

# 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 4 章节，将学会 Pandas 完成对数据集缺失值的处理工作。

### 1.2 实验知识点

- 缺失值标记
- 缺失值填充
- 缺失值插值

### 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。

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、认识缺失值

在真实的生产环境中，我们需要处理的数据文件往往没有想象中的那么美好。其中，很大几率会遇到的情况就是缺失值。

### 2.1 什么是缺失值？

缺失值主要是指数据丢失的现象，也就是数据集中的某一块数据不存在。除此之外，存在但明显不正确数据也被归为缺失值一类。例如，在一个时间序列数据集中，某一段数据突然发生了时间流错乱，那么这一小块数据就是毫无意义的，可以被归为缺失值。

当然，除了原始数据集就已经存在缺失值以外。当我们用到前面章节中的提到的索引对齐（`reindex()`）的方法时，也容易人为导致缺失值的产生。举个例子：

首先，我们生成一个 DataFrame。

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.rand(5, 5), index=list('cafed'), columns=list('ABCDE'))

print df
```

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

	A	B	C	D	E
c	0.868389	0.183820	0.700115	0.019156	0.409585
a	0.983965	0.949280	0.429262	0.479958	0.258734
f	0.045322	0.648676	0.652105	0.101033	0.181820
e	0.346455	0.158358	0.703988	0.001316	0.698206
d	0.964799	0.402322	0.777566	0.340223	0.887303

实验楼  
shiyantalou.com

然后，我们使用 `reindex()` 完成索引对齐。

```
print df.reindex(list('abcde'))
```

```
In [5]: print df.reindex(list('abcde'))
```

	A	B	C	D	E
a	0.983965	0.949280	0.429262	0.479958	0.258734
b	NaN	NaN	NaN	NaN	NaN
c	0.868389	0.183820	0.700115	0.019156	0.409585
d	0.964799	0.402322	0.777566	0.340223	0.887303
e	0.346455	0.158358	0.703988	0.001316	0.698206



由于原始数据集中，没有索引 b，所以对齐之后，b 后面全部为缺失值，也就造成了数据缺失。

## 2.2 检测缺失值

Pandas 为了方便地检测缺失值，将不同类型数据的缺失均采用 NaN 标记。这里的 NaN 代表 Not a Number，它仅仅是作为一个标记。例外是，在时间序列里，时间戳的丢失采用 NaT 标记。

Pandas 中用于检测缺失值主要用到两个方法，分别是：isnull() 和 notnull()，故名思意就是「是缺失值」和「不是缺失值」。默认会返回布尔值用于判断。

下面，演示一下这两个方法的作用，我们这里沿用上面进行索引对齐后的数据。

```
df2 = df.reindex(list('abcde'))
```

```
df2.isnull()
```

```
df2.notnull()
```

```
In [12]: df2.isnull()
Out[12]:
```

	A	B	C	D	E
a	False	False	False	False	False
b	True	True	True	True	True
c	False	False	False	False	False
d	False	False	False	False	False
e	False	False	False	False	False

```
In [13]: df2.notnull()
Out[13]:
```

	A	B	C	D	E
a	True	True	True	True	True
b	False	False	False	False	False
c	True	True	True	True	True
d	True	True	True	True	True
e	True	True	True	True	True



然后，我们来看一下对时间序列缺失值的检测，对上面的 df2 数据集进行稍微修改。

```
# 插入 T 列，并打上时间戳
df2.insert(value=pd.Timestamp('2017-10-1'),loc=0,column='T')

# 将 T 列的 1, 3, 5 行置为缺失值
df2.loc[['a','c','e'],['T']] = np.nan
```

这里，我们也更清晰看到，时间序列的缺失值用 NaT 标记。我们对 df2 进行缺失值检测。

```
df2.isnull()
df2.notnull()
```

```
In [9]: print df2
```

	T	A	B	C	D	E
a	NaT	0.243705	0.135536	0.551735	0.124507	0.569908
b	2017-10-01	NaN	NaN	NaN	NaN	NaN
c	NaT	0.441094	0.730500	0.870138	0.548461	0.996243
d	2017-10-01	0.705067	0.470127	0.957446	0.943162	0.154077
e	NaT	0.574563	0.288122	0.618536	0.523963	0.394824

```
In [10]: df2.isnull()
```

```
Out[10]:
```

	T	A	B	C	D	E
a	True	False	False	False	False	False
b	False	True	True	True	True	True
c	True	False	False	False	False	False
d	False	False	False	False	False	False
e	True	False	False	False	False	False

```
In [11]: df2.notnull()
```

```
Out[11]:
```

	T	A	B	C	D	E
a	False	True	True	True	True	True
b	True	False	False	False	False	False
c	False	True	True	True	True	True
d	True	True	True	True	True	True
e	False	True	True	True	True	True



## 三、填充和清除缺失值

上面已经对缺省值的产生、检测进行了介绍。那么，我们面对缺失值时，到底有哪些实质性的措施呢？接下来，就来看一看如何完成对缺失值填充和清除。

填充和清除都是两个极端。如果你感觉有必要保留缺失值所在的列或行，那么就需要对缺失值进行填充。如果没有必要保留，就可以选择清除缺失值。

Pandas 中，填充缺失的方法为 `fillna()`，清除为 `dropna()`。

### 3.1 填充缺失值 `fillna()`

首先，我们看一看 `fillna()` 的使用方法。重新打开一个 ipython 终端，我们生成和上面相似的数据。



```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.rand(9, 5), columns=list('ABCDE'))

# 插入 T 列, 并打上时间戳
df.insert(value=pd.Timestamp('2017-10-1'), loc=0, column='Time')

# 将 1, 3, 5 列的 1, 3, 5 行置为缺失值
df.iloc[[1,3,5,7], [0,2,4]] = np.nan

# 将 2, 4, 6 列的 2, 4, 6 行置为缺失值
df.iloc[[2,4,6,8], [1,3,5]] = np.nan
```

```
In [7]: df
Out[7]:
```

	Time	A	B	C	D	E
0	2017-10-01	0.592195	0.458845	0.918613	0.559202	0.628064
1	NaT	0.356603	NaN	0.888663	NaN	0.428797
2	2017-10-01	NaN	0.641348	NaN	0.242371	NaN
3	NaT	0.520168	NaN	0.040097	NaN	0.733489
4	2017-10-01	NaN	0.644487	NaN	0.926258	NaN
5	NaT	0.461074	NaN	0.471301	NaN	0.130112
6	2017-10-01	NaN	0.072048	NaN	0.444503	NaN
7	NaT	0.836771	NaN	0.495634	NaN	0.103467
8	2017-10-01	NaN	0.788112	NaN	0.109669	NaN

我们用相同的标量值替换 NaN , 比如用 0。

```
df.fillna(0)
```

```
In [10]: df.fillna(0)
Out[10]:
```

	Time	A	B	C	D	E
0	2017-10-01	0.592195	0.458845	0.918613	0.559202	0.628064
1	1970-01-01	0.356603	0.000000	0.888663	0.000000	0.428797
2	2017-10-01	0.000000	0.641348	0.000000	0.242371	0.000000
3	1970-01-01	0.520168	0.000000	0.040097	0.000000	0.733489
4	2017-10-01	0.000000	0.644487	0.000000	0.926258	0.000000
5	1970-01-01	0.461074	0.000000	0.471301	0.000000	0.130112
6	2017-10-01	0.000000	0.072048	0.000000	0.444503	0.000000
7	1970-01-01	0.836771	0.000000	0.495634	0.000000	0.103467
8	2017-10-01	0.000000	0.788112	0.000000	0.109669	0.000000

注意, 这里的填充并不会直接覆盖原数据集, 你可以重新输出 df 比较结果。

除了直接填充值，我们还可以通过参数，将缺失值前面或者后面的值填充给相应的缺失值。例如使用缺失值前面的值进行填充：

```
df.fillna(method='pad')
```

```
In [11]: df.fillna(method='pad')
```

```
Out[11]:
```

	Time	A	B	C	D	E
0	2017-10-01	0.592195	0.458845	0.918613	0.559202	0.628064
1	2017-10-01	0.356603	0.458845	0.888663	0.559202	0.428797
2	2017-10-01	0.356603	0.641348	0.888663	0.242371	0.428797
3	2017-10-01	0.520168	0.641348	0.040097	0.242371	0.733489
4	2017-10-01	0.520168	0.644487	0.040097	0.926258	0.733489
5	2017-10-01	0.461074	0.644487	0.471301	0.926258	0.130112
6	2017-10-01	0.461074	0.072048	0.471301	0.444503	0.130112
7	2017-10-01	0.836771	0.072048	0.495634	0.444503	0.103467
8	2017-10-01	0.836771	0.788112	0.495634	0.109669	0.103467

或者是后面的值：

```
df.fillna(method='bfill')
```

```
In [12]: df.fillna(method='bfill')
```

```
Out[12]:
```

	Time	A	B	C	D	E
0	2017-10-01	0.592195	0.458845	0.918613	0.559202	0.628064
1	2017-10-01	0.356603	0.641348	0.888663	0.242371	0.428797
2	2017-10-01	0.520168	0.641348	0.040097	0.242371	0.733489
3	2017-10-01	0.520168	0.644487	0.040097	0.926258	0.733489
4	2017-10-01	0.461074	0.644487	0.471301	0.926258	0.130112
5	2017-10-01	0.461074	0.072048	0.471301	0.444503	0.130112
6	2017-10-01	0.836771	0.072048	0.495634	0.444503	0.103467
7	2017-10-01	0.836771	0.788112	0.495634	0.109669	0.103467
8	2017-10-01	NaN	0.788112	NaN	0.109669	NaN

最后一行由于没有对于的后序值，自然继续存在缺失值。

上面的例子中，我们的缺失值是间隔存在的。那么，如果存在连续的缺失值是怎样的情况呢？试一试。首先，我们将数据集的第 2，4，6 列的第 3，5 行也置为缺失值。

```
df.iloc[[3,5], [1,3,5]] = np.nan
```

```
In [13]: df.iloc[[3,5], [1,3,5]] = np.nan
```

```
In [14]: df
```

```
Out[14]:
```

	Time	A	B	C	D	E
0	2017-10-01	0.592195	0.458845	0.918613	0.559202	0.628064
1	NaT	0.356603	NaN	0.888663	NaN	0.428797
2	2017-10-01	NaN	0.641348	NaN	0.242371	NaN
3	NaT	NaN	NaN	NaN	NaN	NaN
4	2017-10-01	NaN	0.644487	NaN	0.926258	NaN
5	NaT	NaN	NaN	NaN	NaN	NaN
6	2017-10-01	NaN	0.072048	NaN	0.444503	NaN
7	NaT	0.836771	NaN	0.495634	NaN	0.103467
8	2017-10-01	NaN	0.788112	NaN	0.109669	NaN

实验楼  
shiyancelou.com

然后来正向填充：

```
df.fillna(method='pad')
```

```
In [15]: df.fillna(method='pad')
```

```
Out[15]:
```

	Time	A	B	C	D	E
0	2017-10-01	0.592195	0.458845	0.918613	0.559202	0.628064
1	2017-10-01	0.356603	0.458845	0.888663	0.559202	0.428797
2	2017-10-01	0.356603	0.641348	0.888663	0.242371	0.428797
3	2017-10-01	0.356603	0.641348	0.888663	0.242371	0.428797
4	2017-10-01	0.356603	0.644487	0.888663	0.926258	0.428797
5	2017-10-01	0.356603	0.644487	0.888663	0.926258	0.428797
6	2017-10-01	0.356603	0.072048	0.888663	0.444503	0.428797
7	2017-10-01	0.836771	0.072048	0.495634	0.444503	0.103467
8	2017-10-01	0.836771	0.788112	0.495634	0.109669	0.103467

实验楼  
shiyancelou.com

可以看到，连续缺失值也是按照前序数值进行填充的，并且完全填充。这里，我们可以通过 `limit=` 参数设置连续填充的限制数量。

```
df.fillna(method='pad', limit=1)
```



```
In [16]: df.fillna(method='pad', limit=1)
```

```
Out[16]:
```

	Time	A	B	C	D	E
0	2017-10-01	0.592195	0.458845	0.918613	0.559202	0.628064
1	2017-10-01	0.356603	0.458845	0.888663	0.559202	0.428797
2	2017-10-01	0.356603	0.641348	0.888663	0.242371	0.428797
3	2017-10-01	NaN	0.641348	NaN	0.242371	NaN
4	2017-10-01	NaN	0.644487	NaN	0.926258	NaN
5	2017-10-01	NaN	0.644487	NaN	0.926258	NaN
6	2017-10-01	NaN	0.072048	NaN	0.444503	NaN
7	2017-10-01	0.836771	0.072048	0.495634	0.444503	0.103467
8	2017-10-01	0.836771	0.788112	0.495634	0.109669	0.103467

除了上面的填充方式，还可以通过 Pandas 自带的求平均值方法来填充特定列或行。举个例子：

```
df.fillna(df.mean()['C':'E'])
```

对 C 列和 E 列用平均值填充。

```
In [17]: df.fillna(df.mean()['C':'E'])
```

```
Out[17]:
```

	Time	A	B	C	D	E
0	2017-10-01	0.592195	0.458845	0.918613	0.559202	0.628064
1	NaT	0.356603	NaN	0.888663	0.456401	0.428797
2	2017-10-01	NaN	0.641348	0.767637	0.242371	0.386776
3	NaT	NaN	NaN	0.767637	0.456401	0.386776
4	2017-10-01	NaN	0.644487	0.767637	0.926258	0.386776
5	NaT	NaN	NaN	0.767637	0.456401	0.386776
6	2017-10-01	NaN	0.072048	0.767637	0.444503	0.386776
7	NaT	0.836771	NaN	0.495634	0.456401	0.103467
8	2017-10-01	NaN	0.788112	0.767637	0.109669	0.386776

## 3.2 清除缺失值 `dropna()`


上面演示了缺失值填充。但有些时候，缺失值比较少或者是填充无意义时，就可以直接清除了。

由于填充和清除在赋值之前，均不会影响原有的数据。所以，我们这里依旧延续使用上面的 `df`。

```
df.dropna()
```

```
In [18]: df.dropna()
Out[18]:
```

	Time	A	B	C	D	E
0	2017-10-01	0.592195	0.458845	0.918613	0.559202	0.628064




我们可以看到，`dropna()` 方法带来的默认效果就是，凡是存在缺失值的行均被直接移除。此时，`dropna()` 里面有一个默认参数是 `axis=0`，代表依据行来移除。

如果我们像将凡是有缺失值的列直接移除，可以将 `axis=1`，试一试。

```
df.dropna(axis=1)
```

由于上面的 `df` 中，每一列都有缺失值，所以全部被移除了。

```
In [19]: df.dropna(axis=1)
Out[19]:
Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4, 5, 6, 7, 8]
```



## 四、插值 `interpolate()`

插值是数值分析中一种方法。简而言之，就是借助于一个函数（线性或非线性），再根据已知数据去求解未知数据的值。插值在数据领域非常常见，它的好处在于，可以尽量去还原数据本身的样子。

Pandas 中的插值，通过 `interpolate()` 方法完成，默认为线性插值，即 `method='linear'`。除此之外，还有 `{'linear', 'time', 'index', 'values', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'barycentric', 'krogh', 'polynomial', 'spline', 'piecewise_polynomial', 'from_derivatives', 'pchip', 'akima'}` 等插值方法可供选择。

举个例子：

```
import pandas as pd
import numpy as np

# 生成一个 DataFrame
df = pd.DataFrame({'A': [1.1, 2.2, np.nan, 4.5, 5.7, 6.9], 'B': [.21,
np.nan, np.nan, 3.1, 11.7, 13.2]})
```

```
In [4]: df
Out[4]:
```

	A	B
0	1.1	0.21
1	2.2	NaN
2	NaN	NaN
3	4.5	3.10
4	5.7	11.70
5	6.9	13.20



对于上面存在的缺失值，如果通过前后值，或者平均值来填充是不太能反映出趋势的。这时候，插值最好使。我们用默认的线性插值试一试。

```
df.interpolate()
```

```
In [5]: df.interpolate()
Out[5]:
```

	A	B
0	1.10	0.210000
1	2.20	1.173333
2	3.35	2.136667
3	4.50	3.100000
4	5.70	11.700000
5	6.90	13.200000



如果你熟悉 Matplotlib，我们可以将数据绘制成图看一看趋势。图中，第 2，3 点的坐标是我们插值的结果。



上面提到了许多插值的方法，也就是 `method=`。下面给出几条选择的建议：

1. 如果你的数据增长速率越来越快，可以选择 `method='quadratic'` 二次插值。
2. 如果数据集呈现出累计分布的样子，推荐选择 `method='pchip'`。
3. 如果需要填补缺省值，以平滑绘图为目标，推荐选择 `method='akima'`。

当然，最后提到的 `method='akima'`，需要你的环境中安装了 Scipy 库。除此之外，`method='barycentric'` 和 `method='pchip'` 同样也需要 Scipy 才能使用。

## 五、实验总结

本章节学习了如何通过 Pandas 处理数据集中的缺失值。并了解了检测缺失值、填充缺失值、清除缺失值以及相关的插值方法。当然，这些内容相对于强大的 Pandas 而言仅仅是开始，每一种方法都还包含很多参数。入门之后，需要通过官方文档来学习更高阶的使用方法。

## 六、课后作业



自己尝试生成一个 DataFrame，并制造一些零散的缺失值。最后，通过不同的插值方法完成缺失值的填充。

*\*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 数据选择与过滤 \(/courses/906/labs/3377/document\)](/courses/906/labs/3377/document)

下一节：[Pandas 时间序列分析 \(/courses/906/labs/3394/document\)](/courses/906/labs/3394/document)