Experiment No.: 7

Title: Write a Python program to perform data analysis using Groupby.

Objectives:

1. To learn how to analyze data using Groupby.

Theory:

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

create a DataFrame object and perform all the operations on it

```
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
```

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)

grouping objects can be applied to the DataFrame object

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

View Groups

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

Group by with multiple columns –

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

Iterating through Groups

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

```
grouped = df.groupby('Year')

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

Select a Group

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

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

Aggregation via the aggregate or equivalent **agg** method –

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

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

```
grouped = df.groupby('Team')
print grouped.agg(np.size)
```

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

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

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

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

Filtration

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

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