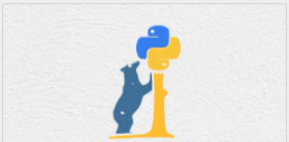

Python Pandas - GroupBy

Previous Page Next Page

Any **groupby** operation involves one of the following operations on the original object. They are –

- **Splitting** the Object
- **Applying** a function
- **Combining** the results

In many situations, we split the data into sets and we apply some functionality on each subset. In the apply functionality, we can perform the following operations –

- **Aggregation** – computing a summary statistic
- **Transformation** – perform some group-specific operation
- **Filtration** – discarding the data with some condition

Let us now create a DataFrame object and perform all the operations on it –

Live Demo

```
#import the pandas library
import pandas as pd

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

print df
```

Its **output** is as follows –

```
    Points   Rank     Team   Year
0      876      1   Riders   2014
1      789      2   Riders   2015
2      863      2   Devils   2014
3      673      3   Devils   2015
4      741      3    Kings   2014
5      812      4    kings   2015
6      756      1    Kings   2016
7      788      1    Kings   2017
8      694      2   Riders   2016
9      701      4   Royals   2014
10     804      1   Royals   2015
11     690      2   Riders   2017
```

Split Data into Groups

Pandas object can be split into any of their objects. There are multiple ways to split an object like –

- obj.groupby('key')
- obj.groupby(['key1','key2'])
- obj.groupby(key,axis=1)

Let us now see how the grouping objects can be applied to the DataFrame object

Example

Live Demo

```
# import the pandas library
import pandas as pd

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

print df.groupby('Team')
```

Its **output** is as follows –

```
<pandas.core.groupby.DataFrameGroupBy object at 0x7fa46a977e50>
```

View Groups

Live Demo

```
# import the pandas library
import pandas as pd

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

print df.groupby('Team').groups
```

Its **output** is as follows –

```
{'Kings': Int64Index([4, 6, 7],      dtype='int64'),
'Devils': Int64Index([2, 3],         dtype='int64'),
'Riders': Int64Index([0, 1, 8, 11],  dtype='int64'),
'Royals': Int64Index([9, 10],        dtype='int64'),
'kings' : Int64Index([5],            dtype='int64')}
```

Example

Group by with multiple columns –

Live Demo

```
# import the pandas library
import pandas as pd

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

print df.groupby(['Team','Year']).groups
```

Its **output** is as follows –

```
{('Kings', 2014): Int64Index([4], dtype='int64'),
 ('Royals', 2014): Int64Index([9], dtype='int64'),
 ('Riders', 2014): Int64Index([0], dtype='int64'),
 ('Riders', 2015): Int64Index([1], dtype='int64'),
 ('Kings', 2016): Int64Index([6], dtype='int64'),
 ('Riders', 2016): Int64Index([8], dtype='int64'),
 ('Riders', 2017): Int64Index([11], dtype='int64'),
 ('Devils', 2014): Int64Index([2], dtype='int64'),
 ('Devils', 2015): Int64Index([3], dtype='int64'),
 ('kings', 2015): Int64Index([5], dtype='int64'),
 ('Royals', 2015): Int64Index([10], dtype='int64'),
 ('Kings', 2017): Int64Index([7], dtype='int64')}
```

Iterating through Groups

With the **groupby** object in hand, we can iterate through the object similar to itertools.obj.

Live Demo

```
# import the pandas library
import pandas as pd

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

grouped = df.groupby('Year')

for name,group in grouped:
   print name
   print group
```

Its **output** is as follows –

```
2014
   Points  Rank     Team   Year
0     876     1   Riders   2014
2     863     2   Devils   2014
4     741     3   Kings    2014
9     701     4   Royals   2014

2015
   Points  Rank     Team   Year
1     789     2   Riders   2015
3     673     3   Devils   2015
5     812     4    kings   2015
10    804     1   Royals   2015

2016
   Points  Rank     Team   Year
6     756     1    Kings   2016
8     694     2   Riders   2016

2017
```

```
   Points  Rank    Team   Year
7     788     1   Kings   2017
11    690     2  Riders   2017
```

By default, the **groupby** object has the same label name as the group name.

Select a Group

Using the **get_group()** method, we can select a single group.

Live Demo

```
# import the pandas library
import pandas as pd

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

grouped = df.groupby('Year')
print grouped.get_group(2014)
```

Its **output** is as follows –

```
   Points  Rank     Team    Year
0     876     1   Riders    2014
2     863     2   Devils    2014
4     741     3   Kings     2014
9     701     4   Royals    2014
```

Aggregations

An aggregated function returns a single aggregated value for each group. Once the **group by** object is created, several aggregation operations can be performed on the grouped data.

An obvious one is aggregation via the aggregate or equivalent **agg** method –

Live Demo

```
# import the pandas library
import pandas as pd
import numpy as np

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

grouped = df.groupby('Year')
print grouped['Points'].agg(np.mean)
```

Its **output** is as follows –

```
Year
2014   795.25
2015   769.50
2016   725.00
2017   739.00
Name: Points, dtype: float64
```

Another way to see the size of each group is by applying the size() function –

Live Demo

```
import pandas as pd
import numpy as np

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

Attribute Access in Python Pandas
grouped = df.groupby('Team')
print grouped.agg(np.size)
```

Its **output** is as follows –

```
         Points   Rank   Year
Team
Devils        2      2      2
Kings         3      3      3
Riders        4      4      4
Royals        2      2      2
kings         1      1      1
```

Applying Multiple Aggregation Functions at Once

With grouped Series, you can also pass a **list** or **dict of functions** to do aggregation with, and generate DataFrame as output –

Live Demo

```
# import the pandas library
import pandas as pd
import numpy as np
```

```
ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

grouped = df.groupby('Team')
print grouped['Points'].agg([np.sum, np.mean, np.std])
```

Its **output** is as follows –

```
Team      sum      mean          std
Devils   1536   768.000000   134.350288
Kings    2285   761.666667    24.006943
Riders   3049   762.250000    88.567771
Royals   1505   752.500000    72.831998
kings     812   812.000000          NaN
```

Transformations

Transformation on a group or a column returns an object that is indexed the same size of that is being grouped. Thus, the transform should return a result that is the same size as that of a group chunk.

Live Demo

```
# import the pandas library
import pandas as pd
import numpy as np

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

grouped = df.groupby('Team')
score = lambda x: (x - x.mean()) / x.std()*10
print grouped.transform(score)
```

Its **output** is as follows –

```
       Points       Rank       Year
0   12.843272 -15.000000 -11.618950
1    3.020286   5.000000  -3.872983
2    7.071068  -7.071068  -7.071068
3   -7.071068   7.071068   7.071068
4   -8.608621  11.547005 -10.910895
5         NaN        NaN        NaN
6   -2.360428  -5.773503   2.182179
7   10.969049  -5.773503   8.728716
8   -7.705963   5.000000   3.872983
9   -7.071068   7.071068  -7.071068
10   7.071068  -7.071068   7.071068
11  -8.157595   5.000000  11.618950
```

Filtration

Filtration filters the data on a defined criteria and returns the subset of data. The **filter()** function is used to filter the data.

Live Demo

```
import pandas as pd
import numpy as np

ipl_data = {'Team': ['Riders', 'Riders', 'Devils', 'Devils', 'Kings',
   'kings', 'Kings', 'Kings', 'Riders', 'Royals', 'Royals', 'Riders'],
   'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
   'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
   'Points':[876,789,863,673,741,812,756,788,694,701,804,690]}
df = pd.DataFrame(ipl_data)

print df.groupby('Team').filter(lambda x: len(x) >= 3)
```

Its **output** is as follows –

```
    Points  Rank     Team   Year
0      876     1   Riders   2014
1      789     2   Riders   2015
4      741     3   Kings    2014
6      756     1   Kings    2016
7      788     1   Kings    2017
8      694     2   Riders   2016
11     690     2   Riders   2017
```

In the above filter condition, we are asking to return the teams which have participated three or more times in IPL.