> 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 04-Pandas文本数据处理 (04-Pandas文本数据处理.ipynb) 介绍了使用 Pandas 处理文本（字符串）数据，这节来看下分类（category）数据如何处理吧。

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

executed in 6ms, finished 09:33:51 2018-06-28

创建对象

在创建分类数据之前，先来了解下什么是分类（Category）数据呢？分类数据直白来说就是取值为有限的，或者说是固定数量的可能值。例如：性别、血型。

这里以血型为例，假定每个用户有以下的血型，我们如何创建一个关于血型的分类对象呢？

一种有效的方法就是明确指定 `dtype="category"`

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

user_info = pd.Series(data=["A", "AB", np.nan, "AB", "O", "B"], index=index, name="blood_type", dtype="category")
user_info
```

executed in 68ms, finished 09:33:51 2018-06-28

Out[2]:
```
name
Tom        A
Bob       AB
Mary     NaN
James     AB
Andy       O
Alice      B
Name: blood_type, dtype: category
Categories (4, object): [A, AB, B, O]
```

我们也可以使用 pd.Categorical 来构建分类数据。

In [3]:
```
pd.Categorical(["A", "AB", np.nan, "AB", "O", "B"])
```

executed in 26ms, finished 09:33:51 2018-06-28

Out[3]:
```
[A, AB, NaN, AB, O, B]
Categories (4, object): [A, AB, B, O]
```

当然了，我们也可以自己制定类别数据所有可能的取值，假定我们认为血型只有 A、B 以及 AB 这三类，那么我们可以这样操作。

In [4]:
```
pd.Categorical(["A", "AB", np.nan, "AB", "O", "B"], categories=["A", "B", "AB"])
```

executed in 11ms, finished 09:33:51 2018-06-28

Out[4]:
```
[A, AB, NaN, AB, NaN, B]
Categories (3, object): [A, B, AB]
```

除了上面这些方法外，经常遇到的情况是已经创建了一个 Series，如何将它转为分类数据呢？来看看 astype 用法吧。

In [5]:
```
user_info = pd.Series(data=["A", "AB", np.nan, "AB", "O", "B"], index=index, name="blood_type")
user_info = user_info.astype("category")
user_info
```

executed in 42ms, finished 09:33:51 2018-06-28

Out[5]:
```
name
Tom        A
Bob       AB
Mary     NaN
James     AB
Andy       O
Alice      B
Name: blood_type, dtype: category
Categories (4, object): [A, AB, B, O]
```

此外，一些其他的方法返回的结果也是分类数据。如 cut 、 qcut。具体可以见 Pandas基本功能详解中的离散化 (02-Pandas基本功能详解.ipynb#离散化)部分。

常用操作

可以对分类数据使用 .describe() 方法，它得到的结果与 string类型的数据相同。

In [6]:
```
user_info.describe()
```

executed in 72ms, finished 09:33:51 2018-06-28

Out[6]:
```
count      5
unique     4
top       AB
freq       2
Name: blood_type, dtype: object
```

解释下每个指标的含义，count 表示非空的数据有5条，unique 表示去重后的非空数据有4条，top 表示出现次数最多的值为 AB，freq 表示出现次数最多的值的次数为2次。

我们可以使用 .cat.categories 来获取分类数据所有可能的取值。

In [7]:
```
user_info.cat.categories
```

executed in 17ms, finished 09:33:51 2018-06-28

Out[7]:
```
Index([u'A', u'AB', u'B', u'O'], dtype='object')
```

你可能会发现，假如你将分类名称写错了，如何修改呢？难道还需要重新构建一次么？

不不不，不需要这么麻烦，你可以直接使用 .cat.rename_categories 方法来重命名分类名称。

In [8]:
```
user_info.cat.rename_categories(["A+", "AB+", "B+", "O+"])
```

executed in 38ms, finished 09:33:51 2018-06-28

Out[8]:
```
name
Tom       A+
Bob      AB+
Mary     NaN
James    AB+
Andy      O+
Alice     B+
Name: blood_type, dtype: category
Categories (4, object): [A+, AB+, B+, O+]
```

类似的，除了重命名，也会遇到添加类别，删除分类的操作，这些都可以通过 .cat.add_categories ，.cat.remove_categories 来实现。

分类数据也是支持使用 value_counts 方法来查看数据分布的。

In [9]:
```
user_info.value_counts()
```

executed in 19ms, finished 09:33:51 2018-06-28

Out[9]:
```
AB    2
O     1
B     1
A     1
Name: blood_type, dtype: int64
```

分类数据也是支持使用 .str 属性来访问的。例如想要查看下是否包含字母 "A"，可以使用 .srt.contains 方法。

In [10]:
```
user_info.str.contains("A")
```

executed in 39ms, finished 09:33:51 2018-06-28

Out[10]:
```
name
Tom       True
Bob       True
Mary       NaN
James     True
Andy     False
Alice    False
Name: blood_type, dtype: object
```

跟多关于 .str 的详细介绍可以见 Pandas文本数据处理 (04-Pandas文本数据处理.ipynb)。

有时候会遇到合并数据的情况，这时候可以借助 pd.concat 来完成。

In [11]:
```
blood_type1 = pd.Categorical(["A", "AB"])
blood_type2 = pd.Categorical(["B", "O"])

pd.concat([pd.Series(blood_type1), pd.Series(blood_type2)])
```

executed in 22ms, finished 09:33:51 2018-06-28

Out[11]:
```
0     A
1    AB
0     B
1     O
dtype: object
```

可以发现，分类数据经过 pd.concat 合并后类型转为了 object 类型。如果想要保持分类类型的话，可以借助 union_categoricals 来完成。

In [12]:
```
from pandas.api.types import union_categoricals

union_categoricals([blood_type1, blood_type2])
```

executed in 35ms, finished 09:33:51 2018-06-28

Out[12]:
```
[A, AB, B, O]
Categories (4, object): [A, AB, B, O]
```

内存使用量的陷阱

Categorical 的内存使用量是与分类数乘以数据长度成正比，object 类型的数据是一个常数乘以数据的长度。

In [13]:
```
blood_type = pd.Series(["AB","O"]*1000)
blood_type.nbytes
```
executed in 24ms, finished 09:33:51 2018-06-28

Out[13]: 16000

In [14]:
```
blood_type.astype("category").nbytes
```
executed in 32ms, finished 09:33:51 2018-06-28

Out[14]: 2016

对比下，是不是发现分类数据非常节省内存。但是当类别的数量接近数据的长度，那么 Categorical 将使用与等效的 object 表示几乎相同或更多的内存。

In [15]:
```
blood_type = pd.Series(['AB%04d' % i for i in range(2000)])
blood_type.nbytes
```
executed in 19ms, finished 09:33:51 2018-06-28

Out[15]: 16000

In [16]:
```
blood_type.astype("category").nbytes
```
executed in 29ms, finished 09:33:51 2018-06-28

Out[16]: 20000

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：**pandas05**。