> 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。

Pandas 有很多高级的功能，但是想要掌握高级功能前，需要先掌握它的基础知识，Pandas 中的数据结构算是非常基础的知识之一了。

Pandas 常用的数据结构有两种：Series 和 DataFrame。这些数据结构构建在 Numpy 数组之上，这意味着它们效率很高。我们来分别看看这些数据结构都长什么样子吧。

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

executed in 5ms, finished 15:22:11 2018-06-04

Series

简介

Series 是一个带有 **名称** 和索引的一维数组，既然是数组，肯定要说到的就是数组中的元素类型，在 Series 中包含的数据类型可以是整数、浮点、字符串、Python对象等。

假定有一个场景是：存储一些用户的信息，暂时只包括年龄信息。

我们可以通过 Series 来存储，这里我们通过 Series 存储了四个年龄：18/30/25/40，**只需将要存储的数据构建成一个数组，然后赋值给data参数即可。**

In [2]:
```
# 存储了 4 个年龄：18/30/25/40
user_age = pd.Series(data=[18, 30, 25, 40])
user_age
```

executed in 49ms, finished 15:22:11 2018-06-04

Out[2]:
```
0    18
1    30
2    25
3    40
dtype: int64
```

可以看到，已经正确将多个年龄存储到 Series 中了，你可能会想，单独存储了年龄有什么用，我怎么知道这个年龄属于哪个用户呢？

我们可以通过 Series 的 index（索引）来解决这个问题。由于有四个年龄，自然地也需要四个姓名，所以我们**需要构建一个与 data 长度相同的数组**，然后通过下面的操作即可满足要求。

In [3]:
```
user_age.index = ["Tom", "Bob", "Mary", "James"]
user_age
```

executed in 30ms, finished 15:22:11 2018-06-04

Out[3]:
```
Tom      18
Bob      30
Mary     25
James    40
dtype: int64
```

你看，现在姓名与年龄已经完全对应上了。虽然说我们自己知道 Tom/Bob 这些是姓名，但是别人不知道啊，我们怎么告诉他人呢？

要想让别人知道，我们可以为 index 起个名字。

In [4]:
```
user_age.index.name = "name"
user_age
```

executed in 35ms, finished 15:22:11 2018-06-04

Out[4]:
```
name
Tom      18
Bob      30
Mary     25
James    40
dtype: int64
```

可能你还会想，如果别人在看我写的代码，怎么能快速的知道我这写的到底是什么玩意呢？

别急，就像我们给index起名字一样，我们也可以给 Series 起个名字。

In [5]:
```
user_age.name="user_age_info"
user_age
```

executed in 22ms, finished 15:22:11 2018-06-04

Out[5]:
```
name
Tom      18
Bob      30
Mary     25
James    40
Name: user_age_info, dtype: int64
```

通过上面一系列的操作，我们对 Series 的结构上有了基本的了解，简单来说，**一个 Series 包括了 data、index 以及 name。**

上面的操作非常方便做演示来使用，如果想要快速实现上面的功能，可以通过以下方式来实现。

In [6]:
```
# 构建索引
name = pd.Index(["Tom", "Bob", "Mary", "James"], name="name")
# 构建 Series
user_age = pd.Series(data=[18, 30, 25, 40], index=name, name="user_age_info")
user_age
```

executed in 23ms, finished 15:22:11 2018-06-04

Out[6]:
```
name
Tom      18
Bob      30
Mary     25
James    40
Name: user_age_info, dtype: int64
```

另外，需要说明的是我们在构造 Series 的时候，并没有设定每个元素的数据类型，这个时候，pandas 会自动判断一个数据类型，并作为 Series 的类型。

当然了，我们也可以自己手动指定数据类型。

In [7]:
```
# 指定类型为浮点型
user_age = pd.Series(data=[18, 30, 25, 40], index=name, name="user_age_info", dtype=float)
user_age
```

executed in 25ms, finished 15:22:11 2018-06-04

Out[7]:
```
name
Tom      18.0
Bob      30.0
Mary     25.0
James    40.0
Name: user_age_info, dtype: float64
```

Series 像什么

Series 包含了 dict 的特点，也就意味着可以使用与 dict 类似的一些操作。我们可以将 index 中的元素看成是 dict 中的 key。

In [8]:
```
# 获取 Tom 的年龄
user_age["Tom"]
```

executed in 31ms, finished 15:22:11 2018-06-04

Out[8]: 18.0

此外，可以通过 get 方法来获取。通过这种方式的好处是当索引不存在时，不会抛出异常。

In [9]:
```
user_age.get("Tom")
```

executed in 32ms, finished 15:22:11 2018-06-04

Out[9]: 18.0

Series 除了像 dict 外，也非常像 ndarray，这也就意味着可以采用切片操作。

In [10]:
```
# 获取第一个元素
user_age[0]
```

executed in 38ms, finished 15:22:11 2018-06-04

Out[10]: 18.0

In [11]:
```
# 获取前三个元素
user_age[:3]
```

executed in 39ms, finished 15:22:11 2018-06-04

Out[11]:
```
name
Tom     18.0
Bob     30.0
Mary    25.0
Name: user_age_info, dtype: float64
```

In [12]:
```
# 获取年龄大于30的元素
user_age[user_age > 30]
```

executed in 25ms, finished 15:22:11 2018-06-04

Out[12]:
```
name
James    40.0
Name: user_age_info, dtype: float64
```

In [13]:
```
# 获取第4个和第二个元素
user_age[[3, 1]]
```

executed in 22ms, finished 15:22:11 2018-06-04

Out[13]:
```
name
James    40.0
Bob      30.0
Name: user_age_info, dtype: float64
```

可以看到，无论我们通过切片如何操作 Series ，它都能够自动对齐 index。

Series 的向量化操作

Series 与 ndarray 一样，也是支持向量化操作的。同时也可以传递给大多数期望 ndarray 的 NumPy 方法。

In [14]:
```
user_age + 1
```

executed in 20ms, finished 15:22:11 2018-06-04

Out[14]:
```
name
Tom      19.0
Bob      31.0
Mary     26.0
James    41.0
Name: user_age_info, dtype: float64
```

In [15]:
```
np.exp(user_age)
```

executed in 31ms, finished 15:22:11 2018-06-04

Out[15]:
```
name
Tom      6.565997e+07
Bob      1.068647e+13
Mary     7.200490e+10
James    2.353853e+17
Name: user_age_info, dtype: float64
```

DataFrame

DataFrame 是一个带有**索引**的二维数据结构，每列可以有自己的名字，并且可以有不同的数据类型。你可以把它想象成一个 excel 表格或者数据库中的一张表，DataFrame 是最常用的 pandas 对象。

我们继续使用之前的实例来讲解 DataFrame，在存储用户信息时，除了年龄之外，我还想存储用户所在的城市。如何通过 DataFrame 实现呢？

可以构建一个 dict，key 是需要存储的信息，value 是信息列表。**然后将 dict 传递给 data 参数。**

In [16]:
```
index = pd.Index(data=["Tom", "Bob", "Mary", "James"], name="name")

data = {
    "age": [18, 30, 25, 40],
    "city": ["BeiJing", "ShangHai", "GuangZhou", "ShenZhen"]
}

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

executed in 97ms, finished 15:22:11 2018-06-04

Out[16]:

name	age	city
Tom	18	BeiJing
Bob	30	ShangHai
Mary	25	GuangZhou
James	40	ShenZhen

可以看到，我们成功构建了一个 DataFrame，这个 DataFrame 的索引是用户性别，还有两列分别是用户的年龄和城市信息。

除了上面这种传入 dict 的方式构建外，我们还可以通过另外一种方式来构建。这种方式是**先构建一个二维数组，然后再生成一个列名称列表。**

In [17]:
```
data = [[18, "BeiJing"],
        [30, "ShangHai"],
        [25, "GuangZhou"],
        [40, "ShenZhen"]]
columns = ["age", "city"]

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

executed in 30ms, finished 15:22:11 2018-06-04

Out[17]:

name	age	city
Tom	18	BeiJing
Bob	30	ShangHai
Mary	25	GuangZhou
James	40	ShenZhen

访问行

在生成了 DataFrame 之后，可以看到，每一行就表示某一个用户的信息，假如我想要访问 Tom 的信息，我该如何操作呢？

一种办法是通过索引名来访问某行，这种办法需要借助 loc 方法。

In [18]:
```
user_info.loc["Tom"]
```

executed in 39ms, finished 15:22:11 2018-06-04

Out[18]:
```
age          18
city    BeiJing
Name: Tom, dtype: object
```

除了直接通过索引名来访问某一行数据之外，还可以通过这行所在的位置来选择这一行。

In [19]:
```
user_info.iloc[0]
```

executed in 24ms, finished 15:22:11 2018-06-04

Out[19]:
```
age          18
city    BeiJing
Name: Tom, dtype: object
```

现在能够访问某一个用户的信息了，那么我如何访问多个用户的信息呢？也就是如何访问多行呢？

借助行切片可以轻松完成，来看这里。

In [20]:
```
user_info.iloc[1:3]
```

executed in 29ms, finished 15:22:11 2018-06-04

Out[20]:

	age	city
name		
Bob	30	ShangHai
Mary	25	GuangZhou

访问列

学会了如何访问行数据之外，自然而然会想到如何访问列。我们可以通过属性（“.列名”）的方式来访问该列的数据，也可以通过[column]的形式来访问该列的数据。

假如我想获取所有用户的年龄，那么可以这样操作。

In [21]:
```
user_info.age
```

executed in 59ms, finished 15:22:11 2018-06-04

Out[21]:
```
name
Tom      18
Bob      30
Mary     25
James    40
Name: age, dtype: int64
```

In [22]:
```
user_info["age"]
```

executed in 39ms, finished 15:22:11 2018-06-04

Out[22]:
```
name
Tom      18
Bob      30
Mary     25
James    40
Name: age, dtype: int64
```

如果想要同时获取年龄和城市该如何操作呢？

In [23]:
```
# 可以变换列的顺序
user_info[["city", "age"]]
```

executed in 33ms, finished 15:22:11 2018-06-04

Out[23]:

	city	age
name		
Tom	BeiJing	18
Bob	ShangHai	30
Mary	GuangZhou	25
James	ShenZhen	40

新增/删除列

在生成了 DataFrame 之后，突然你发现好像缺失了用户的性别这个信息，那么如何添加呢？

如果所有的性别都一样，我们可以通过传入一个标量，pandas 会自动帮我们广播来填充所有的位置。

In [24]:
```
user_info["sex"] = "male"
user_info
```

executed in 28ms, finished 15:22:12 2018-06-04

Out[24]:

name	age	city	sex
Tom	18	BeiJing	male
Bob	30	ShangHai	male
Mary	25	GuangZhou	male
James	40	ShenZhen	male

如果想要删除某一列，可以使用 pop 方法来完成。

In [25]:
```
user_info.pop("sex")
user_info
```

executed in 30ms, finished 15:22:12 2018-06-04

Out[25]:

name	age	city
Tom	18	BeiJing
Bob	30	ShangHai
Mary	25	GuangZhou
James	40	ShenZhen

如果用户的性别不一致的时候，我们可以通过传入一个 like-list 来添加新的一列。

In [26]:
```
user_info["sex"] = ["male", "male", "female", "male"]
user_info
```

executed in 27ms, finished 15:22:12 2018-06-04

Out[26]:

name	age	city	sex
Tom	18	BeiJing	male
Bob	30	ShangHai	male
Mary	25	GuangZhou	female
James	40	ShenZhen	male

通过上面的例子可以看出，我们创建新的列的时候都是在原有的 DataFrame 上修改的，也就是说如果添加了新的一列之后，原有的 DataFrame 会发生改变。

如果想要保证原有的 DataFrame 不改变的话，我们可以通过 assign 方法来创建新的一列。

In [27]:
```
user_info.assign(age_add_one = user_info["age"] + 1)
```
executed in 61ms, finished 15:22:12 2018-06-04

Out[27]:

	age	city	sex	age_add_one
name				
Tom	18	BeiJing	male	19
Bob	30	ShangHai	male	31
Mary	25	GuangZhou	female	26
James	40	ShenZhen	male	41

In [28]:
```
user_info.assign(sex_code = np.where(user_info["sex"] == "male", 1, 0))
```
executed in 34ms, finished 15:22:12 2018-06-04

Out[28]:

	age	city	sex	sex_code
name				
Tom	18	BeiJing	male	1
Bob	30	ShangHai	male	1
Mary	25	GuangZhou	female	0
James	40	ShenZhen	male	1

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：**pandas01**。