S&P 500 Stocks Analysis Summary

Zichen Pan

1. Dataset

Web scraping dataset of S&P500, actually totally 505 stocks

In [5]:
```
# get all S&P 500 stocks from yahoo

# successful stocks
i = 1  # already add stock 'A'

for ticker in tickers[1:]:
    try:
        current_ticker = pdr.get_data_yahoo(symbols=ticker, start = start_date, end = end_date)
    except:
        print("Error : skipping",ticker)
        continue
    current_ticker['Ticker'] = len(current_ticker) * [ticker]
    current_ticker.reset_index('Date',inplace = True)
    sp500 = sp500.append(current_ticker)
    print(ticker,"finished")
    i += 1
```

```
AAL finished
AAP finished
AAPL finished
ABBV finished
ABC finished
ABMD finished
ABT finished
ACN finished
ADBE finished
ADI finished
ADM finished
ADP finished
ADS finished
ADSK finished
AEE finished
AEP finished
AES finished
AFL finished
AGN finished
```

In [6]:
```
i
```

Out[6]: 505

Change column order, drop unnecessary columns and rename the columns, getting the final dataset as requested:

Out[10]:

	Timestamp	Ticker	Daily Closing Price	Daily Volume
0	2018-01-02	A	67.599998	1047800.0
1	2018-01-03	A	69.320000	1698900.0
2	2018-01-04	A	68.800003	2230700.0
3	2018-01-05	A	69.900002	1632500.0
4	2018-01-08	A	70.050003	1613400.0

And the dataset is quite clean so we do not need more actions:

Data Cleaning

```
In [12]: sp500_df.isnull().sum()

Out[12]: Timestamp                0
         Ticker                   0
         Daily Closing Price      0
         Daily Volume             0
         dtype: int64
```

2. SQL Version Table

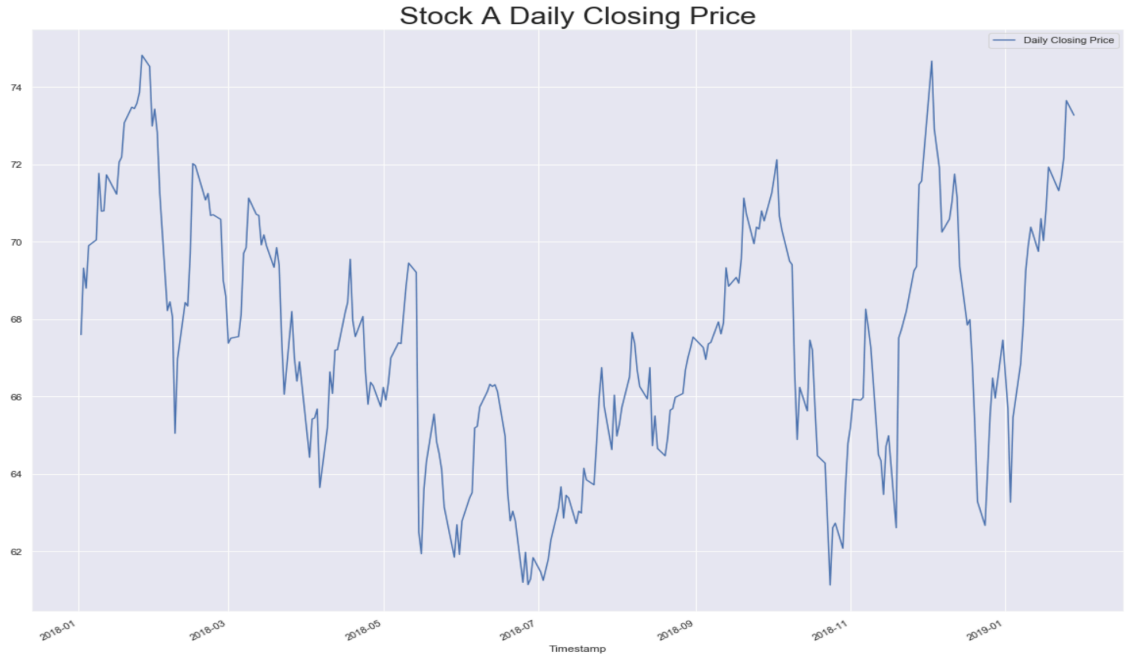
Basically using sqlite3 package, and the result with pretty format:

```
In [22]: # print data
         import pprint
         sql_select = """SELECT * from sp500_df;"""
         crsr.execute(sql_select)
         pprint.pprint(crsr.fetchall())
```

```
[(0, 'A', '2018-01-02 00:00:00', 67.5999984741211, 1047800),
 (1, 'A', '2018-01-03 00:00:00', 69.31999969482422, 1698900),
 (2, 'A', '2018-01-04 00:00:00', 68.80000305175781, 2230700),
 (3, 'A', '2018-01-05 00:00:00', 69.9000015258789, 1632500),
 (4, 'A', '2018-01-08 00:00:00', 70.05000305175781, 1613400),
 (5, 'A', '2018-01-09 00:00:00', 71.7699966430664, 2666100),
 (6, 'A', '2018-01-10 00:00:00', 70.79000091552734, 2957200),
 (7, 'A', '2018-01-11 00:00:00', 70.80000305175781, 1511100),
 (8, 'A', '2018-01-12 00:00:00', 71.7300033569336, 1448100),
 (9, 'A', '2018-01-16 00:00:00', 71.2300033569336, 1702700),
 (10, 'A', '2018-01-17 00:00:00', 72.05999755859375, 1781800),
 (11, 'A', '2018-01-18 00:00:00', 72.19000244140625, 1828300),
 (12, 'A', '2018-01-19 00:00:00', 73.06999969482422, 2266100),
 (13, 'A', '2018-01-22 00:00:00', 73.4800033569336, 1963700),
 (14, 'A', '2018-01-23 00:00:00', 73.44000244140625, 1213400),
 (15, 'A', '2018-01-24 00:00:00', 73.58000183105469, 1754300),
 (16, 'A', '2018-01-25 00:00:00', 73.86000061035156, 1664100),
 (17, 'A', '2018-01-26 00:00:00', 74.81999969482422, 2386900),
 (18, 'A', '2018-01-29 00:00:00', 74.52999877929688, 1321900),
```

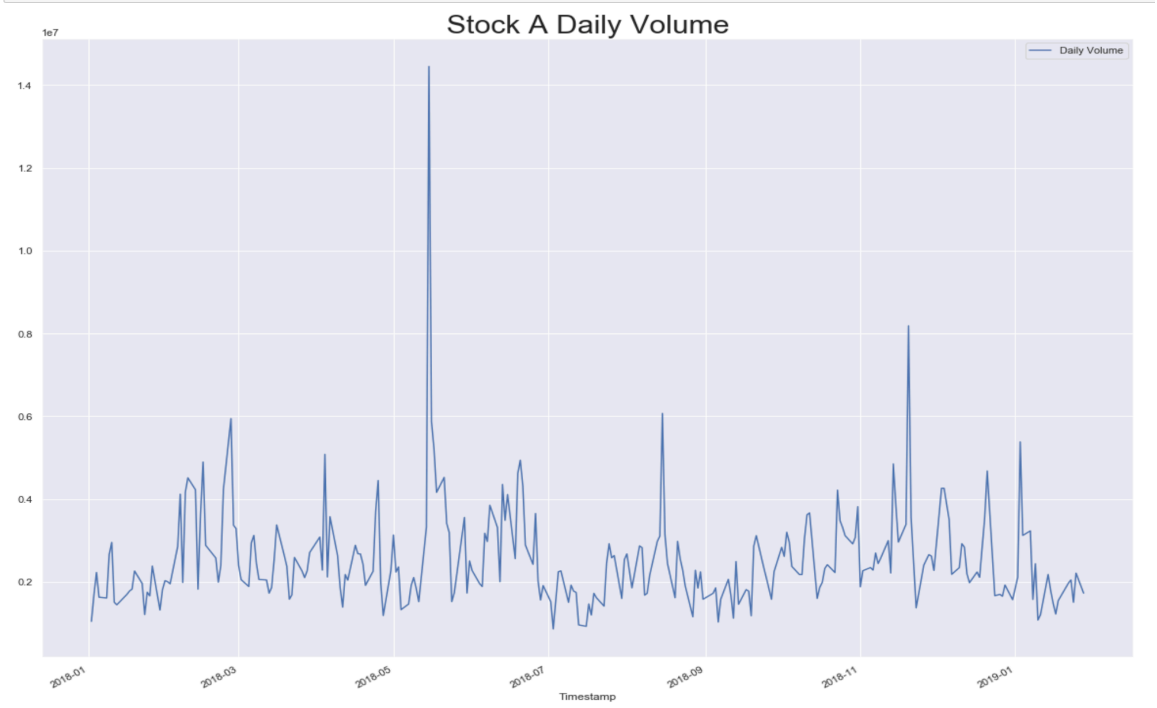
3. EDA and Visualization

I take stock A as an example.
First, daily closing price of stock A:



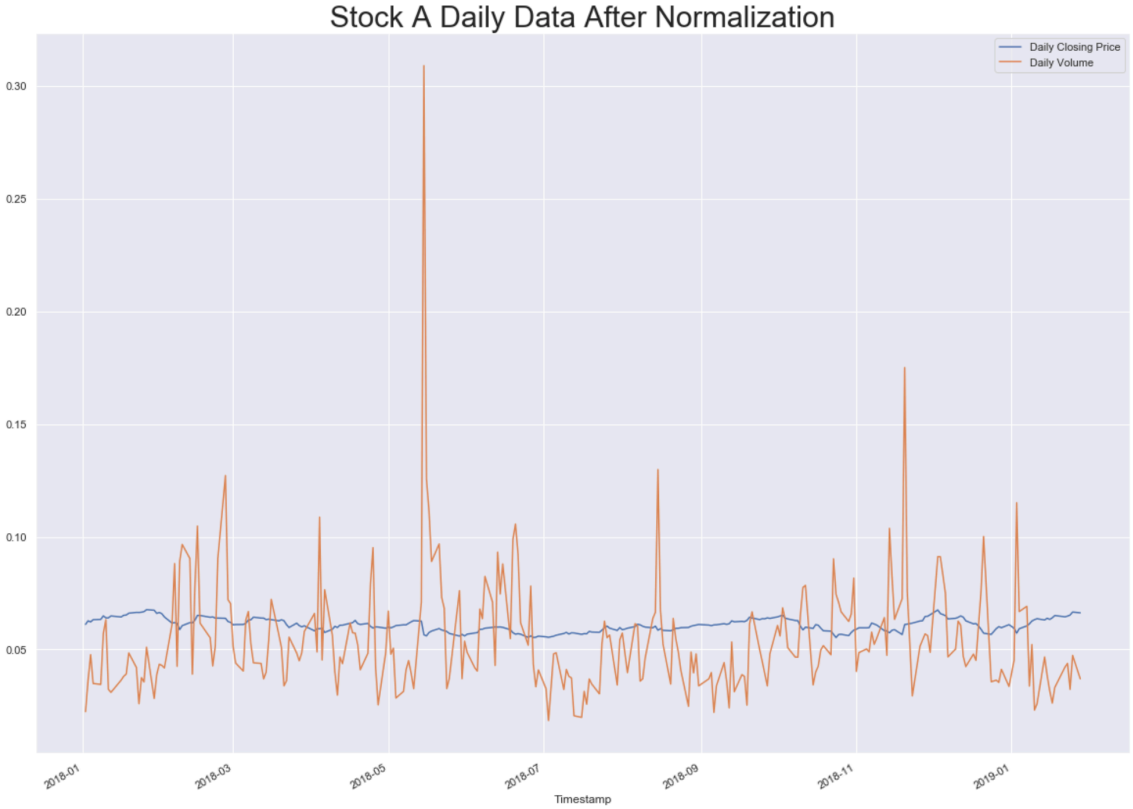
We can figure out the obvious seasonal pattern by month. And the trend first went downward from the beginning of 2018 and began to climb from the summer of 2018.

Next, daily volume of stock A:



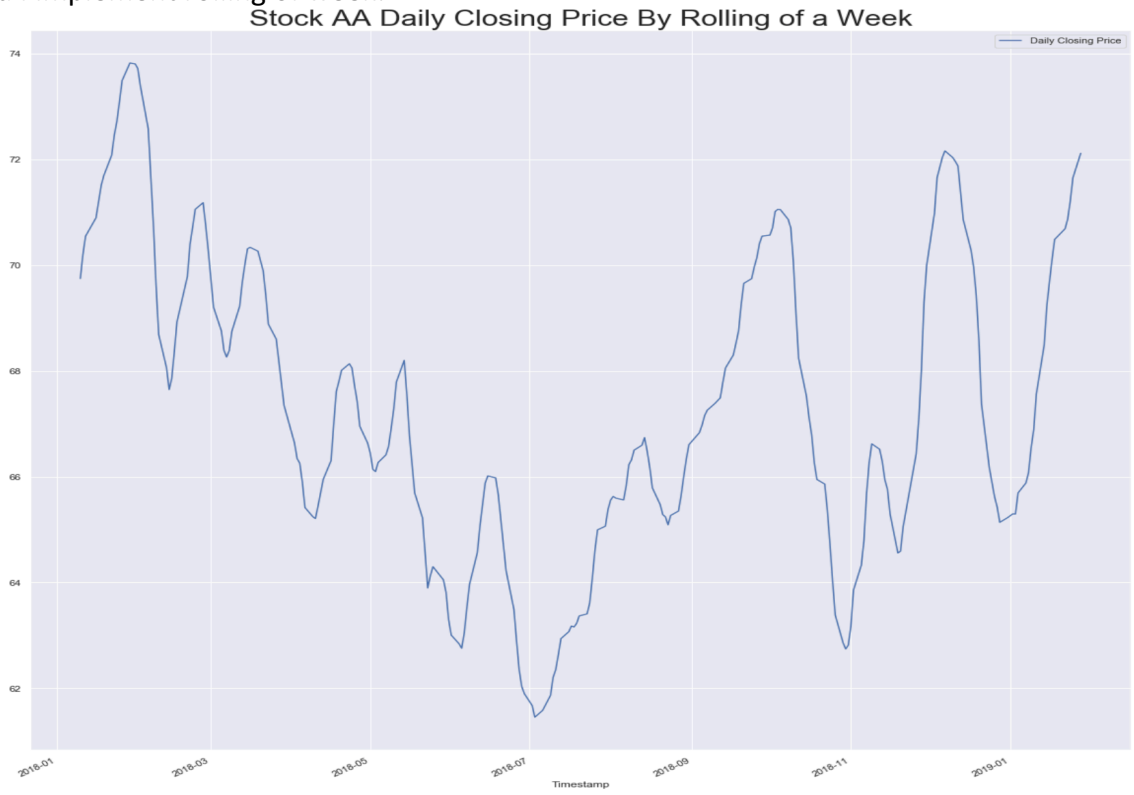
We can figure out the obvious seasonal pattern by week. Also, the peak of the second week in May 2018 is worth to explore.

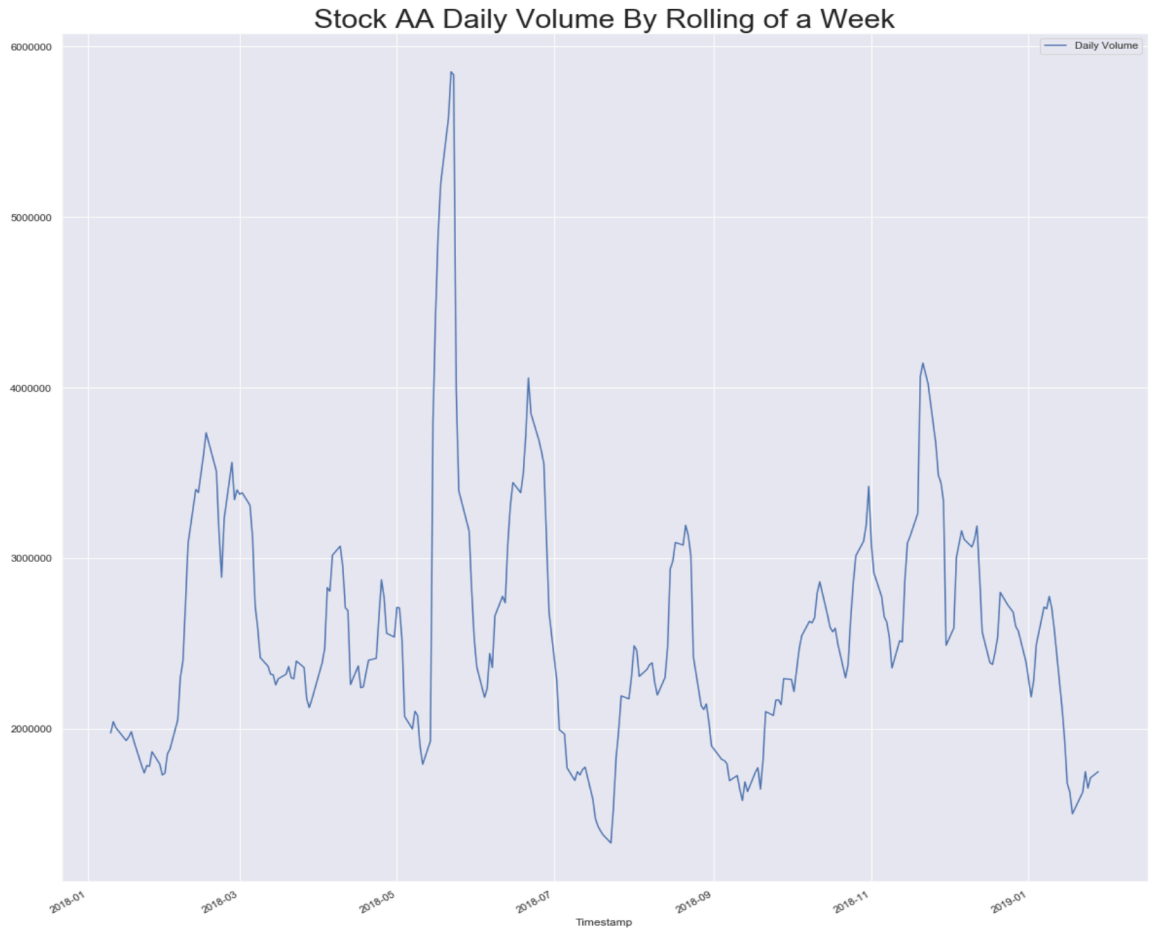
Also, I implemented normalization on both features and put them into one graph:



With some finance knowledge, we may figure out the relationship between volume and closing price. However, I am lack of finance background so I cannot discover more.

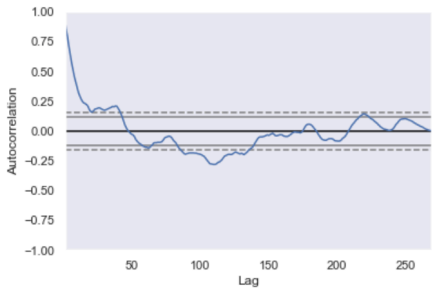
And I implement rolling of week:



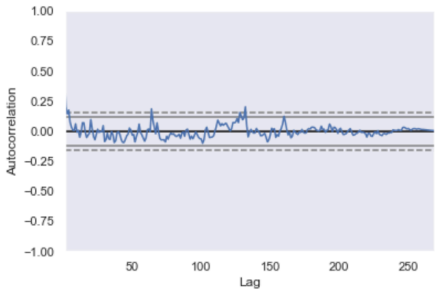


Also, I plot the auto correlation of closing price and volume:

In [36]:
```
df_close = sp500_plot.loc[sp500_plot['Ticker'] == 'A', ['Daily Closing Price']]
pd.plotting.autocorrelation_plot(df_close);
```

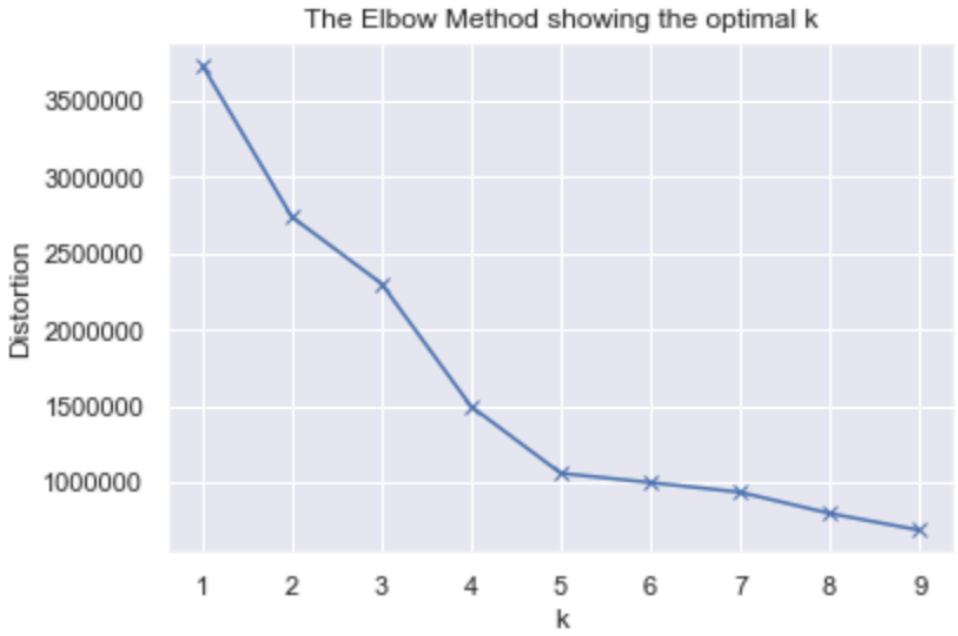


In [37]:
```
df_volume = sp500_plot.loc[sp500_plot['Ticker'] == 'A', ['Daily Volume']]
pd.plotting.autocorrelation_plot(df_volume);
```



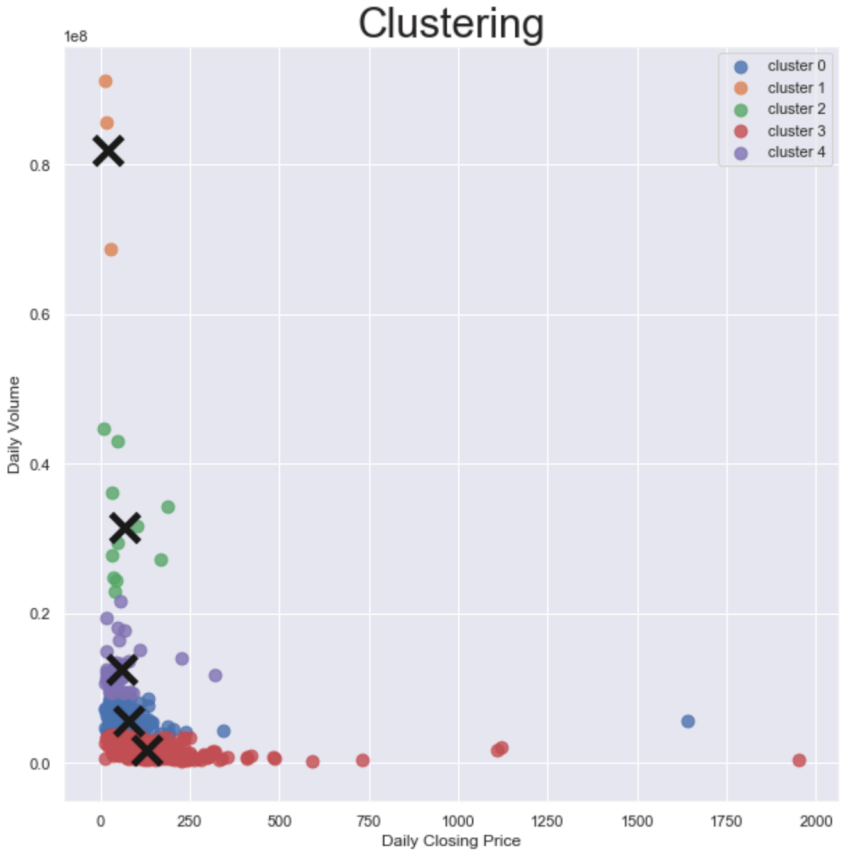
4. Clustering

The common next step should be descriptive statistics. However, the dataset is not labeled or classified, so I would like to figure out the statistics according to the clusters (as labels). Basically I use K-means to do clustering, and apply elbow plot to determine the best K:



Obviously the best K is 5.

And the visualization of clustering:



Add cluster to the original dataframe as a label:

Out[48]:

	Ticker	Daily Closing Price	Daily Volume	Cluster
0	A	67.335837	2.559157e+06	3
1	AAL	42.134944	7.094469e+06	0
2	AAP	140.702528	1.161929e+06	3
3	AAPL	186.630223	3.428477e+07	2
4	ABBV	96.762677	7.148216e+06	0
5	ABC	88.293866	1.540700e+06	3
6	ABMD	339.238439	6.366691e+05	3
7	ABT	64.578178	6.350281e+06	0
8	ACN	158.757658	2.260205e+06	3
9	ADBE	235.177249	3.287117e+06	3
10	ADI	91.788699	2.904904e+06	3
11	ADM	45.510966	3.681111e+06	3
12	ADP	131.646580	2.238858e+06	3
13	ADS	219.013569	5.943829e+05	3
14	ADSK	132.281599	2.136696e+06	3
15	AEE	61.010409	1.573179e+06	3
16	AEP	70.547212	2.948982e+06	3
17	AES	13.085836	7.275496e+06	0
18	AFL	44.948922	3.662462e+06	3
19	AGN	167.723272	2.579266e+06	3
20	AIG	51.876803	6.374819e+06	0

5. Descriptive Statistics

Description of the dataset:

In [50]:
```
sp500_plot.describe()
```

Out[50]:

	Daily Closing Price	Daily Volume
count	135621.000000	1.356210e+05
mean	112.399284	4.493829e+06
std	150.547094	9.234167e+06
min	6.110000	0.000000e+00
25%	47.759998	1.147800e+06
50%	77.629997	2.177200e+06
75%	129.179993	4.479400e+06
max	2206.090088	3.449767e+08

Average daily closing price group by cluster:

In [51]:
```
df_group_daily_close = sp500_mean.groupby('Cluster')['Daily Closing Price']
df_group_daily_close.describe().sort_values('mean')
```

Out[51]:

	count	mean	std	min	25%	50%	75%	max
Cluster								
1	3.0	19.849988	8.716048	12.619740	15.010836	17.401933	23.465111	29.528290
4	34.0	58.642456	60.557344	14.920037	24.050446	45.568085	64.035009	319.609331
2	11.0	68.069324	58.773703	10.278476	34.447807	44.207435	75.106599	186.630223
0	108.0	80.155136	160.268453	13.085836	30.464052	51.569591	78.339870	1641.378849
3	349.0	129.788103	151.673193	14.586877	63.100892	94.937695	150.242714	1952.067553

Average daily volume group by cluster:

In [52]:
```
df_group_daily_volume = sp500_mean.groupby('Cluster')['Daily Volume']
df_group_daily_volume.describe().sort_values('mean')
```

Out[52]:

	count	mean	std	min	25%	50%	75%	max
Cluster								
3	349.0	1.810404e+06	9.029222e+05	1.800059e+05	1.075343e+06	1.670255e+06	2.456785e+06	3.745881e+06
0	108.0	5.727400e+06	1.378366e+06	3.781800e+06	4.554084e+06	5.416085e+06	6.429943e+06	9.062070e+06
4	34.0	1.250684e+07	3.133684e+06	9.311010e+06	9.933736e+06	1.151196e+07	1.354051e+07	2.165800e+07
2	11.0	3.148801e+07	7.361863e+06	2.287167e+07	2.600058e+07	2.939399e+07	3.524688e+07	4.465489e+07
1	3.0	8.185593e+07	1.177164e+07	6.867068e+07	7.712930e+07	8.558791e+07	8.844856e+07	9.130921e+07

Trim mean for daily closing price sorted:

Out[53]:

	Cluster	Daily Closing Price
0	1	19.849988
1	4	46.116119
2	0	57.636126
3	2	61.317096
4	3	105.664704

Trim mean for daily volume sorted:

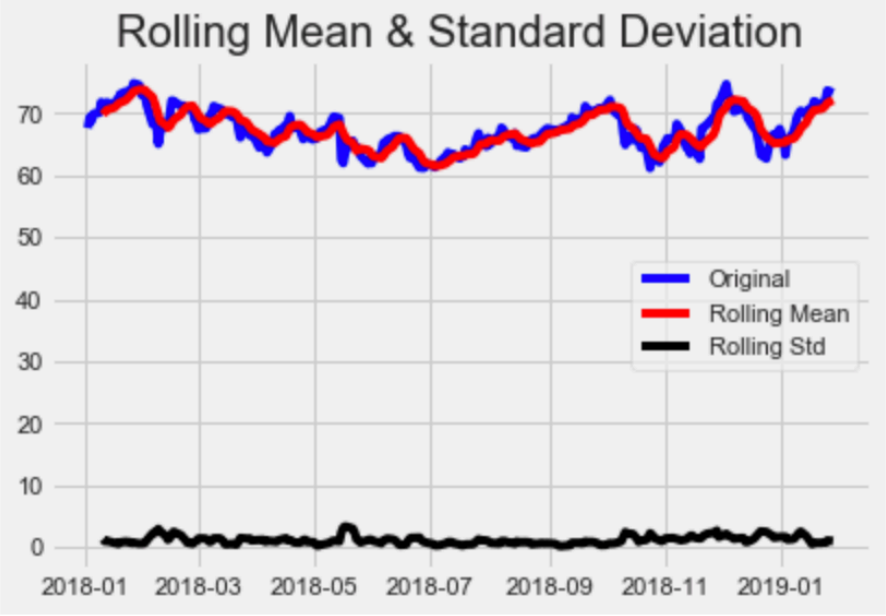
Out[54]:

	Cluster	Daily Volume
0	3	1.759387e+06
1	0	5.606417e+06
2	4	1.207059e+07
3	2	3.098239e+07
4	1	8.185593e+07

6. Prediction

First, I implement a weekly rolling to the dataset.

Check stationarity:



We can tell that the dataset is approximately stationary.

6.1 Prediction by ARIMA

Visualization:

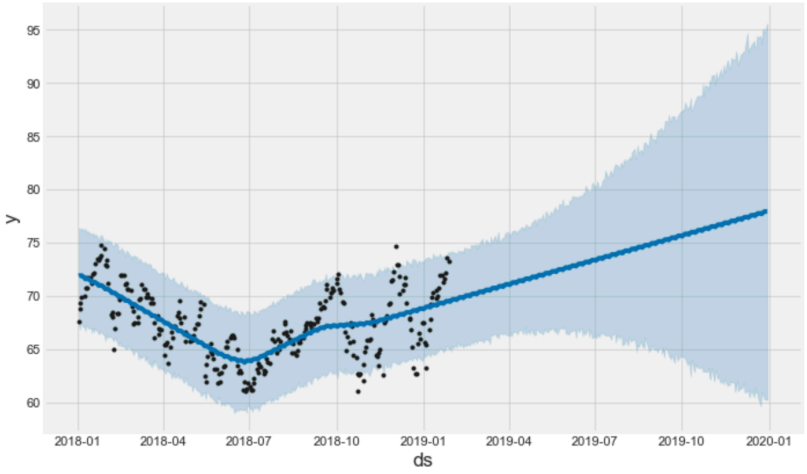
Mean Square Error of forecast:

In [67]:
```
# RMSE
A_forecasted = pred.predicted_mean
A_truth = df_A_weekly['2019-01-01':].loc[:,'Daily Closing Price']
mse = ((A_forecasted - A_truth) ** 2).mean()
print('The Mean Squared Error of our forecasts is {}'.format(round(np.sqrt(mse), 2)))
```

```
The Mean Squared Error of our forecasts is 1.09
```

6.2 Prediction by Prophet

Visualization:



Mean Square Error of forecast:

In [80]:
```
# RMSE
mse = ((A_forecasted - A_truth) ** 2).mean()
print('The Mean Squared Error of our forecasts is {}'.format(round(np.sqrt(mse), 2)))
```

```
The Mean Squared Error of our forecasts is 2.55
```

6.3 Summary of Prediction

	Mean Square of Error
ARIMA	1.09
Prophet	2.55

7. Classification

Training Set: Testing Set = 7 : 3
Also, I use Grid Search to select the best parameter for Random Forest

	Cross Validation Accuracy	Mean Square of Error
Dummy	0.71	2.05
Logistic Regression	0.72	2.45
Decision Tree	0.99	0.01
Gradient Boosting	0.99	0.01
Random Forest	0.98	0.07