

# 7 Uncommon But Useful Pandas Functions

Functions to boost your Pandas skills



Photo by [Erik Mclean](#) on [Unsplash](#)

Pandas is one of the most popular data analysis libraries. There are numerous Pandas functions and methods that ease and expedite the data cleaning and analysis process.

Pandas also provides some functions that are not so common but come in handy for certain tasks. In this post, we will cover 7 uncommon Pandas functions.

The functions that will be discussed are:

- Clip
- Eval
- Combine\_first
- Transform
- Melt
- Diff
- Shift

We always start importing the dependencies.

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

## 1. Clip

Clip function trims a dataframe based on the given upper or lower values. It does not drop the rows that are outside the specified range by the upper or lower values. Instead, if a value is outside the boundaries, the clip function makes them equal to the appropriate boundary value.

Consider the following dataframe.

| df |           |      |          |
|----|-----------|------|----------|
|    | cola      | colb | colc     |
| 0  | 1.278426  | 1    | 0.079890 |
| 1  | -0.617994 | 3    | 0.421217 |
| 2  | 1.095972  | -3   | 0.417555 |
| 3  | 0.159107  | -5   | 0.043130 |
| 4  | 1.802620  | -2   | 0.258976 |
| 5  | -0.287144 | -4   | 0.113189 |

(image by author)

Let's say we do not want to have any negative values and want to make them equal to zero. It can be done by setting the lower parameter of the clip function as 0.

```
df.clip(lower=0)
```

|   | cola     | colb | colc     |
|---|----------|------|----------|
| 0 | 1.278426 | 1    | 0.079890 |
| 1 | 0.000000 | 3    | 0.421217 |
| 2 | 1.095972 | 0    | 0.417555 |
| 3 | 0.159107 | 0    | 0.043130 |
| 4 | 1.802620 | 0    | 0.258976 |
| 5 | 0.000000 | 0    | 0.113189 |

(image by author)

All negative values are equal to zero now. We can also assign an upper limit. For instance, we can trim the values

to be between 0 and 1.

```
df.clip(lower=0, upper=1)
```

(image by author)

## 2. Eval

The eval function allows for manipulating or modifying a dataframe by passing an operation as a string.

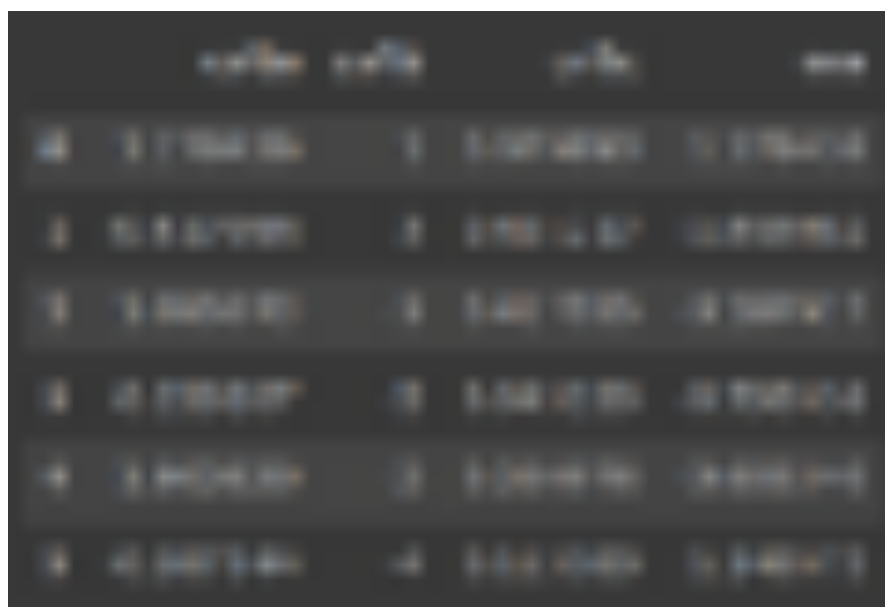
For instance, the following code will modify the values in the "cola".

```
df.eval("cola = cola * 10")
```

(image by author)

We can also create a new column:

```
df.eval("new = cola * colb")
```



|   | cola     | colb | cola     | new      |
|---|----------|------|----------|----------|
| 0 | 1.176471 | 1    | 1.176471 | 1.176471 |
| 1 | 0.842105 | 2    | 0.842105 | 1.684211 |
| 2 | 0.507843 | 3    | 0.507843 | 1.523530 |
| 3 | 0.173478 | 4    | 0.173478 | 0.693913 |
| 4 | 0.842105 | 5    | 0.842105 | 4.210524 |
| 5 | 0.507843 | 3    | 0.507843 | 1.523530 |
| 6 | 0.842105 | 4    | 0.842105 | 3.368421 |

It is important to note that we need to set the inplace parameter to save the changes. Otherwise, the eval function will return a modified version of the dataframe but not change the original one.

### 3. Combine\_first

The combine\_first function updates the missing values in a dataframe by using the values in another dataframe. The matching criterion is the position in terms of row and column.

Consider the following two dataframes.

|   | cola | colb | colc |   | cola | colb | colc |
|---|------|------|------|---|------|------|------|
| 0 | 1    | 3    | a    | 0 | 0.0  | 1.0  | NaN  |
| 1 | 3    | 3    | b    | 1 | 4.0  | NaN  | b    |
| 2 | 4    | 3    | d    | 2 | NaN  | 4.0  | k    |
| 3 | 6    | 4    | g    | 3 | 2.0  | 2.0  | NaN  |
| 4 | 7    | 1    | f    | 4 | 8.0  | 1.0  | f    |

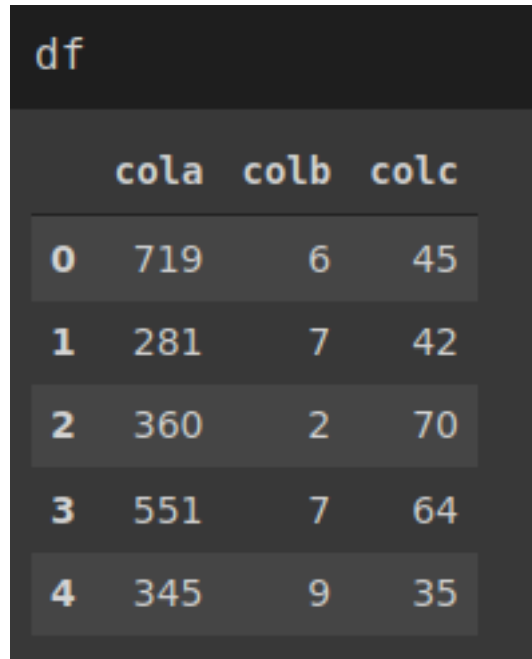
We can update the missing values in the second dataframe based on the first dataframe.

```
df2.combine_first(df1)
```

### 4. Transform

The transform function modifies the values in a dataframe according to the given function. It can be a simple aggregation or a lambda expression.

Consider the following dataframe.



|   | cola | colb | colc |
|---|------|------|------|
| 0 | 719  | 6    | 45   |
| 1 | 281  | 7    | 42   |
| 2 | 360  | 2    | 70   |
| 3 | 551  | 7    | 64   |
| 4 | 345  | 9    | 35   |

(image by author)

We can take the log of each value by using the transform function.

```
df.transform(lambda x: np.log(x))
```

(image by author)

One useful feature of the transform function is that it accepts multiple functions. We can specify them in a list as below.

```
df.transform([lambda x: np.log(x), np.sqrt])
```



(image by author)

## 5. Melt

The melt function converts a dataframe from wide to long format. In the wide form, the similar variables are represented as separate columns. On the other hand, the long format contains a column to store the values of these variables and another column to store the name of the variable.

It is better to have the dataframe in the long format for certain tasks. The melt function provides a quite simple way for this conversion. It will be more clear when we do an example.

Consider the following dataframe in wide format.

|   | name   | day1 | day2 | day3 | day4 | day5 | day6 | day7 | day8 |
|---|--------|------|------|------|------|------|------|------|------|
| 0 | Jane   | 51   | 75   | 61   | 52   | 98   | 30   | 28   | 71   |
| 1 | John   | 31   | 84   | 88   | 63   | 24   | 53   | 27   | 72   |
| 2 | Ashley | 56   | 48   | 33   | 92   | 54   | 52   | 74   | 83   |
| 3 | Adam   | 72   | 21   | 26   | 15   | 57   | 61   | 40   | 57   |
| 4 | Mike   | 31   | 54   | 24   | 96   | 62   | 34   | 6    | 59   |
| 5 | Jenny  | 28   | 48   | 31   | 14   | 58   | 70   | 2    | 10   |

(image by author)

The dataframe contains contains daily measurements for some people. The melt function can be used to convert it to a long format as below.

```
df_long = pd.melt(df, id_vars='name')df_long.head()
```

|   | name   | variable | value |
|---|--------|----------|-------|
| 0 | Jane   | day1     | 51    |
| 1 | John   | day1     | 31    |
| 2 | Ashley | day1     | 56    |
| 3 | Adam   | day1     | 72    |
| 4 | Mike   | day1     | 31    |
| 5 | Jenny  | day1     | 28    |
| 6 | Jane   | day2     | 75    |
| 7 | John   | day2     | 84    |
| 8 | Ashley | day2     | 48    |
| 9 | Adam   | day2     | 21    |

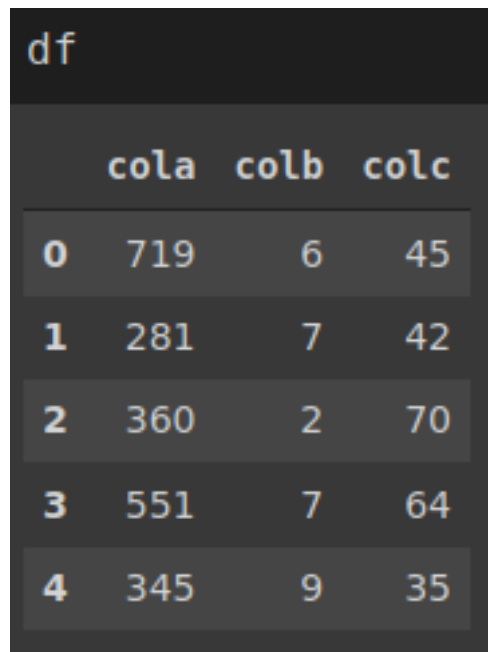
(image by author)



## 6. Diff

The `diff` function is used to calculate the difference between two consecutive rows or columns depending on the `axis` parameter.

Consider the following dataframe.



|   | cola | colb | colc |
|---|------|------|------|
| 0 | 719  | 6    | 45   |
| 1 | 281  | 7    | 42   |
| 2 | 360  | 2    | 70   |
| 3 | 551  | 7    | 64   |
| 4 | 345  | 9    | 35   |

(image by author)

We want to create a new column that contains the difference between the consecutive values in "colc".

```
df['diff_c'] = df['colc'].diff()
```

(image by author)

Since the first row does not have any previous row, the first value of the `diff_c` column is null.

## Conclusion

What we have covered in this article is only a small part of Pandas abilities in data analysis process but will certainly be useful for your tasks.

It is not reasonable to try to learn all at once. Instead, learning small chunks and absorbing the information with practice will help you build comprehensive data analysis skills.

Thank you for reading. Please let me know if you have any feedback.