Day-28

Topic: Pandas - 2

#30daysofpython

Group by operation (Aggregate, Transform & Filter)

In this tutorial, we cover the groupby operation which let's us summarize and filter a dataframe based on groupings of the rows that we specify. There are three types of methods that can be used along with a group by:

- aggregate: compute statistics (mean, stdev, max, min, etc ...) of each group.
- transformations: perform some group specific operation on the data.
- filter: filter the data based on some information form each group.

We will take an Apple stock data for most of 2017.
This data is stored in AAPL.csv.
Let's first read in the data and create a column for the month and weekday.

In [5]:

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

In [6]:

```
df_aapl = pd.read_csv("AAPL.csv", index_col=0, parse_dates=["Date"])

df_aapl.head()
```

Out[6]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2017-01-03	115.800003	116.330002	114.760002	116.150002	114.311760	28781900
1	2017-01-04	115.849998	116.510002	115.750000	116.019997	114.183815	21118100
2	2017-01-05	115.919998	116.860001	115.809998	116.610001	114.764473	22193600
3	2017-01-06	116.779999	118.160004	116.470001	117.910004	116.043915	31751900
4	2017-01-09	117.949997	119.430000	117.940002	118.989998	117.106812	33561900

In [7]:

```
df_aapl.dtypes
```

Out[7]:

```
Date         datetime64[ns]
Open                float64
High                float64
Low                 float64
Close               float64
Adj Close           float64
Volume                int64
dtype: object
```

In [4]:

```
df_aapl.shape
```

Out[4]:

```
(169, 7)
```

In [9]:

```
df_aapl["Month"] = df_aapl["Date"].dt.month

df_aapl.tail()
```

Out[9]:

	Date	Open	High	Low	Close	Adj Close	Volume	Month
164	2017-08-28	160.139999	162.000000	159.929993	161.470001	160.891617	25966000	8
165	2017-08-29	160.100006	163.119995	160.000000	162.910004	162.326462	29516900	8
166	2017-08-30	163.800003	163.889999	162.610001	163.350006	162.764893	27269600	8
167	2017-08-31	163.639999	164.520004	163.479996	164.000000	163.412552	26785100	8
168	2017-09-01	164.800003	164.940002	163.630005	164.050003	163.462372	16591100	9

Let's say I want to the know the average open price for each month. I will groupby month and for each of these groups I want to the know the average open price.

In [10]:

```
df_aapl.groupby(by = "Month").Open.mean()
```

Out[10]:

```
Month
1    119.093499
2    133.234738
3    140.362174
4    143.030001
5    151.965908
6    148.215001
7    148.096500
8    158.946958
9    164.800003
Name: Open, dtype: float64
```

As you see, I get back a series with the avg open price for each month.

What is I want to know both the average open price and the average volume in each month. To do this, I just select the two column columns instead of just the Open column.

In [11]:

```
df_aapl.groupby( by = "Month") ["Open", "Volume"].mean()
```

Out[11]:

	Open	Volume
Month		
1	119.093499	2.815610e+07
2	133.234738	3.026151e+07
3	140.362174	2.441863e+07
4	143.030001	1.964758e+07
5	151.965908	2.971615e+07
6	148.215001	3.109900e+07
7	148.096500	2.109962e+07
8	158.946958	2.874213e+07
9	164.800003	1.659110e+07

In this case, I get back a data frame with the information from the each month given in the two columns.

What if I want the average open price for each month and each day of the week? I can specify a list of column names for the by argument in the group by to accomplish this.

In [12]:

```
df_aapl["Weekday"] = df_aapl["Date"].dt.weekday_name
df_aapl.head()
```

Out[12]:

	Date	Open	High	Low	Close	Adj Close	Volume	Month	Week
0	2017-01-03	115.800003	116.330002	114.760002	116.150002	114.311760	28781900	1	Tues
1	2017-01-04	115.849998	116.510002	115.750000	116.019997	114.183815	21118100	1	Wednes
2	2017-01-05	115.919998	116.860001	115.809998	116.610001	114.764473	22193600	1	Thurs
3	2017-01-06	116.779999	118.160004	116.470001	117.910004	116.043915	31751900	1	Fr
4	2017-01-09	117.949997	119.430000	117.940002	118.989998	117.106812	33561900	1	Mon

In [13]:

```
df_aapl.groupby( by = ["Month", "Weekday"]).Open.mean()
```

Out[13]:

```
Month  Weekday  
1      Friday       119.619999
       Monday       119.626666
       Thursday     118.972500
       Tuesday      118.722000
       Wednesday    118.752499
2      Friday       132.945004
       Monday       133.116669
       Thursday     133.170000
       Tuesday      134.329998
       Wednesday    132.582500
3      Friday       140.850000
       Monday       139.502499
       Thursday     140.982001
       Tuesday      140.345001
       Wednesday    139.956000
4      Friday       143.419998
       Monday       143.072502
       Thursday     142.834999
       Tuesday      142.877502
       Wednesday    143.042503
5      Friday       152.209999
       Monday       151.035000
       Thursday     150.992500
       Tuesday      153.133996
       Wednesday    152.126001
6      Friday       148.426001
       Monday       147.727501
       Thursday     148.444003
       Tuesday      148.234997
       Wednesday    148.132503
7      Friday       147.687500
       Monday       147.658002
       Thursday     148.442501
       Tuesday      148.576665
       Wednesday    148.347500
8      Friday       157.545002
       Monday       158.505001
       Thursday     160.307999
       Tuesday      157.338004
       Wednesday    160.670001
9      Friday       164.800003
Name: Open, dtype: float64
```

I now have a group for every month + weekday combination. The above is a series with multi-level. In the PPT I show how to slice the series in this case. Another way to get around this is to simply reset the index, which will move the multilevel index to columns in your dataframe.

In [15]:

```
df_aapl.groupby( by = ["Month", "Weekday"]).Open.mean().reset_index()
```

Out[15]:

	Month	Weekday	Open
0	1	Friday	119.619999
1	1	Monday	119.626666
2	1	Thursday	118.972500
3	1	Tuesday	118.722000
4	1	Wednesday	118.752499
5	2	Friday	132.945004
6	2	Monday	133.116669
7	2	Thursday	133.170000
8	2	Tuesday	134.329998
9	2	Wednesday	132.582500
10	3	Friday	140.850000
11	3	Monday	139.502499
12	3	Thursday	140.982001
13	3	Tuesday	140.345001
14	3	Wednesday	139.956000
15	4	Friday	143.419998
16	4	Monday	143.072502
17	4	Thursday	142.834999
18	4	Tuesday	142.877502
19	4	Wednesday	143.042503
20	5	Friday	152.209999
21	5	Monday	151.035000
22	5	Thursday	150.992500
23	5	Tuesday	153.133996
24	5	Wednesday	152.126001
25	6	Friday	148.426001
26	6	Monday	147.727501
27	6	Thursday	148.444003
28	6	Tuesday	148.234997
29	6	Wednesday	148.132503
30	7	Friday	147.687500
31	7	Monday	147.658002
32	7	Thursday	148.442501
33	7	Tuesday	148.576665
34	7	Wednesday	148.347500

	Month	Weekday	Open
35	8	Friday	157.545002
36	8	Monday	158.505001
37	8	Thursday	160.307999
38	8	Tuesday	157.338004
39	8	Wednesday	160.670001
40	9	Friday	164.800003

Aggregate

I can use the aggregate method to summarize the groups using multiple functions. Let's say I want to know the avg and standard deviation of the open price for each month. After the groupby, I use the agg() method and inside I can specify a list of functions that will be used to summarize each group.

In [16]:

```
df_aapl.groupby(by="Month").Open.agg( [np.mean, np.std])
```

Out[16]:

	mean	std
Month		
1	119.093499	1.891817
2	133.234738	3.410836
3	140.362174	1.728701
4	143.030001	1.120391
5	151.965908	3.476493
6	148.215001	4.450870
7	148.096500	3.479140
8	158.946958	2.926976
9	164.800003	NaN

Let's now use agg() where we select both Open and Volume:

In [19]:

```
output = df_aapl.groupby(by="Month") ["Open", "Volume"].agg([np.mean,np.std])
output
```

Out[19]:

	Open		Volume	
	mean	**std**	**mean**	**std**
Month				
1	119.093499	1.891817	2.815610e+07	6.572536e+06
2	133.234738	3.410836	3.026151e+07	2.055055e+07
3	140.362174	1.728701	2.441863e+07	7.416385e+06
4	143.030001	1.120391	1.964758e+07	4.026376e+06
5	151.965908	3.476493	2.971615e+07	1.014266e+07
6	148.215001	4.450870	3.109900e+07	1.416738e+07
7	148.096500	3.479140	2.109962e+07	4.182316e+06
8	158.946958	2.926976	2.874213e+07	1.037540e+07
9	164.800003	NaN	1.659110e+07	NaN

This gives back a dataframe with multi-level columns, which can be sliced as follows:

In [20]:

```
output.loc[:, "Volume"]
```

Out[20]:

	mean	std
Month		
1	2.815610e+07	6.572536e+06
2	3.026151e+07	2.055055e+07
3	2.441863e+07	7.416385e+06
4	1.964758e+07	4.026376e+06
5	2.971615e+07	1.014266e+07
6	3.109900e+07	1.416738e+07
7	2.109962e+07	4.182316e+06
8	2.874213e+07	1.037540e+07
9	1.659110e+07	NaN

Let's say that instead of applying both functions to both columns I wanted the mean to be applied to the Open column and a maximum to be applied to the volume column. To do this, I specify a dictionary in the agg()

column and a maximum to be applied to the volume column. To do this, I specify a dictionary in the agg() method where the key is the column and the value is a function to be applied to the groups of that column. In this case, since I specify the columns I want to focus on in the dictionary I do not have to select them after the groupby

In [21]:

```
df_aapl.groupby( by ="Month").agg( {"Open": np.mean, "Volume": np.max})
```

Out[21]:

```
             Open     Volume
Month
    1  119.093499   49201000
    2  133.234738  111985000
    3  140.362174   43885000
    4  143.030001   30379400
    5  151.965908   50767700
    6  148.215001   72307300
    7  148.096500   32476300
    8  158.946958   69936800
    9  164.800003   16591100
```

I can even use the agg() with a customer function. Let's say I wanted the count of the number of days in each month the open price was above the average open price for the month. First I'll write my customer function to be applied to each group.

In [24]:

```
def compare_open_price(col):

    count=0
    avg_price = col.mean()

    for i in list(col.index):
        open_price = col[i]

        if open_price >= avg_price:
            count=count+1

    return count
```

In [25]:

```
df_aapl.groupby(by="Month").agg({"Open": compare_open_price})
```

Out[25]:

```
       Open
Month
    1  11.0
    2  10.0
    3  10.0
    4  11.0
    5  15.0
    6   7.0
    7  11.0
    8  14.0
    9   1.0
```

In the example above, the function Compare_Open is called once for each group (each month), The input group is a series that represents the open price for the given group.

Transform

Instead of aggregating each group we can apply a transformation with transform() method after the groupby. The agg() method uses takes a column of data and spits out a single number summarizing this column based on the specified groups. The transform() method takes a column and returns a back a series that is the same length. For example let's say I use the transform method with the sum function on the column Volume grouping by Month.

In [29]:

```
volume_by_month =  df_aapl.groupby(by="Month").Volume.transform(np.sum)
volume_by_month
```

Out[29]:

```
0      563122000
1      563122000
2      563122000
3      563122000
4      563122000
         ...
164    661069000
165    661069000
166    661069000
167    661069000
168     16591100
Name: Volume, Length: 169, dtype: int64
```

In [32]:

```
volume_by_month =  df_aapl.groupby(by="Month").Volume.agg(np.sum)
volume_by_month
```

Out[32]:

```
Month
1    563122000
2    574968600
3    561628400
4    373304100
5    653755300
6    684178100
7    421992400
8    661069000
9     16591100
Name: Volume, dtype: int64
```

In [30]:

```
volume_by_month.shape
```

Out[30]:

```
(169,)
```

In [31]:

```
df_aapl.shape
```

Out[31]:

```
(169, 9)
```

Notice that this series has 169 rows, which is the same number of days that we have stock information for. What happened is that for each group we compute the sum, but instead of giving a single number for each group, it takes this summarizing number of matches it with the group that each row corresponds to. So let's say

I wanted a column for the fraction of the month's volumne that each day represents. I can first use a transform as I have done above to get a column for the total volume in each month and then I can do the simple division to get the desired column.

In [33]:

```
df_aapl["Total_Month_Volume"] = df_aapl.groupby(by="Month").Volume.transform(np.sum)
df_aapl.head(20)
```

Out[33]:

	Date	Open	High	Low	Close	Adj Close	Volume	Month	We
0	2017-01-03	115.800003	116.330002	114.760002	116.150002	114.311760	28781900	1	Tu
1	2017-01-04	115.849998	116.510002	115.750000	116.019997	114.183815	21118100	1	Wedr
2	2017-01-05	115.919998	116.860001	115.809998	116.610001	114.764473	22193600	1	Th
3	2017-01-06	116.779999	118.160004	116.470001	117.910004	116.043915	31751900	1	
4	2017-01-09	117.949997	119.430000	117.940002	118.989998	117.106812	33561900	1	M
5	2017-01-10	118.769997	119.379997	118.300003	119.110001	117.224907	24462100	1	Tu
6	2017-01-11	118.739998	119.930000	118.599998	119.750000	117.854782	27588600	1	Wedr
7	2017-01-12	118.900002	119.300003	118.209999	119.250000	117.362694	27086200	1	Th
8	2017-01-13	119.110001	119.620003	118.809998	119.040001	117.156021	26111900	1	
9	2017-01-17	118.339996	120.239998	118.220001	120.000000	118.100822	34439800	1	Tu
10	2017-01-18	120.000000	120.500000	119.709999	119.989998	118.090981	23713000	1	Wedr
11	2017-01-19	119.400002	120.089996	119.370003	119.779999	117.884300	25597300	1	Th
12	2017-01-20	120.449997	120.449997	119.730003	120.000000	118.100822	32597900	1	
13	2017-01-23	120.000000	120.809998	119.769997	120.080002	118.179558	22050200	1	M
14	2017-01-24	119.550003	120.099998	119.500000	119.970001	118.071304	23211000	1	Tu
15	2017-01-25	120.419998	122.099998	120.279999	121.879997	119.951073	32377600	1	Wedr
16	2017-01-26	121.669998	122.440002	121.599998	121.940002	120.010132	26337600	1	Th
17	2017-01-27	122.139999	122.349998	121.599998	121.949997	120.019958	20562900	1	
18	2017-01-30	120.930000	121.629997	120.660004	121.629997	119.705017	30377500	1	M
19	2017-01-31	121.150002	121.389999	120.620003	121.349998	119.429459	49201000	1	Tu

In [34]:

```
df_aapl["Frac_Volume"]= df_aapl["Volume"]/df_aapl["Total_Month_Volume"]
df_aapl.head()
```

Out[34]:

	Date	Open	High	Low	Close	Adj Close	Volume	Month	Weel
0	2017-01-03	115.800003	116.330002	114.760002	116.150002	114.311760	28781900	1	Tue
1	2017-01-04	115.849998	116.510002	115.750000	116.019997	114.183815	21118100	1	Wedne
2	2017-01-05	115.919998	116.860001	115.809998	116.610001	114.764473	22193600	1	Thur
3	2017-01-06	116.779999	118.160004	116.470001	117.910004	116.043915	31751900	1	Fr
4	2017-01-09	117.949997	119.430000	117.940002	118.989998	117.106812	33561900	1	Mor

I can also use transform to create standardize columns for each group. So let's say I want to create a standardized open price column where for each open price I substract off the mean open price for the month and divide by the standard deviation. We can use a lambda function to accomplish this.

In [35]:

```
df_aapl["Standardize_Open"]= df_aapl.groupby(by="Month").Open.transform(lambda x: (x-x.mean
df_aapl
```

Out[35]:

	Date	Open	High	Low	Close	Adj Close	Volume	Month	W
0	2017-01-03	115.800003	116.330002	114.760002	116.150002	114.311760	28781900	1	T
1	2017-01-04	115.849998	116.510002	115.750000	116.019997	114.183815	21118100	1	Wec
2	2017-01-05	115.919998	116.860001	115.809998	116.610001	114.764473	22193600	1	T
3	2017-01-06	116.779999	118.160004	116.470001	117.910004	116.043915	31751900	1	
4	2017-01-09	117.949997	119.430000	117.940002	118.989998	117.106812	33561900	1	
...	...	...	...	...	...	...	...	...	
164	2017-08-28	160.139999	162.000000	159.929993	161.470001	160.891617	25966000	8	
165	2017-08-29	160.100006	163.119995	160.000000	162.910004	162.326462	29516900	8	T
166	2017-08-30	163.800003	163.889999	162.610001	163.350006	162.764893	27269600	8	Wec
167	2017-08-31	163.639999	164.520004	163.479996	164.000000	163.412552	26785100	8	T
168	2017-09-01	164.800003	164.940002	163.630005	164.050003	163.462372	16591100	9	

169 rows × 12 columns

The lambda function above is only called only once for each group where the input x will be a series representing the column Open for each of the Months that we grouped by. It should be noted that if you subtract a single number from a series, as is the case when this lambda function is called, then pandas knows to subtract this number from all the numbers in the series. The same game for dividing a series by a single number.

Filtering

The filter() method after a group by let's us only select rows corresponding to each group where a certain criterion regarding the group as a whole is true is satisfied. The filter() method must take in a function that returns a boolean. The function will be run once for each row. For example, lets say I only want to look at rows for days in months where the average opening price for the month was above 140.

In [36]:

```
df_aapl.groupby(by="Month").agg({"Open": np.mean})
```

Out[36]:

	Open
Month	
1	119.093499
2	133.234738
3	140.362174
4	143.030001
5	151.965908
6	148.215001
7	148.096500
8	158.946958
9	164.800003

In [37]:

```
df_aapl.groupby(by="Month").filter(lambda x: x["Open"].mean() >= 150)
```

Out[37]:

	Date	Open	High	Low	Close	Adj Close	Volume	Month	Weekday	Total
81	2017-05-01	145.100006	147.199997	144.960007	146.580002	144.885605	33602900	5	Monday	
82	2017-05-02	147.539993	148.089996	146.839996	147.509995	145.804855	45352200	5	Tuesday	
83	2017-05-03	145.589996	147.490005	144.270004	147.059998	145.360062	45697000	5	Wednesday	
84	2017-05-04	146.520004	147.139999	145.809998	146.529999	144.836182	23371900	5	Thursday	
85	2017-05-05	146.759995	148.979996	146.759995	148.960007	147.238113	27327700	5	Friday	
86	2017-05-08	149.029999	153.699997	149.029999	153.009995	151.241272	48752400	5	Monday	
	2017-									

The lambda function above is called once for each group where the input group will be dataframe with the rows corresponding to the given month that is being run through the lambda function.