Rakamin Academy

HOME CREDIT

Virtual Internship Experience

Programming Language Scripting

Filtering and Aggregation

Filtering

In our machine learning, data science projects, While dealing with datasets in Pandas dataframe, we are often required to perform the filtering operations for accessing the desired data. In this article, we'll learn about pandas functions that help in the filtering of data. We will understand pandas groupby(), where() and filter() along with syntax and examples for proper understanding.

Importing Pandas Library

Let's start this tutorial by first importing the pandas library.

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

Pandas Groupby : groupby()

The pandas groupby function is used for grouping dataframe using a mapper or by series of columns.

Syntax:
pandas.DataFrame.groupby(by, axis, level, as_index, sort, group_keys, squeeze, observed)

Parameter:

- by : mapping, function, label, or list of labels – It is used to determine the groups for groupby.
- axis : {0 or 'index', 1 or 'columns'}, default 0 – The axis along which the operation is applied.
- level : int, level name, or sequence of such, default None – It used to decide if the axis is a MultiIndex (hierarchical), group by a particular level or levels.
- as_index : bool, default True – For aggregated output, return object with group labels as the index.
- sort : bool, default True – This is used for sorting group keys.
- group_keys : bool, default True – When calling apply, this parameter adds group keys to index to identify pieces.

- squeeze : bool, default False – This parameter is used to reduce the dimensionality of the return type if possible.
- observed : bool, default False – This only applies if any of the groupers are Categoricals. If True: only show observed values for categorical groupers. If False: show all values for categorical groupers.

The function returns a groupby object that contains information about the groups

Example 1: Computing mean using groupby() function

Using the pandas groupby function

```
df = pd.DataFrame({'Cars': ['Bentley', 'Bentley',
                            'Aston Martin', 'Aston Martin'],
                   'Max Speed': [380, 370, 275, 350]})
```

Output:

	Cars	Max Speed
0	Bentley	380
1	Bentley	370
2	Aston Martin	275
3	Aston Martin	350

In this example, the mean of max_speed attribute is computed using pandas groupby function using Cars column.

Example 2: Using hierarchical indexes with pandas groupby function

In this example multindex dataframe is created, this is further used to learn about the utility of pandas groupby function.

```
arrays = [['Mclaren', 'Mercedes', 'Mclaren', 'Mercedes'],
          ['Sports', 'Luxury', 'Sports', 'Luxury']]
```

```
index = pd.MultiIndex.from_arrays(arrays, names=('Cars', 'Type'))
```

```
df = pd.DataFrame({'Max Speed': [380, 370, 275, 350]},
                  index=index)
```

Output:

		Max Speed
Cars	Type	
Mclaren	Sports	380
Mercedes	Luxury	370
Mclaren	Sports	275
Mercedes	Luxury	350

Here the groupby function is passed two different values as parameter. In both the examples, level parameter is passed to the groupby function.

Pandas Where: where()

The pandas where function is used to replace the values where the conditions are not fulfilled.

Syntax:

pandas.DataFrame.where(cond, other=nan, inplace=False, axis=None, level=None, try_cast=False)

Parameter:

- cond : bool Series/DataFrame, array-like, or callable – This is the condition used to check for executing the operations.
- other : scalar, Series/DataFrame, or callable – Entries where cond is False are replaced with corresponding value from other.
- inplace : bool, default False – It is used to decide whether to perform the operation in place on the data.
- axis : int, default None – This is used to specify the alignment axis, if needed.
- level : int, default None – This is used to specify the alignment axis, if needed.
- try_cast : bool, default False – This parameter is used to try to cast the result back to the input type.

Example 1: Simple example of pandas where() function

Here the where() function is used for filtering the data on the basis of specific conditions.

```
df = pd.read_csv('players.csv')
```

```
df.head()
```

Output:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0

Example 2: Multi-condition operations in pandas where() function

In the 2nd example of where() function, we will be combining two different conditions into one filtering operation.

```
df_mul = pd.read_csv('players.csv')
```

```
df_mul.sort_values("Team", inplace = True)
```

```
mul_filter1 = df_mul["Team"]=="Boston Celtics"
```

```
mul_filter2 = df_mul["Weight"]>215
```

```
df_mul.where(mul_filter1 & mul_filter2, inplace = True)
```

Output:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
317	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
309	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
310	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
311	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
312	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
313	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
314	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
315	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
316	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
318	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
319	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
320	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
321	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
322	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
323	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
5	Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0
10	Jared Sullinger	Boston Celtics	7.0	C	24.0	6-9	260.0	Ohio State	2569260.0
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
12	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0
13	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14	Tyler Zeller	Boston Celtics	44.0	C	26.0	7-0	253.0	North Carolina	2616975.0
...	...	...	...	...	...	...	...	...	...
454	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
453	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

452	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
451	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
450	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
449	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
448	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
447	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
446	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
445	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
444	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
443	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
456	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
455	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
381	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
380	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
379	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
378	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
377	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
376	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
375	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
374	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
373	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
372	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
371	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
369	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
368	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
382	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
370	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
457	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Pandas Filter : filter()

The pandas filter function helps in generating a subset of the dataframe rows or columns according to the specified index labels.

Syntax:
pandas.DataFrame.filter(items, like, regex, axis)

Parameter:

- items : list-like – This is used for specifying to keep the labels from axis which are in items.
- like : str – This is used for keeping labels from axis for which "like in label == True".

- regex : str (regular expression) – This is used for keeping labels from axis for which re.search(regex, label) == True.
- axis : {0 or 'index', 1 or 'columns', None}, default None – This is the axis over which the operation is applied.

Example 1: Filtering columns by name using pandas filter() function

In this example, the pandas filter operation is applied to the columns for filtering them with their names.

```
df = pd.DataFrame(np.array(([1, 2, 3], [4, 5, 6])),
                  index=['Audi', 'Bentley'],
                  columns=['one', 'two', 'three'])
```

```
df.filter(items=['one', 'three'])
```

Output:

	one	three
Audi	1	3
Bentley	4	6

Example 2: Using regular expression to filter columns

In this example, regex is used along with the pandas filter function. Here, with the help of regex, we are able to fetch the values of column(s) which have column name that has "o" at the end. The '$' is used as a wildcard suggesting that column name should end with "o".

```
df.filter(regex='o$', axis=1)
```

Output:

	two
Audi	2
Bentley	5

Example 3: Filtering rows with "like" parameter

This like parameter helps us to find desired strings in the row values and then filters them accordingly. As we specified the string in the like parameter, we got the desired results. So this is how like parameter is put to use.

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

Output:

	one	two	three
Bentley	4	5	6

Reset Index

Reset the index, or a level of it. Reset the index of the DataFrame, and use the default one instead. If the DataFrame has a MultiIndex, this method can remove one or more levels.

Syntax:
DataFrame.reset_index(level=None, drop=False, inplace=False, col_level=0, col_fill='')

Parameter:

- level: int, str, tuple, or list, default None. Only remove the given levels from the index. Removes all levels by default.
- Drop: bool, default False. Do not try to insert index into dataframe columns. This resets the index to the default integer index.
- Inplace: bool, default False. Modify the DataFrame in place (do not create a new object).
- col_level: int or str, default 0. If the columns have multiple levels, determines which level the labels are inserted into. By default it is inserted into the first level.
- col_fill: object, default ''. If the columns have multiple levels, determines how the other levels are named. If None then the index name is repeated.
- Returns: DataFrame or None. DataFrame with the new index or None if inplace=True.

Example:

```
>>> df = pd.DataFrame([('bird', 389.0),
...                    ('bird', 24.0),
...                    ('mammal', 80.5),
...                    ('mammal', np.nan)],
...                   index=['falcon', 'parrot', 'lion', 'monkey'],
...                   columns=('class', 'max_speed'))
>>> df
         class  max_speed
falcon    bird      389.0
parrot    bird       24.0
lion    mammal       80.5
monkey  mammal        NaN
```

When we reset the index, the old index is added as a column, and a new sequential index is used:

```
>>> df.reset_index()
    index   class  max_speed
0  falcon    bird      389.0
1  parrot    bird       24.0
2    lion  mammal       80.5
3  monkey  mammal        NaN
```

We can use the drop parameter to avoid the old index being added as a column:

```
>>> df.reset_index(drop=True)
    class  max_speed
0    bird      389.0
1    bird       24.0
2  mammal       80.5
3  mammal        NaN
```

Pivot

Return reshaped DataFrame organized by given index / column values. Reshape data (produce a "pivot" table) based on column values. Uses unique values from specified index / columns to form axes of the resulting DataFrame. This function does not support data aggregation, multiple values will result in a MultiIndex in the columns. See the User Guide for more on reshaping.

Syntax:

DataFrame.pivot(index=None, columns=None, values=None)

Parameter:

- Index: str or object or a list of str, optional. Column to use to make new frame's index. If None, uses existing index.
- Columns: str or object or a list of str. Column to use to make new frame's columns.

- Values: str, object or a list of the previous, optional. Column(s) to use for populating new frame's values. If not specified, all remaining columns will be used and the result will have hierarchically indexed columns.

Returns: DataFrame. Returns reshaped DataFrame.
Raises: ValueError: When there are any index, columns combinations with multiple values. DataFrame.pivot_table when you need to aggregate.

Example:

```
>>> df = pd.DataFrame({'foo': ['one', 'one', 'one', 'two', 'two',
...                            'two'],
...                    'bar': ['A', 'B', 'C', 'A', 'B', 'C'],
...                    'baz': [1, 2, 3, 4, 5, 6],
...                    'zoo': ['x', 'y', 'z', 'q', 'w', 't']})
>>> df
    foo   bar  baz  zoo
0   one   A    1    x
1   one   B    2    y
2   one   C    3    z
3   two   A    4    q
4   two   B    5    w
5   two   C    6    t
```

```
>>> df.pivot(index='foo', columns='bar', values='baz')
bar  A   B   C
foo
one  1   2   3
two  4   5   6
```

```
>>> df.pivot(index='foo', columns='bar')['baz']
bar  A   B   C
foo
one  1   2   3
two  4   5   6
```

```
>>> df.pivot(index='foo', columns='bar', values=['baz', 'zoo'])
      baz       zoo
bar   A  B  C   A  B  C
foo
one   1  2  3   x  y  z
two   4  5  6   q  w  t
```

Pivot Table

Create a spreadsheet-style pivot table as a DataFrame. The levels in the pivot table will be stored in MultiIndex objects (hierarchical indexes) on the index and columns of the result DataFrame.

Syntax:

DataFrame.pivot_table(values=None, index=None, columns=None, aggfunc='mean', fill_value=None, margins=False, dropna=True, margins_name='All', observed=False, sort=True)

Parameter:

- Values: column to aggregate, optional
- Index: column, Grouper, array, or list of the previous. If an array is passed, it must be the same length as the data. The list can contain any of the other types (except list). Keys to group by on the pivot table index. If an array is passed, it is being used as the same manner as column values.

- Columns: column, Grouper, array, or list of the previous. If an array is passed, it must be the same length as the data. The list can contain any of the other types (except list). Keys to group by on the pivot table column. If an array is passed, it is being used as the same manner as column values.
- Aggfunc: function, list of functions, dict, default numpy.mean. If list of functions passed, the resulting pivot table will have hierarchical columns whose top level are the function names (inferred from the function objects themselves) If dict is passed, the key is column to aggregate and value is function or list of functions.
- fill_value: scalar, default None. Value to replace missing values with (in the resulting pivot table, after aggregation).
- Margins: default False. Add all row / columns (e.g. for subtotal / grand totals).
- Dropna: bool, default True. Do not include columns whose entries are all NaN.
- margins_name: str, default 'All. Name of the row / column that will contain the totals when margins is True.
- Observed: bool, default False. This only applies if any of the groupers are Categoricals. If True: only show observed values for categorical groupers. If False: show all values for categorical groupers.
- Sort: bool, default True. Specifies if the result should be sorted.

Returns: DataFrame. An Excel style pivot table.

Example:

```
>>> df = pd.DataFrame({"A": ["foo", "foo", "foo", "foo", "foo",
...                          "bar", "bar", "bar", "bar"],
...                    "B": ["one", "one", "one", "two", "two",
...                          "one", "one", "two", "two"],
...                    "C": ["small", "large", "large", "small",
...                          "small", "large", "small", "small",
...                          "large"],
...                    "D": [1, 2, 2, 3, 3, 4, 5, 6, 7],
...                    "E": [2, 4, 5, 5, 6, 6, 8, 9, 9]})
>>> df
     A    B      C  D  E
0  foo  one  small  1  2
1  foo  one  large  2  4
2  foo  one  large  2  5
3  foo  two  small  3  5
4  foo  two  small  3  6
5  bar  one  large  4  6
6  bar  one  small  5  8
7  bar  two  small  6  9
8  bar  two  large  7  9
```

This first example aggregates values by taking the sum.

```
>>> table = pd.pivot_table(df, values='D', index=['A', 'B'],
...                        columns=['C'], aggfunc=np.sum, fill_value=0)
>>> table
C        large  small
A   B
bar one      4      5
    two      7      6
foo one      4      1
    two      0      6
```

We can also fill missing values using the fill_value parameter.

```
>>> table = pd.pivot_table(df, values='D', index=['A', 'B'],
...                        columns=['C'], aggfunc=np.sum, fill_value=0)
>>> table
C        large  small
A   B
bar one      4      5
    two      7      6
foo one      4      1
    two      0      6
```

The next example aggregates by taking the mean across multiple columns.

```
>>> table = pd.pivot_table(df, values=['D', 'E'], index=['A', 'C'],
...                        aggfunc={'D': np.mean,
...                                 'E': np.mean})
>>> table
                D         E
A   C
bar large  5.500000  7.500000
    small  5.500000  8.500000
foo large  2.000000  4.500000
    small  2.333333  4.333333
```

We can also calculate multiple types of aggregations for any given value column.

```
>>> table = pd.pivot_table(df, values=['D', 'E'], index=['A', 'C'],
...                        aggfunc={'D': np.mean,
...                                 'E': [min, max, np.mean]})
>>> table
                   D   E
                mean max      mean  min
A   C
bar large  5.500000   9  7.500000    6
    small  5.500000   9  8.500000    8
foo large  2.000000   5  4.500000    4
    small  2.333333   6  4.333333    2
```

Reference :

1. https://machinelearningknowledge.ai/pandas-tutorial-groupby-where-and-filter/
2. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.reset_index.html
3. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.pivot.html
4. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.pivot_table.html