

# Pandas 使用教程

---

## 一、实验介绍

---

### 1.1 实验内容

Pandas 是非常著名的开源数据处理工具，我们可以通过它对数据集进行快速读取、转换、过滤、分析等一系列操作。除此之外，Pandas 拥有强大的缺失数据处理与数据透视功能，可谓是数据预处理中的必备利器。这是 Pandas 使用教程的第 3 章节，将学会使用 Pandas 对数据进行选择与变换。

### 1.2 实验知识点

- 基于索引数字选择
- 基于标签名称选择
- 数据随机取样
- 条件语句选择
- where() 方法选择
- query() 方法选择

### 1.3 实验环境

- python2.7
- Xfce 终端
- ipython 终端

### 1.4 适合人群

本课程难度为一般，属于初级级别课程，适合具有 Python 基础，并对使用 Pandas 进行数据处理感兴趣的用户。

## 1.5 数据文件

学习本课程之前，请先打开在线环境终端，下载本文可能会用到的数据文件。

```
wget http://labfile.oss.aliyuncs.com/courses/906/los_census.csv
```

`los_census.csv` 为为洛杉矶人口普查数据，仅格式有区别。

下面的内容均在 iPython 交互式终端中演示，你可以通过在线环境左下角的应用程序菜单 > 附件打开。如果你在本地进行练习，推荐使用 Jupyter Notebook 环境。

## 二、数据选择

在数据预处理过程中，我们往往会对数据集进行切分，只将需要的某些行、列，或者数据块保留下来，输出到下一个流程中去。这也就是这里所说的数据选择。

由于 Pandas 的数据结构中存在索引、标签，所以我们可以通过多轴索引完成对数据的选择。

### 2.1 基于索引数字选择

当我们新建一个 DataFrame 之后，如果未自己指定行索引或者列对应的标签，那么 Pandas 会默认从 0 开始以数字的形式作为行索引，并以数据集的第一行作为列对应的标签。其实，这里的「列」也有数字索引，默认也是从 0 开始，只是未显示出来。

所以，我们首先可以基于数字索引对数据集进行选择。这里用到的 Pandas 中的 `.iloc` 方法。该方法可以接受的类型有：

1. 整数。例如：5

2. 整数构成的列表或数组。例如：[1, 2, 3]
3. 布尔数组。
4. 可返回索引值的函数或参数。

下面，我们还是用 `los_census.csv` 数据集演示该方法的使用。如果未下载该数据集，请看 1.5 节。

```
import pandas as pd

df = pd.read_csv("los_census.csv")


print df.head()
```

```
In [7]: print df.head()
```

	Zip Code	Total Population	Median Age	Total Males	Total Females	\
0	91371	1	73.5	0	1	
1	90001	57110	26.6	28468	28642	
2	90002	51223	25.5	24876	26347	
3	90003	66266	26.3	32631	33635	
4	90004	62180	34.8	31302	30878	

	Total Households	Average Household Size
0	1	1.00
1	12971	4.40
2	11731	4.36
3	15642	4.22
4	22547	2.73



首先，我们可以选择前 3 行数据。这和 python 或者 numpy 里面的切片很相似。


```
print df.iloc[:3]
```

```
In [8]: print df.iloc[:3]
```

	Zip Code	Total Population	Median Age	Total Males	Total Females	\
0	91371	1	73.5	0	1	
1	90001	57110	26.6	28468	28642	
2	90002	51223	25.5	24876	26347	

	Total Households	Average Household Size
0	1	1.00
1	12971	4.40
2	11731	4.36



我们还可以选择特定的一行。

```
print df.iloc[5]
```

```
In [11]: print df.iloc[5]
Zip Code      90005.0
Total Population  37681.0
Median Age      33.9
Total Males     19299.0
Total Females   18382.0
Total Households 15044.0
Average Household Size  2.5
Name: 5, dtype: float64
```



那么选择多行是不是 `print df.iloc[1, 3, 5]` 这样呢？答案是错误的。`df.iloc[]` 的 `[[行], [列]]` 里面可以同时接受行和列的位置，如果你直接键入 `df.iloc[1, 3, 5]` 就会报错。

所以，很简单。如果你想要选择 1, 3, 5 行，可以这样做。

```
print df.iloc[[1, 3, 5]]
```

```
In [12]: print df.iloc[[1, 3, 5]]
Zip Code  Total Population  Median Age  Total Males  Total Females  \
1    90001             57110        26.6      28468      28642
3    90003             66266        26.3      32631      33635
5    90005             37681        33.9      19299      18382

Total Households  Average Household Size
1             12971             4.40
3             15642             4.22
5             15044             2.50
```



选择行学会以后，选择列就应该能想到怎么办了。你可以先暂停浏览下面的内容，自己试一试。

例如，我们要选择第 2-4 列。

```
print df.iloc[:, 1:4]
```

```
In [18]: print df.iloc[:, 1:4]
```

	Total Population	Median Age	Total Males
0	0	0.0	0
1	0	0.0	0
2	0	0.0	0
3	66266	26.3	32631
4	62180	34.8	31302
5	37681	33.9	19299
6	59185	32.4	30254
7	40920	24.0	20915
8	32327	39.7	14477
9	3800	37.8	1874



这里选择 2-4 列，输入的却是 1:4。这和 python 或者 numpy 里面的切片操作非常相似。

既然我们能定位行和列，那么只需要组合起来，我们就可以选择数据集中的任何一块数据了。

## 2.2 基于标签名称选择

除了根据数字索引选择，我们还可以直接根据标签对应的名称选择。这里用到的方法和上面的 `iloc` 很相似，少了个 `i` 为 `df.loc[]`。

`df.loc[]` 可以接受的类型有：

1. 单个标签。例如：2 或 'a'，这里的 2 指的是标签而不是索引位置。
2. 列表或数组包含的标签。例如：['A', 'B', 'C']。
3. 切片对象。例如：'A':'E'，注意这里和上面切片的不同支持，首位都包含在内。
4. 布尔数组。
5. 可返回标签的函数或参数。

下面，我们来演示 `df.loc[]` 的用法。我们先随机生成一个 DataFrame。

```
import pandas as pd
import numpy as np # 加载 numpy 模块

df = pd.DataFrame(np.random.randn(6,5),index=list('abcdef'),columns=list('ABCDE'))

print df
```

```
In [4]: print df
```

	A	B	C	D	E
a	1.135156	0.227590	-0.198433	-2.298855	0.342061
b	-0.918968	0.636828	1.841580	-1.082270	0.656800
c	-1.866682	0.611608	-0.874196	1.401121	-0.525998
d	-0.299740	-0.239781	-0.331322	1.625800	0.356279
e	1.220473	0.066768	-0.073860	0.209334	1.003628
f	1.099330	-0.042109	0.197624	1.234342	2.596630



先选择前 3 行：

```
print df.loc['a':'c']
```

```
In [5]: print df.loc['a':'c']
```

	A	B	C	D	E
a	1.135156	0.227590	-0.198433	-2.298855	0.342061
b	-0.918968	0.636828	1.841580	-1.082270	0.656800
c	-1.866682	0.611608	-0.874196	1.401121	-0.525998



再选择 1, 3, 5 行：

```
print df.loc[['a', 'c', 'd']]
```

```
In [6]: print df.loc[['a', 'c', 'd']]
```

	A	B	C	D	E
a	1.135156	0.227590	-0.198433	-2.298855	0.342061
c	-1.866682	0.611608	-0.874196	1.401121	-0.525998
d	-0.299740	-0.239781	-0.331322	1.625800	0.356279



然后，选择 2-4 列：

```
print df.loc[:, 'B':'D']
```



```
In [13]: print df.loc[:, 'B': 'D']
```

	B	C	D
a	0.227590	-0.198433	-2.298855
b	0.636828	1.841580	-1.082270
c	0.611608	-0.874196	1.401121
d	-0.239781	-0.331322	1.625800
e	0.066768	-0.073860	0.209334
f	-0.042109	0.197624	1.234342



最后，选择 1，3 行和 C 后面的列：

```
print df.loc[['a', 'c'], 'C':]
```

```
In [14]: print df.loc[['a', 'c'], 'C':]
```

	C	D	E
a	-0.198433	-2.298855	0.342061
c	-0.874196	1.401121	-0.525998



## 2.3 数据随机取样

上面，的 `.iloc` 和 `.loc` 可用于精准定位数据块。而 Pandas 同样也提供了随机取样的方法，用于满足各种情况。随机取样用 `.sample()` 完成，下面我们就演示一下它的用法。

首先，看一看 Series 数据结构。

```
import pandas as pd

s = pd.Series([0,1,2,3,4,5,6,7,8,9])

print s.sample()
```

```
In [3]: print s.sample()
```

2	2
---	---

```
dtype: int64
```



我们可以看到，默认情况下 `.sample()` 返回了一个数值。注意，前面的 2 是数字索引，后面的 2 才是值。

我们可以通过 `n=` 参数，设定返回值的数量。

```
print s.sample(n=5)
```

```
In [4]: print s.sample(n=5)
2      2
1      1
3      3
0      0
7      7
dtype: int64
```



同样也可以用 `frac=` 参数设定返回数量的比例。

```
print s.sample(frac=.6) # 返回 60% 的数值
```

```
In [8]: print s.sample(frac=.6)
6      6
9      9
5      5
0      0
2      2
1      1
dtype: int64
```



对应 `DataFrame` 而言，过程也很相似，只是需要选择坐标轴。举个例子：

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(6,5),index=list('abcdef'),columns=list('ABCDE'))

print df
print df.sample(n=3)
```



```
In [4]: print df
```

	A	B	C	D	E
a	0.454559	-1.771880	0.097097	-0.585950	0.573461
b	-0.251677	-1.185492	0.557911	0.688526	-1.852775
c	1.125986	0.431218	-1.879841	0.999239	1.366848
d	-0.196214	0.004068	0.435760	-0.002148	0.277887
e	-1.186614	0.740217	1.649656	-0.406492	0.531674
f	1.771335	1.555901	1.724901	0.803414	-3.587212

```
In [5]: print df.sample(n=3)
```

	A	B	C	D	E
f	1.771335	1.555901	1.724901	0.803414	-3.587212
a	0.454559	-1.771880	0.097097	-0.585950	0.573461
b	-0.251677	-1.185492	0.557911	0.688526	-1.852775



默认会返回行，如果要随机返回 3 列。需要添加 `axis=` 参数。

```
print df.sample(n=3, axis=1)
```

```
In [6]: print df.sample(n=3,axis=1)
```

	B	A	C
a	-1.771880	0.454559	0.097097
b	-1.185492	-0.251677	0.557911
c	0.431218	1.125986	-1.879841
d	0.004068	-0.196214	0.435760
e	0.740217	-1.186614	1.649656
f	1.555901	1.771335	1.724901



## 2.4 条件语句选择

数据选择的时候，我们还可以加入一些条件语句，从而达到对数据筛选的目的。这个过程和 numpy 里面的效果很相似。我们先举一个 Series 的例子：

```
import pandas as pd

s = pd.Series(range(-5, 5))

print s
print s[(s < -2) | (s > 1)] # 添加 逻辑或 条件
```

```
In [3]: print s
```

```
0    -5
1    -4
2    -3
3    -2
4    -1
5     0
6     1
7     2
8     3
9     4
```

```
dtype: int64
```

```
In [4]: print s[(s < -2) | (s > 1)]
```

```
0    -5
1    -4
2    -3
7     2
8     3
9     4
```

```
dtype: int64
```



对于 DataFrame 也是相似的。

```
import pandas as pd
import numpy as np
```

```
df = pd.DataFrame(np.random.randn(6,5), index=list('abcdef'), columns=list('ABCDE'))
```

```
print df
```

```
print df[(df['B'] > 0) | (df['D'] < 0)] # 添加条件
```

```
In [8]: print df
```

	A	B	C	D	E
a	-0.782018	-0.241509	0.304918	0.390334	0.202453
b	-0.162390	0.181710	-0.478234	0.917108	1.592034
c	0.367813	-1.366442	-1.499660	-1.992763	0.583795
d	1.047316	-0.840930	-1.547218	0.571404	-0.336317
e	-1.591755	0.539542	-0.748026	-0.090421	0.058902
f	0.490673	-0.124902	-0.117991	0.221871	0.658855

```
In [9]: print df[(df['B'] > 0) | (df['D'] < 0)]
```

	A	B	C	D	E
b	-0.162390	0.181710	-0.478234	0.917108	1.592034
c	0.367813	-1.366442	-1.499660	-1.992763	0.583795
e	-1.591755	0.539542	-0.748026	-0.090421	0.058902



## 2.5 where() 方法选择

接下来，再介绍一种通过 `where()` 方法进行数据选择得方法。`DataFrame` 和 `Series` 都带有 `where()`，可以通过一些判断句来选择数据。举个例子：

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(6,5),index=list('abcdef'),columns=
list('ABCDE'))

print df
print df.where(df < 0) # 添加条件
```

`.where(df < 0)` 会返回所有负值，而非负值就会被置为空值 `NaN`。

```
In [13]: print df
           A          B          C          D          E
a  0.825341 -1.241227  0.328471 -0.499764 -1.808909
b  0.177475 -0.613529 -0.226586 -1.409960  0.690522
c -0.132223  1.026666  0.647785  0.896511  1.324797
d -0.345777  0.253142 -0.112352 -1.641362  0.703383
e -0.396848 -0.010160  0.414950 -0.507433 -2.115783
f -0.715164  1.433390 -0.941574 -0.844121 -0.645845

In [14]: print df.where(df < 0)
           A          B          C          D          E
a         NaN -1.241227         NaN -0.499764 -1.808909
b         NaN -0.613529 -0.226586 -1.409960         NaN
c -0.132223         NaN         NaN         NaN         NaN
d -0.345777         NaN -0.112352 -1.641362         NaN
e -0.396848 -0.010160         NaN -0.507433 -2.115783
f -0.715164         NaN -0.941574 -0.844121 -0.645845
```



你也可以对判断条件以外得值重新替代，例如这里将非负值全部变号为负值。

```
print df.where(df < 0, -df) # 筛选负值并将正值变号
```

```
In [15]: print df.where(df < 0, -df)
          A          B          C          D          E
a -0.825341 -1.241227 -0.328471 -0.499764 -1.808909
b -0.177475 -0.613529 -0.226586 -1.409960 -0.690522
c -0.132223 -1.026666 -0.647785 -0.896511 -1.324797
d -0.345777 -0.253142 -0.112352 -1.641362 -0.703383
e -0.396848 -0.010160 -0.414950 -0.507433 -2.115783
f -0.715164 -1.433390 -0.941574 -0.844121 -0.645845
```



故，where() 实际上期待了匹配和替换得效果。我们可以借助该方法实现对数据的自由设定。

## 2.6 query() 方法选择

针对数据变换和筛选的方法还很多，除了上面的提到的，Pandas 0.13 之后的版本中增加了 query() 实验性方法，该方法也可以被用来选择数据。

query() 是 DataFrame 具有的方法，你可以通过一个比较语句对满足行列条件的值进行选择，举个例子：

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.rand(10, 5), columns=list('abcde'))

print df
print df.query('(a < b) & (b < c)') # 添加 逻辑与 条件
```

上面的判断语句应该很容易看明白，也就是满足 a 列的值需小于 b 列，且 b 列的值小于 c 列所在的行。



```
In [4]: print df
```

	a	b	c	d	e
0	0.059002	0.565043	0.034743	0.154243	0.523622
1	0.268942	0.854591	0.495145	0.624837	0.301047
2	0.686075	0.865926	0.500580	0.562298	0.900873
3	0.639538	0.329158	0.235176	0.747104	0.880388
4	0.614287	0.966424	0.176917	0.412457	0.664658
5	0.278414	0.026753	0.988854	0.008697	0.927102
6	0.181542	0.568155	0.898819	0.680959	0.509856
7	0.612872	0.222697	0.602892	0.502694	0.473312
8	0.978274	0.471334	0.520021	0.426481	0.034709
9	0.985763	0.334789	0.426850	0.893935	0.738529

```
In [5]: print df.query('(a < b) & (b < c)')
```

	a	b	c	d	e
6	0.181542	0.568155	0.898819	0.680959	0.509856



当然，在没有 `query()` 之前，我们也是可以通过前面提到的条件语句选择。

```
print df[(df.a < df.b) & (df.b < df.c)]
```

```
In [8]: df[(df.a < df.b) & (df.b < df.c)]
```

```
Out[8]:
```

	a	b	c	d	e
6	0.181542	0.568155	0.898819	0.680959	0.509856



结果虽然一致，但是 `query()` 语句的确要简洁和自然很多。`query()` 包含很多内容，非常强大。你可以通过官方文档了解，这里就不再赘述了。

### 三、实验总结

本章节学习了针对数据选择的常用方法和手段，这是 Pandas 中十分重要的一部分内容。针对数据集的处理，无外乎就是变换、筛选，最终得到我们想要的数

\*本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。

上一节：Pandas 常用的基本方法 (/courses/906/labs/3376/document)

下一节：Pandas 进行缺失值处理 (</courses/906/labs/3378/document>)