

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 2 章节，将学会 Pandas 中的一些常用的基本方法。

1.2 实验知识点

- 数据读取与存储
- Head & Tail
- 统计方法
- 计算方法
- 标签对齐
- 排序

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两个数据文件。

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

两个文件均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、Pandas 常见的基本方法

2.1 数据读取与存储

Pandas 支持大部分常见数据文件读取与存储。一般清楚下，读取文件的方法以 `pd.read_` 开头，而写入文件的方法以 `pd.to_` 开头。详细的表格如下。

文件格式	读取方法	写入方法
CSV	<i>read_csv</i>	<i>to_csv</i>
JSON	<i>read_json</i>	<i>to_json</i>
HTML	<i>read_html</i>	<i>to_html</i>
Local clipboard	<i>read_clipboard</i>	<i>to_clipboard</i>
MS Excel	<i>read_excel</i>	<i>to_excel</i>
HDF5 Format	<i>read_hdf</i>	<i>to_hdf</i>
Feather Format	<i>read_feather</i>	<i>to_feather</i>
Msgpack	<i>read_msgpack</i>	<i>to_msgpack</i>
Stata	<i>read_stata</i>	<i>to_stata</i>
SAS	<i>read_sas</i>	
Python Pickle Format	<i>read_pickle</i>	<i>to_pickle</i>
SQL	<i>read_sql</i>	<i>to_sql</i>
Google Big Query	<i>read_gbq</i>	<i>to_gbq</i>

拿刚刚下载好的数据文件举例，如果没有下载，请看 1.5 小节。

```
import pandas as pd
```

```
df = pd.read_csv("los_census.csv") #读取 csv 文件
print df
```

```
In [2]: df = pd.read_csv("los_census.csv")
```

```
In [3]: print df
```

	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
5	90005	37681	33.9	19299	18382
6	90006	59185	32.4	30254	28931
7	90007	40920	24.0	20915	20005

可以看到，文件已经读取出来了。由于列数太多，所以分段显示了。输出的最下方会有一个行数和列数的统计。这里是 319 行 X 7 列。

我们可以发现，由 pandas 读取的文件就已经是 DataFrame 结构了。上面演示了 csv 文件的读取，其余格式的文件也很相似。

不过，很多时候我们拿到手的数据是像 `los_census.txt` 文件样式的数据，如下图所示。

```
import pandas as pd
```

```
df = pd.read_table("los_census.txt") #读取 txt 文件
print df
```

```
Zip Code,Total Population,Median Age,Total Males,Total Females,Total Households,Average Household Size
91371,1,73.5,0,1,1,1
90001,57110,26.6,28468,28642,12971,4.490002,51223,25.5,24876,26347,11731,4.3690003,66266,26.3,32631,33635,15642,4.22
90004,62180,34.8,31302,30878,22547,2.7390005,37681,33.9,19299,18382,15044,2.590006,59185,32.4,30254,28931,18617,3.13
90007,40920,24,20915,20005,11944,390008,32327,39.7,14477,17850,13841,2.3390010,3800,37.8,1874,1926,2014,1.87
90011,103892,26.2,52794,51098,22168,4.6790012,31103,36.3,19493,11610,10327,2.1290013,11772,44.6,7629,4143,6416,1.26
90014,7005,44.8,4471,2534,4109,1.3490015,18986,31.3,9833,9153,7420,2.4590016,47596,33.9,22778,24818,16145,2.93
90017,23768,29.4,12818,10950,9338,2.5390018,49310,33.2,23770,25540,15493,3.1290019,64458,35.8,31442,33016,23344,2.7
90020,38967,34.6,19381,19586,16514,2.3590021,3951,44.3,2790,1161,1561,1.5790022,67179,29.8,33216,33963,17023,3.94
90023,45903,28.4,23037,22866,10727,4.2690024,47452,23.6,22248,25204,17903,2.0390025,42147,34.7,20859,21288,21228,1.97
90026,67869,34,34515,33354,24956,2.6890027,45151,38.3,22362,22789,21929,1.9990028,28714,34,16056,12658,14964,1.78
90029,38617,34.6,19575,19042,13883,2.790031,39316,33.5,19546,19770,11156,3.4990032,45786,32.4,22564,23222,12765,3.52
90033,48852,29.2,24425,24427,12924,3.6690034,57964,32.8,28828,29136,25592,2.2390035,28418,37.5,13326,15092,12814,2.19
90036,36865,33.9,17914,18951,18646,1.9690037,62276,28.8,31187,31089,15869,3.8590038,28917,33.1,15383,13534,11928,2.41
90039,28514,38.8,14383,14131,11436,2.4790040,12520,31.2,6129,6391,3317,3.7590041,27425,39,13212,14213,9513,2.71
90042,62430,33.6,30836,31594,19892,3.1190043,44789,38.7,20561,24228,16075,2.7690044,89779,28.6,43128,46651,25144,3.55
90045,39480,35.6,18958,20522,15224,2.3790046,48581,38.5,25854,22727,28534,1.6990047,48606,36.2,22129,26477,16168,2.99
90048,21397,39.2,10132,11265,11821,1.7790049,35482,41.3,16359,19123,16657,2.0990056,7827,48.4,3436,4391,3371,2.32
90057,44998,31.2,24300,20698,15658,2.8190058,3223,26,1555,1668,892,3.690059,40952,25.7,19623,21329,9596,4.19
90061,26872,28.4,13049,13823,6892,3.8590062,32821,31.8,15720,17101,9155,3.5590063,55758,29,27843,27915,13260,4.19
90064,25403,40,12297,13106,10968,2.2990065,45527,36.1,22873,22654,14476,3.1290066,55277,37.3,27714,27563,23985,2.29
90067,2424,65.3,1074,1350,1510,1.6190068,22286,39.4,12018,10268,12326,1.890069,20483,41.5,12153,8330,13364,1.53
90071,15,45.5,13,2,0,090073,539,56.9,506,33,4,1.2590077,9377,47.9,4594,4783,3615,2.5790079,0,0,0,0,0
90089,3217,19.3,1436,1781,31,1.9490090,0,0,0,0,0,090094,5464,33.7,2559,2905,2949,1.8590095,3,52.5,2,1,2,1.5
90201,101279,27.8,50658,50621,24104,4.1690210,21741,47.5,10292,11449,8669,2.4990211,8434,40.6,3849,4585,3706,2.28
90212,11555,41.2,5211,6344,5567,2.0890220,49328,29.8,23773,25555,12741,3.8590221,53704,26.7,26346,27358,11630,4.57
90222,31869,27.3,15375,16494,7520,4.2190230,31766,39.1,14932,16834,12883,2.4590232,15149,38.6,7333,7816,6605,2.28
90240,25876,35.5,12501,13375,7632,3.3690241,42399,33.9,20466,21933,13617,3.0990242,43497,31.6,21207,22290,12687,3.41
90245,16654,39.2,8304,8350,7085,2.3490247,47487,35.5,23217,24270,15830,2.9690248,9947,41.2,4823,5124,3427,2.89
90249,26669,37.2,12897,13772,8880,2.9890250,93193,31.9,45113,48080,31087,2.9890254,19506,37,10273,9233,9550,2.04
90255,75066,29.1,37525,37541,18419,4.0690260,34924,32.7,17509,17415,10429,3.3390262,69745,27.8,33919,35826,14669,4.57
90263,1612,19.7,665,947,0,090265,18116,46.5,9159,8957,7174,2.3990266,35135,40.9,17605,17530,14038,2.5
```

其实 `los_census.txt` 也就是 `los_census.csv` 文件，因为 csv 文件又叫逗号分隔符文件，数据之间采用逗号分割。

那么，我们怎样将这种文件转换为 DataFrame 结构的数据呢？这里就要使用到读取方法中提供的一些参数了，例如 `sep[]` 分隔符参数。

```
import pandas as pd
```

```
df = pd.read_table("los_census.txt", sep=',') #读取 txt 文件
print df
```

```
In [8]: df = pd.read_table("los_census.txt", sep=',')
```

```
In [9]:
```

```
In [9]: print df
```

	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	
5	90005	37681	33.9	19299	18382	

除了 `sep`，读取文件时常用的参数还有：

1. `header=`，用来选择将第几行作为列索引名称。
2. `names=[]`，自定义列索引名称。

例如：

```
import pandas as pd
```

```
df = pd.read_csv("los_census.csv", header=1) #将第二行作为列索引名称。  
print df
```

```
In [11]: df = pd.read_csv("los_census.csv", header=1)
```

```
In [12]: print df
```

	91371	1	73.5	0	1.1	1.2	1.3
0	90001	57110	26.6	28468	28642	12971	4.40
1	90002	51223	25.5	24876	26347	11731	4.36
2	90003	66266	26.3	32631	33635	15642	4.22
3	90004	62180	34.8	31302	30878	22547	2.73

```
import pandas as pd
```

```
df = pd.read_csv("los_census.csv", names=['A', 'B', 'C', 'D', 'E', 'F',  
, 'G']) #自定义列索引名称。  
print df
```

```
In [15]: print df
```

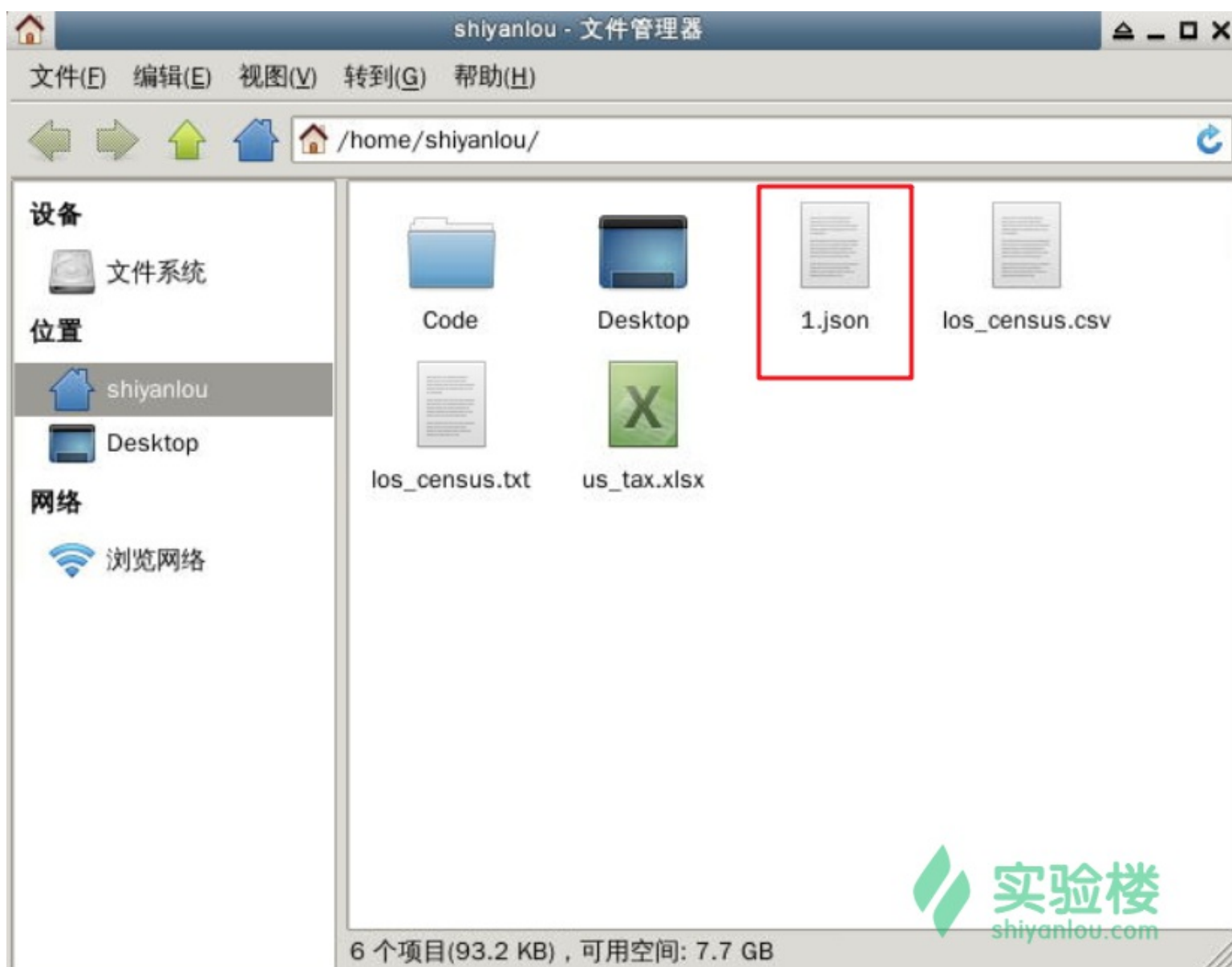
	A	B	C	D	E	\
0	Zip Code	Total Population	Median Age	Total Males	Total Females	
1	91371	1	73.5	0	1	
2	90001	57110	26.6	28468	28642	
3	90002	51223	25.5	24876	26347	
4	90003	66266	26.3	32631	33635	

好了，说了这么久的读取文件，再说一说存储文件。存储文件的方法也很简单。比如我们将 `los_census.csv` 文件，存储为 `json` 格式的文件。

```
import pandas as pd

df = pd.read_csv("los_census.csv") #读取 csv 文件

df.to_json("1.json") # 将其存储为 json 格式文件
```



当然，你也可以通过 `to_excel("1.xlsx")` 储存为 Excel 默认支持的 `.xlsx` 格式。只是，需要注意在线环境会报错。这时候需要再补充安装 `openpyxl` 包就好了：

```
sudo pip install openpyxl
```

2.2 Head & Tail

有些时候，我们读取的文件很大。如果全部输出预览这些文件，既不美观，又很耗时。还好，Pandas 提供了 `head()` 和 `tail()` 方法，它可以帮助我们只预览一小块数据。

顾名思义，`head()` 方法就是从数据集开头预览，不带参数默认显示头部的 5 条数据，你也可以自定义显示条数。

```
import pandas as pd

df = pd.read_csv("los_census.csv") #读取 csv 文件

print df.head() # 默认显示前 5 条
print df.head(7) # 显示前 7 条
```

```
In [3]: 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	

```
In [4]: print df.head(7)
```

	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	
5	90005	37681	33.9	19299	18382	
6	90006	59185	32.4	30254	28931	

`tail()` 方法就是从数据集尾部开始显示了，同样默认 5 条，可自定义。

```
import pandas as pd

df = pd.read_csv("los_census.csv") #读取 csv 文件

print df.tail() # 默认显示后 5 条
print df.tail(7) # 显示后 7 条
```

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

	Zip Code	Total Population	Median Age	Total Males	Total Females
314	93552	38158	28.4	18711	19447
315	93553	2138	43.3	1121	1017
316	93560	18910	32.4	9491	9419
317	93563	388	44.5	263	125
318	93591	7285	30.9	3653	3632

```
In [8]: print df.tail(7)
```

	Zip Code	Total Population	Median Age	Total Males	Total Females
312	93550	74929	27.5	36414	38515
313	93551	50798	37.0	25056	25742
314	93552	38158	28.4	18711	19447
315	93553	2138	43.3	1121	1017
316	93560	18910	32.4	9491	9419
317	93563	388	44.5	263	125
318	93591	7285	30.9	3653	3632

2.3 统计方法

Pandas 提供了几个统计和描述性方法，方便你从宏观的角度去了解数据集。

1. describe()

describe() 相当于对数据集进行概览，会输出该数据集的计数、最大值、最小值等。

```
import pandas as pd

df = pd.read_csv("los_census.csv") #读取 csv 文件

print df.describe()
```

```
In [9]: print df.describe()
```

	Zip Code	Total Population	Median Age	Total Males
count	319.000000	319.000000	319.000000	319.000000
mean	91000.673981	33241.341693	36.527586	16391.564263
std	908.360203	21644.417455	8.692999	10747.495566
min	90001.000000	0.000000	0.000000	0.000000
25%	90243.500000	19318.500000	32.400000	9763.500000
50%	90807.000000	31481.000000	37.100000	15283.000000
75%	91417.000000	44978.000000	41.000000	22219.500000
max	93591.000000	105549.000000	74.000000	52794.000000

例如上面，针对一个 DataFrame 会对每一列的数据单独统计。

2. idxmin() & idxmax()

idxmin() 和 idxmax() 会计算最小、最大值对应的索引标签。

```
import pandas as pd

df = pd.read_csv("los_census.csv") #读取 csv 文件

print df.idxmin()
print df.idxmax()
```

```
In [10]: print df.idxmin()
Zip Code          1
Total Population   64
Median Age         64
Total Males        0
Total Females      64
Total Households   61
Average Household Size  61
dtype: int64

In [11]: print df.idxmax()
Zip Code          318
Total Population   132
Median Age         180
Total Males        10
Total Females      132
Total Households    85
Average Household Size  10
dtype: int64
```



3. count()

count() 用于统计非空数据的数量。

```
import pandas as pd

df = pd.read_csv("los_census.csv") #读取 csv 文件

print df.count()
```

```
In [27]: print df.count()
Zip Code      319
Total Population  319
Median Age      319
Total Males      319
Total Females    319
Total Households 319
Average Household Size  319
dtype: int64
```



4. value_counts()

value_counts() 仅仅针对 Series，它会计算每一个值对应的数量统计。

```
import pandas as pd
import numpy as np

s = pd.Series(np.random.randint(0, 9, size=100)) # 生成一个 Series，并在
0-9 之间生成 100 个随机值。

print s
print s.value_counts()
```

```
In [24]: print s.value_counts()
3      20
6      15
1      11
4      10
0      10
7       9
2       9
8       8
5       8
dtype: int64
```



2.4 计算方法

除了统计类的方法，Pandas 还提供了很多计算类的方法。

1. sum()

sum() 用于计算数值数据的总和。

```
import pandas as pd

df = pd.read_csv("los_census.csv") #读取 csv 文件

print df.sum()
```

```
In [28]: print df.sum()
Zip Code                29029215.00
Total Population        10603988.00
Median Age              11652.30
Total Males             5228909.00
Total Females           5375079.00
Total Households        3497698.00
Average Household Size   902.17
dtype: float64
```



2. mean()

mean() 用于计算数值数据的平均值。

```
import pandas as pd

df = pd.read_csv("los_census.csv") #读取 csv 文件

print df.mean()
```

```
In [29]: print df.mean()
Zip Code                91000.673981
Total Population        33241.341693
Median Age              36.527586
Total Males             16391.564263
Total Females           16849.777429
Total Households        10964.570533
Average Household Size   2.828119
dtype: float64
```



3. median()

median() 用于计算数值数据的算术中值。

```
import pandas as pd

df = pd.read_csv("los_census.csv") #读取 csv 文件

print df.median()
```

```
In [30]: print df.median()
Zip Code          90807.00
Total Population   31481.00
Median Age         37.10
Total Males        15283.00
Total Females      16202.00
Total Households   10968.00
Average Household Size 2.83
dtype: float64
```



除此之外，剩下的一些常见计算方法如下表所示。

方法	描述
mad	平均绝对偏差
min	最小值
max	最大值
abs	绝对值
std	贝塞尔校正样本标准偏差
var	无偏差
sem	平均值的标准误差
skew	样品偏度（3次）
kurt	样品偏度（4次）
quantile	样本分位数（以%计）
cumsum	累计总和
cummax	累计最大值
cummin	累计最小值



2.5 标签对齐

索引标签是 Pandas 中非常重要的特性，有些时候，由于数据的缺失等各种因素导致标签错位的现象，或者想匹配新的标签。于是 Pandas 提供了索引标签对齐的方法 `reindex()`。

`reindex()` 主要有三个作用：

1. 重新排序现有数据以匹配新的一组标签。
2. 在没有标签对应数据的位置插入缺失值（NaN）标记。
3. 特殊情形下，使用逻辑填充缺少标签的数据（与时间序列数据高度相关）。


```
import pandas as pd

s = pd.Series(data=[1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e'])

print s
print s.reindex(['e', 'b', 'f', 'd'])
```

```
In [3]: print s
a      1
b      2
c      3
d      4
e      5
dtype: int64

In [4]: print s.reindex(['e', 'b', 'f', 'd'])
e      5.0
b      2.0
f      NaN
d      4.0
dtype: float64
```



我们可以看到，重新排列的数据中，原有索引对应的数据能自动匹配，而新索引缺失的数据通过 NaN 补全。

当然，对于 DataFrame 类型的数据也是一样的。

```
import pandas as pd

df = pd.DataFrame(data={'one': [1, 2, 3], 'two': [4, 5, 6], 'three': [7, 8, 9]}, index=['a', 'b', 'c'])

print df
```


```
In [7]: print df
   one  three  two
a     1     7    4
b     2     8    5
c     3     9    6
```



```
print df.reindex(index=['b', 'c', 'a'], columns=['three', 'two', 'one'])
```

```
In [8]: print df.reindex(index=['b', 'c', 'a'], columns=['three', 'two', 'one'])
```

	three	two	one
b	8	5	2
c	9	6	3
a	7	4	1




你甚至还可以将上面 Series 的数据按照下面的 DataFrame 的索引序列对齐。

```
print s.reindex(df.index)
```

```
In [11]: print s.reindex(df.index)
```

a	1
b	2
c	3

dtype: int64



2.6 排序

既然是数据处理，就少不了排序这一常用的操作。在 Pandas 中，排序拥有很多「姿势」，下面就一起来看一看。

1. 按索引排序

首先是按照索引排序，其方法为 `Series.sort_index()` 或者是 `DataFrame.sort_index()`。


```
import pandas as pd

df = pd.DataFrame(data={'one': [1, 2, 3], 'two': [4, 5, 6], 'three': [7, 8, 9], 'four': [10, 11, 12]}, index=['a', 'c', 'b'])

print df
```

```
In [17]: print df
```

	four	one	three	two
a	10	1	7	4
c	11	2	8	5
b	12	3	9	6



下面按索引对行重新排序：

```
print df.sort_index()
```

```
In [18]: print df.sort_index()
      four  one  three  two
a      10    1     7    4
b      12    3     9    6
c      11    2     8    5
```



或者添加参数，进行倒序排列：

```
print df.sort_index(ascending=False)
```

```
In [19]: print df.sort_index(ascending=False)
      four  one  three  two
c      11    2     8    5
b      12    3     9    6
a      10    1     7    4
```



1. 按数值排序

第二种是按照数值排序，其方法为 `Series.sort_values()` 或者是 `DataFrame.sort_values()`。举个例子：

```
import pandas as pd

df = pd.DataFrame(data={'one': [1, 2, 3, 7], 'two': [4, 5, 6, 9], 'three': [7, 8, 9, 2], 'four': [10, 11, 12, 5]}, index=['a', 'c', 'b', 'd'])

print df
```

```
In [22]: print df
      four  one  three  two
a      10    1     7    4
c      11    2     8    5
b      12    3     9    6
d       5    7     2    9
```



将第三列按照从小到大排序：

```
print df.sort_values(by='three')
```

```
In [23]: print df.sort_values(by='three')
   four one three two
d      5  7     2   9
a     10  1     7   4
c     11  2     8   5
b     12  3     9   6
```



也可以同时按照两列：

```
print df[['one', 'two', 'three', 'four']].sort_values(by=['one', 'two'])
```

```
In [25]: print df[['one', 'two', 'three', 'four']].sort_values(by=['one', 'two'])
   one two three four
a     1  4     7  10
c     2  5     8  11
b     3  6     9  12
d     7  9     2   5
```



三、实验总结

本章节熟悉了 Pandas 中一些基本方法，这些方法是针对数据集操作过程中经常遇到的。当然，由于不可能面面俱到，这里面提到的方法也只是冰山一角。在数据分析实践中，还需要多多依据需求查阅官方文档。

*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 安装与数据结构 (</courses/906/labs/3375/document>)

下一节：Pandas 数据选择与过滤 (</courses/906/labs/3377/document>)