> 教你学会 Pandas 不是我的目的，**教你轻松玩转 Pandas 才是我的目的。**我会通过一系列实例来带入 Pandas 的知识点，让你在学习 Pandas 的路上不再枯燥。
>
> 声明：我所写的**轻松玩转 Pandas 教程都是免费的**，如果对你有帮助，你可以持续关注我。

数据分析过程中，最常见的一个场景莫过于分组统计了，这个也是学习 Pandas 的一个核心功能。我们一起来看一看吧。

In [1]:
```
# 导入相关库
import numpy as np
import pandas as pd
```

executed in 4ms, finished 02:01:21 2018-07-27

In [2]:
```
index = pd.Index(data=["Tom", "Bob", "Mary", "James", "Andy", "Alice"], name="name")

data = {
    "age": [18, 30, 35, 18, np.nan, 30],
    "city": ["Bei Jing ", "Shang Hai ", "Guang Zhou", "Shen Zhen", np.nan, " "],
    "sex": ["male", "male", "female", "male", np.nan, "female"],
    "income": [3000, 8000, 8000, 4000, 6000, 7000]
}

user_info = pd.DataFrame(data=data, index=index)
user_info
```

executed in 56ms, finished 02:01:21 2018-07-27

Out[2]:

name	age	city	income	sex
Tom	18.0	Bei Jing	3000	male
Bob	30.0	Shang Hai	8000	male
Mary	35.0	Guang Zhou	8000	female
James	18.0	Shen Zhen	4000	male
Andy	NaN	NaN	6000	NaN
Alice	30.0		7000	female

将对象分割成组

在进行分组统计前，首先要做的就是进行分组。既然是分组，就需要依赖于某个信息。

比如，依据性别来分组。直接调用 `user_info.groupby(user_info["sex"])` 即可完成按照性别分组。

In [3]:
```
grouped  = user_info.groupby(user_info["sex"])
grouped.groups
```

executed in 15ms, finished 02:01:21 2018-07-27

Out[3]:
```
{'female': Index(['Mary', 'Alice'], dtype='object', name='name'),
 'male': Index(['Tom', 'Bob', 'James'], dtype='object', name='name')}
```

可以看到，已经能够正确的按照性别来进行分组了。通常我们为了更简单，会使用这种方式来实现相同的功能：`user_info.groupby("sex")` 。

In [4]:
```
grouped  = user_info.groupby("sex")
grouped.groups
```

executed in 21ms, finished 02:01:21 2018-07-27

Out[4]:
```
{'female': Index(['Mary', 'Alice'], dtype='object', name='name'),
 'male': Index(['Tom', 'Bob', 'James'], dtype='object', name='name')}
```

你可能会想，能不能先按照性别来分组，再按照年龄进一步分组呢？答案是可以的，看这里。

In [5]:
```
grouped  = user_info.groupby(["sex", "age"])
grouped.groups
```

executed in 32ms, finished 02:01:21 2018-07-27

Out[5]:
```
{('female', 30.0): Index(['Alice'], dtype='object', name='name'),
 ('female', 35.0): Index(['Mary'], dtype='object', name='name'),
 ('male', 18.0): Index(['Tom', 'James'], dtype='object', name='name'),
 ('male', 30.0): Index(['Bob'], dtype='object', name='name'),
 (nan, nan): Index(['Andy'], dtype='object', name='name')}
```

关闭排序

默认情况下，`groupby` 会在操作过程中对数据进行排序。如果为了更好的性能，可以设置 `sort=False`。

In [6]:
```
grouped  = user_info.groupby(["sex", "age"], sort=False)
grouped.groups
```

executed in 22ms, finished 02:01:21 2018-07-27

Out[6]:
```
{('female', 30.0): Index(['Alice'], dtype='object', name='name'),
 ('female', 35.0): Index(['Mary'], dtype='object', name='name'),
 ('male', 18.0): Index(['Tom', 'James'], dtype='object', name='name'),
 ('male', 30.0): Index(['Bob'], dtype='object', name='name'),
 (nan, nan): Index(['Andy'], dtype='object', name='name')}
```

选择列

在使用 groupby 进行分组后，可以使用切片 [] 操作来完成对某一列的选择。

In [7]:
```
grouped  = user_info.groupby("sex")
grouped
```

executed in 31ms, finished 02:01:21 2018-07-27

Out[7]:
```
<pandas.core.groupby.DataFrameGroupBy object at 0x000002E355D787B8>
```

In [8]:
```
grouped["city"]
```

executed in 39ms, finished 02:01:21 2018-07-27

Out[8]:
```
<pandas.core.groupby.SeriesGroupBy object at 0x000002E355D78470>
```

遍历分组

在对数据进行分组后，可以进行遍历。

In [9]:
```
grouped  = user_info.groupby("sex")

for name, group in grouped:
    print("name: {}".format(name))
    print("group: {}".format(group))
    print("--------------")
```

executed in 37ms, finished 02:01:21 2018-07-27

```
name: female
group:         age        city  income     sex
name
Mary   35.0  Guang Zhou    8000  female
Alice  30.0                7000  female
--------------
name: male
group:         age        city  income   sex
name
Tom    18.0   Bei Jing     3000  male
Bob    30.0  Shang Hai     8000  male
James  18.0   Shen Zhen    4000  male
--------------
```

如果是根据多个字段来分组的，每个组的名称是一个元组。

In [10]:
```
grouped  = user_info.groupby(["sex", "age"])

for name, group in grouped:
    print("name: {}".format(name))
    print("group: {}".format(group))
    print("--------------")
```

executed in 42ms, finished 02:01:21 2018-07-27

```
name: ('female', 30.0)
group:         age city  income     sex
name
Alice  30.0         7000  female
--------------
name: ('female', 35.0)
group:        age        city  income     sex
name
Mary  35.0  Guang Zhou    8000  female
--------------
name: ('male', 18.0)
group:         age       city  income   sex
name
Tom    18.0  Bei Jing     3000  male
James  18.0  Shen Zhen    4000  male
--------------
name: ('male', 30.0)
group:        age        city  income   sex
name
Bob   30.0  Shang Hai     8000  male
--------------
```

选择一个组

分组后，我们可以通过 `get_group` 方法来选择其中的某一个组。

In [11]:
```
grouped  = user_info.groupby("sex")
grouped.get_group("male")
```

executed in 22ms, finished 02:01:21 2018-07-27

Out[11]:

name	age	city	income	sex
Tom	18.0	Bei Jing	3000	male
Bob	30.0	Shang Hai	8000	male
James	18.0	Shen Zhen	4000	male

In [12]:
```
user_info.groupby(["sex", "age"]).get_group(("male", 18))
```

executed in 30ms, finished 02:01:21 2018-07-27

Out[12]:

name	age	city	income	sex
Tom	18.0	Bei Jing	3000	male
James	18.0	Shen Zhen	4000	male

聚合

分组的目的是为了统计，统计的时候需要聚合，所以我们需要在分完组后来看下如何进行聚合。常见的一些聚合操作有：计数、求和、最大值、最小值、平均值等。

想要实现聚合操作，一种方式就是调用 `agg` 方法。

In [13]:
```
# 获取不同性别下所包含的人数
grouped = user_info.groupby("sex")
grouped["age"].agg(len)
```

executed in 55ms, finished 02:01:21 2018-07-27

Out[13]:
```
sex
female    2.0
male      3.0
Name: age, dtype: float64
```

In [14]:
```
# 获取不同性别下包含的最大的年龄
grouped = user_info.groupby("sex")
grouped["age"].agg(np.max)
```

executed in 24ms, finished 02:01:22 2018-07-27

Out[14]:
```
sex
female    35.0
male      30.0
Name: age, dtype: float64
```

如果是根据多个键来进行聚合，默认情况下得到的结果是一个多层索引结构。

In [15]:
```
grouped = user_info.groupby(["sex", "age"])
rs = grouped.agg(len)
rs
```

executed in 31ms, finished 02:01:22 2018-07-27

Out[15]:

		city	income
sex	**age**		
female	**30.0**	1	1
	35.0	1	1
male	**18.0**	2	2
	30.0	1	1

有两种方式可以避免出现多层索引，先来介绍第一种。对包含多层索引的对象调用 `reset_index` 方法。

In [16]:
```
rs.reset_index()
```

executed in 28ms, finished 02:01:22 2018-07-27

Out[16]:

	sex	age	city	income
0	female	30.0	1	1
1	female	35.0	1	1
2	male	18.0	2	2
3	male	30.0	1	1

另外一种方式是在分组时，设置参数 `as_index=False`。

In [17]:
```
grouped = user_info.groupby(["sex", "age"], as_index=False)
grouped.agg(len)
```

executed in 33ms, finished 02:01:22 2018-07-27

Out[17]:

	sex	age	city	income
0	female	30.0	1	1
1	female	35.0	1	1
2	male	18.0	2	2
3	male	30.0	1	1

Series 和 DataFrame 都包含了 `describe` 方法，我们分组后一样可以使用 `describe` 方法来查看数据的情况。

In [18]:
```
grouped = user_info.groupby("sex")
grouped.describe()
```

executed in 68ms, finished 02:01:22 2018-07-27

Out[18]:

	age								income							
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**count**	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
sex																
female	2.0	32.5	3.535534	30.0	31.25	32.5	33.75	35.0	2.0	7500.0	707.106781	7000.0	7250.0	7500.0	7750.0	8000.0
male	3.0	22.0	6.928203	18.0	18.00	18.0	24.00	30.0	3.0	5000.0	2645.751311	3000.0	3500.0	4000.0	6000.0	8000.0

一次应用多个聚合操作

有时候进行分组后，不单单想得到一个统计结果，有可能是多个。比如想统计出不同性别下的一个收入的总和和平均值。

In [19]:
```
grouped = user_info.groupby("sex")
grouped["income"].agg([np.sum, np.mean])
```

executed in 28ms, finished 02:01:22 2018-07-27

Out[19]:

	sum	mean
sex		
female	15000	7500
male	15000	5000

如果想将统计结果进行重命名，可以传入字典。

In [20]:
```
grouped = user_info.groupby("sex")
grouped["income"].agg([np.sum, np.mean]).rename(columns={"sum": "income_sum", "mean": "income_mean"})
```

executed in 29ms, finished 02:01:22 2018-07-27

Out[20]:

	income_sum	income_mean
sex		
female	15000	7500
male	15000	5000

对DataFrame列应用不同的聚合操作

有时候可能需要对不同的列使用不同的聚合操作。例如，想要统计不同性别下人群的年龄的均值以及收入的总和。

In [21]:
```
grouped = user_info.groupby("sex")
grouped.agg({"age": np.mean, "income": np.sum}).rename(columns={"age": "age_mean", "income": "income_sum"})
```

executed in 30ms, finished 02:01:22 2018-07-27

Out[21]:

	age_mean	income_sum
sex		
female	32.5	15000
male	22.0	15000

transform 操作

前面进行聚合运算的时候，得到的结果是一个以分组名作为索引的结果对象。虽然可以指定 `as_index=False` ,但是得到的索引也并不是元数据的索引。如果我们想使用原数组的索引的话，就需要进行 merge 转换。

transform方法简化了这个过程，它会把 func 参数应用到所有分组，然后把结果放置到原数组的索引上（如果结果是一个标量，就进行广播）

In [22]:
```
# 通过 agg 得到的结果的索引是分组名
grouped = user_info.groupby("sex")
grouped["income"].agg(np.mean)
```

executed in 25ms, finished 02:01:22 2018-07-27

Out[22]:
```
sex
female    7500
male      5000
Name: income, dtype: int64
```

In [23]:
```
# 通过 transform 得到的结果的索引是原始索引，它会将得到的结果自动关联上原始的索引
grouped = user_info.groupby("sex")
grouped["income"].transform(np.mean)
```

executed in 22ms, finished 02:01:22 2018-07-27

Out[23]:
```
name
Tom      5000.0
Bob      5000.0
Mary     7500.0
James    5000.0
Andy        NaN
Alice    7500.0
Name: income, dtype: float64
```

可以看到，通过 transform 操作得到的结果的长度与原来保持一致。

apply 操作

除了 transform 操作外，还有更神奇的 apply 操作。

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，最后尝试用 pd.concat() 把结果组合起来。func 的返回值可以是 Pandas 对象或标量，并且数组对象的大小不限。

In [24]:
```
# 使用 apply 来完成上面的聚合
grouped = user_info.groupby("sex")
grouped["income"].apply(np.mean)
```

executed in 20ms, finished 02:01:22 2018-07-27

Out[24]:
```
sex
female    7500.0
male      5000.0
Name: income, dtype: float64
```

来看下 apply 不一样的用法吧。

比如想要统计不同性别最高收入的前n个值，可以通过下面这种方式实现。

In [25]:
```
def f1(ser, num=2):
    return ser.nlargest(num).tolist()

grouped["income"].apply(f1)
```

executed in 24ms, finished 02:01:22 2018-07-27

Out[25]:
```
sex
female    [8000, 7000]
male      [8000, 4000]
Name: income, dtype: object
```

另外，如果想要获取不同性别下的年龄的均值，通过 apply 可以如下实现。

In [26]:
```
def f2(df):
    return df["age"].mean()

grouped.apply(f2)
```

executed in 26ms, finished 02:01:22 2018-07-27

Out[26]:
```
sex
female    32.5
male      22.0
dtype: float64
```

这里我将整篇文章的内容整理成了pdf，想要pdf文件的可以在公众号后台回复关键字：**pandas**。