Class 4 GroupBy

April 13, 2020

1 Groupby

The groupby method allows you to group rows of data together and call aggregate functions

[1]:
```
import pandas as pd
import sys
print('Python version ' + sys.version)
print('Pandas version ' + pd.__version__)
```

```
Python version 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit
(AMD64)]
Pandas version 0.24.2
```

[3]:
```
# Our small data set
d = {'one':[1,2,3,4,5],
     'two':[2,2,2,2,2],
     'letter':['a','a','b','b','c']}

# Create dataframe
df = pd.DataFrame(d)
df
```

[3]:

	one	two	letter
0	1	2	a
1	2	2	a
2	3	2	b
3	4	2	b
4	5	2	c

[4]:
```
# Create group object
one = df.groupby('letter')
print(one)
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001F75AAB8588>
```

[5]:
```
# Apply sum function
one.sum()
```

[5]:
```
        one  two
letter
a         3    4
b         7    4
c         5    2
```

[6]:
```
#Getting Groupby based on two col
one=df.groupby(['letter','two'])
one.sum()
```

[6]:
```
            one
letter two
a      2      3
b      2      7
c      2      5
```

[7]:
```
letterone = df.groupby(['letter','one'], as_index=False).sum()
letterone
```

[7]:
```
  letter  one  two
0      a    1    2
1      a    2    2
2      b    3    2
3      b    4    2
4      c    5    2
```

[8]:
```
letterone.index
```

[8]:
```
Int64Index([0, 1, 2, 3, 4], dtype='int64')
```

1.0.1 Applying Groupby

[2]:
```
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)
```

```
      Team  Rank  Year  Points
0   Riders     1  2014     876
1   Riders     2  2015     789
2   Devils     2  2014     863
3   Devils     3  2015     673
4    Kings     3  2014     741
```

```
5    kings     4  2015     812
6    Kings     1  2016     756
7    Kings     1  2017     788
8   Riders     2  2016     694
9   Royals     4  2014     701
10  Royals     1  2015     804
11  Riders     2  2017     690
```

[3]:
```
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).groupby('Team').sum()
print(df)
```

```
        Rank  Year  Points
Team
Devils     5  4029    1536
Kings      5  6047    2285
Riders     7  8062    3049
Royals     5  4029    1505
kings      4  2015     812
```

[4]:
```
from pprint import pprint
df = pd.DataFrame(ipl_data)
#print(df)

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

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

[5]:
```
#Groupby with Multiple Col values
df = pd.DataFrame(ipl_data)
pprint (df.groupby(['Team','Year']).groups)
```

```
{('Devils', 2014): Int64Index([2], dtype='int64'),
 ('Devils', 2015): Int64Index([3], dtype='int64'),
 ('Kings', 2014): Int64Index([4], dtype='int64'),
 ('Kings', 2016): Int64Index([6], dtype='int64'),
 ('Kings', 2017): Int64Index([7], dtype='int64'),
 ('Riders', 2014): Int64Index([0], dtype='int64'),
 ('Riders', 2015): Int64Index([1], dtype='int64'),
```

```
 ('Riders', 2016): Int64Index([8], dtype='int64'),
 ('Riders', 2017): Int64Index([11], dtype='int64'),
 ('Royals', 2014): Int64Index([9], dtype='int64'),
 ('Royals', 2015): Int64Index([10], dtype='int64'),
 ('kings', 2015): Int64Index([5], dtype='int64')}
```

[4]:
```
grouped = df.groupby('Year')
for ranks,group in grouped:
    print (ranks)
    print (group)
```

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

[18]:
```
grouped = df.groupby('Year')
print (grouped['Points'].mean())
```

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

Aggregate functions deliver a single number to represent a data set. The numbers being used may themselves be products of aggregate functions. Economists use the outputs of data aggregation to plot changes over time and project future trends. The models created out of aggregated data can be used to influence policy and business decisions. **Aggregate is a different way to say add up. When you add up an aggregate, the items you add together should be similar items.** You can calculate the aggregate of your marks by adding the total marks scored by you

of each of your subjects and dividing it by the total number of subjects. This will give you your overall percentage.

[11]:
```
import numpy as np
grouped = df.groupby('Year')
print (grouped['Points'].agg(np.mean))
```

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

[12]:
```
df = pd.DataFrame(ipl_data)

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

```
         sum        mean         std
Team
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
```

[24]:
```
import pandas as pd
# Create dataframe
data = {'Company':['GOOG','GOOG','MSFT','MSFT','FB','FB'],
       'Person':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
       'Sales':[200,120,340,124,243,350]}
```

[25]:
```
df = pd.DataFrame(data)
```

[26]:
```
df
```

[26]:

	Company	Person	Sales
0	GOOG	Sam	200
1	GOOG	Charlie	120
2	MSFT	Amy	340
3	MSFT	Vanessa	124
4	FB	Carl	243
5	FB	Sarah	350

Now you can use the .groupby() method to group rows together based off of a column name. For instance let's group based off of Company. This will create a DataFrameGroupBy object:

[27]:
```
df.groupby('Company')
```

[27]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001F759C22080>
```

[28]:
```
by_comp = df.groupby("Company")
```

[29]:
```
by_comp.mean()
```

[29]:

Company	Sales
FB	296.5
GOOG	160.0
MSFT	232.0

[30]:
```
df.groupby('Company').mean()
```

[30]:

Company	Sales
FB	296.5
GOOG	160.0
MSFT	232.0

What is Standard Deviation? *it is the square root of the Variance.* **What is Variance?** Variance The Variance is defined as:

The average of the squared differences from the Mean.

To calculate the variance follow these steps: 1. Work out the Mean (the simple average of the numbers) 2. Then for each number: subtract the Mean and square the result (the squared difference). 3. Then work out the average of those squared differences. (Why Square?)

you are having different dogs now find the vairance of the dogs

The heights (at the shoulders) are: 600mm, 470mm, 170mm, 430mm and 300mm.

Find out the Mean, the Variance, and the Standard Deviation.

Your first step is to find the Mean:

Mean = 600 + 470 + 170 + 430 + 300 / 5 = 1970/5 = 394

Now Chart for the Mean Difference:

To calculate the Variance, take each difference, square it, and then average the result:

Variance: 2 = (206)^2 + (76)^2 + (224)^2 + 362 + (94)^2 / 5 = 42436 + 5776 + 50176 + 1296 + 8836 / 5 = 108520 / 5 = 21704

And the Standard Deviation is just the square root of Variance, so:

Standard Deviation

= √21704 = 147.32... = 147 (to the nearest mm)

And the good thing about the Standard Deviation is that it is useful. Now we can show which heights are within one Standard Deviation (147mm) of the Mean:

So, using the Standard Deviation we have a "standard" way of knowing what is normal, and what is extra large or extra small.

Following is for Standard Deviation Formula:

1. Work out the Mean (the simple average of the numbers) 2. Then for each number: subtract the Mean and square the result 3. Then work out the mean of those squared differences. 4. Take the square root of that and we are done!

[31]:
```
by_comp.std()
```

[31]:
```
              Sales
Company
FB        75.660426
GOOG      56.568542
MSFT     152.735065
```

[32]:
```
by_comp.min()
```

[32]:
```
          Person  Sales
Company
FB          Carl    243
GOOG     Charlie    120
MSFT         Amy    124
```

[33]:
```
by_comp.max()
```

[33]:
```
          Person  Sales
Company
FB         Sarah    350
GOOG         Sam    200
MSFT     Vanessa    340
```

[35]:
```
by_comp.count()
```

[35]:
```
         Person  Sales
Company
FB            2      2
GOOG          2      2
MSFT          2      2
```

Link to Refer about Describe

[36]:
```
by_comp.describe()
```

[36]:
```
         Sales
         count   mean         std    min     25%    50%     75%    max
Company
FB         2.0  296.5   75.660426  243.0  269.75  296.5  323.25  350.0
GOOG       2.0  160.0   56.568542  120.0  140.00  160.0  180.00  200.0
MSFT       2.0  232.0  152.735065  124.0  178.00  232.0  286.00  340.0
```

[37]:
```
by_comp.describe().transpose()
```

[37]:
```
Company              FB        GOOG        MSFT
Sales count    2.000000    2.000000    2.000000
      mean   296.500000  160.000000  232.000000
      std     75.660426   56.568542  152.735065
      min    243.000000  120.000000  124.000000
      25%    269.750000  140.000000  178.000000
      50%    296.500000  160.000000  232.000000
      75%    323.250000  180.000000  286.000000
      max    350.000000  200.000000  340.000000
```

[38]:
```
by_comp.describe().transpose()['GOOG']
```

[38]:
```
Sales  count      2.000000
       mean     160.000000
       std       56.568542
       min      120.000000
       25%      140.000000
       50%      160.000000
       75%      180.000000
       max      200.000000
Name: GOOG, dtype: float64
```

2 Good Job