

## 处理 NaN

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [1]: import pandas as pd

items = [{'bikes': 20, 'pants': 30, 'watches': 35, 'shirts': 15, 'shoes': 8, 'suits': 45},
         {'watches': 10, 'glasses': 50, 'bikes': 15, 'pants': 5, 'shirts': 2, 'shoes': 5, 'suits': 7},
         {'bikes': 20, 'pants': 30, 'watches': 35, 'glasses': 4, 'shoes': 10}]

store_items = pd.DataFrame(items, index=['store 1', 'store 2', 'store 3'])
store_items
```

Out[1]:

	bikes	glasses	pants	shirts	shoes	suits	watches
store 1	20	NaN	30	15.0	8	45.0	35
store 2	15	50.0	5	2.0	5	7.0	10
store 3	20	4.0	30	NaN	10	NaN	35

Below the output, there is a text box with the prompt "In [ ]: |" and a blue circular icon.

在开始分析数据或用数据训练学习算法之前

00:00 / 04:45

1x CC

正如之前提到的，在能够使用大型数据集训练学习算法之前，我们通常需要先清理数据。也就是说，我们需要通过某个方法检测并更正数据中的错误。虽然任何给定数据集可能会出现各种糟糕的数据，例如离群值或不正确的值，但是我们几乎始终会遇到的糟糕数据类型是缺少值。正如之前看到的，Pandas 会为缺少的值分配 `NaN` 值。在这节课，我们将学习如何检测和处理 `NaN` 值。

首先，我们将创建一个具有一些 `NaN` 值的 DataFrame。

```
# We create a list of Python dictionaries
items2 = [{'bikes': 20, 'pants': 30, 'watches': 35, 'shirts': 15,
          {'watches': 10, 'glasses': 50, 'bikes': 15, 'pants': 5, 'shirts': 2, 'shoes': 5, 'suits': 7},
          {'bikes': 20, 'pants': 30, 'watches': 35, 'glasses': 4, 'shoes': 10}]

# We create a DataFrame and provide the row index
store_items = pd.DataFrame(items2, index = ['store 1', 'store 2',
```

	bikes	glasses	pants	shirts	shoes	suits	watc
store 1	20	NaN	30	15.0	8	45.0	35
store 2	15	50.0	5	2.0	5	7.0	10
store 3	20	4.0	30	NaN	10	NaN	35

可以清晰地看出，我们创建的 DataFrame 具有 3 个 NaN 值：商店 1 中有一个，商店 3 中有两个。但是，如果我们向 DataFrame 中加载非常庞大的数据集，可能有数百万条数据，那么就不太容易直观地发现 NaN 值的数量。对于这些情形，我们结合使用多种方法来计算数据中的 NaN 值的数量。以下示例同时使用了 `.isnull()` 和 `sum()` 方法来计算我们的 DataFrame 中的 NaN 值的数量。

```
# We count the number of NaN values in store_items
x = store_items.isnull().sum().sum()

# We print x
print('Number of NaN values in our DataFrame:', x)
```

Number of NaN values in our DataFrame: 3

在上述示例中，`.isnull()` 方法返回一个大小和 `store_items` 一样的布尔型 DataFrame，并用 `True` 表示具有 NaN 值的元素，用 `False` 表示非 NaN 值的元素。我们来看一个示例：

```
store_items.isnull()
```



<b>store</b> 1	False	True	False	False	False	False	Fal.
<b>store</b> 2	False	False	False	False	False	False	Fal.
<b>store</b> 3	False	False	False	True	False	True	Fal.

在 Pandas 中，逻辑值 `True` 的数字值是 1，逻辑值 `False` 的数字值是 0。因此，我们可以通过数逻辑值 `True` 的数量数出 `NaN` 值的数量。为了数逻辑值 `True` 的总数，我们使用 `.sum()` 方法两次。要使用该方法两次，是因为第一个 `sum()` 返回一个 Pandas Series，其中存储了列上的逻辑值 `True` 的总数，如下所示：

```
store_items.isnull().sum()
```

```
bikes      0
glasses    1
pants      0
shirts     1
shoes      0
suits      1
watches    0
dtype: int64
```

第二个 `sum()` 将上述 Pandas Series 中的 1 相加。

除了数 `NaN` 值的数量之外，我们还可以采用相反的方式，我们可以数 *非 NaN* 值的数量。为此，我们可以使用 `.count()` 方法，如下所示：

```
# We print the number of non-NaN values in our DataFrame
print()
print('Number of non-NaN values in the columns of our DataFrame:\n')
```



```
glasses    2
pants      3
shirts     2
shoes      3
suits      2
watches    3
dtype: int64
```

现在我们已经知道如何判断数据集中是否有任何 NaN 值，下一步是决定如何处理这些 NaN 值。通常，我们有两种选择，可以删除或替换 NaN 值。在下面的示例中，我们将介绍这两种方式。

首先，我们将学习如何从 DataFrame 中删除包含任何 NaN 值的行或列。如果 `axis = 0`，`.dropna(axis)` 方法将删除包含 NaN 值的任何行，如果 `axis = 1`，`.dropna(axis)` 方法将删除包含 NaN 值的任何列。我们来看一些示例：

```
# We drop any rows with NaN values
store_items.dropna(axis = 0)
```

	bikes	glasses	pants	shirts	shoes	suits	watches
store 1	15	50.0	5	2.0	5	7.0	10
store 2	15	50.0	5	2.0	5	7.0	10

```
# We drop any columns with NaN values
store_items.dropna(axis = 1)
```

	bikes	pants	shoes	watches
store 1	20	30	8	35
store 2	15	5	5	10

store 3	20	30	10	35
---------	----	----	----	----

注意，`.dropna()` 方法不在原地地删除具有 `NaN` 值的行或列。也就是说，原始 `DataFrame` 不会改变。你始终可以在 `dropna()` 方法中将关键字 `inplace` 设为 `True`，在原地删除目标行或列。

现在，我们不再删除 `NaN` 值，而是将它们替换为合适的值。例如，我们可以选择将所有 `NaN` 值替换为 0。为此，我们可以使用 `.fillna()` 方法，如下所示。

```
# We replace all NaN values with 0
store_items.fillna(0)
```

	bikes	glasses	pants	shirts	shoes	suits	watc
store 1	20	0.0	30	15.0	8	45.0	35
store 2	15	50.0	5	2.0	5	7.0	10
store 3	20	4.0	30	0.0	10	0.0	35

我们还可以使用 `.fillna()` 方法将 `NaN` 值替换为 `DataFrame` 中的上个值，称之为前向填充。在通过前向填充替换 `NaN` 值时，我们可以使用列或行中的上个值。`.fillna(method = 'ffill', axis)` 将通过前向填充 (`ffill`) 方法沿着给定 `axis` 使用上个已知值替换 `NaN` 值。我们来看一些示例：

```
# We replace NaN values with the previous value in the column
store_items.fillna(method = 'ffill', axis = 0)
```

	bikes	glasses	pants	shirts	shoes	suits	watc
--	-------	---------	-------	--------	-------	-------	------



store 1	20	NaN	30	15.0	8	45.0	35
store 2	15	50.0	5	2.0	5	7.0	10
store 3	20	4.0	30	2.0	10	7.0	35

注意 **store 3** 中的两个 **NaN** 值被替换成了它们所在列中的上个值。但是注意，**store 1** 中的 **NaN** 值没有被替换掉。因为这列前面没有值，因为 **NaN** 值是该列的第一个值。但是，如果使用上个行值进行前向填充，则不会发生这种情况。我们来看看具体情形：

```
# We replace NaN values with the previous value in the row
store_items.fillna(method = 'ffill', axis = 1)
```

	bikes	glasses	pants	shirts	shoes	suits	watches
store 1	20.0	20.0	30.0	15.0	8.0	45.0	35.0
store 2	15.0	50.0	5.0	2.0	5.0	7.0	10.0
store 3	20.0	4.0	30.0	30.0	10.0	10.0	35.0

我们看到，在这种情形下，所有 **NaN** 值都被替换成了之前的行值。

同样，你可以选择用 DataFrame 中之后的值替换 **NaN** 值，称之为**后向填充**。

`.fillna(method = 'backfill', axis)` 将通过后向填充 (`backfill`) 方法沿着给定 `axis` 使用下个已知值替换 **NaN** 值。和前向填充一样，我们可以选择使用行值或列值。我们来看一些示例：



```
store_items.fillna(method = 'backfill', axis = 0)
```

	bikes	glasses	pants	shirts	shoes	suits	watc
store 1	20	50.0	30	15.0	8	45.0	35
store 2	15	50.0	5	2.0	5	7.0	10
store 3	20	4.0	30	NaN	10	NaN	35

注意，**store 1** 中的 **NaN** 值被替换成了它所在列的下个值。但是注意，**store 3** 中的两个 **NaN** 值没有被替换掉。因为这些列中没有下个值，这些 **NaN** 值是这些列中的最后一个值。但是，如果使用下个行值进行后向填充，则不会发生这种情况。我们来看看具体情形：

```
# We replace NaN values with the next value in the row
store_items.fillna(method = 'backfill', axis = 1)
```

	bikes	glasses	pants	shirts	shoes	suits	watc
store 1	20.0	30.0	30.0	15.0	8.0	45.0	35.
store 2	15.0	50.0	5.0	2.0	5.0	7.0	10.
store 3	20.0	4.0	30.0	10.0	10.0	35.0	35.

注意，**.fillna()** 方法不在原地地替换（填充）**NaN** 值。也就是说，原始 DataFrame 不会改变。你始终可以在 **fillna()** 函数中将关键字 **inplace** 设为 **True**，在原地替换 **NaN** 值。

定 `axis` 的值替换 `NaN` 值。我们来看一些示例：

```
# We replace NaN values by using linear interpolation using column
store_items.interpolate(method = 'linear', axis = 0)
```

	<i>bikes</i>	<i>glasses</i>	<i>pants</i>	<i>shirts</i>	<i>shoes</i>	<i>suits</i>	<i>watc</i>
<i>store 1</i>	20	NaN	30	15.0	8	45.0	35
<i>store 2</i>	15	50.0	5	2.0	5	7.0	10
<i>store 3</i>	20	4.0	30	2.0	10	7.0	35

注意，**store 3** 中的两个 `NaN` 值被替换成了线性插值。但是注意，**store 1** 中的 `NaN` 值没有被替换掉。因为该 `NaN` 值是该列中的第一个值，因为它前面没有数据，因此插值函数无法计算值。现在，我们使用行值插入值：

```
# We replace NaN values by using linear interpolation using row va
store_items.interpolate(method = 'linear', axis = 1)
```

	<i>bikes</i>	<i>glasses</i>	<i>pants</i>	<i>shirts</i>	<i>shoes</i>	<i>suits</i>	<i>watc</i>
<i>store 1</i>	20.0	25.0	30.0	15.0	8.0	45.0	35.
<i>store 2</i>	15.0	50.0	5.0	2.0	5.0	7.0	10.
<i>store 3</i>	20.0	4.0	30.0	20.0	10.0	22.5	35.





Search or ask questions in  
[Knowledge](#).

Ask peers or mentors for help in  
[Student Hub](#).

下一项