> 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。

在 01-Pandas数据结构详解 (01-Pandas数据结构详解.ipynb) 介绍了 Pandas 中常用的两种数据结构 Series 以及 DataFrame，这里来看下这些数据结构都有哪些常用的功能。

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

executed in 8ms, finished 06:48:12 2018-06-15

常用的基本功能

当我们构建好了 Series 和 DataFrame 之后，我们会经常使用哪些功能呢？来跟我看看吧。引用上一章节中的场景，我们有一些用户的的信息，并将它们存储到了 DataFrame 中。

因为大多数情况下 DataFrame 比 Series 更为常用，所以这里以 DataFrame 举例说明，但实际上很多常用功能对于 Series 也适用。

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

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

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

executed in 64ms, finished 06:48:12 2018-06-15

Out[2]:

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

一般拿到数据，我们第一步需要做的是了解下数据的整体情况，可以使用 `info` 方法来查看。

In [3]:
```
user_info.info()
```

executed in 24ms, finished 06:48:12 2018-06-15

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, Tom to James
Data columns (total 3 columns):
age     4 non-null int64
city    4 non-null object
sex     4 non-null object
dtypes: int64(1), object(2)
memory usage: 128.0+ bytes
```

如果我们的数据量非常大，我想看看数据长啥样，我当然不希望查看所有的数据了，这时候我们可以采用只看头部的 n 条或者尾部的 n 条。查看头部的 n 条数据可以使用 `head` 方法，查看尾部的 n 条数据可以使用 `tail` 方法。

In [4]:
```
user_info.head(2)
```

executed in 36ms, finished 06:48:12 2018-06-15

Out[4]:

	age	city	sex
name			
Tom	18	BeiJing	male
Bob	30	ShangHai	male

此外，Pandas 中的数据结构都有 ndarray 中的常用方法和属性，如通过 `.shape` 获取数据的形状，通过 `.T` 获取数据的转置。

In [5]:
```
user_info.shape
```

executed in 20ms, finished 06:48:12 2018-06-15

Out[5]: (4, 3)

In [6]:
```
user_info.T
```

executed in 44ms, finished 06:48:12 2018-06-15

Out[6]:

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

如果我们想要通过 DataFrame 来获取它包含的原有数据，可以通过 `.values` 来获取，获取后的数据类型其实是一个 ndarray。

In [7]:
```
user_info.values
```

executed in 32ms, finished 06:48:12 2018-06-15

Out[7]:
```
array([[18, 'BeiJing', 'male'],
       [30, 'ShangHai', 'male'],
       [25, 'GuangZhou', 'female'],
       [40, 'ShenZhen', 'male']], dtype=object)
```

描述与统计

有时候我们获取到数据之后，想要查看下数据的简单统计指标（最大值、最小值、平均值、中位数等），比如想要查看年龄的最大值，如何实现呢？

直接对 age 这一列调用 max方法即可。

In [8]:
```
user_info.age.max()
```

executed in 24ms, finished 06:48:13 2018-06-15

Out[8]:
```
40
```

类似的，通过调用 min、mean、quantile、sum 方法可以实现最小值、平均值、中位数以及求和。可以看到，对一个 Series 调用 这几个方法之后，返回的都只是一个聚合结果。

来介绍个有意思的方法：cumsum，看名字就发现它和 sum 方法有关系，事实上确实如此，cumsum 也是用来求和的，不过它是用来**累加求和**的，也就是说它得到的结果与原始的 Series 或 DataFrame 大小相同。

In [9]:
```
user_info.age.cumsum()
```

executed in 24ms, finished 06:48:13 2018-06-15

Out[9]:
```
name
Tom       18
Bob       48
Mary      73
James    113
Name: age, dtype: int64
```

可以看到，cummax 最后的结果就是将上一次求和的结果与原始当前值求和作为当前值。这话听起来有点绕。举个例子，上面的 73 = 48 + 25。cumsum

也可以用来操作字符串类型的对象。

In [10]:
```
user_info.sex.cumsum()
```

executed in 28ms, finished 06:48:13 2018-06-15

Out[10]:
```
name
Tom                     male
Bob                 malemale
Mary          malemalefemale
James     malemalefemalemale
Name: sex, dtype: object
```

如果想要获取更多的统计方法，可以参见官方链接：Descriptive statistics (http://pandas.pydata.org/pandas-docs/stable/basics.html#descriptive-statistics)

虽然说常见的各种统计值都有对应的方法，如果我想要得到多个指标的话，就需要调用多次方法，是不是显得有点麻烦呢？

Pandas 设计者自然也考虑到了这个问题，**想要一次性获取多个统计指标，只需调用** `describe` **方法即可。**

In [11]:
```
user_info.describe()
```

executed in 36ms, finished 06:48:13 2018-06-15

Out[11]:

	age
count	4.000000
mean	28.250000
std	9.251126
min	18.000000
25%	23.250000
50%	27.500000
75%	32.500000
max	40.000000

可以看到，直接调用 describe 方法后，会显示出数字类型的列的一些统计指标，如 总数、平均数、标准差、最小值、最大值、25%/50%/75% 分位数。如果想要查看非数字类型的列的统计指标，可以设置 include=["object"] 来获得。

In [12]:
```
user_info.describe(include=["object"])
```

executed in 32ms, finished 06:48:13 2018-06-15

Out[12]:

	city	sex
count	4	4
unique	4	2
top	BeiJing	male
freq	1	3

上面的结果展示了非数字类型的列的一些统计指标：总数，去重后的个数、最常见的值、最常见的值的频数。

此外，如果我想要**统计下某列中每个值出现的次数**，如何快速实现呢？调用 value_counts 方法快速获取 Series 中每个值出现的次数。

In [13]:
```
user_info.sex.value_counts()
```
executed in 20ms, finished 06:48:13 2018-06-15

Out[13]:
```
male      3
female    1
Name: sex, dtype: int64
```

如果想要获取某列最大值或最小值对应的索引，可以使用 idxmax 或 idxmin 方法完成。

In [14]:
```
user_info.age.idxmax()
```
executed in 24ms, finished 06:48:13 2018-06-15

Out[14]:
```
'James'
```

离散化

有时候，我们会碰到这样的需求，想要将年龄进行离散化（分桶），直白来说就是将年龄分成几个区间，这里我们想要将年龄分成 3 个区间段。就可以使用 Pandas 的 cut 方法来完成。

In [15]:
```
pd.cut(user_info.age, 3)
```
executed in 40ms, finished 06:48:13 2018-06-15

Out[15]:
```
name
Tom      (17.978, 25.333]
Bob      (25.333, 32.667]
Mary     (17.978, 25.333]
James      (32.667, 40.0]
Name: age, dtype: category
Categories (3, interval[float64]): [(17.978, 25.333] < (25.333, 32.667] < (32.667, 40.0]]
```

可以看到， cut 自动生成了等距的离散区间，如果自己想定义也是没问题的。

In [16]:
```
pd.cut(user_info.age, [1, 18, 30, 50])
```

executed in 20ms, finished 06:48:13 2018-06-15

Out[16]:
```
name
Tom       (1, 18]
Bob      (18, 30]
Mary     (18, 30]
James    (30, 50]
Name: age, dtype: category
Categories (3, interval[int64]): [(1, 18] < (18, 30] < (30, 50]]
```

有时候离散化之后，想要给每个区间起个名字，可以指定 labels 参数。

In [17]:
```
pd.cut(user_info.age, [1, 18, 30, 50], labels=["childhood", "youth", "middle"])
```

executed in 20ms, finished 06:48:13 2018-06-15

Out[17]:
```
name
Tom      childhood
Bob          youth
Mary         youth
James       middle
Name: age, dtype: category
Categories (3, object): [childhood < youth < middle]
```

除了可以使用 cut 进行离散化之外，qcut 也可以实现离散化。cut **是根据每个值的大小来进行离散化的，**qcut **是根据每个值出现的次数来进行离散化的。**

In [18]:
```
pd.qcut(user_info.age, 3)
```

executed in 25ms, finished 06:48:13 2018-06-15

Out[18]:
```
name
Tom      (17.999, 25.0]
Bob        (25.0, 30.0]
Mary     (17.999, 25.0]
James      (30.0, 40.0]
Name: age, dtype: category
Categories (3, interval[float64]): [(17.999, 25.0] < (25.0, 30.0] < (30.0, 40.0]]
```

排序功能

在进行数据分析时，少不了进行数据排序。Pandas 支持两种排序方式：按轴（索引或列）排序和按实际值排序。

先来看下按索引排序：`sort_index` 方法默认是按照索引进行正序排的。

In [19]:
```
user_info.sort_index()
```

executed in 20ms, finished 06:48:13 2018-06-15

Out[19]:

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

如果想要按照列进行倒序排，可以设置参数 `axis=1` 和 `ascending=False`。

In [20]:
```
user_info.sort_index(axis=1, ascending=False)
```

executed in 34ms, finished 06:48:13 2018-06-15

Out[20]:

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

如果想要实现按照实际值来排序，例如想要按照年龄排序，如何实现呢？

使用 `sort_values` 方法，设置参数 `by="age"` 即可。

In [21]:
```
user_info.sort_values(by="age")
```

executed in 24ms, finished 06:48:13 2018-06-15

Out[21]:

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

有时候我们可能需要按照多个值来排序，例如：按照年龄和城市来一起排序，可以设置参数 by 为一个 list 即可。

注意：**list 中每个元素的顺序会影响排序优先级的。**

In [22]:
```
user_info.sort_values(by=["age", "city"])
```

executed in 28ms, finished 06:48:13 2018-06-15

Out[22]:

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

一般在排序后，我们可能需要获取最大的n个值或最小值的n个值，我们可以使用 `nlargest` 和 `nsmallest` 方法来完成，这比先进行排序，再使用 `head(n)` 方法快得多。

In [23]:
```
user_info.age.nlargest(2)
```

executed in 24ms, finished 06:48:13 2018-06-15

Out[23]:
```
name
James    40
Bob      30
Name: age, dtype: int64
```

函数应用

虽说 Pandas 为我们提供了非常丰富的函数，有时候我们可能需要自己定制一些函数，并将它应用到 DataFrame 或 Series。常用到的函数有：`map`、`apply`、`applymap`。

`map` 是 Series 中特有的方法，通过它可以对 Series 中的每个元素实现转换。

如果我想通过年龄判断用户是否属于中年人（30岁以上为中年），通过 `map` 可以轻松搞定它。

In [24]:
```
# 接收一个 lambda 函数
user_info.age.map(lambda x: "yes" if x >= 30 else "no")
```

executed in 24ms, finished 06:48:13 2018-06-15

Out[24]:
```
name
Tom       no
Bob      yes
Mary      no
James    yes
Name: age, dtype: object
```

又比如，我想要通过城市来判断是南方还是北方，我可以这样操作。

In [25]:
```
city_map = {
    "BeiJing": "north",
    "ShangHai": "south",
    "GuangZhou": "south",
    "ShenZhen": "south"
}

# 传入一个 map
user_info.city.map(city_map)
```

executed in 22ms, finished 06:48:13 2018-06-15

Out[25]:
```
name
Tom      north
Bob      south
Mary     south
James    south
Name: city, dtype: object
```

apply 方法既支持 Series，也支持 DataFrame，在对 Series 操作时会作用到每个值上，在对 DataFrame 操作时会作用到所有行或所有列（通过 axis 参数控制）。

In [26]:
```
# 对 Series 来说，apply 方法 与 map 方法区别不大。
user_info.age.apply(lambda x: "yes" if x >= 30 else "no")
```

executed in 22ms, finished 06:48:13 2018-06-15

Out[26]:
```
name
Tom       no
Bob      yes
Mary      no
James    yes
Name: age, dtype: object
```

In [27]:
```
# 对 DataFrame 来说，apply 方法的作用对象是一行或一列数据（一个Series）
user_info.apply(lambda x: x.max(), axis=0)
```

executed in 20ms, finished 06:48:13 2018-06-15

Out[27]:
```
age           40
city    ShenZhen
sex         male
dtype: object
```

applymap 方法针对于 DataFrame，它作用于 DataFrame 中的每个元素，它对 DataFrame 的效果类似于 apply 对 Series 的效果。

In [28]:
```
user_info.applymap(lambda x: str(x).lower())
```

executed in 28ms, finished 06:48:13 2018-06-15

Out[28]:

	age	city	sex
name			
Tom	18	beijing	male
Bob	30	shanghai	male
Mary	25	guangzhou	female
James	40	shenzhen	male

修改列/索引名称

在使用 DataFrame 的过程中，经常会遇到修改列名，索引名等情况。使用 `rename` 轻松可以实现。

修改列名只需要设置参数 `columns` 即可。

In [29]:
```
user_info.rename(columns={"age": "Age", "city": "City", "sex": "Sex"})
```

executed in 27ms, finished 06:48:13 2018-06-15

Out[29]:

	Age	City	Sex
name			
Tom	18	BeiJing	male
Bob	30	ShangHai	male
Mary	25	GuangZhou	female
James	40	ShenZhen	male

类似的，修改索引名只需要设置参数 `index` 即可。

In [30]:
```
user_info.rename(index={"Tom": "tom", "Bob": "bob"})
```

executed in 24ms, finished 06:48:13 2018-06-15

Out[30]:

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

类型操作

如果想要获取每种类型的列数的话，可以使用 `get_dtype_counts` 方法。

In [31]:
```
user_info.get_dtype_counts()
```

executed in 20ms, finished 06:48:13 2018-06-15

Out[31]:
```
int64     1
object    2
dtype: int64
```

如果想要转换数据类型的话，可以通过 `astype` 来完成。

In [32]:
```
user_info["age"].astype(float)
```

executed in 20ms, finished 06:48:13 2018-06-15

Out[32]:
```
name
Tom      18.0
Bob      30.0
Mary     25.0
James    40.0
Name: age, dtype: float64
```

有时候会涉及到将 object 类型转为其他类型，常见的有转为数字、日期、时间差，Pandas 中分别对应 `to_numeric`、`to_datetime`、`to_timedelta` 方法。

这里给这些用户都添加一些关于身高的信息。

In [33]:
```
user_info["height"] = ["178", "168", "178", "180cm"]
user_info
```

executed in 24ms, finished 06:48:13 2018-06-15

Out[33]:

name	age	city	sex	height
Tom	18	BeiJing	male	178
Bob	30	ShangHai	male	168
Mary	25	GuangZhou	female	178
James	40	ShenZhen	male	180cm

现在将身高这一列转为数字，很明显，180cm 并非数字，为了强制转换，我们可以传入 errors 参数，这个参数的作用是当强转失败时的处理方式。

默认情况下，errors='raise'，这意味着强转失败后直接抛出异常，设置 errors='coerce' 可以在强转失败时将有问题的元素赋值为 pd.NaT（对于 datetime和timedelta）或 np.nan（数字）。设置 errors='ignore' 可以在强转失败时返回原有的数据。

In [34]:
```
pd.to_numeric(user_info.height, errors="coerce")
```

executed in 24ms, finished 06:48:13 2018-06-15

Out[34]:
```
name
Tom      178.0
Bob      168.0
Mary     178.0
James      NaN
Name: height, dtype: float64
```

In [35]:
```
pd.to_numeric(user_info.height, errors="ignore")
```

executed in 24ms, finished 06:48:13 2018-06-15

Out[35]:
```
name
Tom        178
Bob        168
Mary       178
James    180cm
Name: height, dtype: object
```

想要学习更多关于人工智能的知识，请关注公众号：**AI派**

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：**pandas02**。