Filtering

Filter DataFrames for values that fit certain conditions.

Chapter Goals:

- Understand how to filter a DataFrame based on filter conditions
- Write code to filter a dataset of MLB statistics

A. Filter conditions

In the Data Manipulation section, we used relation operations on NumPy arrays to create *filter conditions*. These filter conditions returned boolean arrays, which represented the locations of the elements that pass the filter.

In pandas, we can also create filter conditions for DataFrames. Specifically, we can use relation operations on a DataFrame's column features, which will return a boolean Series representing the DataFrame rows that pass the filter.

The code below demonstrates how to use relation operations as filter conditions.

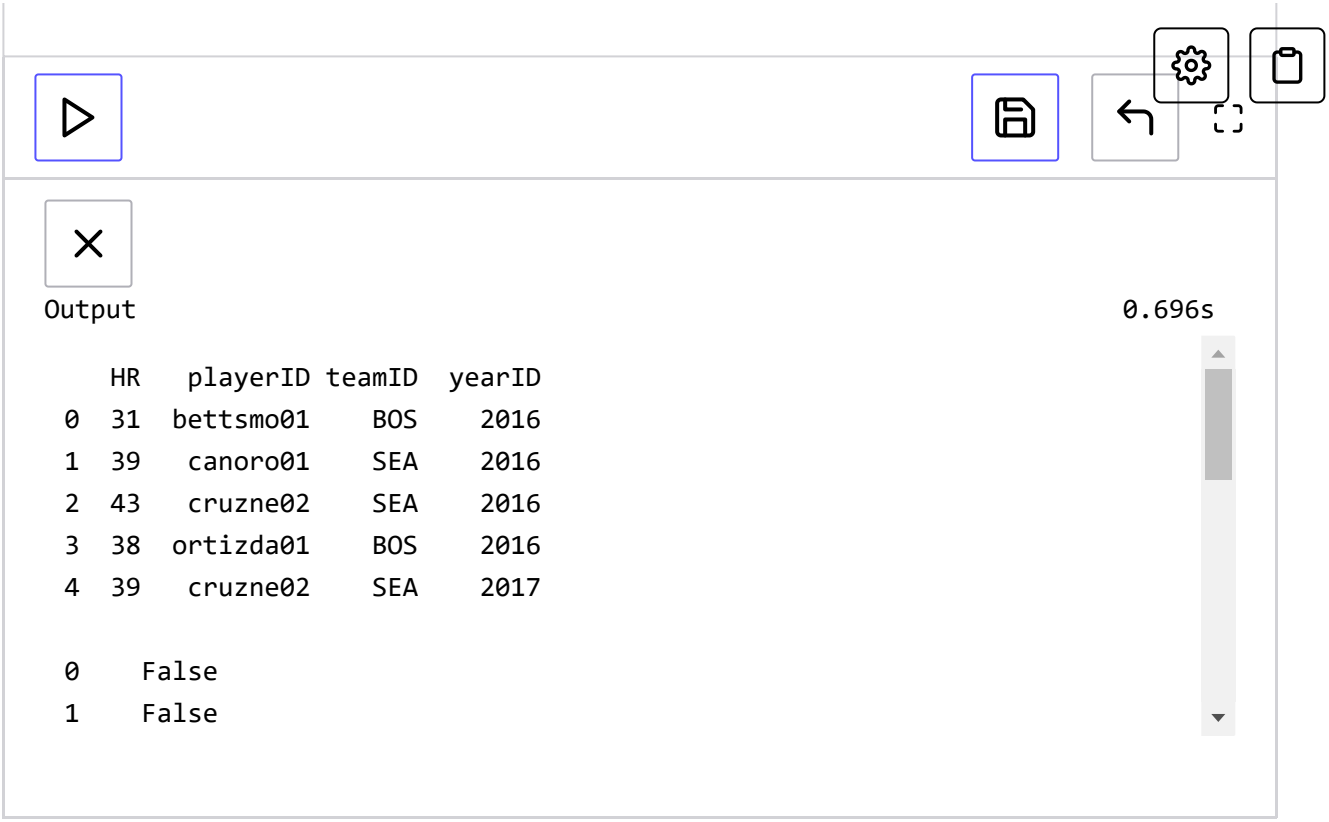
```
df = pd.DataFrame({
  'playerID': ['bettsmo01', 'canoro01', 'cruzne02', 'ortizda01', 'cruzne02'],
  'yearID': [2016, 2016, 2016, 2016, 2017],
  'teamID': ['BOS', 'SEA', 'SEA', 'BOS', 'SEA'],
  'HR': [31, 39, 43, 38, 39]})

print('{}\n'.format(df))

cruzne02 = df['playerID'] == 'cruzne02'
print('{}\n'.format(cruzne02))

hr40 = df['HR'] > 40
print('{}\n'.format(hr40))

notbos = df['teamID'] != 'BOS'
print('{}\n'.format(notbos))
```



In the code above, we created filter conditions for `df` based on the columns labeled `'playerID'`, `'HR'`, and `'teamID'`. The boolean Series outputs have `True` for the rows that pass the filter, and `False` for the rows that don't.

B. Filters from functions

Apart from relation operations, pandas provides various functions for creating specific filter conditions. For columns with string values, we can use `str.startswith` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.startswith.html), `str.endswith` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.endswith.html), and `str.contains` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.contains.html) to filter for specific strings. These functions work the exact same as their namesakes from the Python standard library.

The code below shows various examples of string filter conditions. In the final example using `str.contains`, we prepend the `~` operation, which negates the filter condition. This means our final filter condition checked for player IDs that *do not* contain `'o'`.

```
df = pd.DataFrame({
  'playerID': ['bettsmo01', 'canoro01', 'cruzne02', 'ortizda01', 'cruzne02'],
  'yearID': [2016, 2016, 2016, 2016, 2017],
  'teamID': ['BOS', 'SEA', 'SEA', 'BOS', 'SEA'],
  'HR': [31, 39, 43, 38, 39]})

print('{}\n'.format(df))

str_f1 = df['playerID'].str.startswith('c')
print('{}\n'.format(str_f1))

str_f2 = df['teamID'].str.endswith('S')
print('{}\n'.format(str_f2))

str_f3 = ~df['playerID'].str.contains('o')
print('{}\n'.format(str_f3))
```

Output 0.931s

```
   HR   playerID teamID  yearID
0  31  bettsmo01    BOS    2016
1  39   canoro01    SEA    2016
2  43   cruzne02    SEA    2016
3  38  ortizda01    BOS    2016
4  39   cruzne02    SEA    2017

0   False
1    True
```

We can also create filter conditions that check for values in a specific set, by using the `isin (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.isin.html)` function. The function only takes in one argument, which is a list of values that we want to filter for.

The code below demonstrates how to use the `isin` function for filter conditions.

```
df = pd.DataFrame({
```

```
    'playerID': ['bettsmo01', 'canoro01', 'cruzne02', 'ortizda01', 'cruzne02'],
    'yearID': [2016, 2016, 2016, 2016, 2017],
    'teamID': ['BOS', 'SEA', 'SEA', 'BOS', 'SEA'],
    'HR': [31, 39, 43, 38, 39]})

print('{}\n'.format(df))

isin_f1 = df['playerID'].isin(['cruzne02',
                               'ortizda01'])
print('{}\n'.format(isin_f1))

isin_f2 = df['yearID'].isin([2015, 2017])
print('{}\n'.format(isin_f2))
```

Output 0.706s

```
   HR   playerID teamID  yearID
0  31  bettsmo01    BOS    2016
1  39   canoro01    SEA    2016
2  43   cruzne02    SEA    2016
3  38  ortizda01    BOS    2016
4  39   cruzne02    SEA    2017

0   False
1   False
```

In pandas, when a Series or DataFrame has a missing value at a location, it is represented by `NaN`. The `NaN` value in pandas is equivalent to `np.nan` in NumPy.

Similar to Numpy, we cannot use a relation operation to create a filter condition for `NaN` values. Instead, we use the `isna` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.isna.html) and `notna` (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.notna.html) functions.

```
df = pd.DataFrame({
```

```
    'playerID': ['bettsmo01', 'canoro01', 'doejo01'],
    'yearID': [2016, 2016, 2017],
    'teamID': ['BOS', 'SEA', np.nan],
    'HR': [31, 39, 99]})

print('{}\n'.format(df))

isna = df['teamID'].isna()
print('{}\n'.format(isna))

notna = df['teamID'].notna()
print('{}\n'.format(notna))
```

Output 0.703s

```
   HR   playerID teamID  yearID
0  31  bettsmo01    BOS    2016
1  39   canoro01    SEA    2016
2  99    doejo01    NaN    2017

0    False
1    False
2     True
Name: teamID, dtype: bool
```

The `isna` function returns `True` in the locations that contain `NaN` and `False` in the locations that don't, while the `notna` function does the opposite.

C. Feature filtering

It is really easy to filter a DataFrame's rows based on filter conditions. Similar to direct indexing of a DataFrame, we use square brackets. However, the inside of the square brackets will now contain a filter condition.

When applying filter conditions within square brackets, we retrieve the rows of the DataFrame that pass the filter condition (i.e. the rows for which the filter condition is `True`).

The code below shows how to filter using square brackets and filter conditions.

```
df = pd.DataFrame({
  'playerID': ['bettsmo01', 'canoro01', 'cruzne02', 'ortizda01', 'bettsmo01'],
  'yearID': [2016, 2016, 2016, 2016, 2015],
  'teamID': ['BOS', 'SEA', 'SEA', 'BOS', 'BOS'],
  'HR': [31, 39, 43, 38, 18]})

print('{}\n'.format(df))

hr40_df = df[df['HR'] > 40]
print('{}\n'.format(hr40_df))

not_hr30_df = df[~(df['HR'] > 30)]
print('{}\n'.format(not_hr30_df))

str_df = df[df['teamID'].str.startswith('B')]
print('{}\n'.format(str_df))
```

Output 0.642s

```
   HR   playerID teamID  yearID
0  31  bettsmo01    BOS    2016
1  39   canoro01    SEA    2016
2  43   cruzne02    SEA    2016
3  38  ortizda01    BOS    2016
4  18  bettsmo01    BOS    2015

   HR  playerID teamID  yearID
2  43  cruzne02    SEA    2016
```

Time to Code!

In this chapter's code exercises, we'll apply various filters to a predefined DataFrame, `mlb_df`, which contains MLB statistics.

We'll first filter `mlb_df` for the top MLB hitting seasons in history, which we define as having a batting average above .300.

Set `top_hitters` equal to `mlb_df[]` applied with `mlb_df['BA'] > .300` as the filter condition.

```
1  # CODE HERE
```

Solution

```
1  top_hitters = mlb_df[mlb_df['BA'] > .300]
2
3
```

Next we filter for the players whose player ID *does not* start with the letter **a**.

Set `exclude_a` equal to `mlb_df[]` applied with the negation of `mlb_df['playerID'].str.startswith('a')` as the filter condition.

```
1  # CODE HERE
```

Solution

```
1  exclude_a = mlb_df[~mlb_df['playerID'].str.startswith('a')]
2
3
```

We'll now retrieve the statistics for two specific players. Their player IDs are `'bondsba01'` and `'troutmi01'`.

Set `two_ids` equal to a list containing the two specified player IDs.

Set `two_players` equal to `mlb_df[]` applied with `mlb_df['playerID'].isin(two_ids)` as the filter condition.

```
1  # CODE HERE
```

Solution

```
1  two_ids = ['bondsba01', 'troutmi01']
2  two_players = mlb_df[mlb_df['playerID'].isin(two_ids)]
3
4
```

Back

Features

Next

Sorting

Mark as Completed

22% completed, meet the criteria and claim your course certificate!

Buy Certificate

Report an Issue

Ask a Question (https://discuss.educative.io/tag/filtering__data-analysis-with-pandas__machine-learning-for-software-engineers)