Comparison of ARIMA, LSTM, SMA, and Holt-winters for Stock Price In Time Series Analysis

Table of Contents

Comparison of ARIMA, LSTM, SMA, and Holt-winters for Stock Price In Time Series Analysis

Dylan, N., Setiawan, M., and Khai, F. F.

1.0 Abstract

Utilising time series analysis has always been an essential part of the stock market in an attempt to increase profit and reduce risk. Several algorithms—such as Autoregressive Integrated Moving Average (ARIMA), Long Short-Term Memory (LSTM), Simple Moving Average (SMA), Exponential Smoothing (ES), Vector Auto-Regressive (VAR), Holt-Winters, and Prophet—are available to predict time series. We opted to focus on ARIMA, LSTM, SMA, Holt-winters; as 4 of these algorithms are quite popular and it suits the state of our equipment. This research objective is to find out how 4 of these algorithms compare and whether LSTM, as a deep learning algorithm, can outperform the rest—which are machine learning algorithms. Our studies established that ARIMA performed the best out of all the algorithms.

Keywords: Stock market, time series analysis, ARIMA, LSTM, SMA, Holt-winters

2.0 Introduction

2.1 Briefly describe the broad research area and narrow it down to your particular focus

Machine Learning is the expansion and usage of computer systems that are capable of learning and adapting without the need of human intervention or special orders. Over the past years, it has been one of the main contributors in the advancement of data science, where statistical methods are used, and algorithms are implemented to generate classifications, forecasts, and discover some insights in data mining. Thus, enhancing decision making within applications and businesses. Moreover, technological advancements in storage and processing power have contributed to the emergence of several products hinged on machine learning, for instance Netflix's recommendation engine and self-driving cars. On top of that, there is also a high rate of the usage of machine learning for stock market prediction, since it is focused on forecasting fluctuations in the market, learning about customers behaviour, and scanning the stock price dynamics. Through establishing the relations between variables and using the features to train the model, we may predict the values, or in this case, the stock price [1]. Moreover, the vital reason why stock market prediction is suitable for machine learning is because the stock market is always changing, unstable, and not having a direct pattern. Besides, the stock market has its own challenges since there are several factors that need to be included, namely politics, unexpected occurrences, and the world economy. Yet, these data affluence are the best resources for financial analysts, data scientists, and researchers to utilise these data into diverse analytical methods to discern current trends in the stock market. Therefore, leading to the emergence of algorithmic trading that applies pre-arranged automated plans to conduct trades [2]. To conclude, its role in financial markets now is more important than ever as approximately 20 million amateur investors got into the stock market during the pandemic.

2.2 What you want to achieve and what is interesting to be find out by the reader

Due to the complexity of the stock market, most inexperienced investors might view the stock price predictions as something challenging. In addition to that, the various prediction algorithms which can be found might be overwhelming and perplexing. Through this research we are aspiring to be contributing to the stock price market by reviewing 3 of the most popular prediction algorithm which we discovered—which consists of ARIMA, Holt-Winters, and LSTM—in order to help guide the readers on which algorithm might be the best for their situation and clear out their confusion regarding the algorithms available. In addition, we will also review a less-known algorithm—which is SMA—in order to look into any other algorithm that may be underestimated.

2.3 Summarize the relevant literature on your topic

Referring to a literature piece by Kulkarni, M., et al. [3], the authors utilised 3 models (ARIMA, Simple Moving Average, and Holt-Winters) and 10 years of data (2009-2019) to do time series analysis for stock market prediction and compared those 3 models. The authors encountered that SMA has the lowest prediction error, followed by Holt-Winters. ARIMA is the weakest out of all, but it has the potential to do well in short-term stock prediction.

A research done in 2022 in Indonesia by Soeryawinata, J., et al. [4], attempted to forecast the number of bike sales using ARIMA, LSTM, and Holt-Winters respectively. The authors utilised a 6 year dataset ranging from 2015-2021 which were separated into 6 years of training data and 1 year of testing data. The authors came up with the conclusion that ARIMA is the best algorithm with a RMSE value averaging at around 2.4905. LSTM came up next with its average RMSE around 2.8533, and Holt-Winters with the worst average RMSE of 3.2339.

The author, Andi, H. K. [5], implemented LSTM to predict Bitcoin price using linear regression. Applying a dataset from 2014 to 2019 with 25 unique characteristics, the researcher determined a total of 31942 training set values and 15734 test set values. The aforementioned claimed to have managed to counteract the overfitting issues which enabled the LSTM model to achieve a 97.2% accuracy and 98.32% precision rate.

Ramadhan, R. H., et. al., [6] applied and compared 2 algorithms—Holt-Winters and SMA—to find the best model to predict flood potential in Bekasi, a city within the Jakarta metropolitan area. The authors gathered clipped image object data from the United States Geological Survey using Landsat 8 technology from 1 May 2019 to 1 April 2021 and the NDWI values were produced. Other than that, the precipitation data from 1 May 2019 to 1 April 2021 were also collected to be used as secondary data. Based on the day's low forecast error value, lower value of RMSE, MAE, MAPE, and MSE, Holt-Winters is the superior algorithm.

The authors—Mahmud, N., et. al. [7]—utilised the ARIMA and Holt-Winters (multiplicative and additive) algorithm to do a prediction of a dengue outbreak in Selangor, Malaysia. The data gathered were the number of dengue cases per week in Selangor from January 2018 to November 2020, totalling in 150 weeks' worth of data. The dataset is divided to 70% for training and 30% for validation. They concluded that Additive Holt-Winters is the best model (RMSE = 145.413), followed by ARIMA (RMSE = 151.278), and at last, Multiplicative Holt-Winters (RMSE = 514.347).

Dritsaki, M., et. al. [8] implemented ARIMA and Holt-Winters model and compared their performance forecasting of GDP per capita in five Balkan countries—Romania, Slovenia, Greece, Bulgaria, and Croatia—which are members of the European Union. For the analysis, they gathered the annual GDP per capita at 2015 reference level for each country starting from 1990 until 2020. They concluded that ARIMA has the best performance for all four countries except Romania. In Romania, they found that Holt-Winters has the overall lowest RMSE value. The researchers also noted that Romania has a positive skewness which means that the remote values on GDP per capita is higher than average.

Kucĕra, J., et. al. [9], implemented the closing price from the S&P 500 and Nasdaq Composite Index from 31 March 1992 to 31 March 2022. Any missing data is filled in the gap by duplicating the data from the previous day. Based on the average MAPE, the authors concluded ARIMA is better than SMA at predicting both indexes. ARIMA is 6.11% better than SMA in predicting the NASDAQ Composite Index, meanwhile for S&P 500, ARIMA has a 8.12% more accuracy compared to SMA.

A paper from 2021 by Hu, Z. & Zhao, Y., et al. [10], did a survey and reviewed 86 publications dated after 2015 which had the keywords "CNN", "LSTM", "RNN", "deep learning", or "reinforcement learning" and "Stock/forex". Based on the MSE, LSTM shows promising results with 1 of the papers achieving the best MSE and 40% of the LSTM papers were in the range of 0–0.01. One of the LSTM papers also managed to be the only paper to get an accuracy rate of more than 90%.

The paper of Efrilia, I. [11] aims at forecasting CPI (Consumer Price Index) in Tegal City, Central Java. The additive exponential smoothing Holt-Winters and ARIMA algorithm is used and the data was taken from BPS-Statistics of Tegal City CPI from January 2014 to June 2021. The performance itself will be based on the Mean Absolute Percentage Error (MAPE) value. Their study indicated that additive Holt-Winters has an advantage over ARIMA as Holt-Winters has a MAPE value of 0.281, which is considerably better than the ARIMA's MAPE value of 0.311.

2.4 Describe the current state of the art

Nowadays, machine learning and deep learning have already become an integral part of our life. Referring to Arthur Samuel, machine learning is defined as a "Field of study that gives computers the ability to learn without being explicitly programmed" [12]. Meanwhile, deep learning is a machine learning concept based on artificial neural networks [13].

Even though some studies suggested that deep learning is overtaking machine learning in this recent years [14] [13], we believe that from looking into other similar research that machine learning has been and is still promising and may be the best answer for some certain situations. Machine learning algorithms such as ARIMA and Holt-Winters have been showing various but also promising results. SMA is much less discussed, especially in these recent years.

Based on this situation, we would like to compare 3 of these machine learning algorithms, and also compare them with one deep learning algorithm which has repeatedly shown superior performance which is LSTM.

2.5 Gaps in the literature that your study will address

The studies mentioned above are not without any shortcomings. We believe that by overcoming these shortcomings it is possible to improve the outcome.

Referring to the first paper [3], the researchers were said to be utilising 10 years of price data ranging from 2009 to 2019, yielding a p-value of 0.57 for the ARIMA model, which is significantly more than 0.05 and shows that the data is not stationary enough, and may be a sign of lack of data.

In the second paper [4], the researcher failed to pre-process the data automatically through the code as they lack the hardware needed and also the software required, other than that, the authors failed to test out a bigger dataset on the LSTM model as they only fed 72 data during the training.

Similarly, the lack of a bigger dataset can be found in the author, Andi, H. K.'s research paper [5], with the author only utilising 5 years of data from 2014-2019.

The similarities continue with the fifth study by the author Ramadhan, R. H., et. al. [6], with the datasets only consisting of 25 data of rain precipitation and temperature.

The research by the authors—Mahmud, N., et. al. [7], only utilises the data from 2018 until 2020, the problem with this, the COVID pandemic affected the number of dengue cases spreading. A bigger dataset might be more useful as the results differ from the author's literature review regarding another similar research done in Indonesia, where Holt-Winters came out on top.

2.6 Hypothesis and research question

Hypothesis

In general terms, hypothesis is a statement that is utilised to conduct an experiment on the connection of two or more variables or a suggested clarification about several observation phenomena. Thus, in this report the hypothesis that will be made are stated below:

- The implementation of the SMA algorithm with a longer time frame will result in a (higher/lower) accuracy of the price prediction in the stock price.
- The implementation of the SMA algorithm with a shorter time frame will result in a (higher/lower) accuracy of the price prediction in the stock price
- The implementation of the ARIMA algorithm with a longer time frame will result in a (higher/lower) accuracy of the price prediction in the stock price.
- The implementation of the ARIMA algorithm with a shorter time frame will result in a (higher/lower) accuracy of the price prediction in the stock price
- The implementation of the Holt-winters algorithm with a longer time frame will result in a (higher/lower) accuracy of the price prediction in the stock price.
- The implementation of the Holt-winters algorithm with a shorter time frame will result in a (higher/lower) accuracy of the price prediction in the stock price
- The implementation of the LSTM algorithm with a longer time frame will result in a (higher/lower) accuracy of the price prediction in the stock price.
- The implementation of the LSTM algorithm with a shorter time frame will result in a (higher/lower) accuracy of the price prediction in the stock price

Research Question

Research questions are specified questions where the research is conducted to deliver the answers to, thus normally positioned at the first step of the project. Consequently, research questions emphasize on research, dictate the methodology and hypothesis, and provide the route for all steps in analysis, inquiry, and report. Furthermore, these are the research question that we have come up with are stated below:

- Does changing the range of time frame with SMA algorithm affect the accuracy of the price prediction in the stock price?
- Does changing the range of time frame with ARIMA algorithm affect accuracy of the price prediction in the stock price?
- Does changing the range of time frame with Holt-winters algorithm affect accuracy of the price prediction in the stock price?
- Does changing the range of time frame with LSTM algorithm affect accuracy of the price prediction in the stock price?
- Does the accuracy of the price prediction using SMA algorithm tend to be higher than ARIMA model or vice versa?
- Does the accuracy of the price prediction using SMA algorithm tend to be higher than Holt-winters model or vice versa?
- Does the accuracy of the price prediction using SMA algorithm tend to be higher than LSTM model or vice versa?
- Does the accuracy of the price prediction using ARIMA algorithm tend to be higher than Holt-winters model or vice versa?
- Does the accuracy of the price prediction using ARIMA algorithm tend to be higher than LSTM model or vice versa?
- Does the accuracy of the price prediction using Holt-winters algorithm tend to be higher than LSTM model or vice versa?

2.7 Describe how will we accomplish our aims

In terms of the method used to accomplish the aim of this project, we develop models from the 4 algorithms which are ARIMA, LSTM, SMA, and Holt-winters, together with providing some datasets with several ranges of time frame to check whether longer or shorter time frame will affect the models accuracy. Therefore, we also compare 4 of these models and choose the one with the best accuracy in stock price prediction.

2.8 A preview of the main results and state the contribution of the work

Main results

The accuracy of each model will vary depending on the time frame of the dataset, as we discovered after utilising the four different algorithms ARIMA, LSTM, SMA, and Holt-winters to train our models with various time frames. Additionally, the accuracy of the model may change not only due to the different time frame of the dataset used. We had also discovered that the accuracy of the model may change for a number of reasons, including a different train-test split ratio, a different random state, selecting the incorrect variable to train the model, and others.

Arima Model – We had concluded that ARIMA model trained with a 10-year dataset has the highest accuracy because its MAPE is smaller than the MAPE value of ARIMA model trained with a 15-year dataset.

LSTM Model – We had concluded that LSTM models performed well in producing a prediction on the training set, with an MAPE value of 4.4 for the 10-year dataset, and 6.62 for the 15-year dataset.

Holt-winters Model – Holt-winters model can be broken down into two models, Holt-winters Additive and Holt-winters Multiplicative. We had concluded that Holt-winters Additive model trained with a 2-year dataset has the highest accuracy because its MAPE is smaller than the MAPE value of other Holt-winters models trained with different years of dataset.

SMA Model – We had concluded that SMA20 which has the least rolling window, has 2.69 of MAPE accuracy for 10 years of data and 3.35 for 15 years of data, thus using SMA20 in shorter timeframe will get the better accuracy then SMA20 in 15 year timeframe.

Contribution of work

Arima Model – We had discovered that using a different train-test split ratio – 70% for training and 30% for testing – using a different random state, and selecting the incorrect feature for the model’s training might lower the model’s accuracy.

LSTM Model – We had discovered that when we tested the LSTM on the respective testing sets, it escalated up to 22.75 and 29.61, respectively. This indicates that the LSTM models are less effective in extrapolation, but has the capability to excel in interpolation. In order to corroborate this, we trained another model based on the 2005-1-6 until 2015-1-30 data with 100 epochs. We chose 2015-1-30 as our end date because the range of data becomes too extensive, and the testing set may fall outside the training set range. This resulted in the MAPE in testing value to plummets down to mere 6.48 which reinforced our previous assertion that LSTM is better suited in interpolation, rather than extrapolation.

Holt-winters Model – We had discovered that in three distinct dataset ages – 2, 10, and 15 years – Holt-winters Additive model tends to have a higher accuracy than Holt-winters Multiplicative model. Additionally, we had discovered that using a different train-test split ratio – 80% for training and 20% for testing – using a different random state, using different count of predictions, and selecting the incorrect feature for the model’s training might lower the model’s accuracy. It will be difficult to train an accurate model using Holt-winters model since there are several hyperparameters and parameters to be specify and select.

SMA Model – We had discovered that the test set is actually checking 20% of the lower years of data and not 20% of upper years, so to provide another test set that utilize upper 20% of data, we use ascending method in the data, and after converting the data to ascending, we discovered that the SMA20 MAPE for 10 years is 2.53, which is more accurate than the 2.69, and for 15 years of data is 3.0 which also higher than 3.35, so we have proved that using ascending data can impact the accuracies of SMA20.

3.0 Methodology

Refer to the appendix for the codes

3.1 Arima Model

3.1.1 Data cleaning and preparation

Step 1: Import libraries needed

Step 2: Import dataset. There will be two datasets with different time frame needed, 10 and 15 years

Step 3: Check the data types in the dataset

Step 4: Get the info of the dataset

Step 5: Convert the data types into appropriate data types

Step 6: Check for NULL values

Step 7: Replace the NULL values with mean

Step 8: Check outliers using z score

Step 9: Removing outliers

Step 10: Plot the graph and density plot for price

Step 11: Plot ACF and PACF to determine the parameter q and p

Step 12: Split the dataset with different ratio – 70% for training set and 30% for testing set

Step 13: Feature selection using SelectKBest and crosstab function

Step 14: Selecting feature based on the result generated by SelectKBest and crosstab function

3.1.2 Modelling

Train Model For 15 Years Dataset

Step 1: Repeat from step 1 to step 14 in the Data cleaning and preparation steps

Step 2: Perform Dickey–Fuller test to determine the stationarity of our data. If the p-value is lesser than 0.05, it indicates that our data is stationary. Else if the p-value is greater than 0.05, it indicates that our data is not stationary

Step 3: Use auto_arima to determine our p, d, and q parameter

Step 4: Train our model with the independent variable chose by the selection algorithm and the order determined by auto_arima

Step 5: Making predictions based on the trained model

Train Model For 10 Years Dataset

Step 1: Repeat from step 1 to step 14 in the Data cleaning and preparation steps

Step 2: Perform Dickey–Fuller test to determine the stationarity of our data. If the p-value is lesser than 0.05, it indicates that our data is stationary. Else if the p-value is greater than 0.05, it indicates that our data is not stationary

Step 3: Seasonal decompose our data to view the seasonality, trend, and residual

Step 4: Drop seasonality from our data since ARIMA does not support seasonal data

Step 5: Check for p-value after dropping seasonality

Step 6: Use auto_arima to determine our p, d, and q parameter

Step 7: Train our model with the independent variable chose by the selection algorithm and the order determined by auto_arima

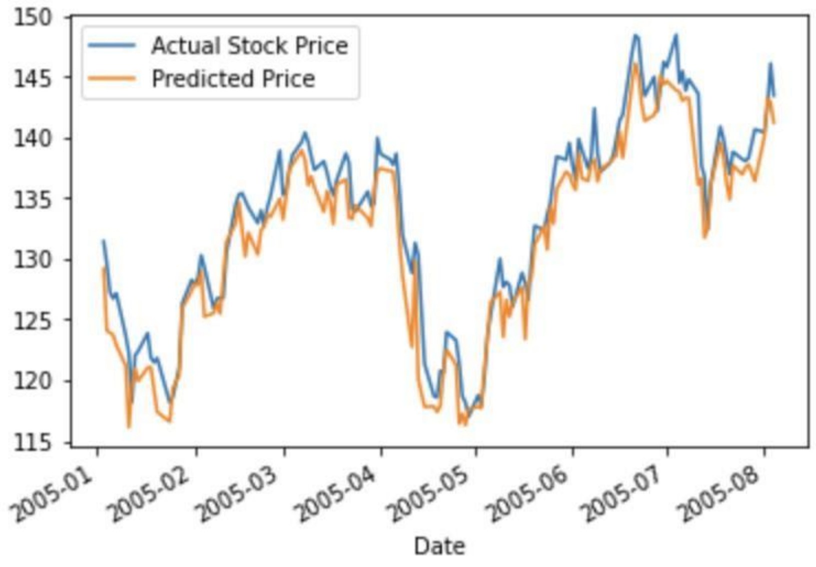
Step 8: Making predictions based on the trained model

3.1.3 Evaluation

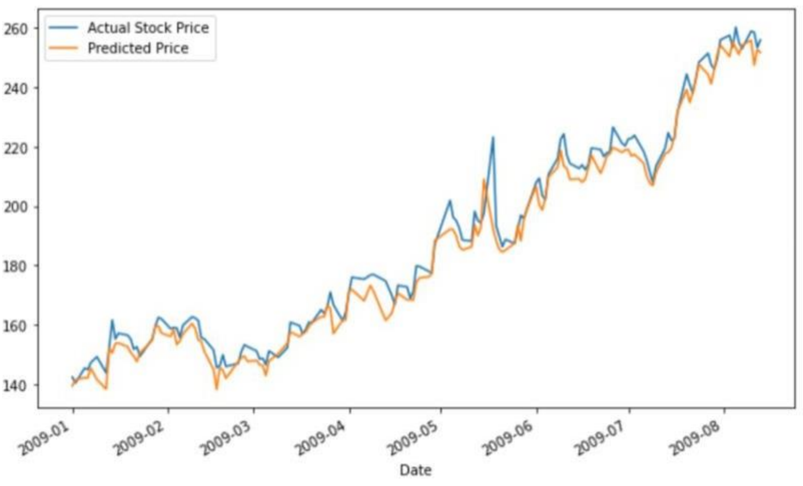
Evaluating Trained Model Performance With 15 and 10 Years Dataset

Step 1: Plotting actual vs predicted stock price graph

15 Years Result



10 Years Result



Step 2: Calculating mape, mae, mse, and rmse value to evaluate our model performance

15 Years Result

mape = 1.5268174983370786
mae = 3.057579650722498
mse = 19.879037505807272
rmse = 4.4585914262025925

10 Years Result

mape = 1.3824412972778934
mae = 3.862040943935004
mse = 25.968461809252496
rmse = 5.095926001155481

3.2 LSTM Model

3.2.1 Data cleaning and preparation

Step 1: Import libraries needed

Step 2: Import dataset. There will be two datasets with different time frame needed, 10 and 15 years

Step 3: Re-sort the data to show the oldest first

Step 4: Set the date into appropriate format

Step 5: Check the data types in the dataset

Step 6: Get the info of the dataset

Step 7: Convert the data types into appropriate data types

Step 8: Check for NULL values

Step 9: Replace the NULL values with mean

Step 10: Check outliers using z score

Step 11: Removing outliers

Step 12: Feature selection using Random Forest Algorithm and Random Forest Classifier

Step 13: Keep only the feature selected by Random Forest Algorithm and Random Forest Classifier in the data

Step 14: Define a function to convert string to datetime.datetime object with year, month, and day values

Step 15: Define a function to produce the windowed data

Step 16: Define the start date, finish date, and call the function data_to_windowed_data

Step 17: Assign windowed_data to windowed_data_to_date_X_y
Note: “High” can be ignored as it will not be used in the modelling

Step 18: Split the dataset with different ratio – 80% for training set, 10% for testing set, and 10% for validation set

3.2.2 Modelling

Train Model For 15 Years and 10 Years Dataset

Step 1: Import libraries needed

Step 2: Train our model with different parameters

Step 3: Fit our model trained with X_train, y_train, epochs, batch_size, and shuffle

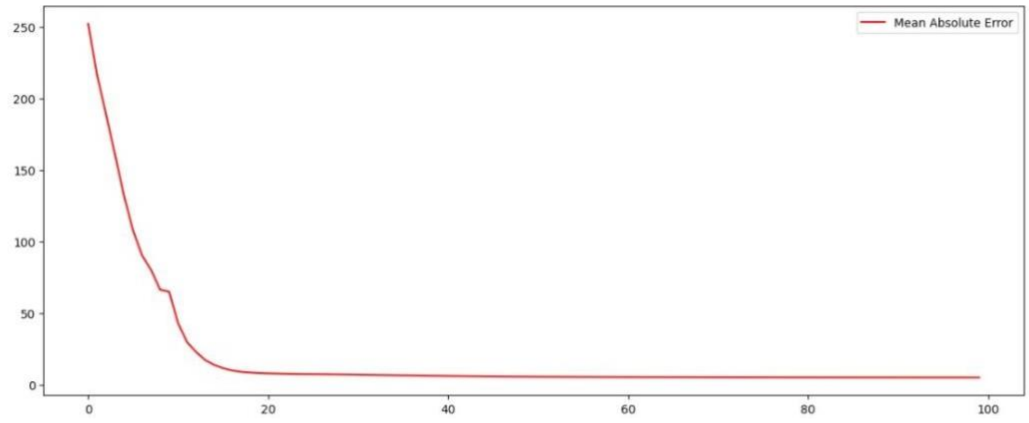
---

3.2.3 Evaluation

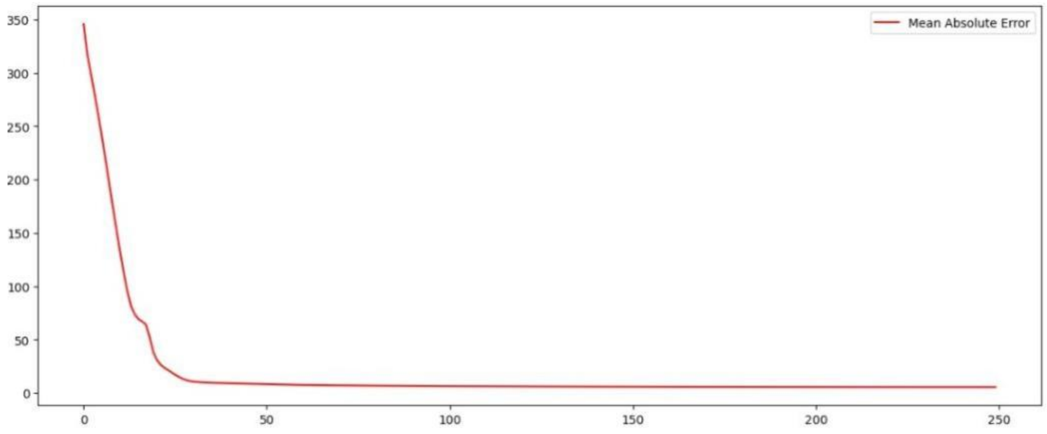
Evaluating Trained Model Performance With 15 and 10 Years Dataset

Step 1: Illustrate the mean absolute error recorded during the training

15 Years Result



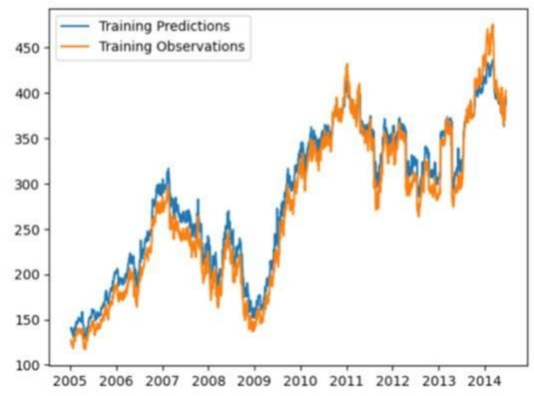
10 Years Result



Step 2: Calculating mape, mae, mse, and rmse value for the training and predicted training data

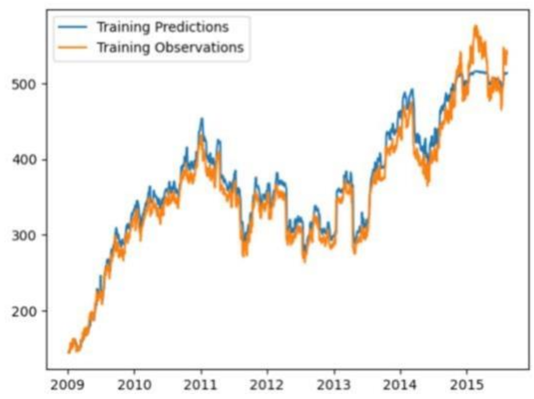
15 Years Result

```
MAPE: 6.5162986516952515
MSE: 296.17767
MAE: 15.409028
RMSE: 17.209814
```



10 Years Result

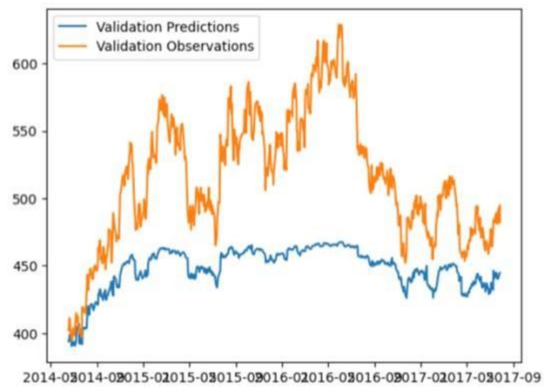
```
MAPE: 0.042587973
MSE: 335.7318
MAE: 15.41715
RMSE: 18.322987
```



Step 3: Calculating mape, mae, mse, and rmse value for the validation and predicted validation data

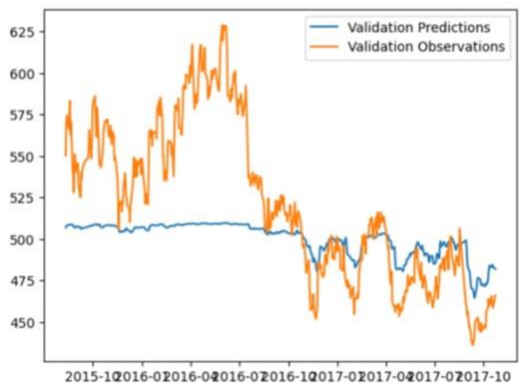
15 Years Result

```
MAPE: 12.54294067621231
MSE: 5676.9834
MAE: 67.16555
RMSE: 75.34576
```



10 Years Result

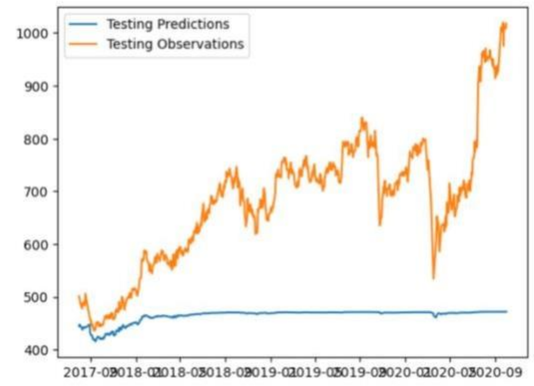
```
MAPE: 6.225723400712013
MSE: 1978.2749
MAE: 34.153286
RMSE: 44.477802
```



Step 4: Calculating mape, mae, mse, and rmse value for the testing and predicted testing data

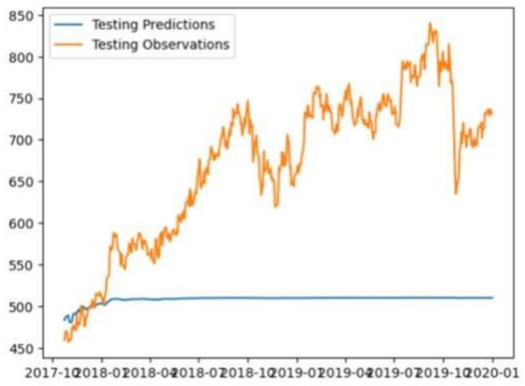
15 Years Result

```
MAPE: 29.832524061203003
MSE: 61085.934
MAE: 218.2509
RMSE: 247.15569
```



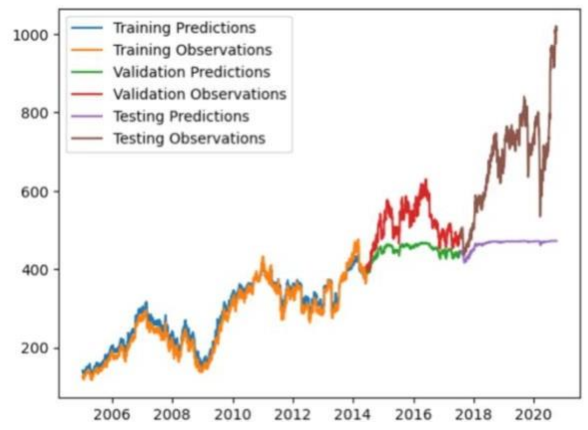
10 Years Result

```
MAPE: 22.75194227695465
MSE: 33314.54
MAE: 161.45644
RMSE: 182.5227
```

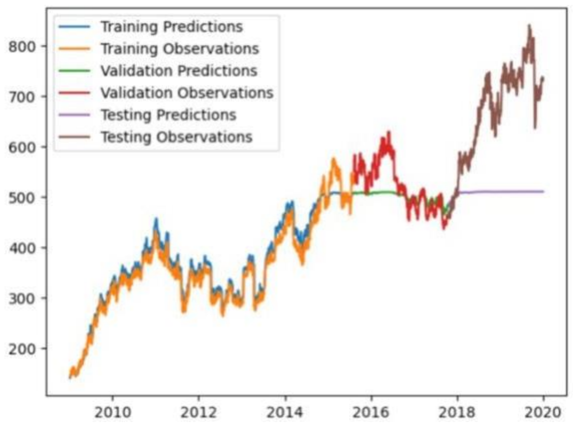


Step 5: Plot all the real and predicted data

15 Years Result



10 Years Result



3.3 Holt-winters Model

3.3.1 Data cleaning and preparation

Step 1: Import libraries needed

Step 2: Import dataset. There will be three datasets with different time frame needed, 2, 10 and 15 years

Step 3: Check the data types in the dataset

Step 4: Get the info of the dataset

Step 5: Convert the data types into appropriate data types

Step 6: Check for NULL values

Step 7: Replace the NULL values with mean

Step 8: Check outliers using z score

Step 9: Removing outliers

Step 10: Plot the graph and density plot for price

Step 11: Split the dataset with different ratio – 80% for training set and 20% for testing set

Step 12: Feature selection using SelectKBest and crosstab function

Step 13: Selecting feature based on the result generated by SelectKBest and crosstab function

Step 14: Group data by month

3.3.2 Modelling

Train Model For 15 Years Dataset

Step 1: Repeat from step 1 to step 14 in the Data cleaning and preparation steps

Step 2: Perform seasonal decompose

Step 3: Train our model with the independent variable, trend, and seasonal chose

Step 4: Making predictions based on the trained model

Train Model For 10 Years Dataset

Step 1: Repeat from step 1 to step 14 in the Data cleaning and preparation steps

Step 2: Perform seasonal decompose

Step 3: Train our model with the independent variable, trend, and seasonal chose

Step 4: Making predictions based on the trained model

Train Model For 2 Years Dataset

Step 1: Repeat from step 1 to step 14 in the Data cleaning and preparation steps

Step 2: Perform seasonal decompose

Step 3: Train our model with the independent variable, trend, and seasonal chose

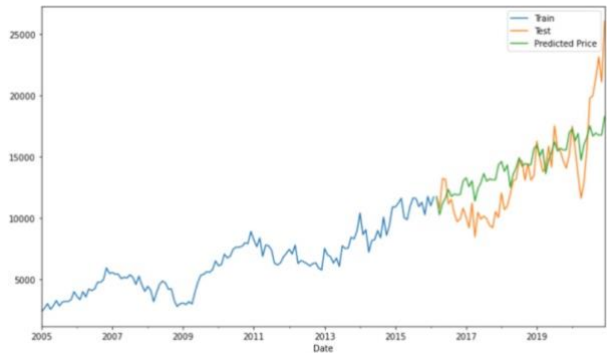
Step 4: Making predictions based on the trained model

3.3.3 Evaluation

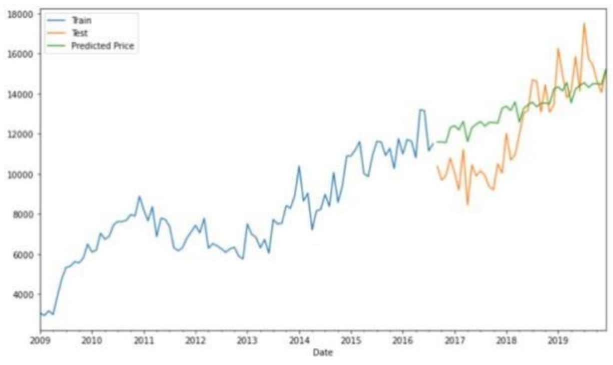
Evaluating Trained Model Performance With 15, 10, and 2 Years Dataset

Step 1: Plotting actual vs predicted stock price graph

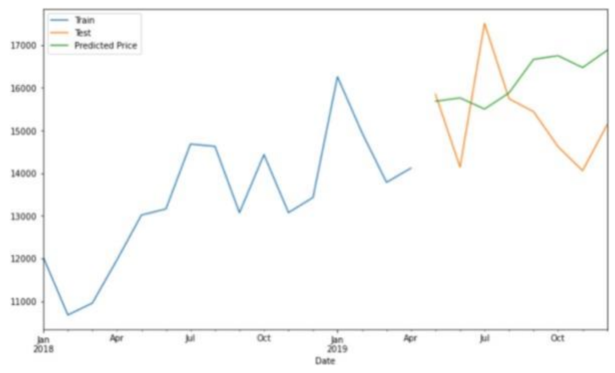
15 Years Result



10 Years Result



2 Years Result



Step 2: Calculating mape, mae, mse, and rmse value to evaluate our model performance

15 Years Result

mape = 14.834549922411117
mae = 2010.2546658826177
mse = 7576030.861757702
rmse = 2752.4590572354937

10 Years Result

mape = 14.019705783276692
mae = 1545.1651757634138
mse = 3366428.911640145
rmse = 1834.783069368187

2 Years Result

mape = 8.058564286425598
mae = 1226.380317095024
mse = 1960469.1109908004
rmse = 1400.1675296159387

3.4 SMA Model

3.4.1 Data cleaning and preparation

Step 1: Import libraries needed

Step 2: Import dataset. There will be three datasets with different time frame needed, 10 and 15 years

Step 3: Check the data types in the dataset

Step 4: Get the info of the dataset

Step 5: Convert the data types into appropriate data types

Step 6: Check for NULL values

Step 7: Replace the NULL values with mean

Step 8: Check outliers using z score

Step 9: Removing outliers

Step 10: Plot the graph for price, low, and high to see the differences between them

Step 11: Feature selection using Recursive Feature Elimination function

Step 12: Split the dataset with different ratio – 80% for training set and 20% for testing set

---

3.4.2 Modelling

Train Model For 15 Years and 10 Years Dataset

Step 1: Train our model with the independent variable chose

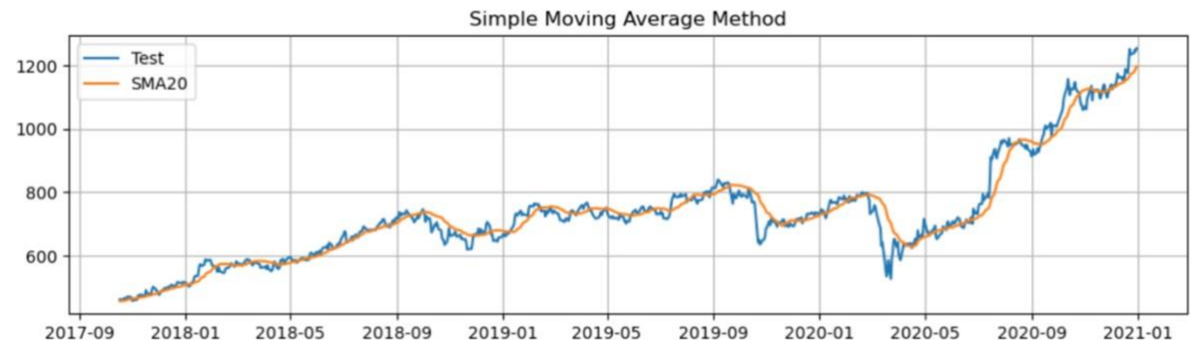
3.4.3 Evaluation

Evaluating Trained Model Performance With 15 and 10 Years Dataset

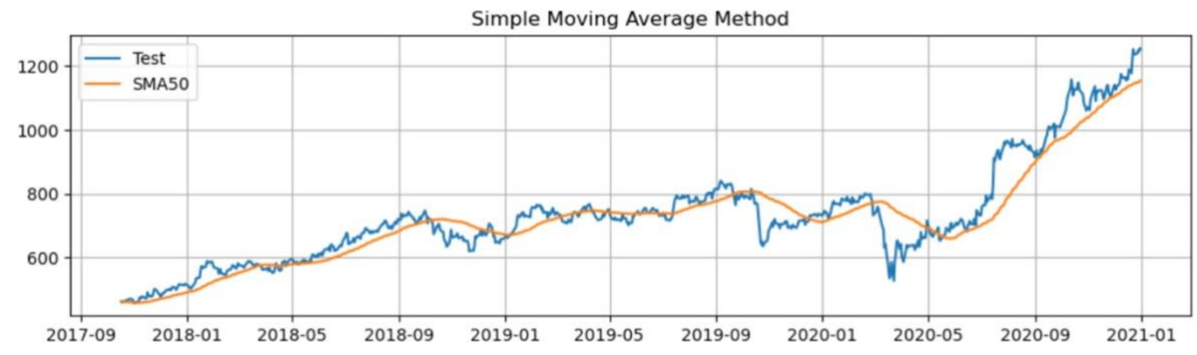
Step 1: Plot the figure to compare price with either SMA20, SMA50, or SMA200
Note: If you want to change to SMA50, the [‘SMA20’] in the code provided in the appendix must be changed to [‘SMA50’] and the same rule applied to SMA200

15 Years Result

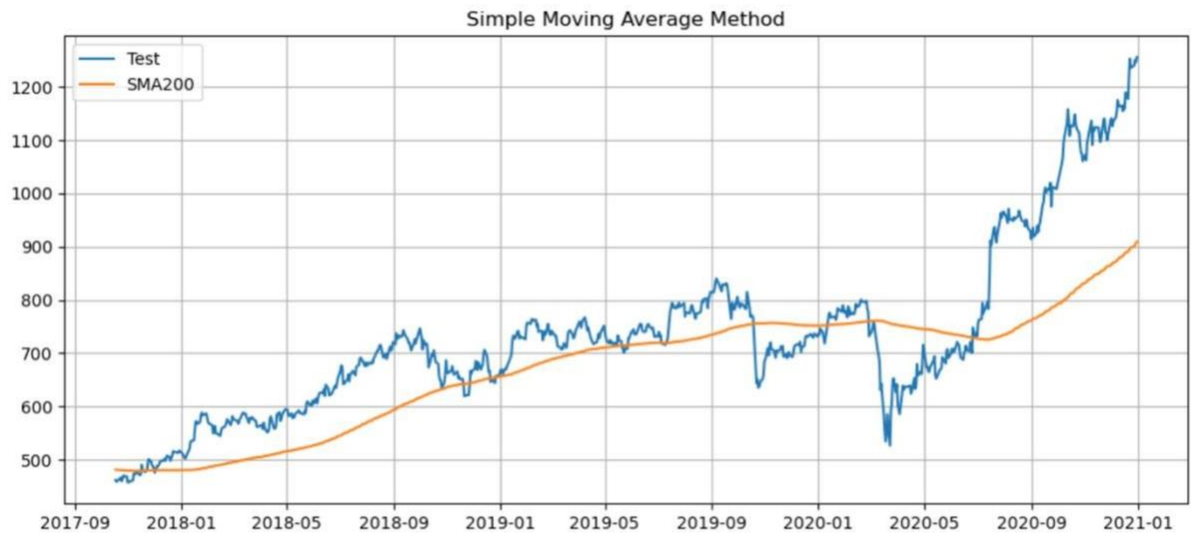
SMA20



SMA50

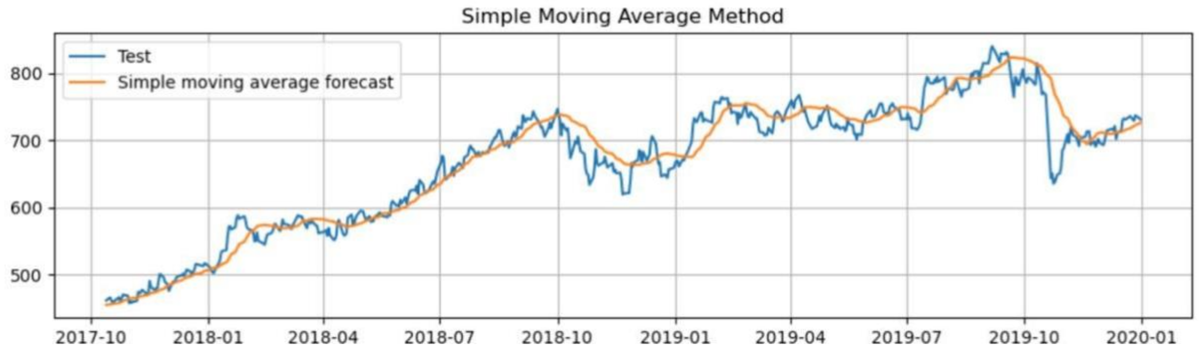


SMA200

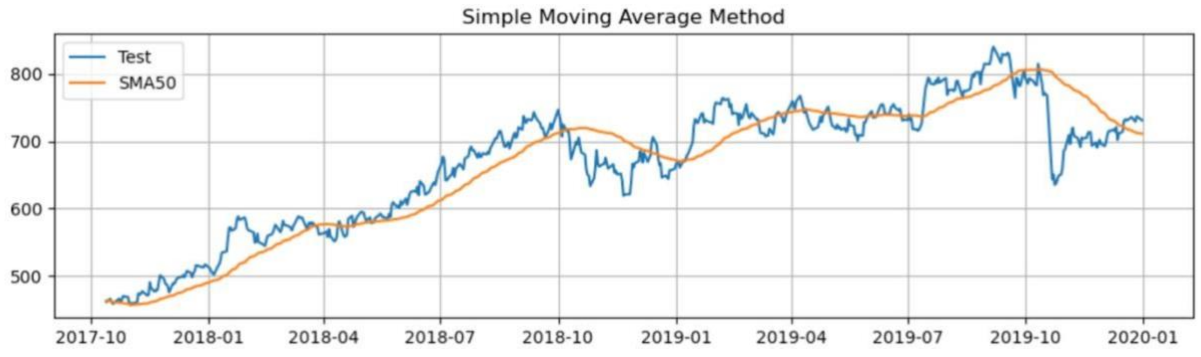


10 Years Result

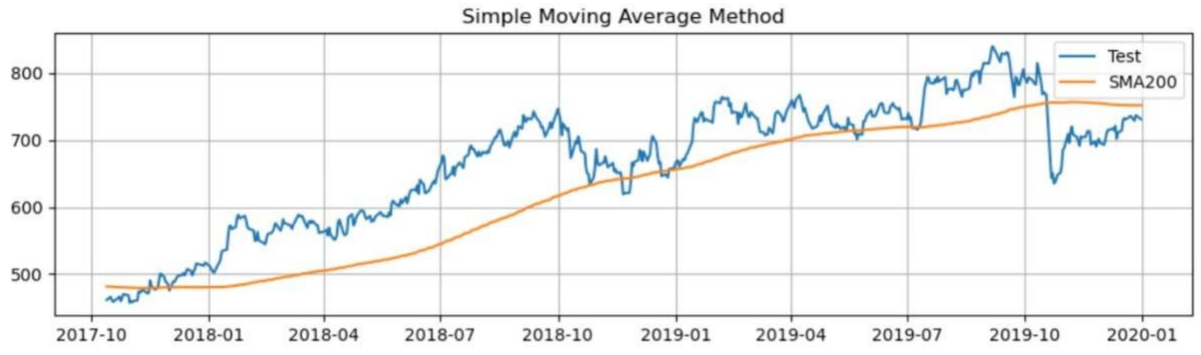
SMA20



SMA50



SMA200



Step 2: Calculating mape, mae, mse, and rmse value to evaluate our model performance
Note: Repeat this step with SMA50 and SMA200 to evaluate the performance

Table 1.0: MAPE, MAE, MSE, and RMSE Values

	Time Frame							
	15 years				**10 years**			
	Evaluation Metrics				**Evaluation Metrics**			
Model	MAPE	MAE	MSE	RMSE	MAPE	MAE	MSE	RMSE
SMA20	3.0	22.0	1175.0	34.27	2.53	17.0	599.0	24.47
SMA50	5.23	39.0	2979.0	54.58	4.12	27.0	1285.0	35.84
SMA200	10.51	83.0	13128.0	114.58	8.22	55.0	4231.0	65.04

4.0 Results

Table 2.0: Main Results For Different Models With 15 Years Dataset

	Time Frame			
	15 Years			
	Evaluation Metrics			
Model	MAPE	MAE	MSE	RMSE
ARIMA	1.53	3.06	19.88	4.46
LSTM	29.83	61085.93	218.25	247.16
SMA20	3.0	22.0	1175.0	34.27
SMA50	5.23	39.0	2979.0	54.58
SMA200	10.51	83.0	13128.0	114.58
HW Additive	14.83	2010.25	7576030.86	2752.46
HW Multiplicative	42.79	5611.45	37901409.86	6156.41

Table 3.0: Main Results For Different Models With 10 Years Dataset

	Time Frame			
	10 Years			
	Evaluation Metrics			
Model	MAPE	MAE	MSE	RMSE
ARIMA	1.38	3.86	25.97	5.10
LSTM	22.75	33314.54	161.46	182.52
SMA20	2.53	17.0	599.0	24.47
SMA50	4.12	27.0	1285.0	35.84
SMA200	8.22	55.0	4231.0	65.04
HW Additive	14.02	1545.17	3366428.91	1834.78
HW Multiplicative	19.33	2160.06	6023625.24	2454.31

Table 4.0: Main Results For Different Models With Different Years Dataset

	Time Frame			
	Different Years			
	Evaluation Metrics			
Model	MAPE	MAE	MSE	RMSE
LSTM (2005/1/6–2015/1/30)	6.48	21.311	506.417	22.5
HW Additive (2 Years)	8.06	1226.38	1960469.11	1400.17
HW Multiplicative (2 Years)	9.51	1431.06	2703504.36	1644.23

The Evaluation Metrics Used Are:

- MAPE: Mean Absolute Percentage Error
- MAE: Mean Absolute Error
- MSE: Mean Squared Error
- RMSE: Root Mean Squared Error

5.0 Discussion

5.1 Briefly describe the broad research area and narrow it down to your particular focus

Based on the MAPE, RMSE, MAE, and the MSE in table 1.0 and table 2.0 in the previous section, our ARIMA model with the 10-year dataset is superior, compared to the one with 15-year data. Other than that, we concluded that a train-test split ratio of 7:3 is the most superior ratio. The feature selected is “Low” based on the SelectKBest and Crosstab. Using different random states, and selecting another feature might lower the accuracy.

Meanwhile, in our LSTM model, we implemented both Random Forest Classifier and Regressor to accomplish feature selection, with both of these algorithms selecting the “High” as the output. From various experiments, we determined that the optimizer “Adam” with the learning rate value of 0.00001 and the “linear” activation function. These parameters have been picked as the others show inferior performance. Concerning the LSTM characteristics, based on the 3 models created—10 years, 15 years, 2005-2015—we concluded that LSTM is suited more to interpolation, rather than extrapolation.

Evidently, based on our results, we concluded that both additive Holt-Winters and multiplicative Holt-Winters fare better with a smaller dataset and that the latter performed worse compared to additive Holt-Winters. Comparable with ARIMA, in Holt-Winters, using different random states, and selecting another feature might lower the accuracy, and hence, it is challenging to create an accurate model as the hyperparameters and parameters which need to be selected are sensitive.

Meanwhile, for our last model—SMA, we determined that out of SMA20, SMA50, SMA200, the SMA20—with the least rolling window—has the best performance. On top of that, using a smaller dataset will produce better accuracy.

5.2 Comparison between our findings and previous studies

Looking back at the paper reviewed in Section 2.3 Literature Review, we can see that some points either support or contradict with our findings.

Referring to the first paper by Kulkarni, M., et al. [3], using 10 years of data of stock, SMA and Holt-Winters performed better than ARIMA with SMA on the top. This contravened with our results which put ARIMA at the top, followed by SMA, and Holt-Winters at the last. But again, looking at Section 2.5 regarding gaps in these studies, the authors failed to keep the p-value under 0.05 and the ARIMA model is displaying a sign of lack of data. This might be the reason why the ARIMA underperformed and made SMA came out on top.

This theme continues also with the fifth paper reviewing ARIMA and Holt-Winters [7]. The authors Mahmud, N., et. al., determined that additive Holt-Winters, ARIMA, and at last Multiplicative Holt-Winters, the best to worse in that particular order. The data utilised were only a 2-year dataset and the difference in RMSE value between ARIMA’s and additive Holt-Winters’s RMSE was a mere 3.135. Even though the finding that ARIMA is worse that additive Holt-Winters is contradicting with our findings, it still supports our conclusion in which additive Holt-Winters is better than multiplicative Holt-Winters.

The discrepancies between our findings and the papers does not stop there. Examining the preceding paper, we can see that in the paper by Soeryawinata, J., et al. [4], which used and compared ARIMA, LSTM, and Holt-Winters performance. The result was aligning with ours in where ARIMA is the best algorithm, but their LSTM exhibited greater accuracy than the Holt-Winters, which is different than what we got in our results.

Back to comparing both ARIMA and Holt-Winters in the research done by Dritsaki, M., et. al. [8], shows that in forecasting GDP of 5 Balkan countries, ARIMA has the best performance for 4 countries except in Romania where Holt-Winters performed better. The researchers noted only Romania data has a positive skewness. Overall, in general, this result aligned with ours.

This alignment with our findings continued to persist in the other paper by Kucĕra, J., et. al. [9], where the authors concluded that their ARIMA model outperformed the use of SMA in predicting the NASDAQ Composite Index and S&P 500 by 6.11% and 8.12%, respectively.

Another paper from Hu, Z. & Zhao, Y., et al. [10] concluded that based on 86 other publications, LSTM is one of the best algorithms with the highest accuracy reaching more than 90%—the only algorithm managed to do so. Unfortunately, this contradicts with our finding in where the LSTM model failed to extrapolate, hence, putting it at the last spot.

LSTM was also seen as promising in the third paper in Section 2.3 by Andi, H. K. [5], in which based on a 5-year dataset with 25 unique characteristics, the model managed to counteract overfitting and achieved an impressive 97.2% accuracy and 98.32% precision rate. Definitely, this statement is at odds with our own findings.

Another contradiction happened with the research done by the authors Ramadhan, R. H., et. al. [6], where the Holt-Winters was determined to be the model with the least error value, least RMSE, MSE, MAE, and MAPE. Meanwhile, in our findings, additive Holt-Winters and multiplicative Holt-Winters ranked third and fourth, respectively, with SMA sitting comfortably at the second position.

Yet another incongruity can be found in the last paper. Efrilia, I. conducted a study in Jakarta which [11] revealed that the additive Holt-Winters model outperforms ARIMA, with a difference of 0.03 in their mean absolute percentage error (MAPE). Meanwhile, in our study, the discrepancy between ARIMA and Holt-Winters is reasonably high.

5.3 Strengths and limitations of this study

Although this study may have contributed to the society and helped providing several additional insights into these algorithms, we do acknowledge that is by means far from perfect and there are some improvements that could be done.

Firstly, the dataset maybe too small, consisting of stocks data ranging from 2005 until 2020. In the future, a bigger dataset may be used in order to see the effects on these algorithms.

Secondly, we also failed to determine whether the model itself would work on future, real unseen data as our testing set itself may not be sufficient due to the lack of data.

5.4 Discuss any unexpected findings

In terms of using Arima algorithm, we had discovered that using different train-test split ratio and different random state used could affect our model accuracy. Hence, the random state (hyperparameter) should be set randomly according to the size of our dataset. The larger our dataset, the higher the random state. To conclude, the size of dataset is directly proportional to the random state.

Reviewing the papers in Section 2.3, lead us to believe that LSTM would be the best performing algorithm. This is surely wrong as can be seen from our findings. We did not expect that the LSTM would fail miserably at doing extrapolation and it is more suited in doing interpolation as proved with the LSTM model with the 2005-2015 dataset. Alas, due to the terrible performance in extrapolating, our LSTM model sits comfortably at the last position.

In terms of using Holt-winters algorithm, we had discovered that the ratio used in dividing our training and testing set could affect our model accuracy. 70% for training set and 30% for testing set seems to be more accurate as compare to 80% for training set and 20% for testing set. In addition, we had also found out that the numbers to be forecast will also affect our model accuracy. The lesser numbers to be forecast, the more accurate our model is. To conclude, the numbers to be forecast is indirectly proportional to the accuracy of the model.

In terms of using SMA algorithm, we had discovered that the data used for testing is taking the 20% of the lower years of data, thus we tried using ascending to reverse the timeframe and the result is tested by utilizing the 20% of the upper years of data. Surprisingly, we found out that implementing ascending data can affect the accuracy of the data. Moreover, after we compare the accuracies for those SMA algorithms in table 1.0 and table 2.0 in previous part with the new accuracies in the ascending data, the MAPE results in the ascending data for SMA20, SMA50, and SMA200 is lower than the accuracies of the SMA in the table 1.0 and table 2.0 which is better since the lower the MAPE the more accurate the model is.

6.0 Conclusion

6.1 Summary of the hypothesis and purpose of the study

The purpose why we conduct this extensive study is to deepen our understanding in 4 algorithms which are ARIMA, LSTM, SMA, and Holt-Winters, along with utilizing these algorithms with various timeframes of historical stock data to find their respective accuracies and compare which one gives the best accuracy, and what algorithms are best suited for different situations. Thus, after observing into section 2.6 in hypothesis and research question, and finished training all the algorithms in the methodology, we can conclude that LSTM, ARIMA, Holt-Winters, and SMA have higher accuracies in the shorter timeframe than using longer timeframe. Lastly, the algorithms with the best accuracy is ARIMA, followed by SMA, then Holt-Winters, and lastly LSTM.

On the other hand, for LSTM, the length of the timeframe itself does not really affect the accuracy, rather the range of the data itself is the biggest factor. If the testing set range is well outside of the training set range, it will result in the LSTM having worse accuracy.

6.2 Highlight and the significance of the study

Firstly, stocks provide investors the highest change of growth overtime, thus they are willing to stay with the stock over longer timeframe, since normally the result will be profitable. Moreover, there are not just long-term stock investors, mid-term, and short-term investors, or you can call traders are constantly buying and selling their stocks to get immediate profit rather than staying in stocks for a long time. Consequently, the downside of holding onto stocks or trading stocks frequently is investors might not get profits but instead, lose all the funds that they had put into the stocks. Thus, this research project helps to improve the chance of getting profits when buying and selling the stocks. The findings of this study are to help investors to prevent losses in stocks by incorporating machine learning algorithms such as ARIMA, SMA, LSTM, and Holt-Winters into stock price predictions. Thus, by utilizing these machine learning algorithms, we can provide predictions on how the stocks will turn out in the future by training using the historical stock data. So, investors will have a clearer image and able to choose better decisions in terms to hold, buy, or sell the stock to gain the most profit.

6.3 Discuss unanswered questions and potential future enhancements

Arima Model – According to our research questions, there aren't any unanswered questions. The future enhancement that is needed to be aware of is the time frame of the dataset chose, feature used to train the model, random state chose, and the use of different train test split ratio.

LSTM Model – Unfortunately, one of our research question could not be answered which is the effect of the length of data time range could not be fully comprehended. Rather, we concluded the data range itself holds a bigger influence on the accuracy as the LSTM model is not really suitable for extrapolating. Further investigation is needed in order to determine the relationship between the length of the data time range with the model's accuracy and also the ways to apply the knowledge to improve the model's accuracy.

Holt-winters Model – According to our research questions, there aren't any unanswered questions. The future enhancement that is needed to be aware of is the time frame of the dataset chose and the model used. A longer time frame and using Holt-winters multiplicative model will lead to a lower accuracy. In addition, the stationary of the data is not important when using Holt-winters algorithms because this algorithms are appropriate for non-stationary data.

SMA Model – According to our research questions, there aren't any unanswered questions. Moreover, several timeframes will require different periods of SMA. Since we only provide SMA20, SMA50 and SMA200, it doesn't mean that these algorithms are the best suited for the timeframe chose. Thus, due to the fact that SMA rolling window is too subjective, we will conduct trial and error to discover the most precise and appropriate SMA periods for different timeframes in the future. .

Acknowledgement

[blank]

7.0 References

[1] *A. Pawar, “Stock Market Analysis using Supervised Machine Learning,” 2019.*

[2] *K. Gülen, “Dataconomy,” 11 January 2023. [Online]. Available: https://dataconomy.com/blog/2023/01/11/stock-prediction-machine-learning/. [Accessed 15 March 2023].*

[3] *M. S. Kulkarni, A. Jadha and D. Dhingra, "Time Series Data Analysis for Stock Market Prediction," 2020.*

[4] *J. Soeryawinata, H. N. Palit and L. W. Santoso, "Sales Forecasting pada Dealer Motor X Dengan LSTM, ARIMA dan Holt-Winters Exponential Smoothing," 2022.*

[5] *H. K. Andi, "An Accurate Bitcoin Price Prediction Using Logistic Regression with LSTM Machine Model," 2021.*

[6] *R. H. Ramadhan, R. Yusman and G. T. Pranoto, "Comparison of Holt Winters and Simple Moving Average Models to Identify the Best Model for Predicting Flood Potential Based on the Normalized Difference Water Index," 2022.*

[7] *N. Mahmud, N. S. M. Pazil, H. Jamaluddin and N. A. Ali, "Prediction of Dengue Outbreak: A Comparison Between ARIMA and Holt-Winters Methods," 2021.*

[8] *M. Dritsaki and C. Dritsaki, "Comparison of the Holt-Winters Exponential Smoothing Method with ARIMA Models: Forecasting of GDP per Capita in Five Balkan Countries Members of European Union (EU) Post COVID," 2021.*

[9] *J. Kučera, E. Kalinová and L. Divoká, "Profability of Current Investments in Stock Indexes," 2022.*

[10] *Z. Hu, Y. Zhao and M. Khushi, "A Survey of Forex and Stock Price Prediction Using Deep Learning," 2021.*

[11] *I. Efrilia, "Comparison Of ARIMA And Exponential Smoothing Holt-Winters Methods For Forecasting CPI In The Tegal City, Central Java," Tegal, 2021.*

[12] *D. N. Skoff, "Exploring Potential Flaws and Dangers Involving Machine," Missouri, 2017.*

[13] *C. Janiesch, P. Zschech and K. Heinrich, "Machine Learning and Deep Learning," 2021.*

[14] *L. Zhang, J. Tan, D. Han and H. Zhu, "From machine learning to deep learning: progress in machine intelligence for rational drug discovery," 2017.*

8.0 Appendix

8.1 Arima Model

8.1.1 Data cleaning and preparation

Step 1

```
import os
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from pmdarima.arima import auto_arima
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.utils import shuffle
from scipy import stats
```

Step 2

15 Years

```
data = pd.read_csv('/Users/_fangkhai/Documents/INFY Historical Data (2005-2019).csv').fillna(0)
data["Date"] = pd.to_datetime(data.Date)
data.set_index ("Date", inplace = True)
```

10 Years

```
data = pd.read_csv('/Users/_fangkhai/Documents/INFY Historical Data (2009-2019).csv').fillna(0)
data["Date"] = pd.to_datetime(data.Date)
data.set_index ("Date", inplace = True)
```

Step 3

```
data.dtypes
```

Step 4

```
data.info()
```

Step 5

15 Years

```
data["Price"] = data["Price"].str.replace(",", "")
data["Open"] = data["Open"].str.replace(",", "")
data["High"] = data["High"].str.replace(",", "")
data["Low"] = data["Low"].str.replace(",", "")
data["Vol."] = data["Vol."].str.replace("M", "")
data["Change %"] = data["Change %"].str.replace("%", "")
```

```
data['Price'] = pd.to_numeric(data['Price'], errors='coerce')
data['Open'] = pd.to_numeric(data['Open'], errors='coerce')
data['High'] = pd.to_numeric(data['High'], errors='coerce')
data['Low'] = pd.to_numeric(data['Low'], errors='coerce')
data['Vol.'] = pd.to_numeric(data['Vol.'], errors='coerce')
data['Change %'] = pd.to_numeric(data['Change %'], errors='coerce')
```

10 Years

```
data["Vol."] = data["Vol."].str.replace("M", "")
data["Change %"] = data["Change %"].str.replace("%", "")
```

```
data['Vol.'] = pd.to_numeric(data['Vol.'], errors='coerce')
data['Change %'] = pd.to_numeric(data['Change %'], errors='coerce')
```

Step 6

```
data.isnull().sum()
```

Step 7

```
data = data.fillna(data["Vol."].mean())
```

Step 8

```
z = np.abs(stats.zscore(data))
(np.abs(stats.zscore(data)) > 3).sum()
```

Step 9

```
data_clean = data[((np.abs(stats.zscore(data)))<3).all(axis=1)]
data_clean
```

Step 10

Graph Plot

```
plt.figure(figsize=(12,7))
plt.grid(True)
plt.xlabel("Date")
plt.ylabel("Prices")
plt.plot(data["Price"])
plt.title("Infosys Share Price")
plt.show()
```

Density Plot

```
df = data["Price"]
df.plot(kind = "kde")
```

Step 11

ACF Plot – To determine the parameter q

```
plot_acf(data["Price"], lags = 35)
plt.show()
```

PACF Plot – To determine the parameter p

```
plot_pacf(data["Price"], lags = 35)
plt.show()
```

Step 12

For 15 Years

```
shuffled_data = shuffle(data, random_state = 800)

#X = Independent Variable
target_col = "Price"
X = data.drop("Price", axis = 1)

#y = Dependent Variable
y = shuffled_data[target_col]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 800)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

For 10 Years

```
shuffled_data = shuffle(data, random_state = 700)

#X = Independent Variable
target_col = "Price"
X = data.drop("Price", axis = 1)

#y = Dependent Variable
y = shuffled_data[target_col]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 700)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

Step 13

```
open_price_crosstab = pd.crosstab(data["Open"], data["Price"], margins = True)
open_price_crosstab
```

```
high_price_crosstab = pd.crosstab(data["High"], data["Price"], margins = True)
high_price_crosstab
```

```
vol_price_crosstab = pd.crosstab(data["Vol."], data["Price"], margins = True)
vol_price_crosstab
```

```
low_price_crosstab = pd.crosstab(data["Low"], data["Price"], margins = True)
low_price_crosstab
```

```
change_price_crosstab = pd.crosstab(data["Change %"], data["Price"], margins = True)
change_price_crosstab
```

Step 14

Get the same crosstab as the crosstab produced in step 13 to select our feature

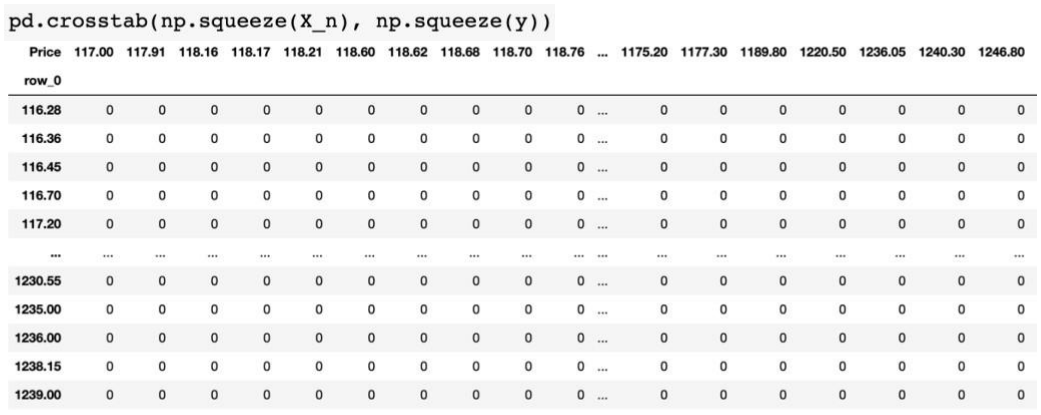
```
X_n = SelectKBest(f_classif, k = 1).fit_transform(X, y)
X_new = SelectKBest(f_classif, k = 1).fit(X_train, y_train)
X_train.columns[X_new.get_support()]
```

15 Years Result

SelectKBest Selection

```
Index(['Low'], dtype='object')
```

Crosstab Function Selection



```
pd.crosstab(np.squeeze(X_n), np.squeeze(y))
```

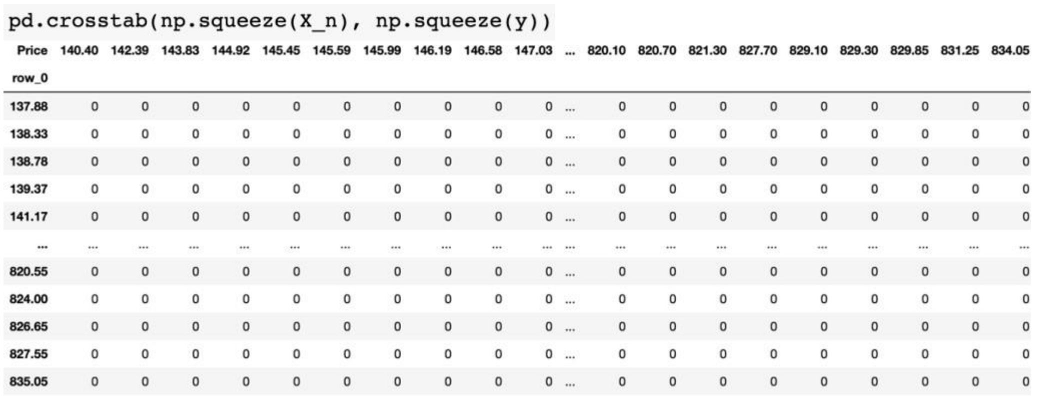
Price	117.00	117.91	118.16	118.17	118.21	118.60	118.62	118.68	118.70	118.76	...	1175.20	1177.30	1189.80	1220.50	1236.05	1240.30	1246.80
row_0																		
116.28	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
116.36	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
116.45	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
116.70	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
117.20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1230.55	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1235.00	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1236.00	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1238.15	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1239.00	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0

10 Years Result

SelectKBest Selection

```
Index(['Low'], dtype='object')
```

Crosstab Function Selection



```
pd.crosstab(np.squeeze(X_n), np.squeeze(y))
```

Price	140.40	142.39	143.83	144.92	145.45	145.59	145.99	146.19	146.58	147.03	...	820.10	820.70	821.30	827.70	829.10	829.30	829.85	831.25	834.05
row_0																				
137.88	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
138.33	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
138.78	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
139.37	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
141.17	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
820.55	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
824.00	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
826.65	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
827.55	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
835.05	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0

8.1.2 Modelling

Train Model For 15 Years Dataset

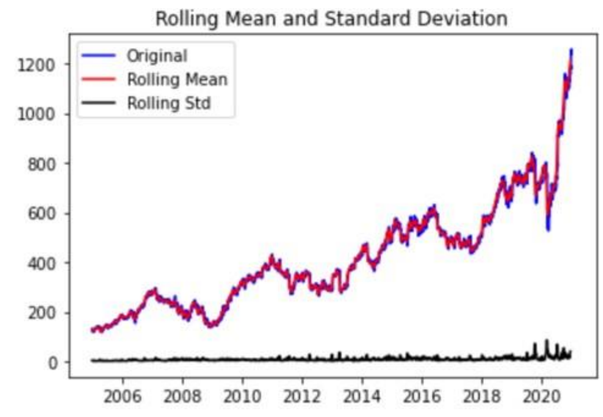
Step 2

```python
def test_stationarity(timeseries):

    #Determing rolling statistics
    rolmean = timeseries.rolling(12).mean()
    rolstd = timeseries.rolling(12).std()

    #Plot rolling statistics:
    plt.plot(timeseries, color='blue',label='Original')
    plt.plot(rolmean, color='red', label='Rolling Mean')
    plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean and Standard Deviation')
    plt.show(block=False)

    #Perform dickey fuller test
    print("Results of dickey fuller test")
    adft = adfuller(timeseries,autolag='AIC')
    output = pd.Series(adft[0:4],index=['Test Statistics','p-value','No. of lags used','Number of observations used'])
    for key,values in adft[4].items():
        output['critical value (%s)'%key] =  values
    print(output)
test_stationarity(data["Price"])
```



```
Results of dickey fuller test
Test Statistics                  -3.553030
p-value                           0.006725
No. of lags used                 26.000000
Number of observations used    3941.000000
critical value (1%)              -3.432010
critical value (5%)              -2.862274
critical value (10%)             -2.567161
dtype: float64
```

The p-value indicates that our data is stationary. Hence, we don’t have to transform our data to stationary

Step 3

```
train_size = int(len(data) * 0.8)
train, test = data.iloc[:train_size], data.iloc[train_size:]
```

```
stepwise_fit = auto_arima(data["Low"], trace=True, suppress_warning=True)
print(stepwise_fit.summary())
```

```
 Performing stepwise search to minimize aic
  ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=27680.877, Time=2.20 sec
  ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=27681.827, Time=0.05 sec
  ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=27677.039, Time=0.12 sec
  ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=27676.891, Time=0.17 sec
  ARIMA(0,1,0)(0,0,0)[0]             : AIC=27684.765, Time=0.04 sec
  ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=27677.697, Time=0.75 sec
  ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=27678.463, Time=0.46 sec
  ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=27678.945, Time=1.16 sec
  ARIMA(0,1,1)(0,0,0)[0]             : AIC=27679.440, Time=0.09 sec

 Best model:  ARIMA(0,1,1)(0,0,0)[0] intercept
 Total fit time: 5.062 seconds
                               SARIMAX Results
 ==============================================================================
 Dep. Variable:                      y   No. Observations:                 3968
 Model:               SARIMAX(0, 1, 1)   Log Likelihood              -13835.446
 Date:                Tue, 18 Apr 2023   AIC                          27676.891
 Time:                        18:11:23   BIC                          27695.749
 Sample:                             0   HQIC                         27683.579
                                - 3968
 Covariance Type:                  opg
 ==============================================================================
                  coef    std err          z      P>|z|      [0.025      0.975]
 ------------------------------------------------------------------------------
 intercept     -0.2794      0.133     -2.098      0.036      -0.540      -0.018
 ma.L1          0.0422      0.008      5.365      0.000       0.027       0.058
 sigma2        62.6366      0.357    175.691      0.000      61.938      63.335
 ===================================================================================
 Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):            156866.96
 Prob(Q):                              0.98   Prob(JB):                         0.00
 Heteroskedasticity (H):               0.18   Skew:                             1.04
 Prob(H) (two-sided):                  0.00   Kurtosis:                        33.74
 ===================================================================================
```

Step 4

```
model = sm.tsa.arima.ARIMA(data["Low"], order = (0, 1, 1))
model_fit = model.fit()
```

Step 5

```
start = len(train)
end = len(train) + len(test) - 1
predictions = model_fit.predict(start=start, end=end)
predictions
```

```
s = pd.Series(predictions, index = data.index[-150:])
s
```

Train Model For 10 Years Dataset

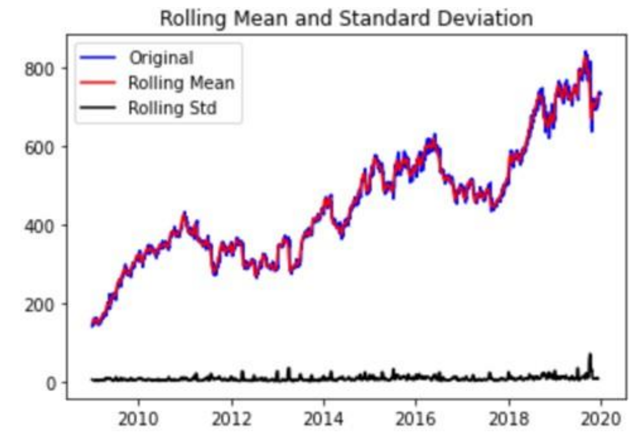
Step 2

```python
def test_stationarity(timeseries):

    #Determing rolling statistics
    rolmean = timeseries.rolling(12).mean()
    rolstd = timeseries.rolling(12).std()

    #Plot rolling statistics:
    plt.plot(timeseries, color='blue',label='Original')
    plt.plot(rolmean, color='red', label='Rolling Mean')
    plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean and Standard Deviation')
    plt.show(block=False)

    #Perform dickey fuller test
    print("Results of dickey fuller test")
    adft = adfuller(timeseries,autolag='AIC')
    output = pd.Series(adft[0:4],index=['Test Statistics','p-value','No. of lags used','Number of observations used'])
    for key,values in adft[4].items():
        output['critical value (%s)'%key] =  values
    print(output)
test_stationarity(data["Price"])
```

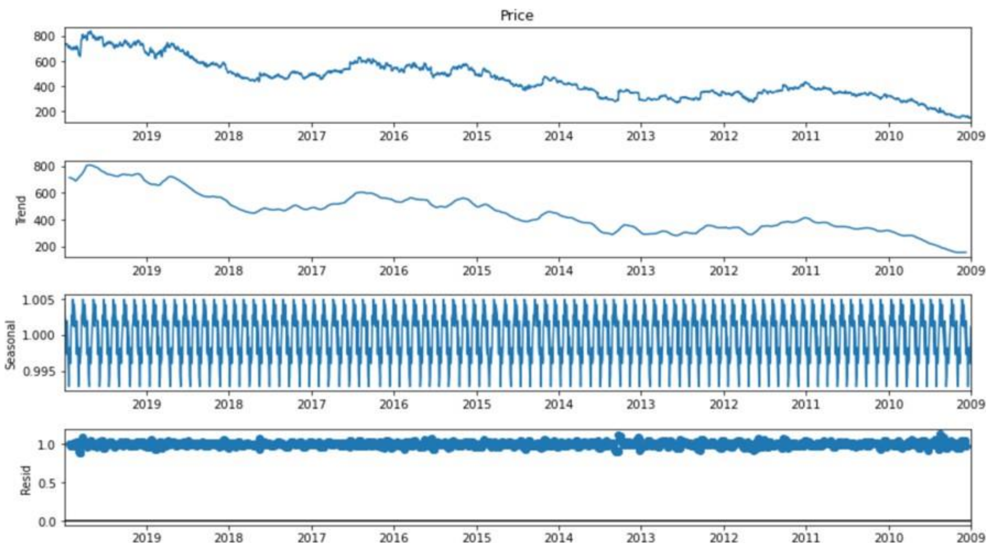


```
Results of dickey fuller test
Test Statistics                   -1.187057
p-value                            0.679139
No. of lags used                   2.000000
Number of observations used     2718.000000
critical value (1%)               -3.432758
critical value (5%)               -2.862604
critical value (10%)              -2.567336
dtype: float64
```

The p-value indicates that our data is not stationary. Hence, we have to transform our data to stationary

Step 3

```
result = seasonal_decompose(df, model='multiplicative', period = 30)
fig = plt.figure()
fig = result.plot()
fig.set_size_inches(12, 7)
```



Step 4

```
from pylab import rcParams
rcParams['figure.figsize'] = 10, 6
df_log = np.log(df)
moving_avg = df_log.rolling(12).mean()
std_dev = df_log.rolling(12).std()
plt.legend(loc='best')
plt.title('Moving Average')
plt.plot(std_dev, color ="black", label = "Standard Deviation")
plt.plot(moving_avg, color="red", label = "Mean")
plt.legend()
plt.show()
```

```
y_train_diff = y_train.diff().dropna()
y_train_diff.plot()
```

```
acf_diff = plot_acf(y_train_diff)

pacf_diff = plot_pacf(y_train_diff)
```

Step 5

```
adf_test = adfuller(y_train_diff)
print(f'p-value: {adf_test[1]}')
```

Step 6

```
train_size = int(len(data) * 0.8)
train, test = data.iloc[:train_size], data.iloc[train_size:]
```

```
stepwise_fit = auto_arima(data["Low"], trace=True, suppress_warning=True)
print(stepwise_fit.summary())
```

```
 Performing stepwise search to minimize aic
  ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=18676.082, Time=1.46 sec
  ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=18678.984, Time=0.04 sec
  ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=18678.165, Time=0.10 sec
  ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=18678.034, Time=0.16 sec
  ARIMA(0,1,0)(0,0,0)[0]             : AIC=18679.244, Time=0.04 sec
  ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=18674.081, Time=0.69 sec
  ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=18678.765, Time=0.24 sec
  ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=18679.178, Time=0.58 sec
  ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=18676.081, Time=1.11 sec
  ARIMA(0,1,3)(0,0,0)[0] intercept   : AIC=18679.437, Time=0.30 sec
  ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=18674.092, Time=1.27 sec
  ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=18677.577, Time=0.95 sec
  ARIMA(1,1,2)(0,0,0)[0]             : AIC=18675.424, Time=0.34 sec

 Best model:  ARIMA(1,1,2)(0,0,0)[0] intercept
 Total fit time: 7.302 seconds
                               SARIMAX Results
 ==============================================================================
 Dep. Variable:                      y   No. Observations:                 2721
 Model:               SARIMAX(1, 1, 2)   Log Likelihood               -9332.041
 Date:                Tue, 18 Apr 2023   AIC                          18674.081
 Time:                        20:59:23   BIC                          18703.623
 Sample:                             0   HQIC                         18684.761
                                - 2721
 Covariance Type:                  opg
 ==============================================================================
                  coef    std err          z      P>|z|      [0.025      0.975]
 ------------------------------------------------------------------------------
 intercept     -0.0213      0.018     -1.201      0.230      -0.056       0.013
 ar.L1          0.9009      0.059     15.296      0.000       0.785       1.016
 ma.L1         -0.8706      0.060    -14.489      0.000      -0.988      -0.753
 ma.L2         -0.0514      0.012     -4.306      0.000      -0.075      -0.028
 sigma2        55.9176      0.475    117.661      0.000      54.986      56.849
 ===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):            152493.60
Prob(Q):                              1.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.30   Skew:                             2.58
Prob(H) (two-sided):                  0.00   Kurtosis:                        39.32
===================================================================================
```

Step 7

```
model = sm.tsa.arima.ARIMA(data["Low"], order = (1, 1, 2))
model_fit = model.fit()
```

Step 8: Making predictions based on the trained model

```
start = len(train)
end = len(train) + len(test) - 1
predictions = model_fit.predict(start=start, end=end)
predictions
```

```
s = pd.Series(predictions, index = data.index[-150:])
s
```

8.1.3 Evaluation

Step 1

```
data["Price"][-150:].plot(label = "Actual Stock Price", legend = True)
s.plot(label = "Predicted Price", legend = True)
```

Step 2

```
mape = np.mean(np.abs((test["Price"] - predictions) / test["Price"])) * 100
mape
```

```
mae = mean_absolute_error(test["Price"], predictions)
mae
```

```
mse = mean_squared_error(test["Price"], predictions)
mse
```

```
rmse = math.sqrt(mse)
rmse
```

8.2 LSTM Model

8.2.1 Data cleaning and preparation

Step 1

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime
%matplotlib inline
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectFromModel
from sklearn import preprocessing
from scipy import stats
```

Step 2

15 and 10 Years

```
data = pd.read_csv("C:/Users/ndyla/Downloads/INFY Historical Data (1).csv")
```

Step 3

```
ori_data = data[::-1]
ori_data.head()
```

Step 4

```
def str_to_datetime1(s):
  split = s.split('/')
  month, day, year = int(split[0]), int(split[1]), int(split[2])
  return datetime.datetime(year=year, month=month, day=day)
```

```
ori_data['Date'] = ori_data['Date'].apply(str_to_datetime1)
```

```
ori_data.index = ori_data.pop('Date')
data.index=ori_data.index
```

Step 5

```
data.dtypes
```

Step 6

```
data.info()
```

Step 7

15 and 10 Years

```
ori_data["Price"] = ori_data["Price"].str.replace(",", "")
ori_data["Open"] = ori_data["Open"].str.replace(",", "")
ori_data["High"] = ori_data["High"].str.replace(",", "")
ori_data["Low"] = ori_data["Low"].str.replace(",", "")
ori_data["Vol."] = ori_data["Vol."].str.replace("M", "")
ori_data["Change %"] = ori_data["Change %"].str.replace("%", "")
```

```
ori_data['Price'] = pd.to_numeric(ori_data['Price'], errors='coerce')
ori_data['Open'] = pd.to_numeric(ori_data['Open'], errors='coerce')
ori_data['High'] = pd.to_numeric(ori_data['High'], errors='coerce')
ori_data['Low'] = pd.to_numeric(ori_data['Low'], errors='coerce')
ori_data['Vol.'] = pd.to_numeric(ori_data['Vol.'], errors='coerce')
ori_data['Change %'] = pd.to_numeric(ori_data['Change %'], errors='coerce')
```

Step 8

```
ori_data.isna().sum()
```

Step 9

```
data = ori_data.fillna(ori_data['Vol.'].mean())
```

Step 10

```
z = np.abs(stats.zscore(data))
(np.abs(stats.zscore(data)) > 3).sum()
```

Step 11

```
data_clean = data[((np.abs(stats.zscore(data)))<3).all(axis=1)]
data_clean
```

Step 12

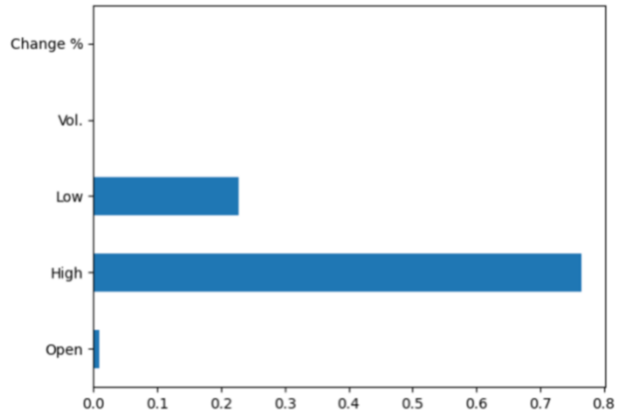
```
rf=RandomForestRegressor(n_estimators = 100, random_state=42)
rf.fit(data_clean.drop('Price', axis=1).astype('int'), data_clean.Price.astype('int'))
rf.feature_importances_
```

```
(pd.Series(rf.feature_importances_, index=data_clean.drop('Price', axis=1).columns).plot(kind='barh'))
```

```
sel = SelectFromModel(RandomForestClassifier(n_estimators = 100,random_state=42))
sel.fit(data_clean.drop('Price', axis=1).astype(int), data_clean.Price.astype(int))
selected_feat= data_clean.drop('Price', axis=1).columns[(sel.get_support())]
print(selected_feat)
```

15 and 10 Years Result

Random Forest Regressor Selection



15 and 10 Years Result

Random Forest Classifier Selection

```
Index(['Open', 'High', 'Low'], dtype='object')
```

Step 13

```
variable = 'High'
data_clean = data_clean[[variable,'Price']]
```

Step 14

```
def str_to_datetime(s):
  split = s.split('-')
  year, month, day = int(split[0]), int(split[1]), int(split[2])
  return datetime.datetime(year=year, month=month, day=day)
```

Step 15

```
def data_to_windowed_data(dataframe, first_date_str, last_date_str, n=3):
  first_date = str_to_datetime(first_date_str)
  last_date  = str_to_datetime(last_date_str)

  target_date = first_date

  dates = []
  X, Y, Z = [], [], []

  last_time = False
  while True:
    data_subset = dataframe.loc[:target_date].tail(3+1)

    if len(data_subset) != n+1:
      print(f'Error: Window of size {n} is too large for date {target_date}')
      return

    values = data_subset[variable].to_numpy()
    price = data_subset['Price'].to_numpy()
    x, y, z = values[:-1], values[-1], price[-1]

    if target_date in dataframe.index:
        dates.append(target_date)
        X.append(x)
        Y.append(y)
        Z.append(z)

    next_week = data.loc[target_date:target_date+datetime.timedelta(days=7)]
    next_datetime_str = str(next_week.head(2).tail(1).index.values[0])
    next_date_str = next_datetime_str.split('T')[0]
    year_month_day = next_date_str.split('-')
    year, month, day = year_month_day
    next_date = datetime.datetime(day=int(day), month=int(month), year=int(year))

    if last_time:
      break

    target_date = next_date

    if target_date == last_date:
      last_time = True

  ret_data = pd.DataFrame({})
  ret_data['Target Date'] = dates

  X = np.array(X)
  for i in range(0, n):
    X[:, i]
    ret_data[f'High-{n-i}'] = X[:, i]

  ret_data[variable] = Y
  ret_data['Price'] = Z
  return ret_data
```

Step 16

15 Years

```
start_date = '2005-1-7'
finish_date = '2020-10-1'
windowed_data = data_to_windowed_data(data_clean,
                                start_date,
                                finish_date,
                                n=3)
windowed_data
```

10 Years

```
start_date = '2009-1-6'
finish_date = '2019-12-31'
windowed_data = data_to_windowed_data(data_clean,
                                start_date,
                                finish_date,
                                n=3)
windowed_data
```

Step 17

```
def windowed_data_to_date_X_y(windowed_dataframe):
  data_as_np = windowed_data.to_numpy()

  dates = data_as_np[:, 0]

  middle_matrix = data_as_np[:, 1:-2]

  X = middle_matrix.reshape((len(dates), middle_matrix.shape[1], 1))

  Y = data_as_np[:, -1]

  high = data_as_np[:, -2]

  return dates, X.astype(np.float32), Y.astype(np.float32), high.astype(np.float32)

dates, X, y, high = windowed_data_to_date_X_y(windowed_data)

dates.shape, X.shape, y.shape
```

Step 18

```
q_60 = int(len(dates) * .6)
q_80 = int(len(dates) * .8)

dates_train, X_train, y_train, h_train = dates[:q_60], X[:q_60], y[:q_60], high[:q_60]

dates_val, X_val, y_val, h_val = dates[q_60:q_80], X[q_60:q_80], y[q_60:q_80], high[q_60:q_80]
dates_test, X_test, y_test, h_test = dates[q_80:], X[q_80:], y[q_80:], high[q_80:]

plt.plot(dates_train, y_train)
# plt.plot(dates_train, h_train)

plt.plot(dates_val, y_val)
# plt.plot(dates_val, h_val)

plt.plot(dates_test, y_test)
# plt.plot(dates_test, h_test)


plt.legend(['Train (80%)', 'Validation (10%)', 'Test (10%)'])
q_60
y_train.shape
```

8.2.2 Modelling

Train Model For 15 and 10 Years Dataset

Step 1

```
from tensorflow import keras
from tensorflow.keras.layers import Input, Bidirectional, Dropout, Activation, Dense, LSTM
from tensorflow.keras.models import Sequential
from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_absolute_percentage_error
```

Step 2

```
model = Sequential([Input((3,1)),
    LSTM(128,return_sequences=True),
    LSTM(128,return_sequences=False),
    Dense(units=25, activation='linear'),
    Dense(units=1)
])
```

```
from tensorflow.keras.optimizers import Adam
adamNew = Adam(learning_rate=0.00001)

from keras.optimizers import SGD
opt = SGD(learning_rate=0.0001)

model.compile(
    loss='mse',
    optimizer=adamNew,
    metrics=['mean_absolute_error']
)

model.summary()
```

Step 3

15 Years

```
model.fit(
    X_train,
    y_train,
    epochs=100,
    batch_size=8,
    shuffle=False
)
```

10 Years

```
model.fit(
    X_train,
    y_train,
    epochs=250,
    batch_size=1,
    shuffle=False
)
```

8.2.3 Evaluation

Step 1

```
plt.figure(figsize=(15, 6))
plt.plot(history.history['mean_absolute_error'], label='mae', color='red')
plt.legend()
plt.show()
```

Step 2

```
train_predictions = model.predict(X_train).flatten()

plt.plot(dates_train, train_predictions)
plt.plot(dates_train, y_train)
plt.legend(['Training Predictions', 'Training Observations'])

mape = mean_absolute_percentage_error(y_train, train_predictions)
mse = mean_squared_error(y_train, train_predictions)
mae = mean_absolute_error(y_train, train_predictions)
rmse = np.sqrt(mse)

print("MAPE:", mape)
print("MSE:", mse)
print("MAE:", mae)
print("RMSE:", rmse)
```

Step 3

```
val_predictions = model.predict(X_val).flatten()

plt.plot(dates_val, val_predictions)
plt.plot(dates_val, y_val)
plt.legend(['Validation Predictions', 'Validation Observations'])

mape = mean_absolute_percentage_error(y_val, val_predictions)*100
mse = mean_squared_error(y_val, val_predictions)
mae = mean_absolute_error(y_val, val_predictions)
rmse = np.sqrt(mse)

print("MAPE:", mape)
print("MSE:", mse)
print("MAE:", mae)
print("RMSE:", rmse)
```

Step 4

```
test_predictions = model.predict(X_test).flatten()

plt.plot(dates_test, test_predictions)
plt.plot(dates_test, y_test)
plt.legend(['Testing Predictions', 'Testing Observations'])

mape = mean_absolute_percentage_error(y_test, test_predictions)*100
mse = mean_squared_error(y_test, test_predictions)
mae = mean_absolute_error(y_test, test_predictions)
rmse = np.sqrt(mse)

print("MAPE:", mape)
print("MSE:", mse)
print("MAE:", mae)
print("RMSE:", rmse)
```

Step 5

```
plt.plot(dates_train, train_predictions)
plt.plot(dates_train, y_train)
plt.plot(dates_val, val_predictions)
plt.plot(dates_val, y_val)
plt.plot(dates_test, test_predictions)
plt.plot(dates_test, y_test)
plt.legend(['Training Predictions',
            'Training Observations',
            'Validation Predictions',
            'Validation Observations',
            'Testing Predictions',
            'Testing Observations'])
```

8.3 Holt-winters Model

8.3.1 Data cleaning and preparation

Step 1

```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_absolute_percentage_error
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.utils import shuffle
from scipy import stats
```

Step 2

15 Years

```
data = pd.read_csv('/Users/_fangkhai/Documents/INFY Historical Data (2005-2019).csv').fillna(0)
```

```
data["Date"] = pd.to_datetime(data["Date"])
data.set_index("Date")
```

10 Years

```
data = pd.read_csv('/Users/_fangkhai/Documents/INFY Historical Data (2009-2019).csv').fillna(0)
```

```
data["Date"] = pd.to_datetime(data["Date"])
data.set_index("Date")
```

2 Years

```
data = pd.read_csv('/Users/_fangkhai/Documents/INFY Historical Data (2005-2019).csv').fillna(0)
```

```
data["Date"] = pd.to_datetime(data["Date"])
data.set_index("Date")
data = data[data["Date"].between('2017-12-31', '2019-12-31')]
```

Step 3

```
data.dtypes
```

Step 4

```
data.info()
```

Step 5

15 Years

```
data["Price"] = data["Price"].str.replace(",", "")
data["Open"] = data["Open"].str.replace(",", "")
data["High"] = data["High"].str.replace(",", "")
data["Low"] = data["Low"].str.replace(",", "")
data["Vol."] = data["Vol."].str.replace("M", "")
data["Change %"] = data["Change %"].str.replace("%", "")
```

```
data['Price'] = pd.to_numeric(data['Price'], errors='coerce')
data['Open'] = pd.to_numeric(data['Open'], errors='coerce')
data['High'] = pd.to_numeric(data['High'], errors='coerce')
data['Low'] = pd.to_numeric(data['Low'], errors='coerce')
data['Vol.'] = pd.to_numeric(data['Vol.'], errors='coerce')
data['Change %'] = pd.to_numeric(data['Change %'], errors='coerce')
```

10 Years and 2 Years

```
data["Vol."] = data["Vol."].str.replace("M", "")
data["Change %"] = data["Change %"].str.replace("%", "")
```

```
data['Vol.'] = pd.to_numeric(data['Vol.'], errors='coerce')
data['Change %'] = pd.to_numeric(data['Change %'], errors='coerce')
```

Step 6

```
data.isnull().sum()
```

Step 7

```
data = data.fillna(data["Vol."].mean())
```

Step 8

```
z = np.abs(stats.zscore(data))
(np.abs(stats.zscore(data)) > 3).sum()
```

Step 9

```
data_clean = data[((np.abs(stats.zscore(data)))<3).all(axis=1)]
data_clean
```

Step 10

Graph Plot

```
plt.figure(figsize=(12,7))
plt.grid(True)
plt.xlabel("Date")
plt.ylabel("Prices")
plt.plot(data["Price"])
plt.title("Infosys Share Price")
plt.show()
```

Density Plot

```
df = data["Price"]
df.plot(kind = "kde")
```

Step 11

For 15 Years

```
shuffled_data = shuffle(data, random_state = 800)

#X = Independent Variable
target_col = "Price"
X = data.drop("Price", axis = 1)

#y = Dependent Variable
y = shuffled_data[target_col]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 800)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

For 10 Years

```
shuffled_data = shuffle(data, random_state = 700)

#X = Independent Variable
target_col = "Price"
X = data.drop("Price", axis = 1)

#y = Dependent Variable
y = shuffled_data[target_col]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 700)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

For 2 Years

```
shuffled_data = shuffle(data, random_state = 100)

#X = Independent Variable
target_col = "Price"
X = data.drop("Price", axis = 1)

#y = Dependent Variable
y = shuffled_data[target_col]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 100)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

Step 12

```
open_price_crosstab = pd.crosstab(data["Open"], data["Price"], margins = True)
open_price_crosstab
```

```
high_price_crosstab = pd.crosstab(data["High"], data["Price"], margins = True)
high_price_crosstab
```

```
vol_price_crosstab = pd.crosstab(data["Vol."], data["Price"], margins = True)
vol_price_crosstab
```

```
low_price_crosstab = pd.crosstab(data["Low"], data["Price"], margins = True)
low_price_crosstab
```

```
change_price_crosstab = pd.crosstab(data["Change %"], data["Price"], margins = True)
change_price_crosstab
```

Step 13
Get the same crosstab as the crosstab produced in step 12 to select our feature

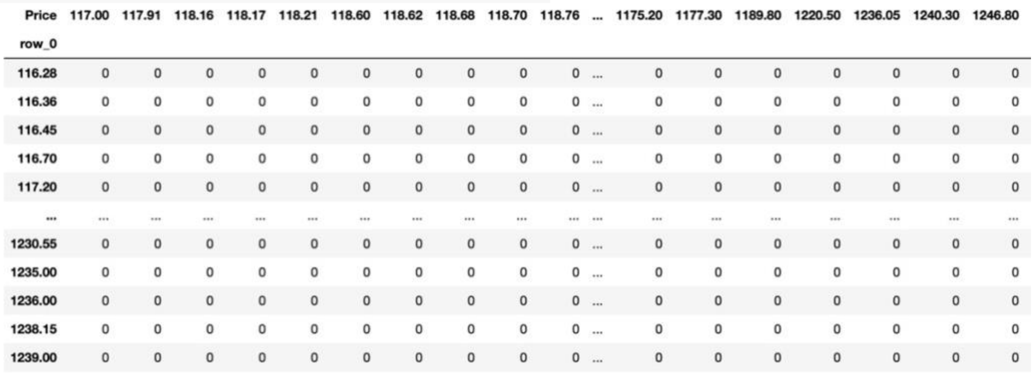
```
X_n = SelectKBest(f_classif, k = 1).fit_transform(X, y)
X_new = SelectKBest(f_classif, k = 1).fit(X_train, y_train)
X_train.columns[X_new.get_support()]
```

15 Years Result
SelectKBest Selection

```
Index(['Low'], dtype='object')
```

Crosstab Function Selection

```
pd.crosstab(np.squeeze(X_n), np.squeeze(y))
```



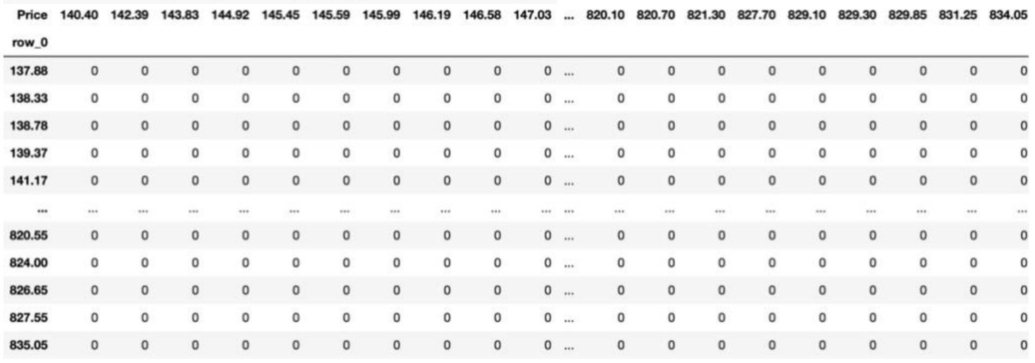
Price	117.00	117.91	118.16	118.17	118.21	118.60	118.62	118.68	118.70	118.76	...	1175.20	1177.30	1189.80	1220.50	1236.05	1240.30	1246.80
row_0																		
116.28	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
116.36	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
116.45	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
116.70	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
117.20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1230.55	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1235.00	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1236.00	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1238.15	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
1239.00	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0

10 Years Result
SelectKBest Selection

```
Index(['Low'], dtype='object')
```

Crosstab Function Selection

```
pd.crosstab(np.squeeze(X_n), np.squeeze(y))
```



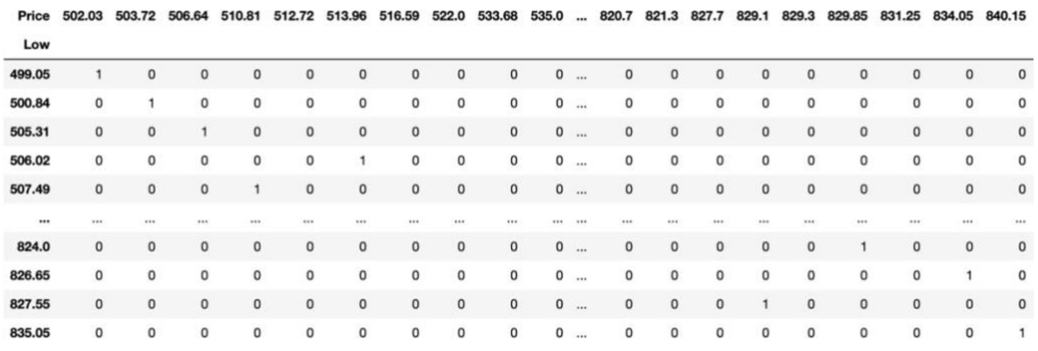
Price	140.40	142.39	143.83	144.92	145.45	145.59	145.99	146.19	146.58	147.03	...	820.10	820.70	821.30	827.70	829.10	829.30	829.85	831.25	834.05
row_0																				
137.88	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
138.33	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
138.78	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
139.37	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
141.17	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
820.55	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
824.00	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
826.65	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
827.55	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
835.05	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0

2 Years Result

SelectKBest Selection

```
Index(['Low'], dtype='object')
```

Crosstab Function Selection



Price	502.03	503.72	506.64	510.81	512.72	513.96	516.59	522.0	533.68	535.0	...	820.7	821.3	827.7	829.1	829.3	829.85	831.25	834.05	840.15
Low																				
499.05	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
500.84	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
505.31	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
506.02	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0
507.49	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
824.0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	1	0	0	0
826.65	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0
827.55	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0
835.05	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	1

Step 14

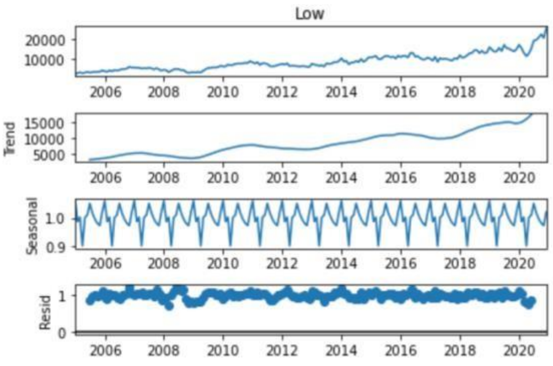
```
data = data.resample(rule = 'MS').sum()
```

8.3.2 Modelling

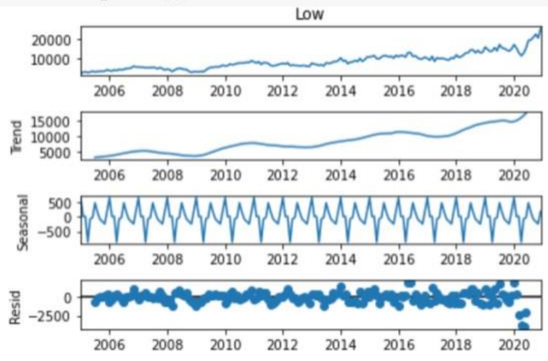
Train Model For 15 Years Dataset

Step 2

```
result = seasonal_decompose(data["Low"], model = "multiplicable", period = 12)
result.plot()
```



```
result = seasonal_decompose(data["Low"], model = "additive", period = 12)
result.plot()
```



Step 3

```
train_size = int(len(data) * 0.7)
train, test = data.iloc[:train_size], data.iloc[train_size:]
```

```
model = ExponentialSmoothing(train["Low"], trend = "add", seasonal = "add", seasonal_periods = 12).fit()
```

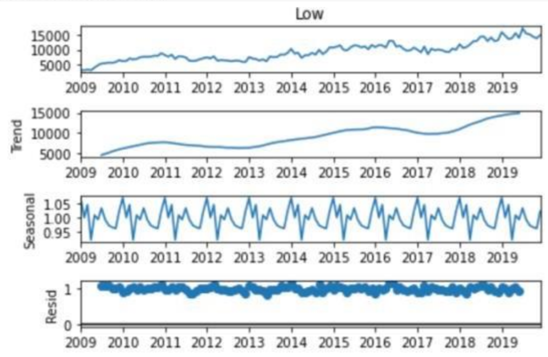
Step 4

```
test_pred = model.forecast(58)
```

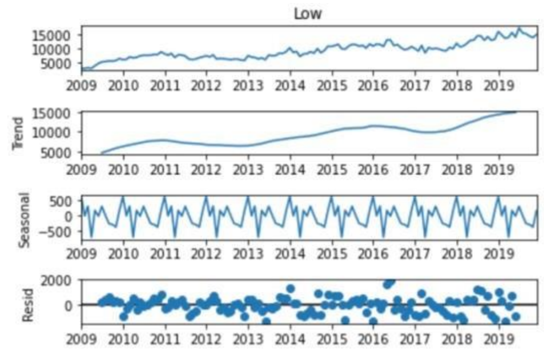
Train Model For 10 Years Dataset

Step 2

```
result = seasonal_decompose(data["Low"], model = "multiplicable", period = 12)
result.plot()
```



```
result = seasonal_decompose(data["Low"], model = "additive", period = 12)
result.plot()
```



Step 3

```
train_size = int(len(data) * 0.7)
train, test = data.iloc[:train_size], data.iloc[train_size:]
```

```
model = ExponentialSmoothing(train["Low"], trend = "add", seasonal = "add", seasonal_periods = 12).fit()
```

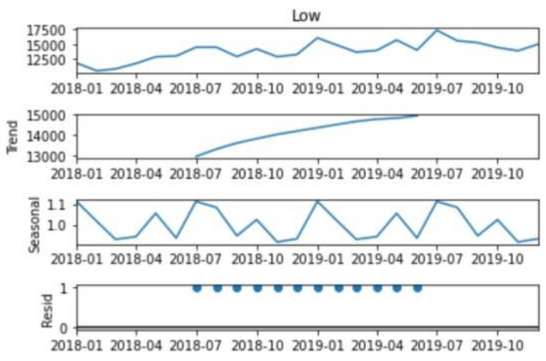
Step 4

```
test_pred = model.forecast(40)
```

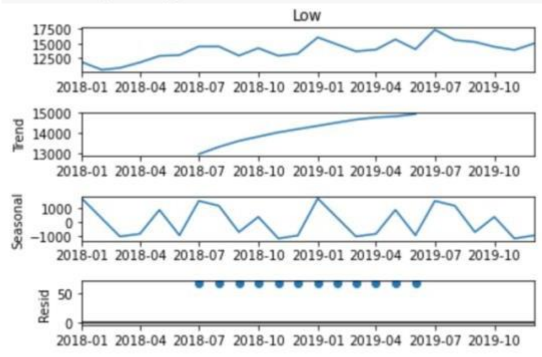
Train Model For 2 Years Dataset

Step 2

```
result = seasonal_decompose(data["Low"], model = "multiplicable", period = 12)
result.plot()
```



```
result = seasonal_decompose(data["Low"], model = "additive", period = 12)
result.plot()
```



Step 3

```
train_size = int(len(data) * 0.7)
train, test = data.iloc[:train_size], data.iloc[train_size:]
```

```
model = ExponentialSmoothing(train["Low"], trend = "add", seasonal = "add", seasonal_periods = 4).fit()
```

Step 4

```
test_pred = model.forecast(8)
```

8.3.3 Evaluation

Step 1

```
train["Price"].plot(legend = True, label = "Train", figsize = (12, 7))
test["Price"].plot(legend = True, label = "Test")
test_pred.plot(legend = True, label = "Predicted Price")
```

Step 2

```
mape = np.mean(np.abs((test["Price"] - test_pred) / test["Price"])) * 100
mape
```

```
mae = mean_absolute_error(test["Price"], test_pred)
mae
```

```
mse = mean_squared_error(test["Price"], test_pred)
mse
```

```
rmse = math.sqrt(mse)
rmse
```

8.4 SMA Model

8.4.1 Data cleaning and preparation

Step 1

```
import numpy as np
import pandas as pd
from pandas import read_csv
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.ensemble import RandomForestRegressor
from sklearn.utils import shuffle
from sklearn.feature_selection import RFE
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_absolute_percentage_error
from sklearn.utils import shuffle
from operator import itemgetter
```

Step 2

15 Years

```
data = read_csv('/Users/HP/Downloads/INFY Historical Data.csv')
data["Date"] = pd.to_datetime(data.Date)
data.set_index ("Date", inplace = True)
data = data.sort_values(by='Date', ascending=True)
```

10 Years

```
data = read_csv('/Users/HP/Downloads/INFY Historical Data (2009-2019) (1).csv')
data["Date"] = pd.to_datetime(data.Date)
data.set_index ("Date", inplace = True)
data = data.sort_values(by='Date', ascending=True)
```

Step 3

```
data.dtypes
```

Step 4

```
data.info()
```

Step 5

15 Years

```
data["Price"] = data["Price"].str.replace(",", "")
data["Open"] = data["Open"].str.replace(",", "")
data["High"] = data["High"].str.replace(",", "")
data["Low"] = data["Low"].str.replace(",", "")
data["Vol."] = data["Vol."].str.replace("M", "")
data["Change %"] = data["Change %"].str.replace("%", "")
```

```
data['Price'] = pd.to_numeric(data['Price'], errors='coerce')
data['Open'] = pd.to_numeric(data['Open'], errors='coerce')
data['High'] = pd.to_numeric(data['High'], errors='coerce')
data['Low'] = pd.to_numeric(data['Low'], errors='coerce')
data['Vol.'] = pd.to_numeric(data['Vol.'], errors='coerce')
data['Change %'] = pd.to_numeric(data['Change %'], errors='coerce')
```

10 Years

```
data["Vol."] = data["Vol."].str.replace("M", "")
data["Change %"] = data["Change %"].str.replace("%", "")
```

```
data['Vol.'] = pd.to_numeric(data['Vol.'], errors='coerce')
data['Change %'] = pd.to_numeric(data['Change %'], errors='coerce')
```

Step 6

```
data.isnull().sum()
```

Step 7

```
data = data.fillna(data["Vol."].mean())
```

Step 8

```
z = np.abs(stats.zscore(data))
(np.abs(stats.zscore(data)) > 3).sum()
```

Step 9

```
data_clean = data[((np.abs(stats.zscore(data)))<3).all(axis=1)]
data_clean
```

Step 10

```
plt.figure(figsize=(16,8))
plt.title('Price History', fontsize = 18)
plt.plot(data['High'])
plt.plot(data['Price'])
plt.plot(data['Low'])
plt.xlabel('Date', fontsize=18)
plt.ylabel('High and Low Price', fontsize=18)
```

Step 11

```
shuffled_data = shuffle(data, random_state = 800)

#X = Independent Variable
target_col = "Price"
X = data.drop("Price", axis = 1)

#y = Dependent Variable
y = shuffled_data[target_col]
```

```
regressor = RandomForestRegressor(n_estimators=100, max_depth=10)
n_features_to_select = 1
rfe = RFE(regressor, n_features_to_select=n_features_to_select)
rfe.fit(X, y)
```

```
features = X.columns.to_list()
for x, y in (sorted(zip(rfe.ranking_ , features), key=itemgetter(0))):
    print(x, y)
```

15 Years and 10 Years Result

Recursive Feature Elimination Selection

```
1 Low
2 Change %
3 Vol.
4 Open
5 High
```

Step 12

```
train, test = train_test_split(data, test_size=0.2, random_state=800)
train_size = int(0.8 * len(data))
train, test = data.iloc[:train_size], data.iloc[train_size:]
```

8.4.2 Modelling

Train Model For 15 Years and 10 Years Dataset

Step 1

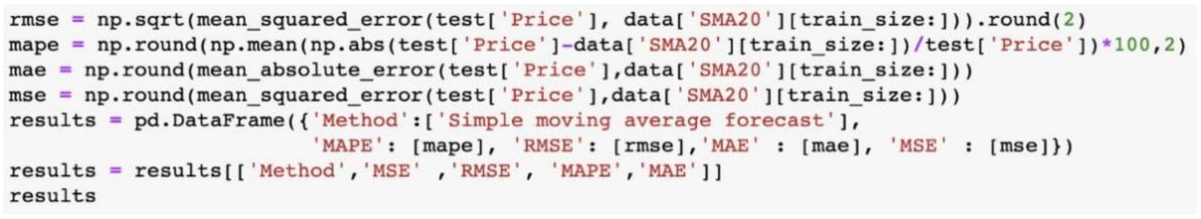
```
data['SMA20'] = data['Low'].rolling(20).mean()
data['SMA50'] = data['Low'].rolling(50).mean()
data['SMA200'] = data['Low'].rolling(200).mean()
```

8.4.3 Evaluation

Step 1

```
plt.figure(figsize=(12,3))
plt.grid()
plt.plot(test['Price'], label='Price')
plt.plot(data['SMA20'][train_size:], label='SMA20')
plt.legend(loc='best')
plt.title('Test Set of SMA20 VS Price')
plt.show()
```

Step 2



```
rmse = np.sqrt(mean_squared_error(test['Price'], data['SMA20'][train_size:])).round(2)
mape = np.round(np.mean(np.abs(test['Price']-data['SMA20'][train_size:])/test['Price'])*100,2)
mae = np.round(mean_absolute_error(test['Price'],data['SMA20'][train_size:]))
mse = np.round(mean_squared_error(test['Price'],data['SMA20'][train_size:]))
results = pd.DataFrame({'Method':['Simple moving average forecast'],
                        'MAPE': [mape], 'RMSE': [rmse],'MAE' : [mae], 'MSE' : [mse]})
results = results[['Method','MSE' ,'RMSE', 'MAPE','MAE']]
results
```