TASK REPORT

DESCRIPTION:

Dataset was taken from the website

1.https://finance.yahoo.com/quote/AAPL/history?p=AAPL.

2. https://in.finance.yahoo.com/quote/RELIANCE.NS/history?

The data was taken from the year 1-1-2018 to 1-1-2019 for Apple and Reliance. The dataset consist of Date, open, close, volume, High, Low. With these parameters we can be able to find the forecasting for the next few days.

TASK:

Part-1:

1.Create 4,16,....,52 week moving average(closing price) for each stock and index. This should happen through a function.

APPLE :

```
In [44]: plot_time_series(aapl)

Calculated Moving Averages: for 4 weeks:Date
2018-01-07    173.129998
2018-01-14    175.067999
2018-01-21    178.252503
2018-01-28    174.175998
2018-02-04    166.128000
2018-02-11    158.123999
2018-02-18    167.967999
2018-02-25    172.730003
2018-03-04    177.338000
2018-03-11    177.088000
2018-03-18    179.360000
2018-03-25    171.120004
2018-04-01    168.842499
2018-04-08    169.572000
2018-04-15    172.922000
2018-04-22    174.084002
2018-04-29    163.674002
2018-05-06    174.330002
2018-05-13    187.439999
```

```
In [44]: plot_time_series(aapl)

Calculated Moving Averages: for 16 weeks:Date
2018-01-07    173.129998
2018-01-14    175.067999
2018-01-21    178.252503
2018-01-28    174.175998
2018-02-04    166.128000
2018-02-11    158.123999
2018-02-18    167.967999
2018-02-25    172.730003
2018-03-04    177.338000
2018-03-11    177.088000
2018-03-18    179.360000
2018-03-25    171.120004
2018-04-01    168.842499
2018-04-08    169.572000
2018-04-15    172.922000
2018-04-22    174.084002
2018-04-29    163.674002
2018-05-06    174.330002
2018-05-13    187.439999
```

```
In [44]: plot_time_series(aapl)
```

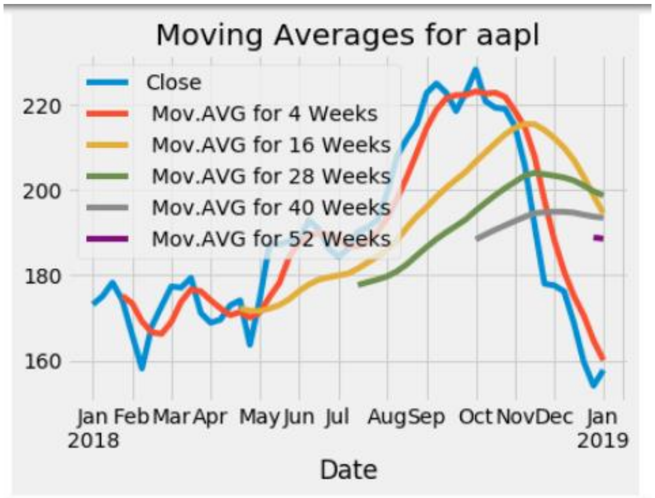
```
Calculated Moving Averages: for 28 weeks:Date
2018-01-07    173.129998
2018-01-14    175.067999
2018-01-21    178.252503
2018-01-28    174.175998
2018-02-04    166.128000
2018-02-11    158.123999
2018-02-18    167.967999
2018-02-25    172.730003
2018-03-04    177.338000
2018-03-11    177.088000
2018-03-18    179.360000
2018-03-25    171.120004
2018-04-01    168.842499
2018-04-08    169.572000
2018-04-15    172.922000
2018-04-22    174.084002
2018-04-29    163.674002
2018-05-06    174.330002
2018-05-13    187.439999
```

```
In [44]: plot_time_series(aapl)
```

```
Calculated Moving Averages: for 40 weeks:Date
2018-01-07    173.129998
2018-01-14    175.067999
2018-01-21    178.252503
2018-01-28    174.175998
2018-02-04    166.128000
2018-02-11    158.123999
2018-02-18    167.967999
2018-02-25    172.730003
2018-03-04    177.338000
2018-03-11    177.088000
2018-03-18    179.360000
2018-03-25    171.120004
2018-04-01    168.842499
2018-04-08    169.572000
2018-04-15    172.922000
2018-04-22    174.084002
2018-04-29    163.674002
2018-05-06    174.330002
2018-05-13    187.439999
```

```
In [44]: plot_time_series(aapl)
```

```
Calculated Moving Averages: for 52 weeks:Date
2018-01-07    173.129998
2018-01-14    175.067999
2018-01-21    178.252503
2018-01-28    174.175998
2018-02-04    166.128000
2018-02-11    158.123999
2018-02-18    167.967999
2018-02-25    172.730003
2018-03-04    177.338000
2018-03-11    177.088000
2018-03-18    179.360000
2018-03-25    171.120004
2018-04-01    168.842499
2018-04-08    169.572000
2018-04-15    172.922000
2018-04-22    174.084002
2018-04-29    163.674002
2018-05-06    174.330002
2018-05-13    187.439999
```



The Moving average for 4 weeks is somewhat closer to the actual Data, But the Moving average for 52 weeks is varies with the actual Data. The values for 52 weeks is linearly increasing and linearly decreasing.

RELIANCE:

```
In [45]: plot_time_series(reliance)

Calculated Moving Averages: for 4 weeks:Date
2018-01-07     915.850000
2018-01-14     939.719995
2018-01-21     929.520007
2018-01-28     971.750000
2018-02-04     944.879993
2018-02-11     898.209998
2018-02-18     926.525009
2018-02-25     927.209986
2018-03-04     948.037506
2018-03-11     909.849988
2018-03-18     920.609986
2018-03-25     896.850012
2018-04-01     894.533325
2018-04-08     901.320007
2018-04-15     926.289990
2018-04-22     937.900000
2018-04-29     969.639990
2018-05-06     962.887497
2018-05-13     977.040002
```

```
In [45]: plot_time_series(reliance)

Calculated Moving Averages: for 16 weeks:Date
2018-01-07     915.850000
2018-01-14     939.719995
2018-01-21     929.520007
2018-01-28     971.750000
2018-02-04     944.879993
2018-02-11     898.209998
2018-02-18     926.525009
2018-02-25     927.209986
2018-03-04     948.037506
2018-03-11     909.849988
2018-03-18     920.609986
2018-03-25     896.850012
2018-04-01     894.533325
2018-04-08     901.320007
2018-04-15     926.289990
2018-04-22     937.900000
2018-04-29     969.639990
2018-05-06     962.887497
2018-05-13     977.040002
```

```
In [45]: plot_time_series(reliance)
```

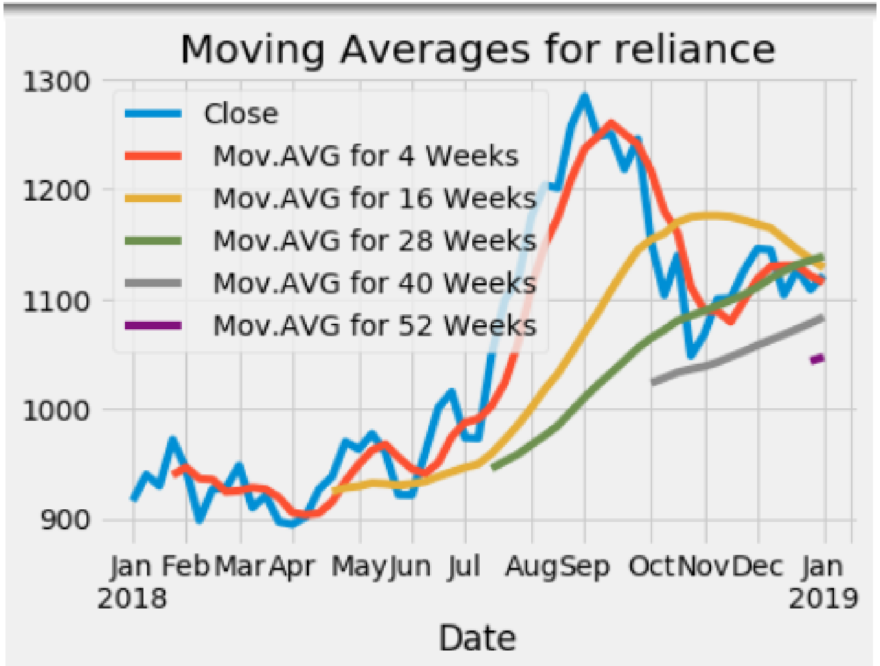
```
Calculated Moving Averages: for 28 weeks:Date
2018-01-07     915.850000
2018-01-14     939.719995
2018-01-21     929.520007
2018-01-28     971.750000
2018-02-04     944.879993
2018-02-11     898.209998
2018-02-18     926.525009
2018-02-25     927.209986
2018-03-04     948.037506
2018-03-11     909.849988
2018-03-18     920.609986
2018-03-25     896.850012
2018-04-01     894.533325
2018-04-08     901.320007
2018-04-15     926.289990
2018-04-22     937.900000
2018-04-29     969.639990
2018-05-06     962.887497
2018-05-13     977.040002
```

```
In [45]: plot_time_series(reliance)
```

```
Calculated Moving Averages: for 40 weeks:Date
2018-01-07     915.850000
2018-01-14     939.719995
2018-01-21     929.520007
2018-01-28     971.750000
2018-02-04     944.879993
2018-02-11     898.209998
2018-02-18     926.525009
2018-02-25     927.209986
2018-03-04     948.037506
2018-03-11     909.849988
2018-03-18     920.609986
2018-03-25     896.850012
2018-04-01     894.533325
2018-04-08     901.320007
2018-04-15     926.289990
2018-04-22     937.900000
2018-04-29     969.639990
2018-05-06     962.887497
2018-05-13     977.040002
```

```
In [45]: plot_time_series(reliance)
```

```
Calculated Moving Averages: for 52 weeks:Date
2018-01-07     915.850000
2018-01-14     939.719995
2018-01-21     929.520007
2018-01-28     971.750000
2018-02-04     944.879993
2018-02-11     898.209998
2018-02-18     926.525009
2018-02-25     927.209986
2018-03-04     948.037506
2018-03-11     909.849988
2018-03-18     920.609986
2018-03-25     896.850012
2018-04-01     894.533325
2018-04-08     901.320007
2018-04-15     926.289990
2018-04-22     937.900000
2018-04-29     969.639990
2018-05-06     962.887497
2018-05-13     977.040002
```



2. Create rolling window of size 10 on each stock/index. Handle unequal time series due to stock market holidays. You should look to increase your rolling window size to 75 and see how the data looks like.

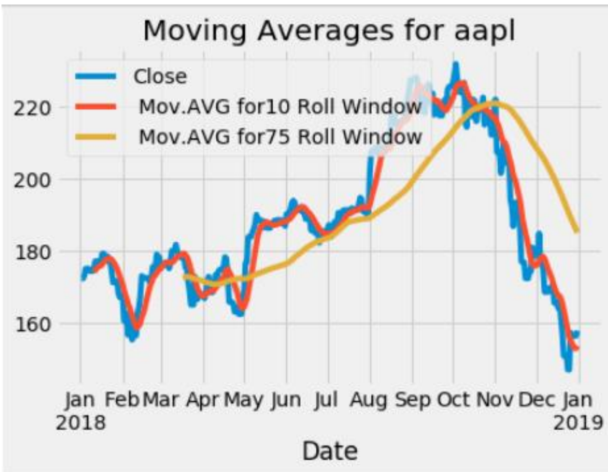
APPLE:

```
In [50]: plot_roll_win(aapl)

         Calculated Moving Averages: for 10 weeks:Date
         2018-01-02    172.259995
         2018-01-03    172.229996
         2018-01-04    173.029999
         2018-01-05    175.000000
         2018-01-06    175.000000
         2018-01-07    175.000000
         2018-01-08    174.350006
         2018-01-09    174.330002
         2018-01-10    174.289993
         2018-01-11    175.279999
         2018-01-12    177.089996
         2018-01-13    177.089996
         2018-01-14    177.089996
         2018-01-15    177.089996
         2018-01-16    176.190002
         2018-01-17    179.100006
         2018-01-18    179.259995
         2018-01-19    178.460007
         2018-01-20    178.460007
```

```
In [50]: plot_roll_win(aapl)
```

```
Calculated Moving Averages: for 75 weeks:Date
2018-01-02    172.259995
2018-01-03    172.229996
2018-01-04    173.029999
2018-01-05    175.000000
2018-01-06    175.000000
2018-01-07    175.000000
2018-01-08    174.350006
2018-01-09    174.330002
2018-01-10    174.289993
2018-01-11    175.279999
2018-01-12    177.089996
2018-01-13    177.089996
2018-01-14    177.089996
2018-01-15    177.089996
2018-01-16    176.190002
2018-01-17    179.100006
2018-01-18    179.259995
2018-01-19    178.460007
2018-01-20    178.460007
```

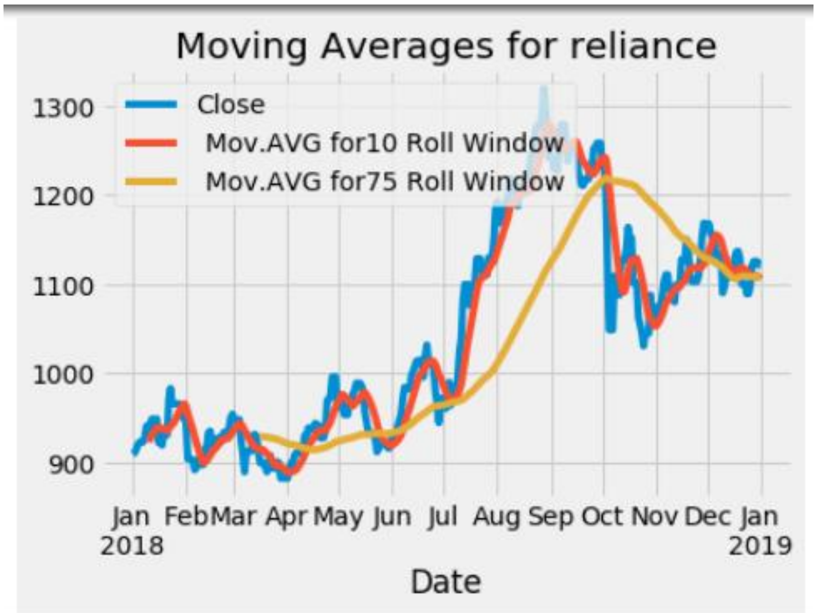


From Above graph plottings, we can visualize that, as much as the rolling window is small, the moving average 10 is somehow significant and closer to the actual data.

Rolling window for Reliance:

```
Calculated Moving Averages: for 10 weeks:Date
2018-01-01    909.750000
2018-01-02    911.150024
2018-01-03    914.799988
2018-01-04    920.299988
2018-01-05    923.250000
2018-01-06    923.250000
2018-01-07    923.250000
2018-01-08    928.549988
2018-01-09    940.950012
2018-01-10    942.349976
2018-01-11    937.750000
2018-01-12    949.000000
2018-01-13    949.000000
2018-01-14    949.000000
2018-01-15    949.150024
2018-01-16    922.950012
2018-01-17    924.500000
2018-01-18    919.700012
2018-01-19    931.299988
```

```
Calculated Moving Averages: for 75 weeks:Date
2018-01-01      909.750000
2018-01-02      911.150024
2018-01-03      914.799988
2018-01-04      920.299988
2018-01-05      923.250000
2018-01-06      923.250000
2018-01-07      923.250000
2018-01-08      928.549988
2018-01-09      940.950012
2018-01-10      942.349976
2018-01-11      937.750000
2018-01-12      949.000000
2018-01-13      949.000000
2018-01-14      949.000000
2018-01-15      949.150024
2018-01-16      922.950012
2018-01-17      924.500000
2018-01-18      919.700012
2018-01-19      931.299988
```



From Above graph plottings, we can visualize that, as much as the rolling window is small, the moving average 10 is somehow significant and closer to the actual data

3. Create the following dummy time series:

- Volume shocks - If volume traded is 10% higher/lower than previous day - make a 0/1 boolean time series for shock, 0/1 dummy-coded time series for direction of shock.

Date	Date	Open	High	Low	Close	Adj Close	Volume	Year	Month	Day	WeekOfYear	vol_t+1	volume_shock
2018-01-02	2018-01-02	170.160004	172.300003	169.259995	172.259995	168.987320	25555900	2018	1	2	1	NaN	0
2018-01-03	2018-01-03	172.529999	174.550003	171.960007	172.229996	168.957886	29517900	2018	1	3	1	25555900.0	1
2018-01-04	2018-01-04	172.539993	173.470001	172.080002	173.029999	169.742706	22434600	2018	1	4	1	29517900.0	1
2018-01-05	2018-01-05	173.440002	175.369995	173.050003	175.000000	171.675278	23660000	2018	1	5	1	22434600.0	0
2018-01-06	2018-01-05	173.440002	175.369995	173.050003	175.000000	171.675278	23660000	2018	1	5	1	23660000.0	0
2018-01-07	2018-01-05	173.440002	175.369995	173.050003	175.000000	171.675278	23660000	2018	1	5	1	23660000.0	0
2018-01-08	2018-01-08	174.350006	175.610001	173.929993	174.350006	171.037628	20567800	2018	1	8	2	23660000.0	1
2018-01-09	2018-01-09	174.550003	175.059998	173.410004	174.330002	171.018005	21584000	2018	1	9	2	20567800.0	0
2018-01-10	2018-01-10	173.160004	174.300003	173.000000	174.289993	170.978760	23959900	2018	1	10	2	21584000.0	0
2018-01-11	2018-01-11	174.589996	175.490005	174.490005	175.279999	171.949951	18667700	2018	1	11	2	23959900.0	1
2018-01-12	2018-01-12	176.179993	177.360001	175.649994	177.089996	173.725571	25226000	2018	1	12	2	18667700.0	1
2018-01-13	2018-01-12	176.179993	177.360001	175.649994	177.089996	173.725571	25226000	2018	1	12	2	25226000.0	0
2018-01-14	2018-01-12	176.179993	177.360001	175.649994	177.089996	173.725571	25226000	2018	1	12	2	25226000.0	0
2018-01-15	2018-01-12	176.179993	177.360001	175.649994	177.089996	173.725571	25226000	2018	1	12	2	25226000.0	0
2018-01-16	2018-01-16	177.899994	179.389999	176.139999	176.190002	172.842682	29565900	2018	1	16	3	25226000.0	1
2018-01-17	2018-01-17	176.149994	179.250000	175.070007	179.100006	175.697388	34386800	2018	1	17	3	29565900.0	1

- Price shocks - If closing price at T vs T+1 has a difference > 2%, then 0/1 boolean time series for shock, 0/1 dummy-coded time series for direction of shock.

```
In [26]: vol_shock_direction(aapl)
```

ut[26]:

Date	Date	Open	High	Low	Close	Adj Close	Volume	Year	Month	Day	WeekOfYear	vol_t+1	volume_shock	VOL_SHOCK_DIR
2018-01-02	2018-01-02	170.160004	172.300003	169.259995	172.259995	168.987320	25555900	2018	1	2	1	NaN	0	NaN
2018-01-03	2018-01-03	172.529999	174.550003	171.960007	172.229996	168.957886	29517900	2018	1	3	1	25555900.0	1	0.0
2018-01-04	2018-01-04	172.539993	173.470001	172.080002	173.029999	169.742706	22434600	2018	1	4	1	29517900.0	1	1.0
2018-01-05	2018-01-05	173.440002	175.369995	173.050003	175.000000	171.675278	23660000	2018	1	5	1	22434600.0	0	NaN
2018-01-06	2018-01-05	173.440002	175.369995	173.050003	175.000000	171.675278	23660000	2018	1	5	1	23660000.0	0	NaN
2018-01-07	2018-01-05	173.440002	175.369995	173.050003	175.000000	171.675278	23660000	2018	1	5	1	23660000.0	0	NaN
2018-	2018-	174.350006	175.610001	173.929993	174.350006	171.037628	20567800	2018	1	8	2	23660000.0	1	1.0

```
In [34]: vol_shock_direction(reliance)
```

]:

Date	Date	Open	High	Low	Close	Adj Close	Volume	Year	Month	Day	WeekOfYear	VOL_SHOCK_DIR	vol_t+1	volume_shock
2018-01-01	2018-01-01	922.700012	922.700012	907.500000	909.750000	904.174133	4321686	2018	1	1	1	NaN	NaN	0
2018-01-02	2018-01-02	913.000000	919.549988	906.400024	911.150024	905.565552	4342815	2018	1	2	1	NaN	4321686.0	0
2018-01-03	2018-01-03	925.000000	926.000000	913.049988	914.799988	909.193176	6175312	2018	1	3	1	0.0	4342815.0	1
2018-01-04	2018-01-04	918.150024	921.799988	915.700012	920.299988	914.659424	4118581	2018	1	4	1	1.0	6175312.0	1
2018-01-05	2018-01-05	921.799988	926.900024	920.250000	923.250000	917.591370	3401905	2018	1	5	1	1.0	4118581.0	1
2018-01-06	2018-01-05	921.799988	926.900024	920.250000	923.250000	917.591370	3401905	2018	1	5	1	NaN	3401905.0	0
2018-	2018-	921.799988	926.900024	920.250000	923.250000	917.591370	3401905	2018	1	5	1	NaN	3401905.0	0

- Pricing black swan - If closing price at T vs T+1 has a difference > 2%, then 0/1 boolean time series for shock, 0/1 dummy-coded time series for direction of shock.

```
In [28]: price_shocks(aapl)
```

igh	Low	Close	Adj Close	Volume	Year	Month	Day	WeekOfYear	vol_t+1	volume_shock	VOL_SHOCK_DIR	price_t+1	price_shock	price_black_swan
003	169.259995	172.259995	168.987320	25555900	2018	1	2	1	NaN	0	NaN	NaN	0	0
003	171.960007	172.229996	168.957886	29517900	2018	1	3	1	25555900.0	1	0.0	172.259995	0	0
001	172.080002	173.029999	169.742706	22434600	2018	1	4	1	29517900.0	1	1.0	172.229996	0	0
995	173.050003	175.000000	171.675278	23660000	2018	1	5	1	22434600.0	0	NaN	173.029999	0	0
995	173.050003	175.000000	171.675278	23660000	2018	1	5	1	23660000.0	0	NaN	175.000000	0	0
995	173.050003	175.000000	171.675278	23660000	2018	1	5	1	23660000.0	0	NaN	175.000000	0	0
001	173.929993	174.350006	171.037628	20567800	2018	1	8	2	23660000.0	1	1.0	175.000000	0	0

```
In [36]: price_shocks(reliance)
```

Date	Date	Open	High	Low	Close	Adj Close	Volume	Year	Month	Day	WeekOfYear	price_t+1	price_shock	price_black_swan
2018-01-01	2018-01-01	922.700012	922.700012	907.500000	909.750000	904.174133	4321686	2018	1	1	1	NaN	0	0
2018-01-02	2018-01-02	913.000000	919.549988	906.400024	911.150024	905.565552	4342815	2018	1	2	1	909.750000	0	0
2018-01-03	2018-01-03	925.000000	926.000000	913.049988	914.799988	909.193176	6175312	2018	1	3	1	911.150024	0	0
2018-01-04	2018-01-04	918.150024	921.799988	915.700012	920.299988	914.659424	4118581	2018	1	4	1	914.799988	0	0
2018-01-05	2018-01-05	921.799988	926.900024	920.250000	923.250000	917.591370	3401905	2018	1	5	1	920.299988	0	0
2018-01-06	2018-01-05	921.799988	926.900024	920.250000	923.250000	917.591370	3401905	2018	1	5	1	923.250000	0	0
2018-	2018-	921.799988	926.900024	920.250000	923.250000	917.591370	3401905	2018	1	5	1	923.250000	0	0

Pricing shock without volume shock - based on points a & b - Make a 0/1 dummy time series.

```
In [33]: price_shock_wo_vol_shock(aapl)
```

Volume	Year	Month	Day	WeekOfYear	vol_t+1	volume_shock	VOL_SHOCK_DIR	price_t+1	price_shock	price_black_swan	not_vol_shock	price_shock_w/0_vol_shock
5555900	2018	1	2	1	NaN	0	NaN	NaN	0	0	1	0
9517900	2018	1	3	1	25555900.0	1	0.0	172.259995	0	0	0	0
2434600	2018	1	4	1	29517900.0	1	1.0	172.229996	0	0	0	0
3660000	2018	1	5	1	22434600.0	0	NaN	173.029999	0	0	1	0
3660000	2018	1	5	1	23660000.0	0	NaN	175.000000	0	0	1	0
3660000	2018	1	5	1	23660000.0	0	NaN	175.000000	0	0	1	0
0567800	2018	1	8	2	23660000.0	1	1.0	175.000000	0	0	0	0

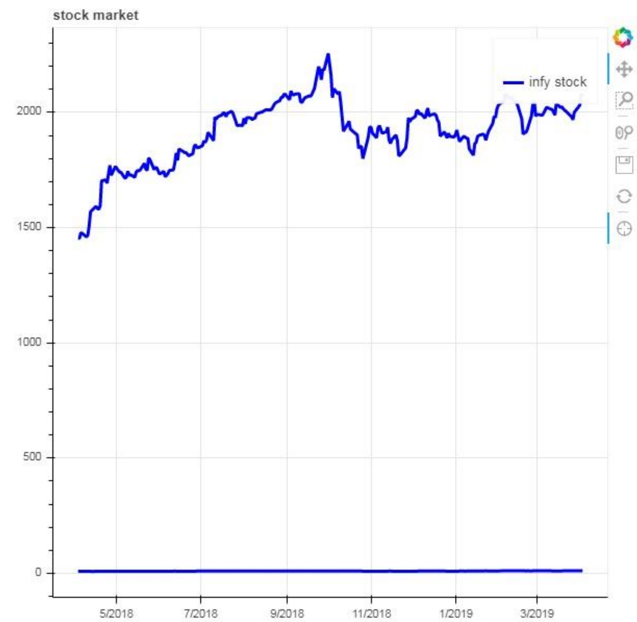
DATASET:

Infosys and TCS are 2 dataset which has been taken. The Period was from 3-april-2018 to 3-april-2019.With these values of the Dataset we can be able to visualize the how the trend is set on which Date and we can able to predict that company is going on with the stock values very well.

Part 2 (data visualization):

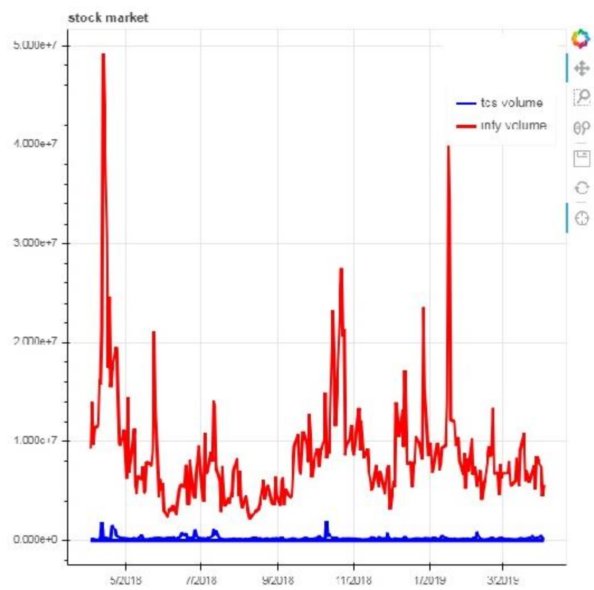
1. Create timeseries plot of close prices of stocks/indices with the following features:
2. Color timeseries in simple blue color

From the TCS stock we can we saying that the values is exponetially increasing in start of the April .The Stock of the TCS is Linearly Increasing and decreasing .



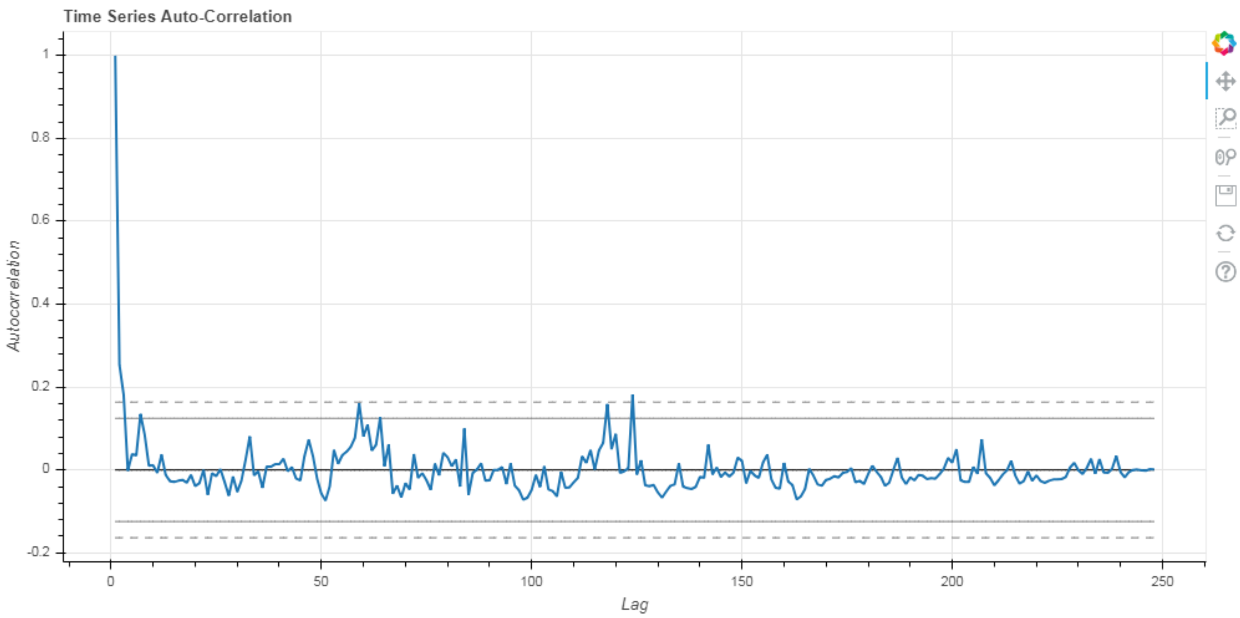
In INFY the stock values has gradually increasing and constantly maintaining the stock with the increasing and decreasing of the value, but not exponentially decreasing.

3. Color timeseries between two volume shocks in a different color (Red)

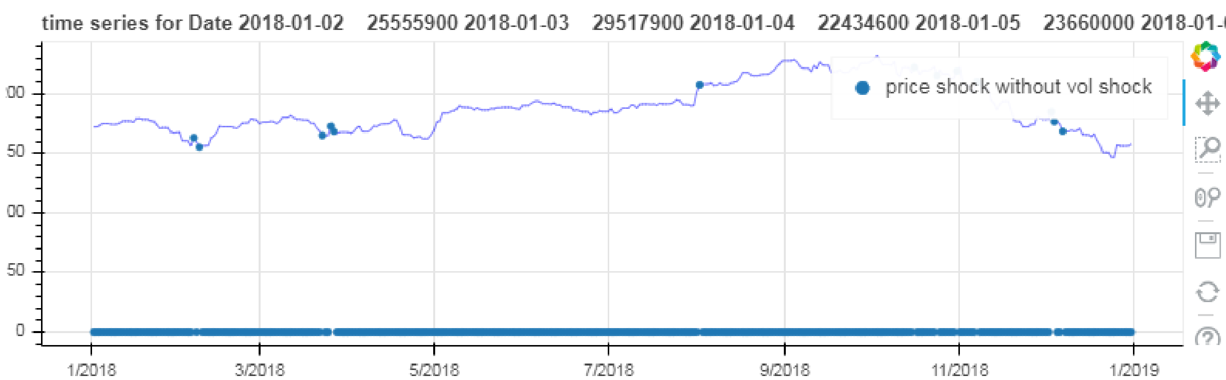


From the graph we can clearly say that the volume of Infosys is more and more higher compared to the TCS stock.The Infosys volume was exponentially increasing and exponentially decreasing.

4. Hand craft partial autocorrelation plot for each stock/index on upto all lookbacks on bokeh



5. Mark closing Pricing shock without volume shock to identify volumeless price movement.



Part-3:

1. Quick build any two models. Quick build is defined as grid search of less than 9 permutation combinations. You can choose the two options of multiple multivariate models from those mentioned below. The goal is to to predict INFY,TCS prices for tomorrow.
2. 1.1 http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLars.html#sklearn.linear_model.LassoLars

TCS:

Lasso

```
In [15]: from sklearn import linear_model
         clf = linear_model.Lasso(alpha=0.1)
         clf.fit(X_train,y_train)

Out[15]: Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,
            normalize=False, positive=False, precompute=False, random_state=None,
            selection='cyclic', tol=0.0001, warm_start=False)
```

```
In [17]: from sklearn.model_selection import GridSearchCV
         param_grid = [
             {'alpha': [0.01,0.10,1.0], 'max_iter': [200,400,600,800]},
             {'normalize' : [True],'alpha': [0.01,0.10,1.0], 'max_iter': [200,400,600,800]}
         ]

         grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='neg_mean_squared_error')
         grid_search.fit(X_train, y_train)

C:\Users\shrini\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_descent.py:491: ConvergenceWarning: Objective di
d not converge. You might want to increase the number of iterations. Fitting data with very small alpha may cause precision
problems.
  ConvergenceWarning)
C:\Users\shrini\Anaconda3\lib\site-packages\sklearn\linear_model\coordinate_descent.py:491: ConvergenceWarning: Objective di
d not converge. You might want to increase the number of iterations. Fitting data with very small alpha may cause precision
problems.
```

```
In [18]: grid_search.best_params_

Out[18]: {'alpha': 1.0, 'max_iter': 800}
```

```
In [19]: cv_scores = grid_search.cv_results_
         for mean_score, params in zip(cv_scores["mean_test_score"], cv_scores["params"]):
           print(np.sqrt(-mean_score), params)

193.34202781928934 {'alpha': 0.01, 'max_iter': 200}
194.2757844772466 {'alpha': 0.01, 'max_iter': 400}
195.39978795944998 {'alpha': 0.01, 'max_iter': 600}
196.6941104237667 {'alpha': 0.01, 'max_iter': 800}
192.95624846737806 {'alpha': 0.1, 'max_iter': 200}
193.5105943481795 {'alpha': 0.1, 'max_iter': 400}
194.105094588869 {'alpha': 0.1, 'max_iter': 600}
194.71243519148877 {'alpha': 0.1, 'max_iter': 800}
188.67470844797688 {'alpha': 1.0, 'max_iter': 200}
186.4482857598098 {'alpha': 1.0, 'max_iter': 400}
184.23315686637318 {'alpha': 1.0, 'max_iter': 600}
182.503837708933 {'alpha': 1.0, 'max_iter': 800}
191.40894569578182 {'alpha': 0.01, 'max_iter': 200, 'normalize': True}
191.4150534482485 {'alpha': 0.01, 'max_iter': 400, 'normalize': True}
191.40857429027247 {'alpha': 0.01, 'max_iter': 600, 'normalize': True}
191.41307621407591 {'alpha': 0.01, 'max_iter': 800, 'normalize': True}
190.39568454406606 {'alpha': 0.1, 'max_iter': 200, 'normalize': True}
190.1149089356979 {'alpha': 0.1, 'max_iter': 400, 'normalize': True}
189.88717649120483 {'alpha': 0.1, 'max_iter': 600, 'normalize': True}
189.7118300252203 {'alpha': 0.1, 'max_iter': 800, 'normalize': True}
```

The best parameter is the alpha: 1.0, max_iter:800 because it has the value which is the least one and it will taken consider for the next forecasting. The function will take all possible values it will be given the best one to predict the values

INFY:

```
In [8]: def train_test_split(a, n): return a[:n], a[n:]

        n_valu = 50
        n_traini = len(infy)-n_valu
        X_Train, X_valu = train_test_split(infy.drop(columns=["Close"]), n_traini)
        y_Train, y_valu = train_test_split(infy["Close"], n_traini)

        X_Train.shape, y_Train.shape, X_valu.shape, y_valu.shape

Out[8]: ((202, 13), (202,), (50, 13), (50,))

In [9]: from sklearn.model_selection import GridSearchCV
        param_grid = [
            {'n_estimators': [3,10,30], 'max_features': [2,4,6,8]},
            {'bootstrap' : [False],'n_estimators': [3,10], 'max_features': [2, 3, 4]}
        ]

        grid_search = GridSearchCV(model, param_grid, cv=5, scoring='neg_mean_squared_error')
        grid_search.fit(X_Train, y_Train)

Out[9]: GridSearchCV(cv=5, error_score='raise',
               estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                   max_features='auto', max_leaf_nodes=None,
                   min_impurity_decrease=0.0, min_impurity_split=None,
                   min_samples_leaf=1, min_samples_split=2,
                   min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                   oob_score=False, random_state=None, verbose=0, warm_start=False),
               fit_params=None, iid=True, n_jobs=1,
```

```
In [10]: grid_search.best_params_

Out[10]: {'max_features': 8, 'n_estimators': 3}

In [11]: cv_scores = grid_search.cv_results_
         for mean_score, params in zip(cv_scores["mean_test_score"], cv_scores["params"]):
           print(np.sqrt(-mean_score), params)

         0.44205957366249443 {'max_features': 2, 'n_estimators': 3}
         0.42444942351098985 {'max_features': 2, 'n_estimators': 10}
         0.44356358260702805 {'max_features': 2, 'n_estimators': 30}
         0.5237140860186879 {'max_features': 4, 'n_estimators': 3}
         0.4283526685610422 {'max_features': 4, 'n_estimators': 10}
         0.43764533355366425 {'max_features': 4, 'n_estimators': 30}
         0.382407710847639 {'max_features': 6, 'n_estimators': 3}
         0.3892764693048571 {'max_features': 6, 'n_estimators': 10}
         0.4200618113207073 {'max_features': 6, 'n_estimators': 30}
         0.3652390826003509 {'max_features': 8, 'n_estimators': 3}
         0.41954106674218217 {'max_features': 8, 'n_estimators': 10}
         0.3916269458827803 {'max_features': 8, 'n_estimators': 30}
         0.43763217396801307 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
         0.39472075603839823 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
         0.4281564016693304 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
         0.4117682182904107 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
         0.42820667888997793 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
         0.4381297490244162 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

3. 1.2 http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression

TCS:

```
In [14]: from sklearn.linear_model import LinearRegression
         reg=LinearRegression().fit(X_train,y_train)
         reg.score(X_train,y_train)

Out[14]: 0.4679461175865742
```

With the help of the Training Dataset we can be able to predict the next forecasting values for the company

INFY:

```
In [13]: from sklearn.linear_model import LinearRegression
         reg=LinearRegression().fit(X_Train,y_Train)
         reg.score(X_Train,y_Train)

Out[13]: 0.2552882424284256
```

4. 1.3 http://scikit-learn.org/stable/modules/linear_model.html#ridge-regression

TCS:

```
In [26]: grid_search.best_params_

Out[26]: {'alpha': 1.0, 'max_iter': 600, 'normalize': True}

In [27]: cv_scores = grid_search.cv_results_
         for mean_score, params in zip(cv_scores["mean_test_score"], cv_scores["params"]):
           print(np.sqrt(-mean_score), params)

4740.707394564728 {'alpha': 0.01, 'max_iter': 200}
4740.70740018896 {'alpha': 0.01, 'max_iter': 400}
4740.70739209991 {'alpha': 0.01, 'max_iter': 600}
4740.707394564728 {'alpha': 0.01, 'max_iter': 800}
925.389213131196 {'alpha': 0.1, 'max_iter': 200}
925.389212777705 {'alpha': 0.1, 'max_iter': 400}
925.389213131196 {'alpha': 0.1, 'max_iter': 600}
925.389213131196 {'alpha': 0.1, 'max_iter': 800}
167.84417693368852 {'alpha': 1.0, 'max_iter': 200}
167.84417694173652 {'alpha': 1.0, 'max_iter': 400}
167.84417693780236 {'alpha': 1.0, 'max_iter': 600}
167.84417693780236 {'alpha': 1.0, 'max_iter': 800}
173.65126529289702 {'alpha': 0.01, 'max_iter': 200, 'normalize': True}
173.65126529289398 {'alpha': 0.01, 'max_iter': 400, 'normalize': True}
173.65126529289512 {'alpha': 0.01, 'max_iter': 600, 'normalize': True}
173.65126529289702 {'alpha': 0.01, 'max_iter': 800, 'normalize': True}
171.3204217075831 {'alpha': 0.1, 'max_iter': 200, 'normalize': True}
```

INFY:

```
In [22]: grid_search.best_params_

Out[22]: {'alpha': 1.0, 'max_iter': 400, 'normalize': True}

In [23]: cv_scores = grid_search.cv_results_
         for mean_score, params in zip(cv_scores["mean_test_score"], cv_scores["params"]):
           print(np.sqrt(-mean_score), params)

9.398239320995993 {'alpha': 0.01, 'max_iter': 200}
9.398239314502877 {'alpha': 0.01, 'max_iter': 400}
9.398239320995993 {'alpha': 0.01, 'max_iter': 600}
9.398239320995993 {'alpha': 0.01, 'max_iter': 800}
1.998762808272583 {'alpha': 0.1, 'max_iter': 200}
1.9987628101160435 {'alpha': 0.1, 'max_iter': 400}
1.9987628088695615 {'alpha': 0.1, 'max_iter': 600}
1.998762808272583 {'alpha': 0.1, 'max_iter': 800}
0.8054646099486573 {'alpha': 1.0, 'max_iter': 200}
0.8054646099486567 {'alpha': 1.0, 'max_iter': 400}
0.8054646099403424 {'alpha': 1.0, 'max_iter': 600}
0.8054646099486573 {'alpha': 1.0, 'max_iter': 800}
0.8458041849793334 {'alpha': 0.01, 'max_iter': 200, 'normalize': True}
0.8458041849792722 {'alpha': 0.01, 'max_iter': 400, 'normalize': True}
0.8458041849792803 {'alpha': 0.01, 'max_iter': 600, 'normalize': True}
0.8458041849793334 {'alpha': 0.01, 'max_iter': 800, 'normalize': True}
0.8369096058763711 {'alpha': 0.1, 'max_iter': 200, 'normalize': True}
0.8369096058763571 {'alpha': 0.1, 'max_iter': 400, 'normalize': True}
0.8369096058763822 {'alpha': 0.1, 'max_iter': 600, 'normalize': True}
0.8369096058763711 {'alpha': 0.1, 'max_iter': 800, 'normalize': True}
```