towards
data science

8 Ways to Filter a Pandas DataFrame by a Partial String or Pattern

How to inspect a DataFrame and return only the desired rows

Susan Maina 1 day ago · 10 min read★

Photo by Maurício Mascaro from Pexels

Filtering a DataFrame refers to checking its contents and returning only those that fit certain criteria. It is part of the data analysis task known as data wrangling and is efficiently done using the Pandas library of Python.

The idea is that once you have filtered this data, you can analyze it separately and gain insights that might be unique to this group, and inform the predictive modeling steps of the project moving forward.

In this article, we will use functions such as `Series.isin()` and `Series.str.contains()` to filter the data. I minimized the use of `apply()` and `Lambda` functions which use more code and are confusing to many people including myself. However, I will explain the code and include links to related articles.

We will use the Netflix dataset from Kaggle which contains details of TV shows and movies including the `title`, `director`, the `cast`, age `rating`, `year` of release, and `duration`. Let us now import the required libraries and load the dataset into our Jupyter notebook.

```
import pandas as pd

data = pd.read_csv('netflix_titles_nov_2019.csv')

data.head()
```

	show_id	title	director	cast	country	date_added	release_year	rating	duration	listed_in	description	type
0	81193313	Chocolate	NaN	Ha Ji-won, Yoon Kye-sang, Jang Seung-jo, Kang ...	South Korea	November 30, 2019	2019	TV-14	1 Season	International TV Shows, Korean TV Shows, Roman...	Brought together by meaningful meals in the pa...	TV Show
1	81197050	Guatemala: Heart of the Mayan World	Luis Ara, Ignacio Jaunsolo	Christian Morales	NaN	November 30, 2019	2019	TV-G	67 min	Documentaries, International Movies	From Sierra de las Minas to Esquipulas, explor...	Movie
2	81213894	The Zoya Factor	Abhishek Sharma	Sonam Kapoor, Dulquer Salmaan, Sanjay Kapoor, ...	India	November 30, 2019	2019	TV-14	135 min	Comedies, Dramas, International Movies	A goofy copywriter unwittingly convinces the l...	Movie
3	81082007	Atlantics	Mati Diop	Mama Sane, Amadou Mbow, Ibrahima Traore, Nicol...	France, Senegal, Belgium	November 29, 2019	2019	TV-14	106 min	Dramas, Independent Movies, International Movies	Arranged to marry a rich man, young Ada is cru...	Movie
4	80213643	Chip and Potato	NaN	Abigail Oliver, Andrea Libman, Briana Buckmast...	Canada, United Kingdom	NaN	2019	TV-Y	2 Seasons	Kids' TV	Lovable pug Chip starts kindergarten, makes ne...	TV Show

The netflix DataFrame by author

1. Filter rows that match a given String in a column

Here, we want to filter by the contents of a particular column. We will use the `Series.isin([list_of_values] )` function from Pandas which returns a 'mask' of `True` for every element in the column that *exactly* matches or `False` if it does not match any of the list values in the `isin()` function. Note that you must always include the value(s) in square brackets even if it is just one.

```
mask = data['type'].isin(['TV Show'])

#display the mask
mask.head()
```


Image by author

We then apply this mask to the whole DataFrame to return the rows where the condition was `True`. Note how the index positions where the mask was `True` above are the only rows returned in the filtered DataFrame below.

```
#Display first 5 rows
#of the filtered data

data[mask].head()
```

Note: `df.loc[mask]` generates the same results as `df[mask]`. This is especially useful when you want to select a few columns to display.

```
data.loc[mask, ['title','country','duration']]
```

Image by author

Other ways to generate the mask above;

- If you do not want to deal with a mix of upper and lowercase letters in the `isin()` function, first convert all the column's elements into lowercase.

```
mask = data['type'].str.lower().isin(['tv show'])
```

- We can also use the == equality operator which compares if two objects are the same. This will compare whether each element in a column is equal to the string provided.

```
mask = (data['type'] == 'TV Show')
```

- We can provide a list of strings like `isin(['str1','str2'])` and check if a column's elements match any of them. The two masks below return the same results.

```
mask = data['type'].isin(['Movie','TV Show'])

#or

mask = (data['type'] == 'Movie') | (data['type'] == 'TV Show')
```

The mask returned will be all Trues because the ‘type’ column contains only ‘Movie’ and ‘TV Show’ categories.

2. Filter rows where a partial string is present

Here, we want to check if a sub-string is present in a column.

For example, the `‘listed-in’` column contains the genres that each movie or show belongs to, separated by commas. I want to filter and return only those that have a ‘horror’ element in them because right now Halloween is upon us.

We will use the string method `Series.str.contains(‘pattern’, case=False, na=False)` where `‘pattern’` is the substring to search for, and `case=False` implies case insensitivity. `na=False` means that any `NaN` values in the column will be returned as False (meaning without the pattern) instead of as `NaN` which removes the boolean identity from the mask.

```
mask = data['listed_in'].str.contains('horror', case=False,
na=False)
```

We will then apply the mask to our data and display three sample rows of the filtered dataframe.

```
data[mask].sample(3)
```


Image by author

Other examples:

- We can also check for the presence of symbols in a column. For example, the `‘cast’` column contains the actors separated by commas, and we can check for rows where there is only one actor. This is by checking for rows with a comma (,) and then applying the filtering mask to the data using a tilde (~) to negate the statement.

```
mask = data['cast'].str.contains(',', na=False)

data[~mask].head()
```

Image by author

But now these results have `NaNs` because we used `na=False` and the tilde returns all rows where the mask was False. We will use `df.dropna(axis=0, subset='cast')` to the filtered data. We use `axis=0` to mean row-wise because we want to drop the row not the column. `subset=['cast']` checks only this column for `NaNs`.

```
data[~mask].dropna(axis=0, subset=['cast'])
```

Image by author

Note: To check for special characters such as `+` or `^`, use `regex=False` (the default is `True`) so that all characters are interpreted as normal strings not regex patterns. You can alternatively use the backslash escape character.

```
df['a'].str.contains('^', regex=False)

#or

df['a'].str.contains('\^')
```

3. Filter rows with either of two partial strings (OR)

You can check for the presence of any two *or* more strings and return `True` if any of the strings are present.

Let us check for either 'horrors' or 'stand-up comedies' to complement our emotional states after each watch.

We use `str.contains()` and pass the two strings separated by a vertical bar (|) which means 'or'.

```
pattern = 'horror|stand-up'

mask = data['listed_in'].str.contains(pattern, case=False, na=False)

data[mask].sample(3)
```

Image by author

- We can also use the long-form where we create two masks and pass them into our data using `|`.

Image by author

Note: You can create many masks and pass them into the data using the symbols `|` or `&`. The `&` means combine the masks and return `True` where *both* masks are True, while `|` means return True where *any* of the masks is `True`.

```
mask1 = (data['listed_in'].str.contains('horror', case=False,
na=False))

mask2 = (data['type'].isin(['TV Show']))

data[mask1 & mask2].head(3)
```

Filtered DataFrame by author

4. Filter rows where both strings are present (AND)

Sometimes, we want to check if multiple sub-strings appear in the elements of a column.

In this example, we will search for movies that were filmed in both US and Mexico countries.

- The code `str.contains('str1.*str2')` uses the symbols `.*` to search if both strings appear *strictly* in that order, where `str1` must appear first to return True.

```
pattern = 'states.*mexico'

mask = data['country'].str.contains(pattern, case=False, na=False)

data[mask].head()
```

Image by author

Note how ‘United States’ always appears first in the filtered rows.

- Where the order does not matter (Mexico can appear first in a row), use `str.contains('str1.*str2|str2.*str1').` The `|` means ‘return rows where `str1` appears first, or `str2` appears first’.

```
pattern = 'states.*mexico|mexico.*states'

mask = data['country'].str.contains(pattern, case=False, na=False)

data[mask].head()
```

Image by author

See how in the fourth row ‘Mexico’ appears first.

- You can also create a mask for each country, then pass the masks into the data using `&` symbol. The code below displays the same DataFrame as above.

```
mask1 = (data['country'].str.contains('states', case=False,
na=False))


mask2 = (data['country'].str.contains('mexico', case=False,
na=False))

data[mask1 & mask2].head()
```

5. Filter rows with numbers in a particular column

We might also want to check for numbers in a column using the regex pattern `‘[0–9]’`. The code looks like `str.contains(‘[0–9]’)`.

In the next example, we want to check the age rating and return those with specific ages after the dash such as `TV-14, PG-13, NC-17` and leave out `TV-Y7` and `TV-Y7-FV`. We, therefore, add a dash (-) before the number pattern.

```
pattern = '-[0-9]'

mask = data['rating'].str.contains(pattern, na=False)
```

```
data[mask].sample(3)
```


Image by author

6. Filter rows where a partial string is present in multiple columns

We can check for rows where a sub-string is present in two or more given columns.

- For example, let us check for the presence of 'tv' in three columns (`'rating','listed_in'` and `'type'`) and return rows where it's present in *all* of them. The easiest way is to create three masks each for a specific column, and filter the data using `&` symbol meaning 'and' (use | symbol to return True if it's in at least one column).

```
mask1 = data['rating'].str.contains('tv', case=False, na=False)

mask2 = data['listed_in'].str.contains('tv', case=False, na=False)

mask3 = data['type'].str.contains('tv', case=False, na=False)

data[mask1 & mask2 & mask3].head()
```


Image by author

See how 'tv' is present in all three columns in the filtered data above.

- Another way is using the slightly complicated apply() and Lambda functions. Read the article Lambda Functions with Practical Examples in Python for clarity on how these two functions work.

```
cols_to_check = ['rating','listed_in','type']

pattern = 'tv'

mask = data[cols_to_check].apply(
        lambda col:col.str.contains(
        pattern, na=False, case=False)).all(axis=1)
```

The code for the mask above says that for every column in the list `cols_to_check`, apply `str.contains('tv')` function. It then uses `.all(axis=1)` to consolidate the three masks into one mask (or column) by returning `True` for every row where *all* the columns are `True`. (use `.any()` to return `True` for presence in at-least one column).

Pandas .all() and .any() by author

The filtered DataFrame is the same as the one displayed previously.

7. Filter for rows where values in one column are present in another column.

We can check whether the value in one column is present as a partial string in another column.

Using our Netflix dataset, let us check for rows where the `'director'` also appeared in the `'cast'` as an actor.

In this example, we will use `df.apply()`, `lambda`, and the `'in'` keyword which checks if a certain value is present in a given sequence of strings.

```
mask = data.apply(
    lambda x: str(x['director']) in str(x['cast']),
    axis=1)
```

`df.apply()` above holds a lambda function which says that for every row (x), check if the value in `'director'` is present in the `'cast'` column and return `True` or `False`. I wrapped the columns with `str()` to convert each value into a String because it raised a `TypeError` probably because of `NaNs`. We use `axis=1` to mean column-wise, therefore the operation is done for every row and the result will be a column (or series).

```
data[mask].head()
```

NaNs after apply() by author

Whoops! That's a lot of `NaNs`. Let's drop them using the director's column as the `subset` and display afresh.

```
data[mask].dropna(subset=['director'])
```


Directors in cast by author

For other ways to check if values in one column match those in another, read this article.

8. Checking column names (or index values) for a given sub-string

We can check for the presence of a partial string in column headers and return those columns.

Filter column names

- In the example below, we will use `df.filter(like=pattern, axis=1)` to return column names with the given pattern. We can also use `axis=columns`. Note that this returns the filtered data and no mask is generated.

```
data.filter(like='in', axis=1)
```

Filtered column names with ‘in’ sub-string

- We can also use `df.loc` where we display all the rows but only the columns with the given sub-string.

```
data.loc[:, data.columns.str.contains('in')]
```

This code generates the same results like the image above. Read this article for how `.loc` works.

Filter by index values

Let us first set the title as the index, then filter by the word ‘Love’. We will use the same methods as above with slight adjustments.

Use `df.filter(like='Love', axis=0)` . We can also use `axis=index`.

```
df = data.set_index('title')

df.filter(like='Love', axis=0)
```


Filter index with ‘Love’

Use `df.loc[]` to display the same results as above. Here we choose the desired rows in the first part of `.loc` and return all columns in the second part.

```
df.loc[df.index.str.contains('Love'), :]
```

. . .

Other filtering methods

- **Using the pandas query() function**

This is a data filtering method especially favored by SQL ninjas. The syntax is

`df.query('expression')` and the result is a modified DataFrame. The cool thing is that aside from filtering by individual columns as we've done earlier, you can reference a local variable and call methods such as mean() inside the expression. It also offers performance advantages to other complex masks.

```
#Example
data.query('country == "South Africa"')
```


Query filter by author

I rarely use this approach myself, but many people find it simpler and more readable. I encourage you to explore more as it has compelling advantages. This and this articles are a good place to start.

- **Using other string (Series.str.) example functions**

These methods can also filter data to return a mask

- `Series.str.len() > 10`
- `Series.str.startswith('Nov')`
- `Series.str.endswith('2019')`
- `Series.str.isnumeric()`
- `Series.str.isupper()`
- `Series.str.islower()`

. . .

Conclusion

As a data professional, chances are you will often need to separate data based on its contents. In this article, we looked at 8 ways to filter a DataFrame by the string values present in the columns. We used Pandas, `Lambda` functions, and the `'in'` keyword. We also used the | and & symbols, and the tilde (~) to negate a statement.

We learned that these functions return a mask (a column) of `True` and `False` values. We then pass this mask into our DataFrame using square brackets like `df[mask]` or using the .loc function like `df.loc[mask]`. You can download the full code here from Github.

I hope you enjoyed the article. To receive more like these whenever I publish a new one, subscribe here. If you are not yet a medium member and would like to support me as a writer, follow this link and I will earn a small commission. Thank you for reading!

References

10 Ways to Filter Pandas DataFrame

Python Pandas String Operations— Working with Text Data

Select by partial string from a pandas DataFrame

Data Science Pandas Machine Learning Technology Python