```
import pandas as pd

import numpy as np
```

在数据分析中，经常会遇到这样的情况：

```
根据某一列（或多列）标签把数据划分为不同的组别，然后再对其进行数据分析。

比如，某网站对注册用户的性别或者年龄等进行分组，从而研究出网站用户的画像（特点）。在 Pandas 中，要完成数据的分组操作，需要使用 groupby() 函数，它和 SQL 的GROUP BY操作非常相似。
```

在划分出来的组（group）上应用一些统计函数，从而达到数据分析的目的，比如对分组数据进行聚合、转换，或者过滤。这个过程主要包含以下三步：

- 拆分（Spliting）：表示对数据进行分组；
- 应用（Applying）：对分组数据应用聚合函数，进行相应计算；
- 合并（Combining）：最后汇总计算结果。

模拟生成的10个样本数据，代码和数据如下：

```
company=["A","B","C"]
data=pd.DataFrame({
        "company":[company[x] for x in
np.random.randint(0,len(company),10)],
        "salary":np.random.randint(5,50,10),
        "age":np.random.randint(15,50,10)
    }
)
data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	company	salary	age
0	A	32	18
1	A	30	29
2	B	34	38
3	C	44	37
4	B	30	31
5	C	28	19
6	A	44	26
7	A	6	34
8	B	48	18
9	A	37	33

一、Groupby的基本原理

在pandas中，实现分组操作的代码很简单，仅需一行代码，在这里，将上面的数据集按照company字段进行划分：

```
group = data.groupby("company")
group
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at
0x000001E9B57A3C88>
```

> 将上述代码输入ipython后，会得到一个DataFrameGroupBy对象

那这个生成的DataFrameGroupBy是啥呢？

对data进行了groupby后发生了什么？

ipython所返回的结果是其内存地址，并不利于直观地理解，为了看看group内部究竟是什么，这里把group转换成list的形式来看一看：

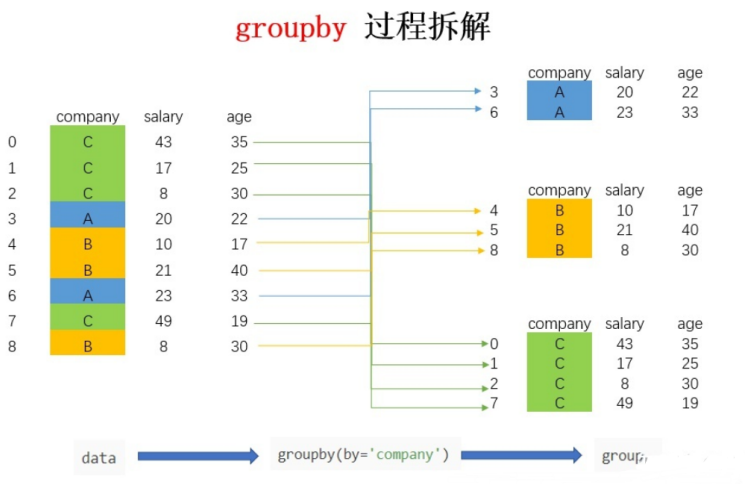
```
list(group)
```

```
[('A',
    company  salary  age
  0       A      32   18
  1       A      30   29
  6       A      44   26
  7       A       6   34
  9       A      37   33),
 ('B',
    company  salary  age
  2       B      34   38
  4       B      30   31
  8       B      48   18),
 ('C',
    company  salary  age
  3       C      44   37
  5       C      28   19)]
```

转换成列表的形式后，可以看到，列表由三个元组组成，每个元组中，

第一个元素是组别（这里是按照company进行分组，所以最后分为了A,B,C），

第二个元素的是对应组别下的DataFrame，整个过程可以图解如下：



总结来说，groupby的过程就是将原有的DataFrame按照groupby的字段（这里是company），划分为若干个分组DataFrame，被分为多少个组就有多少个分组DataFrame。所以说，在groupby之后的一系列操作（如agg、apply等），均是基于子DataFrame的操作。

理解了这点，也就基本摸清了Pandas中groupby操作的主要原理。下面来讲讲groupby之后的常见操作

二、agg 聚合操作

聚合(Aggregation)操作是groupby后非常常见的操作，会写SQL的朋友对此应该是非常熟悉了。聚合操作可以用来求和、均值、最大值、最小值等，下面的表格列出了Pandas中常见的聚合操作。

函数	用途
min	最小值
max	最大值
sum	求和
mean	均值
median	中位数
std	标准差
var	方差
count	计数

针对样例数据集，如果我想求不同公司员工的平均年龄和平均薪水，可以按照下方的代码进行：

```
data.groupby("company").agg('mean')
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

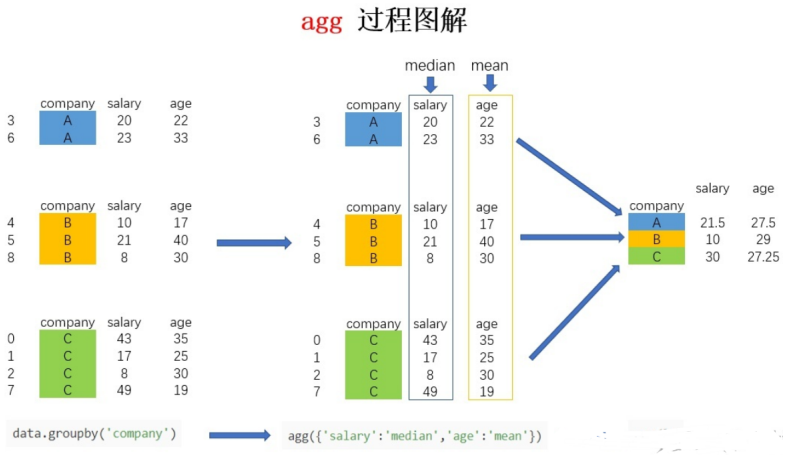
	salary	age
company		
A	29.800000	28.0
B	37.333333	29.0
C	36.000000	28.0

如果想对针对不同的列求不同的值，比如要计算不同公司员工的平均年龄以及薪水的中位数，可以利用字典进行聚合操作的指定：

```
data.groupby('company').agg({'salary':'median','age':'mean'})
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	salary	age
company		
A	32	28
B	34	29
C	36	28



三、transform 转换值

transform是一种什么数据操作？

和agg有什么区别呢？

为了更好地理解transform和agg的不同，下面从实际的应用场景出发进行对比。

在上面的agg中，我们学会了如何求不同公司员工的平均薪水，

如果现在需要在原数据集中新增一列avg_salary，代表员工所在的公司的平均薪水（相同公司的员工具有一样的平均薪水），该怎么实现呢？

如果按照正常的步骤来计算，需要先求得不同公司的平均薪水，然后按照员工和公司的对应关系填充到对应的位置，不用transform的话

```
# to_dict将表格中的数据转换成字典格式
avg_salary_dict= data.groupby('company')['salary'].mean().to_dict()
avg_salary_dict
```

```
{'A': 29.8, 'B': 37.333333333333336, 'C': 36.0}
```

```
# map()函数可以用于Series对象或DataFrame对象的一列，接收函数作为或字典对象作为参数，返回经过函数或字典映射处理后的值。
data['avg_salary'] = data['company'].map(avg_salary_dict)
data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	company	salary	age	avg_salary
0	A	32	18	29.800000
1	A	30	29	29.800000
2	B	34	38	37.333333
3	C	44	37	36.000000
4	B	30	31	37.333333
5	C	28	19	36.000000
6	A	44	26	29.800000
7	A	6	34	29.800000
8	B	48	18	37.333333
9	A	37	33	29.800000

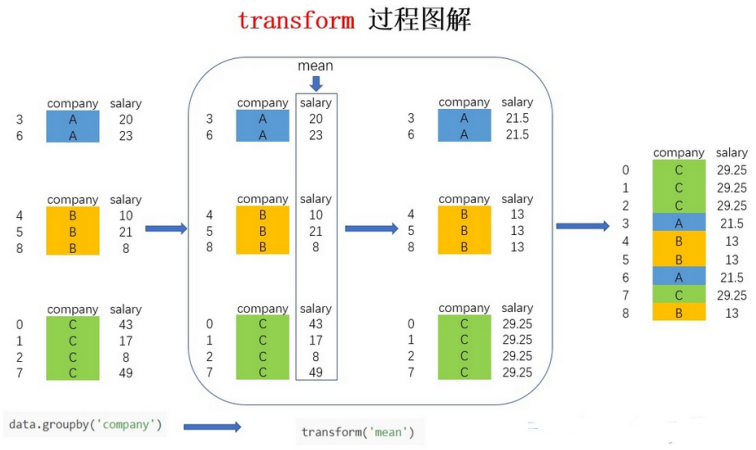
如果使用transform的话，仅需要一行代码：

```
data['avg_salary1'] = data.groupby('company')
['salary'].transform('mean')
data
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	company	salary	age	avg_salary	avg_salary1
0	A	32	18	29.800000	29.800000
1	A	30	29	29.800000	29.800000
2	B	34	38	37.333333	37.333333
3	C	44	37	36.000000	36.000000
4	B	30	31	37.333333	37.333333
5	C	28	19	36.000000	36.000000
6	A	44	26	29.800000	29.800000
7	A	6	34	29.800000	29.800000
8	B	48	18	37.333333	37.333333
9	A	37	33	29.800000	29.800000

还是以图解的方式来看看进行groupby后transform的实现过程（为了更直观展示，图中加入了company列，实际按照上面的代码只有salary列）：



图中的大方框是transform和agg所不一样的地方，对agg而言，会计算得到A，B，C公司对应的均值并直接返回，

但对transform而言，则会对每一条数据求得相应的结果，同一组内的样本会有相同的值，

组内求完均值后会按照原索引的顺序返回结果，如果有不理解的可以拿这张图和agg那张对比一下。

四、apply

它相比agg和transform而言更加灵活，能够传入任意自定义的函数，实现复杂的数据操作

对于groupby后的apply，以分组后的子DataFrame作为参数传入指定函数的，基本操作单位是DataFrame

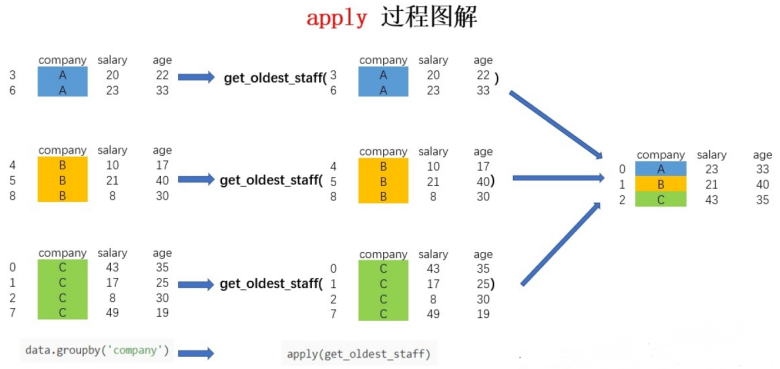
假设我现在需要获取各个公司年龄最大的员工的数据，该怎么实现呢？可以用以下代码实现：

```
def get_oldest_staff(x):
    # 输入的数据按照age字段进行排序
    df = x.sort_values(by = 'age',ascending=True)
    # 返回最后一条数据
    return df.iloc[-1,:]

oldest_staff =
data.groupby('company',as_index=False).apply(get_oldest_staff)
oldest_staff
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	company	salary	age	avg_salary	avg_salary1
company					
A	A	6	34	29.800000	29.800000
B	B	34	38	37.333333	37.333333
C	C	44	37	36.000000	36.000000

这样便得到了每个公司年龄最大的员工的数据，整个流程图解如下：



虽然说apply拥有更大的灵活性，但apply的运行效率会比agg和transform更慢。所以，groupby之后能用agg和transform解决的问题还是优先使用这两个方法，实在解决不了了才考虑使用apply进行操作