditionally on some label or index: this is implemented in the so-called `groupby` operation. The name "group by" comes from a command in the SQL database language, but it is perhaps more illuminative to think of it in the terms first coined by Hadley Wickham of Rstats fame: *split, apply, combine*.

简单的聚合可以提供数据集的基础特征，但是通常我们更希望依据一些标签或索引条件进行聚合操作：这可以通过 `groupby` 操作实现。"group by"的名称来自于SQL，但是将它想成是由Hadley Wickham首先创造的R数据统计术语会更合适：*拆分*、*应用*、*组合*。

Split, apply, combine

拆分、应用、组合

> A canonical example of this split-apply-combine operation, where the "apply" is a summation aggregation, is illustrated in this figure:

作为拆分-应用-组合操作的一个典型例子，下图展示了当进行求和的"应用"聚合操作时的情况：

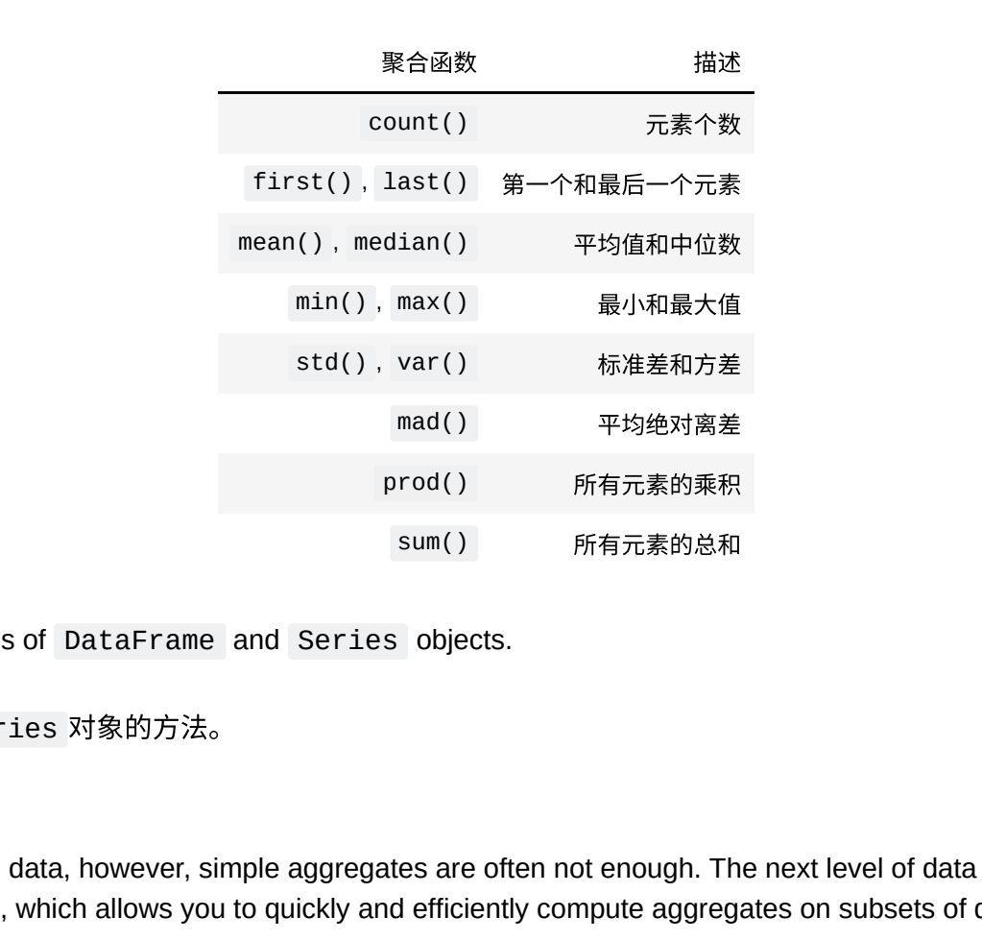

figure source in Appendix

附录：生成图像的源代码

> This makes clear what the `groupby` accomplishes:
>
> - The *split* step involves breaking up and grouping a `DataFrame` depending on the value of the specified key.
> - The *apply* step involves computing some function, usually an aggregate, transformation, or filtering, within the individual groups.
> - The *combine* step merges the results of these operations into an output array.

上图很清晰地展示了 `groupby` 完成的工作：

- 拆分*split*步骤表示按照指定键上的值对 `DataFrame` 进行拆分和分组的功能。
- 应用*apply*步骤表示在每个独立的分组上调用某些函数进行计算，通常是聚合、转换或过滤。
- 组合*combine*步骤将上述计算的结果重新合并在一起输出。

> While this could certainly be done manually using some combination of the masking, aggregation, and merging commands covered earlier, an important realization is that *the intermediate splits do not need to be explicitly instantiated*. Rather, the `GroupBy` can (often) do this in a single pass over the data, updating the sum, mean, count, min, or other aggregate for each group along the way. The power of the `GroupBy` is that it abstracts away these steps: the user need not think about *how* the computation is done under the hood, but rather thinks about the *operation as a whole*.

虽然这可以通过将前面介绍过的遮盖、聚合和组合指令组合在一起来实现，`groupby` 的一个重要的实现是*拆分的中间结果不需要真正的创建出来*。而且，`groupby`（通常）可以在一次过程中处理完所有的数据分组的总和、平均值、计数、最小是或其他聚合操作。`groupby` 的强大在于它将这些步骤抽象了出来：用户不需要思考这些计算是*如何*进行的，只需要认为*这些操作是一个整体*。

> As a concrete example, let's take a look at using Pandas for the computation shown in this diagram. We'll start by creating the input `DataFrame`:

作为一个具体的例子，我们来看一下使用Pandas来实现上面的这些计算，首先创建一个输入 `DataFrame`：

In [11]:
```python
df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],
                   'data': range(6)}, columns=['key', 'data'])
df
```

Out[11]:

	key	data
0	A	0
1	B	1
2	C	2
3	A	3
4	B	4
5	C	5

> The most basic split-apply-combine operation can be computed with the `groupby()` method of `DataFrame` s, passing the name of the desired key column:

最基础的拆分-应用-组合操作可以使用 `DataFrame` 的 `groupby()` 方法来实现，方法中传递作为键来运算的列名：

In [12]:
```python
df.groupby('key')
```

Out[12]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fd196fe1d30>
```

> Notice that what is returned is not a set of `DataFrame` s, but a `DataFrameGroupBy` object. This object is where the magic is: you can think of it as a special view of the `DataFrame`, which is poised to dig into the groups but does no actual computation until the aggregation is applied. This "lazy evaluation" approach means that common aggregates can be implemented very efficiently in a way that is almost transparent to the user.

上面运行的结果不是一个 `DataFrame`，而是一个 `DataFrameGroupBy` 对象。这个对象就是上述步骤魔术的所在：你可以认为它是 `DataFrame` 对象的一个特殊的视图，使用它可以很容易的研究分组的数据，但是除非聚合操作发生，否则它不会进行真实的运算。这种"懒运算"的方式意味着通用的聚合可以实现得非常的高效，而对用户来说几乎是透明的。

> To produce a result, we can apply an aggregate to this `DataFrameGroupBy` object, which will perform the appropriate apply/combine steps to produce the desired result:

要产生结果，我们可以将一个聚合操作应用到该 `DataFrameGroupBy` 对象上，这样就会在分组上执行应用/组合的步骤，并产生需要的结果：

In [13]:
```python
df.groupby('key').sum()
```

Out[13]:

key	data
A	3
B	5
C	7

> The `sum()` method is just one possibility here; you can apply virtually any common Pandas or NumPy aggregation function, as well as virtually any valid `DataFrame` operation, as we will see in the following discussion.

`sum()` 方法仅是其中一个可能的操作；你可以在这里应用几乎所有的Pandas或NumPy的通用聚合函数，也可以应用集合所有正确的 `DataFrame` 操作，我们在下面马上就会看到。

The GroupBy object

GroupBy 对象

> The `GroupBy` object is a very flexible abstraction. In many ways, you can simply treat it as if it's a collection of `DataFrame` s, and it does the difficult things under the hood. Let's see some examples using the Planets data.

`GroupBy` 对象是一个很灵活的抽象。在很多情况下，你可以将它简单的看成 `DataFrame` 的集合，它在底层做了很多复杂的工作。我们用行星数据集来看几个例子。

> Perhaps the most important operations made available by a `GroupBy` are *aggregate*, *filter*, *transform*, and *apply*. We'll discuss each of these more fully in "Aggregate, Filter, Transform, Apply", but before that let's introduce some of the other functionality that can be used with the basic `GroupBy` operation.

也许对 `GroupBy` 对象最重要的操作是*聚合*、*过滤*、*转换*和*应用*。我们会在聚合、过滤、转换、应用中逐个介绍它们，在这之前首先介绍一些其他用于 `GroupBy` 对象的基础操作。

Column indexing

列索引

> The `GroupBy` object supports column indexing in the same way as the `DataFrame`, and returns a modified `GroupBy` object. For example:

`GroupBy` 对象支持列索引，与 `DataFrame` 相同，返回的是修改后的 `GroupBy` 对象。例如：

In [14]:
```python
planets.groupby('method')
```

Out[14]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fd196fa0358>
```

In [15]:
```python
planets.groupby('method')['orbital_period']
```

Out[15]:
```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x7fd19700ddd8>
```

> Here we've selected a particular `Series` group from the original `DataFrame` group by reference to its column name. As with the `GroupBy` object, no computation is done until we call some aggregate on the object:

上例中我们在原始的 `DataFrame` 中选择了特定的 `Series`，这个 `Series` 是按照提供的列名进行分组的。当然，`GroupBy` 对象在调用聚合操作之前是不会进行计算的：

In [16]:
```python
planets.groupby('method')['orbital_period'].median()
```

Out[16]:
```
method
Astrometry                         631.180000
Eclipse Timing Variations         4343.500000
Imaging                          27500.000000
Microlensing                      3300.000000
Orbital Brightness Modulation        0.342887
Pulsar Timing                       66.541900
Pulsation Timing Variations       1170.000000
Radial Velocity                    360.200000
Transit                              5.714932
Transit Timing Variations           57.011000
Name: orbital_period, dtype: float64
```

> This gives an idea of the general scale of orbital periods (in days) that each method is sensitive to.

结果给出了一个不同测量方法对公转周期进行测量的大概范围。

Iteration over groups

在分组上进行迭代

> The `GroupBy` object supports direct iteration over the groups, returning each group as a `Series` or `DataFrame`:

`GroupBy` 对象支持在分组上直接进行迭代，每次迭代返回分组的一个 `Series` 或 `DataFrame` 对象：

In [17]:
```python
for (method, group) in planets.groupby('method'):
    print("{0:30s} shape={1}".format(method, group.shape))
```

```
Astrometry                     shape=(2, 6)
Eclipse Timing Variations      shape=(9, 6)
Imaging                        shape=(38, 6)
Microlensing                   shape=(23, 6)
Orbital Brightness Modulation  shape=(3, 6)
Pulsar Timing                  shape=(5, 6)
Pulsation Timing Variations    shape=(1, 6)
Radial Velocity                shape=(553, 6)
Transit                        shape=(397, 6)
Transit Timing Variations      shape=(4, 6)
```

> This can be useful for doing certain things manually, though it is often much faster to use the built-in `apply` functionality, which we will discuss momentarily.

这种做法在某些需要手动实现的情况下很有用，虽然通常来说使用内建的 `apply` 函数会快很多，我们马上会介绍到 `apply` 函数。

Dispatch methods

扩展方法

> Through some Python class magic, any method not explicitly implemented by the `GroupBy` object will be passed through and called on the groups, whether they are `DataFrame` or `Series` objects. For example, you can use the `describe()` method of `DataFrame` s to perform a set of aggregations that describe each group in the data:

通过一些Python面向对象的魔术技巧，任何非显式定义在 `GroupBy` 对象上的方法，无论是 `DataFrame` 还是 `Series` 对象的，都可以给分组来调用。例如，你可以在数据分组上调用 `DataFrame` 的 `describe()` 方法，对所有分组进行通用的聚合运算：

译者注：作者下面代码多加了 `unstack()` 方法，应该是笔误。

In [18]:
```python
planets.groupby('method')['year'].describe()
```

Out[18]:

method	count	mean	std	min	25%	50%	75%	max
Astrometry	2.0	2011.500000	2.121320	2010.0	2010.75	2011.5	2012.25	2013.0
Eclipse Timing Variations	9.0	2010.000000	1.414214	2008.0	2009.00	2010.0	2011.00	2012.0
Imaging	38.0	2009.131579	2.781901	2004.0	2008.00	2009.0	2011.00	2013.0
Microlensing	23.0	2009.782609	2.859697	2004.0	2008.00	2010.0	2012.00	2013.0
Orbital Brightness Modulation	3.0	2011.666667	1.154701	2011.0	2011.00	2011.0	2012.00	2013.0
Pulsar Timing	5.0	1998.400000	8.384510	1992.0	1992.00	1994.0	2003.00	2011.0
Pulsation Timing Variations	1.0	2007.000000	NaN	2007.0	2007.00	2007.0	2007.00	2007.0
Radial Velocity	553.0	2007.518987	4.249052	1989.0	2005.00	2009.0	2011.00	2014.0
Transit	397.0	2011.236776	2.077867	2002.0	2010.00	2012.0	2013.00	2014.0
Transit Timing Variations	4.0	2012.500000	1.290994	2011.0	2011.75	2012.5	2013.25	2014.0

> Looking at this table helps us to better understand the data: for example, the vast majority of planets have been discovered by the Radial Velocity and Transit methods, though the latter only became common (due to new, more accurate telescopes) in the last decade. The newest methods seem to be Transit Timing Variation and Orbital Brightness Modulation, which were not used to discover a new planet until 2011.

查看上表，能帮助我们更好的理解数据：例如，发现行星最多的方法是径向速度和凌日法，虽然后者是近十年才变得普遍（因为新的更精准的望远镜的作用）。最新的方法应该是凌日时间变分法和轨道亮度调制法，它们直至2011年才开始发现新的行星。

> This is just one example of the utility of dispatch methods. Notice that they are applied *to each individual group*, and the results are then combined within `GroupBy` and returned. Again, any valid `DataFrame` / `Series` method can be used on the corresponding `GroupBy` object, which allows for some very flexible and powerful operations!

这只是一个使用扩展方法的例子。你需要知道的是这些方法会被应用到*每一个独立的分组*上，然后计算得到的结果会在 `GroupBy` 对象中合并并返回。再次提示，任何正确的 `DataFrame` 或 `Series` 方法都能在相应的 `GroupBy` 对象上使用，这种扩展方法的方式提供了非常灵活及强大的操作。

Aggregate, filter, transform, apply

聚合、过滤、转换、应用

> The preceding discussion focused on aggregation for the combine operation, but there are more options available. In particular, `GroupBy` objects have `aggregate()`, `filter()`, `transform()`, and `apply()` methods that efficiently implement a variety of useful operations before combining the grouped data.

前面的讨论聚焦在组合操作相应的聚合函数上，但实际上还有更多的可能选项。特别是 `GroupBy` 对象有 `aggregate()`、`filter()`、`transfrom` 和 `apply()` 方法，它们能在组合分组数据之前有效地实现大量有用的操作。

> For the purpose of the following subsections, we'll use this `DataFrame`:

对于下面的部分内容，我们将使用下述的 `DataFrame`：

In [19]:
```python
rng = np.random.RandomState(0)
df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],
                   'data1': range(6),
                   'data2': rng.randint(0, 10, 6)},
                   columns = ['key', 'data1', 'data2'])
df
```

Out[19]:

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

Aggregation

聚合

> We're now familiar with `GroupBy` aggregations with `sum()`, `median()`, and the like, but the `aggregate()` method allows for even more flexibility. It can take a string, a function, or a list thereof, and compute all the aggregates at once. Here is a quick example combining all these:

我们已经熟悉了 `GroupBy` 使用 `sum()`、`median()` 等方法进行聚合的做法，但是 `aggregate()` 方法能提供更多的灵活性。它能接受字符串、函数或者一个列表，然后一次性计算出所有的聚合结果。下面是一个简单的例子：

In [20]:
```python
df.groupby('key').aggregate(['min', np.median, max])
```

Out[20]:

	data1			data2		
	min	median	max	min	median	max
key						
A	0	1.5	3	3	4.0	5
B	1	2.5	4	0	3.5	7
C	2	3.5	5	3	6.0	9

> Another useful pattern is to pass a dictionary mapping column names to operations to be applied on that column:

还可以将一个字典，里面是列名与操作的对应关系，传递给 `aggregate()` 来进行一次性的聚合运算：

In [21]:
```python
df.groupby('key').aggregate({'data1': 'min',
                             'data2': 'max'})
```

Out[21]:

key	data1	data2
A	0	5
B	1	7
C	2	9

Filtering

过滤

> A filtering operation allows you to drop data based on the group properties. For example, we might want to keep all groups in which the standard deviation is larger than some critical value:

过滤操作能在分组数据上移除一些你不需要的数据。例如，我们可能希望保留标准差大于某个阈值的所有的分组：

译者注：你可以认为 `filter()` 类似于SQL中的HAVING。

In [22]:
```python
def filter_func(x):
    return x['data2'].std() > 4

display('df', "df.groupby('key').std()", "df.groupby('key').filter(filter_func)")
```

Out[22]:

`df`

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

`df.groupby('key').std()`

key	data1	data2
A	2.12132	1.414214
B	2.12132	4.949747
C	2.12132	4.242641

`df.groupby('key').filter(filter_func)`

	key	data1	data2
1	B	1	0
2	C	2	3
4	B	4	7
5	C	5	9

> The filter function should return a Boolean value specifying whether the group passes the filtering. Here because group A does not have a standard deviation greater than 4, it is dropped from the result.

用来进行过滤的函数必须返回一个布尔值，表示分组是否能够通过过滤条件。上例中A分组的标准差不是大于4，因此整个分组在结果中被移除了。

Transformation

转换

> While aggregation must return a reduced version of the data, transformation can return some transformed version of the full data to recombine. For such a transformation, the output is the same shape as the input. A common example is to center the data by subtracting the group-wise mean:

聚合返回的是分组简化后的数据集，而转换可以返回完整数据转换后并重新合并的数据集。因此转换操作的结果和输入数据集具有相同的形状。一个通用例子是将整个数据集通过减去每个分组的平均值进行中心化：

In [23]:
```python
df.groupby('key').transform(lambda x: x - x.mean())
```

Out[23]:

	data1	data2
0	-1.5	1.0
1	-1.5	-3.5
2	-1.5	-3.0
3	1.5	-1.0
4	1.5	3.5
5	1.5	3.0

The apply() method

应用

> The `apply()` method lets you apply an arbitrary function to the group results. The function should take a `DataFrame`, and return either a Pandas object (e.g., `DataFrame`, `Series`) or a scalar; the combine operation will be tailored to the type of output returned.

`apply()` 方法能让你将分组的结果应用到任意的函数上。该函数必须接受一个 `DataFrame` 参数，返回一个Pandas对象（如 `DataFrame`、`Series`）或者一个标量；组合操作会根据返回的类型进行适配。

> For example, here is an `apply()` that normalizes the first column by the sum of the second:

例如，下面采用 `apply()` 使用 `data2` 的分组总和来正则化 `data1` 的值：

In [24]:
```python
def norm_by_data2(x):
    # x is a DataFrame of group values
    x['data1'] /= x['data2'].sum()
    return x

display('df', "df.groupby('key').apply(norm_by_data2)")
```

Out[24]:

`df`

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

`df.groupby('key').apply(norm_by_data2)`

	key	data1	data2
0	A	0.000000	5
1	B	0.142857	0
2	C	0.166667	3
3	A	0.375000	3
4	B	0.571429	7
5	C	0.416667	9

> `apply()` within a `GroupBy` is quite flexible: the only criterion is that the function takes a `DataFrame` and returns a Pandas object or scalar; what you do in the middle is up to you!

`GroupBy` 对象的 `apply()` 方法是非常灵活的：唯一的限制就是应用的函数要接受一个 `DataFrame` 参数并且返回一个Pandas对象或者标量；函数体内做什么工作完全是自定义的。

Specifying the split key

指定拆分键

> In the simple examples presented before, we split the `DataFrame` on a single column name. This is just one of many options by which the groups can be defined, and we'll go through some other options for group specification here.

在前面的简单例子中，我们使用一个列名对 `DataFrame` 进行拆分。这只是分组的众多方式的其中之一，我们下面继续探讨其他的选项。

A list, array, series, or index providing the grouping keys

使用列表、数组、序列或索引指定分组键

> The key can be any series or list with a length matching that of the `DataFrame`. For example:

分组使用的键可以使任何的序列或列表，只要长度和 `DataFrame` 的长度互相匹配即可。例如：

In [25]:
```python
L = [0, 1, 0, 1, 2, 0]
display('df', 'df.groupby(L).sum()')
```

Out[25]:

`df`

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

`df.groupby(L).sum()`

	data1	data2
0	7	17
1	4	3
2	4	7

> Of course, this means there's another, more verbose way of accomplishing the `df.groupby('key')` from before:

当然，这就表明，前面的 `df.groupby('key')` 语法还有另外一种更加有含义的方式来实现：

```
In [26]: display('df', "df.groupby(df['key']).sum()")
```

Out[26]:

df

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

df.groupby(df['key']).sum()

	data1	data2
key		
A	3	8
B	5	7
C	7	12

A dictionary or series mapping index to group

使用字典或映射索引的序列来分组

> Another method is to provide a dictionary that maps index values to the group keys:

还有一种方法是提供一个字典，将索引值映射成分组键：

```
In [27]: df2 = df.set_index('key')
         mapping = {'A': 'vowel', 'B': 'consonant', 'C': 'consonant'}
         display('df2', 'df2.groupby(mapping).sum()')
```

Out[27]:

df2

	data1	data2
key		
A	0	5
B	1	0
C	2	3
A	3	3
B	4	7
C	5	9

df2.groupby(mapping).sum()

	data1	data2
consonant	12	19
vowel	3	8

Any Python function

任何**Python**函数

> Similar to mapping, you can pass any Python function that will input the index value and output the group:

类似映射，你可以传递任何Python函数将输入的索引值变成输出的分组键：

```
In [28]: display('df2', 'df2.groupby(str.lower).mean()')
```

Out[28]:

df2

	data1	data2
key		
A	0	5
B	1	0
C	2	3
A	3	3
B	4	7
C	5	9

df2.groupby(str.lower).mean()

	data1	data2
a	1.5	4.0
b	2.5	3.5
c	3.5	6.0

A list of valid keys

正确键的列表

> Further, any of the preceding key choices can be combined to group on a multi-index:

还有，任何前面的多个分组键可以组合并输出成一个多重索引的结果：

```
In [29]: df2.groupby([str.lower, mapping]).mean()
```

Out[29]:

		data1	data2
a	**vowel**	1.5	4.0
b	**consonant**	2.5	3.5
c	**consonant**	3.5	6.0

Grouping example

分组例子

> As an example of this, in a couple lines of Python code we can put all these together and count discovered planets by method and by decade:

作为分组的例子，我们将前面介绍的内容用几行Python代码写出来用于计算通过不同方法在不同年代发现的行星的个数：

```
In [30]: decade = 10 * (planets['year'] // 10)
         decade = decade.astype(str) + 's'
         decade.name = 'decade'
         planets.groupby(['method', decade])['number'].sum().unstack().fillna(0)
```

Out[30]:

decade	1980s	1990s	2000s	2010s
method				
Astrometry	0.0	0.0	0.0	2.0
Eclipse Timing Variations	0.0	0.0	5.0	10.0
Imaging	0.0	0.0	29.0	21.0
Microlensing	0.0	0.0	12.0	15.0
Orbital Brightness Modulation	0.0	0.0	0.0	5.0
Pulsar Timing	0.0	9.0	1.0	1.0
Pulsation Timing Variations	0.0	0.0	1.0	0.0
Radial Velocity	1.0	52.0	475.0	424.0
Transit	0.0	0.0	64.0	712.0
Transit Timing Variations	0.0	0.0	0.0	9.0

> This shows the power of combining many of the operations we've discussed up to this point when looking at realistic datasets. We immediately gain a coarse understanding of when and how planets have been discovered over the past several decades!

这个例子展示了我们结合前面介绍过的多种操作之后，我们能在真实的数据集上完成多强大的操作。我们立即获得了过去几十年间我们是如何发现行星的大概统计。

> Here I would suggest digging into these few lines of code, and evaluating the individual steps to make sure you understand exactly what they are doing to the result. It's certainly a somewhat complicated example, but understanding these pieces will give you the means to similarly explore your own data.

作者建议你深入研究上面的几行代码，逐步的执行它们，直到你完全理解了这些代码是如何最终产生结果的。当然上面是一个稍微复杂的例子，但是理解这个例子会让你在研究自己的数据集时知道如何进行操作。

< 组合数据集：Merge 和 Join | 目录 | 数据透视表 >

Open in Colab