> 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 03-Pandas缺失值处理 (03-Pandas缺失值处理.ipynb) 介绍了 Pandas 中缺失值的处理，这一节我们来看一看如何处理 Pandas 中的文本（字符串）。

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

executed in 5ms, finished 16:48:36 2018-06-26

为什么要用str属性

文本数据也就是我们常说的字符串，Pandas 为 Series 提供了 `str` 属性，通过它可以方便的对每个元素进行操作。

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

data = {
    "age": [18, 30, np.nan, 40, np.nan, 30],
    "city": ["Bei Jing ", "Shang Hai ", "Guang Zhou", "Shen Zhen", np.nan, " "],
    "sex": [None, "male", "female", "male", np.nan, "unknown"],
    "birth": ["2000-02-10", "1988-10-17", None, "1978-08-08", np.nan, "1988-10-17"]
}

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

# 将出生日期转为时间戳
user_info["birth"] = pd.to_datetime(user_info.birth)
user_info
```

executed in 90ms, finished 16:48:37 2018-06-26

Out[2]:

name	age	birth	city	sex
Tom	18.0	2000-02-10	Bei Jing	None
Bob	30.0	1988-10-17	Shang Hai	male
Mary	NaN	NaT	Guang Zhou	female
James	40.0	1978-08-08	Shen Zhen	male
Andy	NaN	NaT	NaN	NaN
Alice	30.0	1988-10-17		unknown

在之前已经了解过，在对 Series 中每个元素处理时，我们可以使用 `map` 或 `apply` 方法。

比如，我想要将每个城市都转为小写，可以使用如下的方式。

In [3]:
```
user_info.city.map(lambda x: x.lower())
```

executed in 292ms, finished 16:48:37 2018-06-26

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-3-e661c5ad5c48> in <module>()
----> 1 user_info.city.map(lambda x: x.lower())

C:\soft\py3\lib\site-packages\pandas\core\series.py in map(self, arg, na_action)
   2156         else:
   2157             # arg is a function
-> 2158             new_values = map_f(values, arg)
   2159
   2160         return self._constructor(new_values,

pandas/_libs/src\inference.pyx in pandas._libs.lib.map_infer()

<ipython-input-3-e661c5ad5c48> in <lambda>(x)
----> 1 user_info.city.map(lambda x: x.lower())

AttributeError: 'float' object has no attribute 'lower'
```

What？竟然出错了，错误原因是因为 float 类型的对象没有 lower 属性。这是因为缺失值（np.nan）属于float 类型。

这时候我们的 str 属性操作来了，来看看如何使用吧。

In [4]:
```
# 将文本转为小写
user_info.city.str.lower()
```

executed in 7ms, finished 16:48:55 2018-06-26

Out[4]:
```
name
Tom       bei jing
Bob      shang hai
Mary     guang zhou
James     shen zhen
Andy            NaN
Alice
Name: city, dtype: object
```

可以看到，通过 `str` 属性来访问之后用到的方法名与 Python 内置的字符串的方法名一样。并且能够自动排除缺失值。

我们再来试试其他一些方法。例如，统计每个字符串的长度。

In [6]:
```
user_info.city.str.len()
```

executed in 8ms, finished 16:49:23 2018-06-26

Out[6]:
```
name
Tom       9.0
Bob      10.0
Mary     10.0
James     9.0
Andy      NaN
Alice     1.0
Name: city, dtype: float64
```

替换和分割

使用 `.srt` 属性也支持替换与分割操作。

先来看下替换操作，例如：将空字符串替换成下划线。

In [7]:
```
user_info.city.str.replace(" ", "_")
```

executed in 34ms, finished 16:49:23 2018-06-26

Out[7]:
```
name
Tom       Bei_Jing_
Bob      Shang_Hai_
Mary     Guang_Zhou
James     Shen_Zhen
Andy            NaN
Alice             _
Name: city, dtype: object
```

replace 方法还支持正则表达式，例如将所有开头为 S 的城市替换为空字符串。

In [8]:
```
user_info.city.str.replace("^S.*", " ")
```

executed in 24ms, finished 16:49:24 2018-06-26

Out[8]:
```
name
Tom       Bei Jing
Bob
Mary     Guang Zhou
James
Andy            NaN
Alice
Name: city, dtype: object
```

再来看下分割操作，例如根据空字符串来分割某一列。

In [9]:
```
user_info.city.str.split(" ")
```
executed in 54ms, finished 16:49:24 2018-06-26

Out[9]:
```
name
Tom       [Bei, Jing, ]
Bob      [Shang, Hai, ]
Mary      [Guang, Zhou]
James      [Shen, Zhen]
Andy                NaN
Alice              [, ]
Name: city, dtype: object
```

分割列表中的元素可以使用 get 或 [] 符号进行访问：

In [10]:
```
user_info.city.str.split(" ").str.get(1)
```
executed in 55ms, finished 16:49:24 2018-06-26

Out[10]:
```
name
Tom      Jing
Bob       Hai
Mary     Zhou
James    Zhen
Andy      NaN
Alice
Name: city, dtype: object
```

In [11]:
```
user_info.city.str.split(" ").str[1]
```
executed in 41ms, finished 16:49:24 2018-06-26

Out[11]:
```
name
Tom      Jing
Bob       Hai
Mary     Zhou
James    Zhen
Andy      NaN
Alice
Name: city, dtype: object
```

设置参数 `expand=True` 可以轻松扩展此项以返回 DataFrame。

In [12]:
```
user_info.city.str.split(" ", expand=True)
```

executed in 65ms, finished 16:49:24 2018-06-26

Out[12]:

	0	1	2
name			
Tom	Bei	Jing	
Bob	Shang	Hai	
Mary	Guang	Zhou	None
James	Shen	Zhen	None
Andy	NaN	None	None
Alice			None

提取子串

既然是在操作字符串，很自然，你可能会想到是否可以从一个长的字符串中提取出子串。答案是可以的。

提取第一个匹配的子串

`extract` 方法接受一个正则表达式并至少包含一个捕获组，指定参数 `expand=True` 可以保证每次都返回 DataFrame。

例如，现在想要匹配空字符串前面的所有的字母，可以使用如下操作：

In [13]:
```
user_info.city.str.extract("(\w+)\s+", expand=True)
```

executed in 33ms, finished 16:49:24 2018-06-26

Out[13]:

	0
name	
Tom	Bei
Bob	Shang
Mary	Guang
James	Shen
Andy	NaN
Alice	NaN

如果使用多个组提取正则表达式会返回一个 DataFrame，每个组只有一列。

例如，想要匹配出空字符串前面和后面的所有字母，操作如下：

In [14]:
```
user_info.city.str.extract("(\w+)\s+(\w+)", expand=True)
```

executed in 40ms, finished 16:49:24 2018-06-26

Out[14]:

	0	1
name		
Tom	Bei	Jing
Bob	Shang	Hai
Mary	Guang	Zhou
James	Shen	Zhen
Andy	NaN	NaN
Alice	NaN	NaN

匹配所有子串

`extract` 只能够匹配出第一个子串，使用 `extractall` 可以匹配出所有的子串。

例如，将所有组的空白字符串前面的字母都匹配出来，可以如下操作。

In [15]:
```
user_info.city.str.extractall("(\w+)\s+")
```

executed in 35ms, finished 16:49:24 2018-06-26

Out[15]:

		0
name	**match**	
Tom	**0**	Bei
	1	Jing
Bob	**0**	Shang
	1	Hai
Mary	**0**	Guang
James	**0**	Shen

测试是否包含子串

除了可以匹配出子串外，我们还可以使用 `contains` 来测试是否包含子串。例如，想要测试城市是否包含子串 “Zh”。

In [16]:
```
user_info.city.str.contains("Zh")
```

executed in 44ms, finished 16:49:24 2018-06-26

Out[16]:
```
name
Tom      False
Bob      False
Mary      True
James     True
Andy       NaN
Alice    False
Name: city, dtype: object
```

当然了，正则表达式也是支持的。例如，想要测试是否是以字母 “S” 开头。

In [17]:
```
user_info.city.str.contains("^S")
```

executed in 50ms, finished 16:49:24 2018-06-26

Out[17]:
```
name
Tom      False
Bob       True
Mary     False
James     True
Andy       NaN
Alice    False
Name: city, dtype: object
```

生成哑变量

这是一个神奇的功能，通过 get_dummies 方法可以将字符串转为哑变量，sep 参数是指定哑变量之间的分隔符。来看看效果吧。

In [18]:
```
user_info.city.str.get_dummies(sep=" ")
```

executed in 48ms, finished 16:49:24 2018-06-26

Out[18]:

	Bei	Guang	Hai	Jing	Shang	Shen	Zhen	Zhou
name								
Tom	1	0	0	1	0	0	0	0
Bob	0	0	1	0	1	0	0	0
Mary	0	1	0	0	0	0	0	1
James	0	0	0	0	0	1	1	0
Andy	0	0	0	0	0	0	0	0
Alice	0	0	0	0	0	0	0	0

这样，它提取出了 Bei, Guang, Hai, Jing, Shang, Shen, Zhen, Zhou 这些哑变量，并对每个变量下使用 0 或 1 来表达。实际上与 One-Hot（狂热编码）是一回事。听不懂没关系，之后将机器学习相关知识时会详细介绍这里。

方法摘要

这里列出了一些常用的方法摘要。

方法	描述
cat()	连接字符串
split()	在分隔符上分割字符串
rsplit()	从字符串末尾开始分隔字符串
get()	索引到每个元素（检索第i个元素）
join()	使用分隔符在系列的每个元素中加入字符串
get_dummies()	在分隔符上分割字符串，返回虚拟变量的DataFrame
contains()	如果每个字符串都包含pattern / regex，则返回布尔数组
replace()	用其他字符串替换pattern / regex的出现
repeat()	重复值（s.str.repeat(3)等同于x * 3 t2 >）
pad()	将空格添加到字符串的左侧，右侧或两侧
center()	相当于str.center
ljust()	相当于str.ljust
rjust()	相当于str.rjust
zfill()	等同于str.zfill
wrap()	将长长的字符串拆分为长度小于给定宽度的行
slice()	切分Series中的每个字符串
slice_replace()	用传递的值替换每个字符串中的切片
count()	计数模式的发生
startswith()	相当于每个元素的str.startswith(pat)
endswith()	相当于每个元素的str.endswith(pat)
findall()	计算每个字符串的所有模式/正则表达式的列表

方法	描述
match()	在每个元素上调用re.match，返回匹配的组作为列表
extract()	在每个元素上调用re.search，为每个元素返回一行DataFrame，为每个正则表达式捕获组返回一列
extractall()	在每个元素上调用re.findall，为每个匹配返回一行DataFrame，为每个正则表达式捕获组返回一列
len()	计算字符串长度
strip()	相当于str.strip
rstrip()	相当于str.rstrip
lstrip()	相当于str.lstrip
partition()	等同于str.partition
rpartition()	等同于str.rpartition
lower()	相当于str.lower
upper()	相当于str.upper
find()	相当于str.find
rfind()	相当于str.rfind
index()	相当于str.index
rindex()	相当于str.rindex
capitalize()	相当于str.capitalize
swapcase()	相当于str.swapcase
normalize()	返回Unicode标准格式。相当于unicodedata.normalize
translate()	等同于str.translate
isalnum()	等同于str.isalnum
isalpha()	等同于str.isalpha
isdigit()	相当于str.isdigit
isspace()	等同于str.isspace
islower()	相当于str.islower

方法	描述
isupper()	相当于str.isupper
istitle()	相当于str.istitle
isnumeric()	相当于str.isnumeric
isdecimal()	相当于str.isdecimal

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：**pandas04。**