8 Ways to Filter Pandas Dataframes

A practical guide for efficient data analysis

Photo by Daphné Be Frenchie on Unsplash

Pandas is a popular data analysis and manipulation library for Python. The core data structure of Pandas is dataframe which stores data in tabular form with labelled rows and columns.

A common operation in data analysis is to filter values based on a condition or multiple conditions. Pandas provides a variety of ways to filter data points (i.e. rows). In this article, we will cover 8 different ways to filter a dataframe.

We start by importing the libraries.

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

Let's create a sample dataframe for the examples.

```
df = pd.DataFrame({name':['Jane','John','Ashley',
'ctg':['A','A','C','B','B','C','B'],
'val':np.random.random(7).round(2),
'val2':np.random.randint(1,10, size=7)})
```

	name	ctg	val	val2
0	Jane	A	0.43	1
1	John	A	0.67	1
2	Ashley	C	0.40	7
3	Mike	B	0.91	5
4	Emily	B	0.99	8
5	Jack	C	0.02	7
6	Catlin	B	1.00	3

df (image by author)

1. Logical operators

We can use the logical operators on column values to filter rows.

```
df[df.val > 0.5]        name        ctg        val
```

```
-------------------------------------------
1       John        A       0.67        1
3       Mike        B       0.91        5
4       Emily       B       0.99        8
6       Catlin      B       1.00        3
```

We have selected the rows in which the value in “val” column is greater than 0.5.

The logical operators also works on strings.

```
df[df.name > 'Jane']        name       ctg      \
-------------------------------------------
1       John        A       0.67        1
3       Mike        B       0.91        5
```

Only the names that come after ‘Jane’ in alphabetical order are selected.

2. Multiple logical operators

Pandas allows for combining multiple logical operators. For instance, we can apply conditions on both val and val2 columns as below.

```
df[(df.val > 0.5) & (df.val2 == 1)]        name
-------------------------------------------
1       John        A       0.67        1
```

The “&” signs stands for “and” , the “|” stands for “or”.

```
df[(df.val < 0.5) | (df.val2 == 7)]          name
-------------------------------------------------
0       Jane         A       0.43        1
2     Ashley         C       0.40        7
5       Jack         C       0.02        7
```

3. Isin

The isin method is another way of applying multiple condition for filtering. For instance, we can filter the names that exist in a given list.

```
names = ['John','Catlin','Mike']df[df.name.isin(n
-------------------------------------------------
1       John         A       0.67        1
3       Mike         B       0.91        5
6     Catlin         B       1.00        3
```

4. Str accessor

Pandas is a highly efficient library on textual data as well. The functions and methods under the str accessor provide flexible ways to filter rows based on strings.

For instance, we can select the names that start with the letter “J”.

```
df[df.name.str.startswith('J')]          name
-------------------------------------------------
0       Jane         A       0.43        1
```

```
1       John        A        0.67        1
5       Jack        C        0.02        7
```

The contains function under the str accessor returns the values that contain a given set of characters.

```
df[df.name.str.contains('y')]          name       c
---------------------------------------------------
2       Ashley        C        0.40        7
4        Emily        B        0.99        8
```

We can pass a longer set of characters to the contains function depending on the strings in the data.

5. Tilde (~)

The tilde operator is used for “not” logic in filtering. If we add the tilde operator before the filter expression, the rows that do not fit the condition are returned.

```
df[~df.name.str.startswith('J')]         name
---------------------------------------------------
2     Ashley        C        0.40        7
3       Mike        B        0.91        5
4      Emily        B        0.99        8
6     Catlin        B        1.00        3
```

We get the names that do not start with the letter “J”.

6. Query

The query function offers a little more flexibility at writing the conditions for filtering. We can pass the conditions as a string.

For instance, the following code returns the rows that belong to the B category and have a value higher than 0.5 in the val column.

```
df.query('ctg == "B" and val > 0.5')          name
------------------------------------------------
3        Mike        B        0.91        5
4       Emily        B        0.99        8
6      Catlin        B        1.00        3
```

7. Nlargest or nsmallest

In some cases, we do not have a specific range for filtering but just need the largest or smallest values. The nlargest and nsmallest functions allow for selecting rows that have the largest or smallest values in a column, respectively.

```
df.nlargest(3, 'val')          name       ctg       \
------------------------------------------------
6      Catlin        B        1.00        3
4       Emily        B        0.99        8
3        Mike        B        0.91        5
```

We specify the number of largest or smallest values to be selected and the name of the column.

```
df.nsmallest(2, 'val2')        name      ctg
-----------------------------------------
0       Jane       A       0.43       1
1       John       A       0.67       1
```

8. Loc and iloc

The loc and iloc methods are used to select rows or columns based on index or label.

- loc: select rows or columns using labels
- iloc: select rows or columns using indices

Thus, they can be used for filtering. However, we can only select a particular part of the dataframe without specifying a condition.

```
df.iloc[3:5, :] #rows 3 and 4, all columns
-----------------------------------------
3       Mike       B       0.91       5
4      Emily       B       0.99       8
```

If the dataframe has integer index, the indices and labels of the rows are the same. Thus, both loc and iloc accomplished the same thing on the rows.

```
df.loc[3:5, :] #rows 3 and 4, all columns
-----------------------------------------
3       Mike       B       0.91       5
4      Emily       B       0.99       8
```

Let's update the index of the dataframe to demonstrate the difference between loc and iloc better.

```
df.index = ['a','b','c','d','e','f','g']
```

	name	ctg	val	val2
a	Jane	A	0.43	1
b	John	A	0.67	1
c	Ashley	C	0.40	7
d	Mike	B	0.91	5
e	Emily	B	0.99	8
f	Jack	C	0.02	7
g	Catlin	B	1.00	3

df (image by author)

We cannot pass integers to the loc method now because the labels of indices are letters.

```
df.loc['b':'d', :]
```

	name	ctg	val	val
b	John	A	0.67	1
c	Ashley	C	0.40	7
d	Mike	B	0.91	5

Conclusion

We have covered 8 different ways of filtering rows in a

dataframe. All of them are useful and come in handy for particular cases.

Pandas is a powerful library for both data analysis and manipulation. It provides numerous functions and methods to handle data in tabular form. As with any other tool, the best way to learn Pandas is through practicing.

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