Stock Price Predictions

MACHINE LEARNING AND ARTIFICIAL NEURAL NETWORK MODELS

Pardeep Kumar | Capstone Project-I Report | Mentor: Dipanjan Sarkar

# ABSTRACT

The main objective of this study was to learn and evaluate prediction accuracy of different time series forecasting models. A decade long time history of three technology stocks: Apple, Netflix, and Google were obtained from Nasdaq website (Nasdaq, 2019). Univariate closing price time series (measured daily) was selected as the variable of interest for this study. All results and visualizations were performed using *Python 3.7*, which has in-built libraries for time series analysis and predictions. The time series were log transformed to reduce the variance, and the data was pre-processed to fit the coding requirements of individual *Python* function. Time series analysis, artificial neural networks and decision tree methods were compared and for the given data set decision trees outperformed the other forecasting models.

Table of Contents

[1 Introduction 3](#_Toc9607502)

[1.1 Objective 3](#_Toc9607503)

[1.2 Data Mining 3](#_Toc9607504)

[1.3 Preliminary Analysis 3](#_Toc9607505)

[1.4 Model Selection and Evaluation 4](#_Toc9607506)

[2 Time Series Analysis 5](#_Toc9607507)

[2.1 Time Series Decomposition 5](#_Toc9607508)

[2.1.1 Additive Decomposition 6](#_Toc9607509)

[2.1.2 Multiplicative Decomposition 8](#_Toc9607510)

[2.2 Time Series Transformation 10](#_Toc9607511)

[2.3 Benchmark Models 11](#_Toc9607512)

[2.3.1 Naïve Method 11](#_Toc9607513)

[2.3.2 Simple Exponential Method 13](#_Toc9607514)

[2.3.3 Holt’s Linear Trend Method 15](#_Toc9607515)

[2.3.4 Holt’s Damped Trend Method 16](#_Toc9607516)

[2.4 Autoregressive Integrated Moving Average Models 18](#_Toc9607517)

[2.4.1 Autoregressive (AR) Model 18](#_Toc9607518)

[2.4.2 Moving Average (MA) Model 18](#_Toc9607519)

[2.4.3 Stationary Time Series 18](#_Toc9607520)

[2.4.4 Autocorrelation Function 19](#_Toc9607521)

[2.4.5 Partial Autocorrelation Function 21](#_Toc9607522)

[2.4.6 Unit Root Test 21](#_Toc9607523)

[2.4.7 ARIMA Model 23](#_Toc9607524)

[3 Decision Trees 26](#_Toc9607525)

[3.1 Introduction 26](#_Toc9607526)

[3.2 Features Space 26](#_Toc9607527)

[3.3 Simple Moving Averages (TF) 27](#_Toc9607528)

[3.4 Exponential Moving Averages (TF) 28](#_Toc9607529)

[3.5 Moving Average Convergence Divergence (TF/MF) 29](#_Toc9607530)

[3.6 Relative Strength Index (MF) 30](#_Toc9607531)

[3.7 Stochastic Oscillator 31](#_Toc9607532)

[3.8 Commodity Channel Index (TF/MF) 32](#_Toc9607533)

[3.9 Random Forest 34](#_Toc9607534)

[3.9.1 Hyperparameter Tuning 34](#_Toc9607535)

[3.9.2 Model Fit and Predictions 34](#_Toc9607536)

[3.10 Boosting 36](#_Toc9607537)

[4 Long Short-Term Memory Method 38](#_Toc9607538)

[4.1 Introduction 38](#_Toc9607539)

[4.2 Pre-Processing the Data 38](#_Toc9607540)

[5 Conclusion 40](#_Toc9607541)

[Appendix A 42](#_Toc9607542)

[Quandl API 42](#_Toc9607543)

[Bibliography 44](#_Toc9607544)

# Introduction

The stock market is one of the most crucial components of the modern world. With the support of technology (apps, websites, etc.), anyone can participate in this financial framework. What started as a simple trust-based investment approach has expanded into a significant driving force that affects every aspect of society. It is impossible to account and control every factor that affects the future of a business. Thus, the possibility of losing investment exists in case the company did not meet the expectations. Various mathematical parameters have been developed to help an investor decide when to enter, leave, or stay put in the stock market.

This report attempts to understand the real-world applications of data science. There are three main chapters in this report, followed by conclusions:

* Time series analysis
* Decision trees considering technical parameters
* Regression-based artificial neural networks

## Objective

The objective of this report was to compare the different statistical model to predict and forecast future data value. Performance of different forecasting methods for the three stocks: Apple, Netflix, and Google were studied to evaluate the effectiveness of each technique.

## Data Mining

The first and obvious step in any data science project is to obtain the data. For this study, three stocks were chosen from the IT sector: *Apple, Netflix*, and *Google*. All three companies are major US stocks and capture the market complexities and volatility quite well. The Nasdaq webpage (Nasdaq, 2019) (Springboard, 2019) (Quandl, 2019) has one financial time histories for all three stocks. All analysis models explore the idea of predicting present/future from past observations.

## Preliminary Analysis

CSV files with decade long financial time histories were obtained from the Nasdaq website. The raw data file had six columns: date, closing price, the volume of stocks traded, opening price, highest price, and lowest price. Note that all values were adjusted values. For more details on this, refer to Appendix 1. The start and end date of data were 02-22-2019 and 02-23-2009. The file contained some missing values for weekends and holidays. Table 1.1 shows the most recent 5-days data for Google stock. Data from 02-16-2019 (Saturday), 02-17-2019 (Sunday) and 02-18-2019 (holiday) were missing.

Depending on the prediction model used, the data was pre-processed to fit the model-input requirements. This study focuses on the closing price of the stocks. Prediction models were trained to predict the next day closing price of a given stock.

Table 1.1 First five data rows for Google.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| date | close | volume | open | high | low |
| 2/22/19 | 1116.56 | 1471185 | 1109.7 | 1117.25 | 1100.5 |
| 2/21/19 | 1104.21 | 1663456 | 1118.78 | 1119.15 | 1097.98 |
| 2/20/19 | 1120.59 | 1203703 | 1128.88 | 1130.93 | 1111.75 |
| 2/19/19 | 1126.51 | 1098679 | 1116.64 | 1129.64 | 1116.64 |
| 2/15/19 | 1119.63 | 1390397 | 1139.3 | 1139.3 | 1116.72 |

Figure 1.1 shows the closing prices of three stocks over the last decade. The magnitude range of Google is higher than Apple and Netflix. All three stocks show an overall increasing closing price trend.

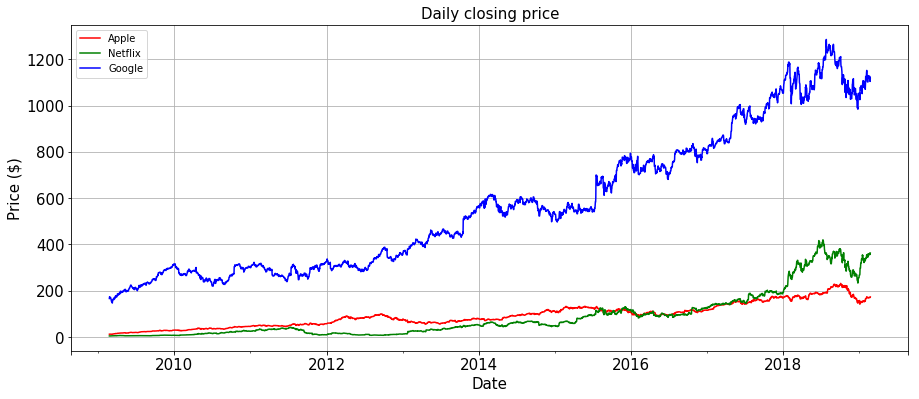


Figure 1.1 Daily (adjusted) closing price of the three stocks.

## Model Selection and Evaluation

Different models were evaluated against the Naïve model to measure the improvements and limitations of each model. Chapter 2 discusses the time series analysis using exponential smoothing and ARIMA models. Chapter 3 explores the decision trees to evaluate the importance of various technical parameters and evaluate the ability of decision trees to predict stock prices. Chapter 4 presents a brief application of the artificial neural network models for forecasting. Each time series was split into training and test set with 2019-02-01 as the split point. Observations before 2019-02-01 were used to train the prediction models, and after were used to evaluate their performance. Performance of each model was assessed using the Mean Absolute Percentage Error (MAPE) measure.

…Equation 1‑1

# Time Series Analysis

The closing price time series is a univariate financial time series, where observations are recorded at the regular time intervals (trading days). Time series usually have four main underlying components:

1. *Trend component* represents the overall increasing or decreasing pattern (trend) of the time series. For example, Google had an overall increasing trend from 2009 to 2019 (Figure 1.1).
2. *The cyclic component* is variation over a period longer than that of the seasonal part. The trend and cyclic components are often combined as one.
3. *The seasonal component* is related to a similar variation with a fixed frequency. For example, sales of winter clothes are usually high during winter and low during other seasons. Thus, there is consistent ‘seasonal’ variation in the sales each year.
4. Whatever remains in the time series after removing all other components, is the random component (also called noise). Similar to linear regression, the randomness comprises information that is not explained by other decomposition components.

## Time Series Decomposition

For time series analysis, the ‘missing’ data on weekends and holidays were filled using the closing price measured on the previous trading day. For example, Friday's stock closing price was assigned to the following Saturday and Sunday. The purpose of this adjustment was to simplify the historical pattern as simpler patterns provide better forecasts. Time series decomposition is one of the first steps to understand the underlying structure and is often a good visualization tool to derive information from the data. There are two main ways to decompose a time series:

*Additive decomposition* where the components can be added to obtain the original time series. Additive decomposition is appropriate when the seasonal variation of time series remains more or less the same over the period.

…Equation 2‑1

*In multiplicative decomposition*, all product of all components gives back the original time series. Multiplicative decomposition is more suited where seasonal variation is not consistent over time.

…Equation 2‑2

### Additive Decomposition

Figure 2.1, Figure 2.2, and Figure 2.3 show the additive decomposition of Apple, Netflix, and Google stock histories, respectively, with the time histories and histograms, side by side.

All three-time series had an overall increasing trend. Trend histograms od Apple was uniformly distributed (in an approximate sense) but were skewed for Netflix and Google. The histogram of Apple suggested that the closing price of the stock increased at a relatively constant rate. Google and Netflix had the same closing and adjusted closing prices, but in the case of Apple, these were not the same time series (see Appendix A).

Frequency histogram of the seasonal component was visually more informative compared to the time series plot due to the high number of data points plotted in the limited space. The maximum magnitude of seasonal components was low compared to the others, for all three stocks. Figure 2.4 shows the boxplot of the three stocks (based on the original series) to verify the seasonal impact. The mean of all three stocks had slight variation, and thus the seasonal component was not significant at all.

Residual components for the three stocks had zero mean. The variance and standard deviation of residual time series give the mean-square error (MSE) and root-mean-square error (RMSE), respectively. The residual component variance of all three stocks increased after 2015 for all three stocks. The variance of the residual component was higher for Google stocks and lowest for Apple.

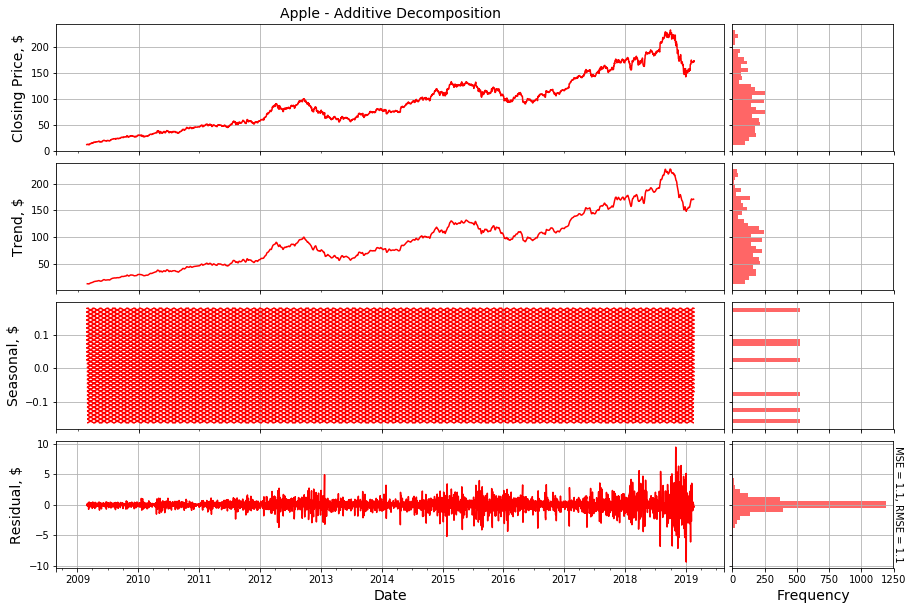


Figure 2.1 Additive decomposition of Apple.

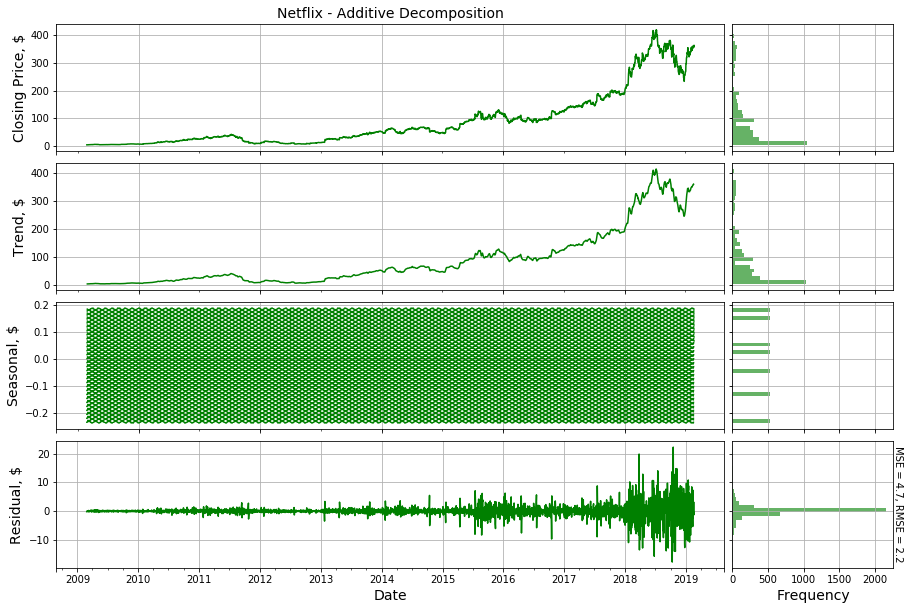


Figure 2.2 Additive decomposition of Netflix.

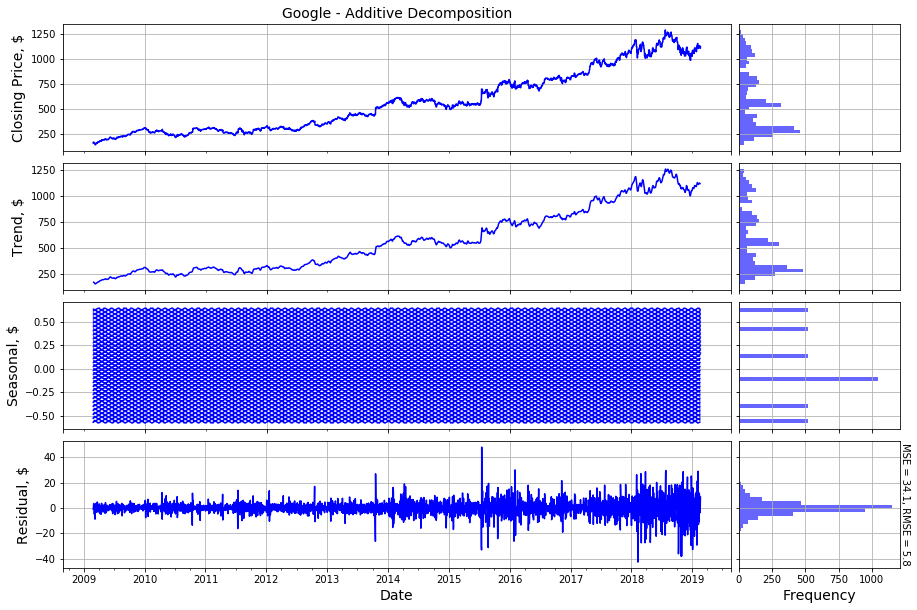


Figure 2.3 Additive decomposition of Google.

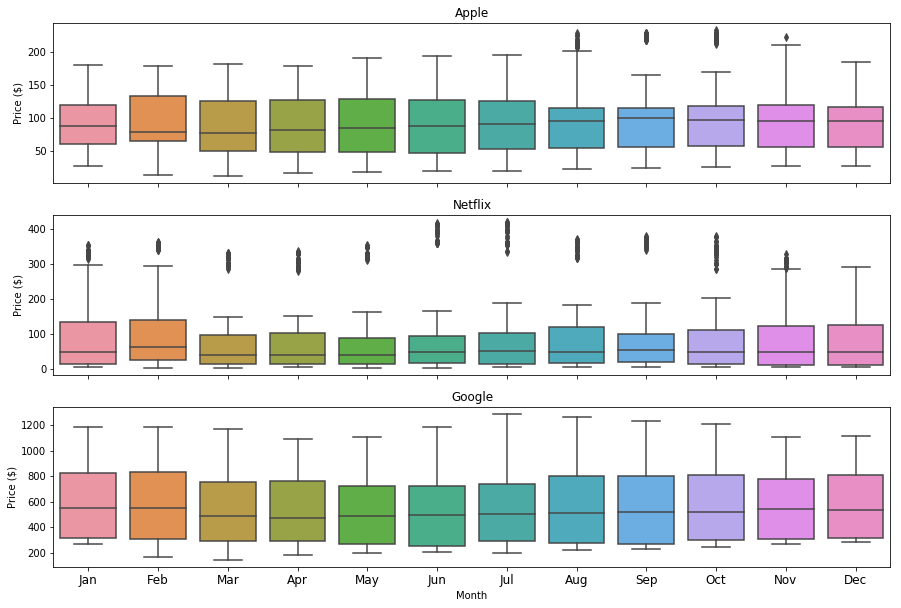


Figure 2.4 Seasonality boxplots.

### Multiplicative Decomposition

Figure 2.5, Figure 2.6, and Figure 2.7 show the multiplicative decomposition of the three stocks. The magnitude of the seasonal component was almost unity with negligible variance.

The residual components had relatively uniform variance (magnitude 1) along the timeline with unit mean. Based on both decompositions, the trend component was most influential for all three stocks. The MSE (and RMSE) for the multiplicative decomposition was lower than the additive decomposition as listed in Table 2.1. Based on this data, a multiplicative decomposition model was well suited for three stocks.

Table 2.1 Mean-square and root-mean-square errors.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Apple | | Netflix | | Google | |
|  | Additive | Multiplicative | Additive | Multiplicative | Additive | Multiplicative |
| MSE | 1.1 | 1 | 4.7 | 1 | 34.1 | 1 |
| RMSE | 1 | 1 | 2.2 | 1 | 5.8 | 1 |

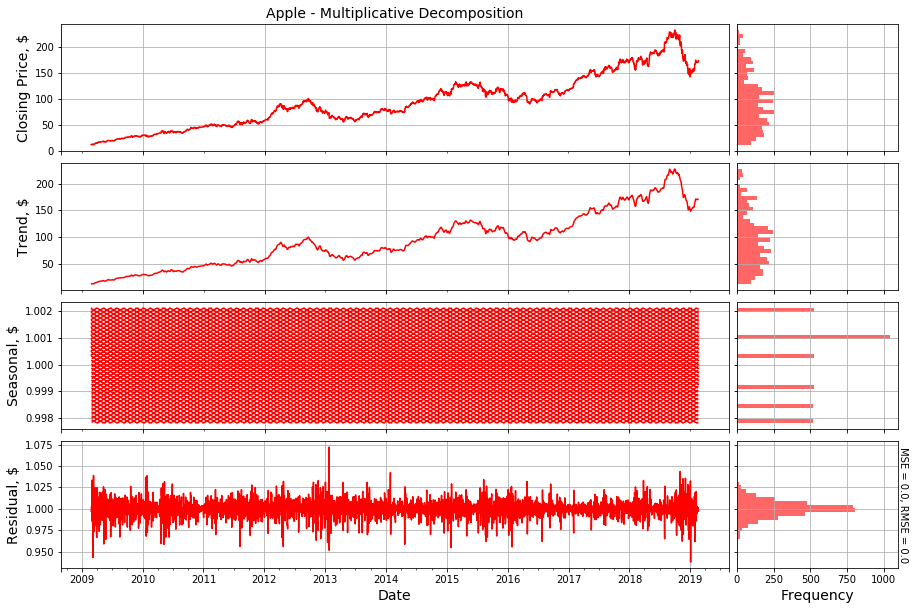


Figure 2.5 Multiplicative decomposition of Apple.

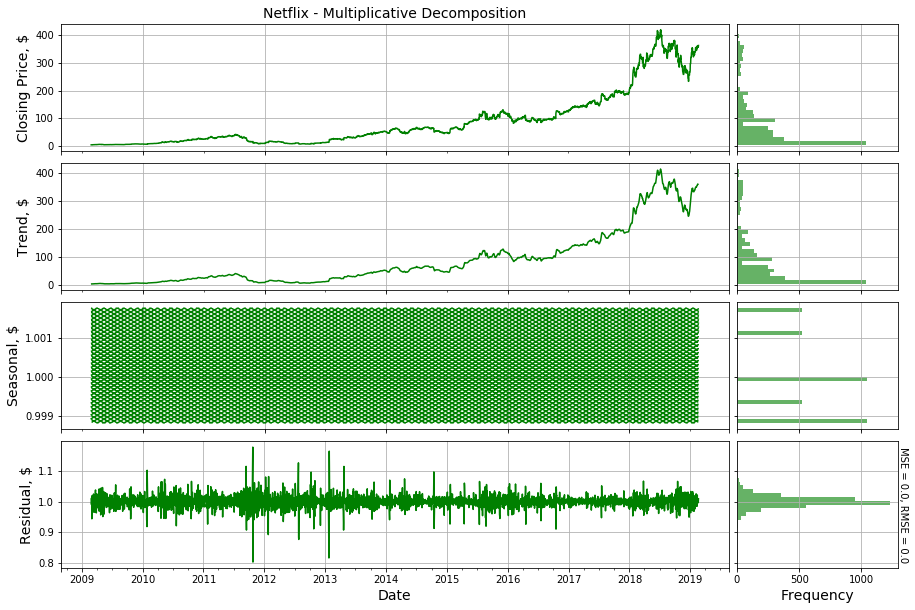


Figure 2.6 Multiplicative decomposition of Netflix.

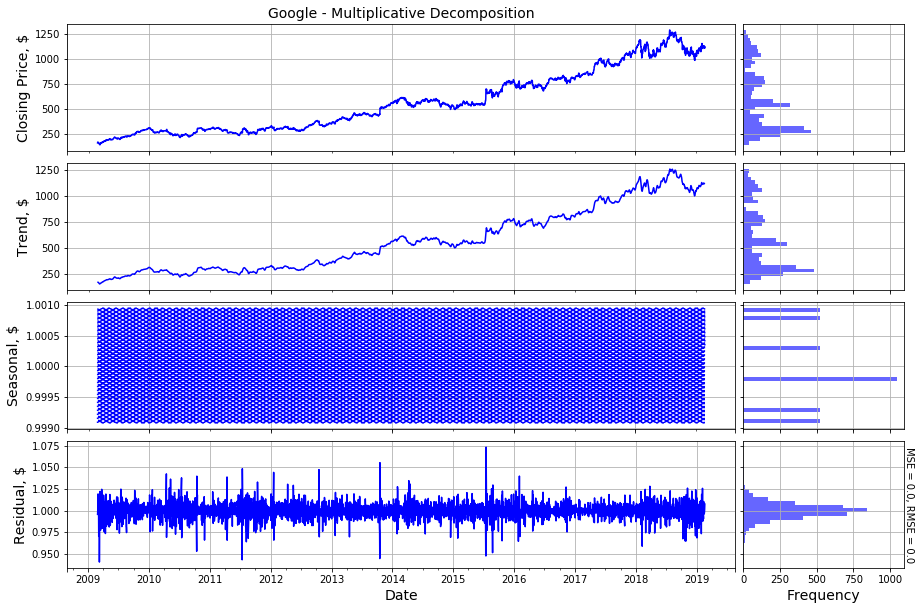


Figure 2.7 Multiplicative decomposition of Google.

## Time Series Transformation

The magnitude of the closing price of the three stocks had different magnitude range. The maximum closing price of Google and Netflix was, respectively, about six times and two times higher than Apple. After logarithmic transformation, the three stock time histories were in close range. Figure 2.8 plots the log-transformed series along with their residual components. As discussed in the previous section, the multiplicative decomposition was better suited for the three stock. After logarithmic transformation, the multiplicative decomposition transforms into an additive decomposition of logged time series,

…Equation 2‑3

All residual components had low variance with zero means. The frequency distributions of the log-transformed series were not normally distributed but were less skewed compared to the original time series.

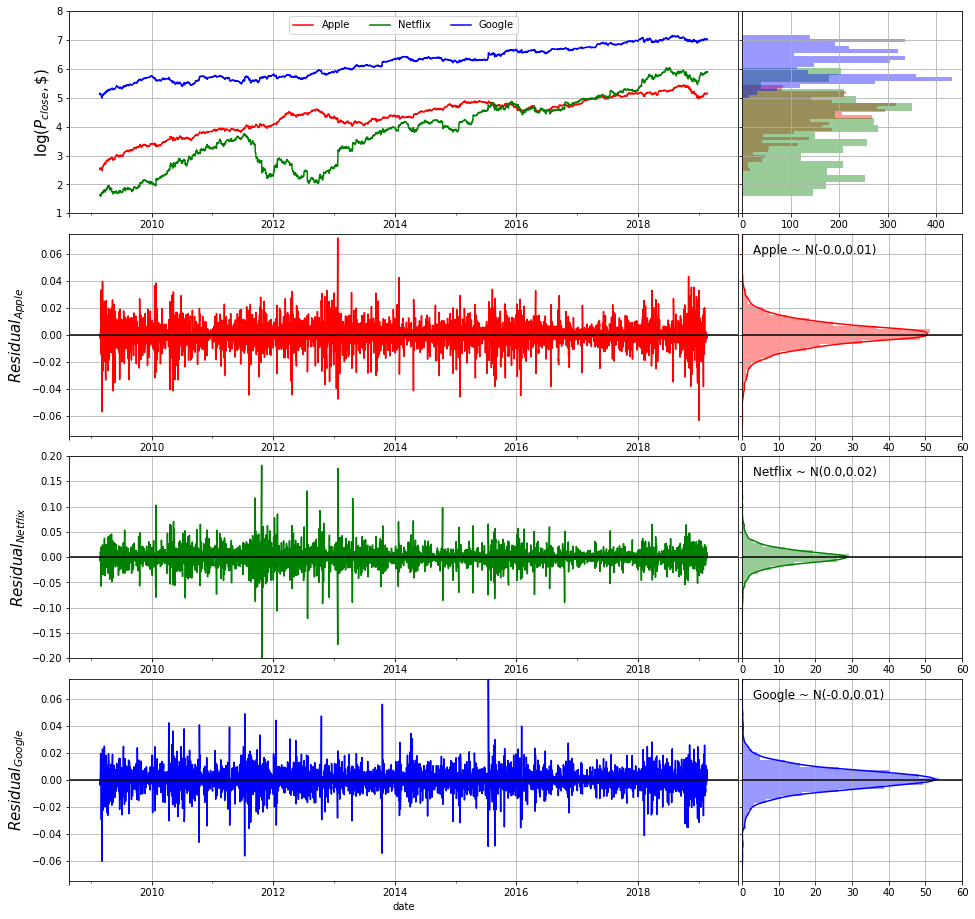


Figure 2.8 Log-transformed time series and residual components.

## Benchmark Models

Benchmark models are the base models that are simple and easy to implement. The naïve and average method are two simple models used for financial time series with the naïve approach being popular than the average method.

### Naïve Method

The naïve method is a forecasting method for the stocks prices that assigns last trading day observation as the prediction for next day’s closing price. In the long-term forecast, all future predictions are equal to the last observed value, i.e.

Equation 2‑4

where is the value of time series at time , and is the predicted value of time series at the next step. In terms of regression, the naïve method assigns a unit weight to the observation at the previous step and zero weight to the rest of past values. Figure 2.9 shows the naïve forecast of the stock time histories for the year 2019. For training data, the predictions were lagged by one-step and for the test set. Last training set observation was assigned as the prediction for all future stock prices. Another variation of the naïve method is the *average method* that assigns equal weights () to all past observations for fitting.

Equation 2‑5

Compared to the naïve method, the average method had higher MAPE error and the predictions were significantly lower than the actual test observations, Figure 2.10, due to the time series trend.

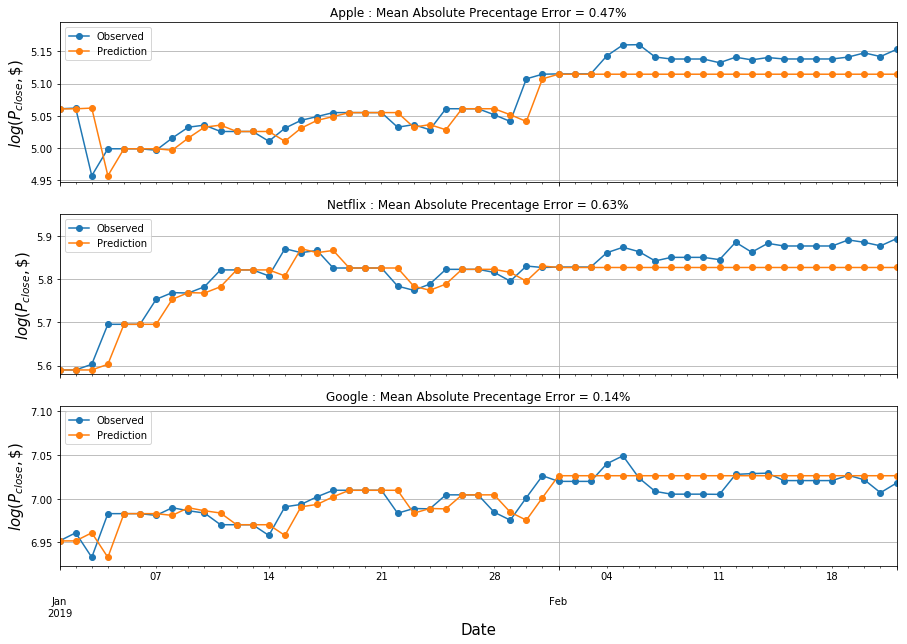


Figure 2.9 Naïve method predictions.

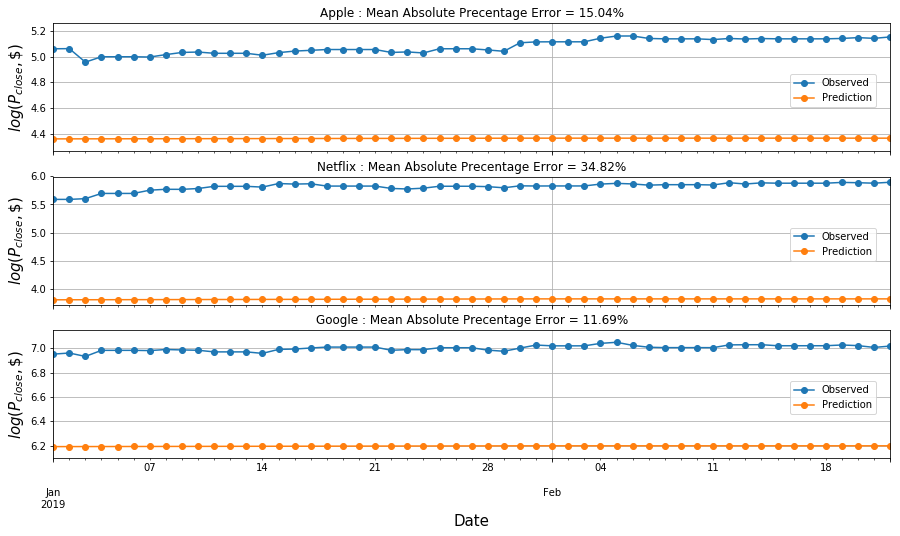


Figure 2.10 Average method predictions.

### Simple Exponential Method

In the naïve method, unit weight is assigned to the last measurements and rest of the past observations are assigned zero weight. In comparison, the average method assigns equal weights to all the observations. In both methods, the total of all the weights is one. Simple Exponential Smoothing (SES) method is based on the idea that more recent data points have a higher impact on the present observations compared to observations further back in the past. To take advantage of available past data without including the entire time history, SES assigns exponentially decaying weights to the past observations as shown in the following equations,

Equation 2‑6

Smoothing Equation: Equation 2‑7

Forecasting Equation: Equation 2‑8

Where is the level or smoothed value at time . For the training data, we computed the fitted values using one-step increment, i.e., . The two unknowns are the initial boundary condition, and the smoothing parameter. The magnitude of smoothing parameters is between zero and one. The naïve method is a special case of SES with smoothing parameter equal to one. sum-of-squares error (SSE) optimization code computes the unknown parameters,

Equation 2‑9

The model predictions are computed using the following flat prediction equation:

Equation 2‑10

Unlike the naïve or the average method, SES adjusts the level of the prediction data. Python’s *statsmodel* library has an in-built function for the SES model. Figure 2.11 plots the SSE against the smoothing parameter. The optimal α-values (based on minimum train SSE), along with the MAPE measures (Table 2.2). For the three-time series, the smoothing parameter was unity. Figure 2.12 compares the observed data and the model fit. With smoothing parameter being one, SES predictions were exactly the same as the naïve method.

Table 2.2 Comparison of the naïve and SES method.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naïve Method | | SES Method | |
|  | Smoothing Parameter | MAPE | Smoothing Parameter | MAPE |
| Apple | 1.0 | 0.47 | 1.0 | 0.47 |
| Netflix | 1.0 | 0.63 | 0.9 | 0.63 |
| Google | 1.0 | 0.14 | 0.8 | 0.11 |

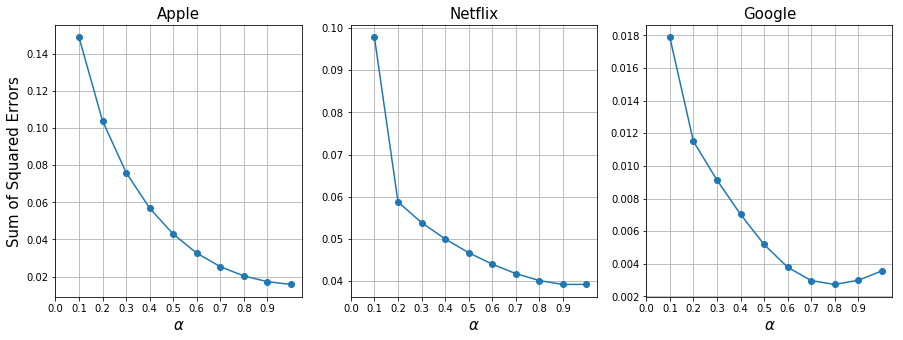


Figure 2.11 SSE vs. smoothing parameter.

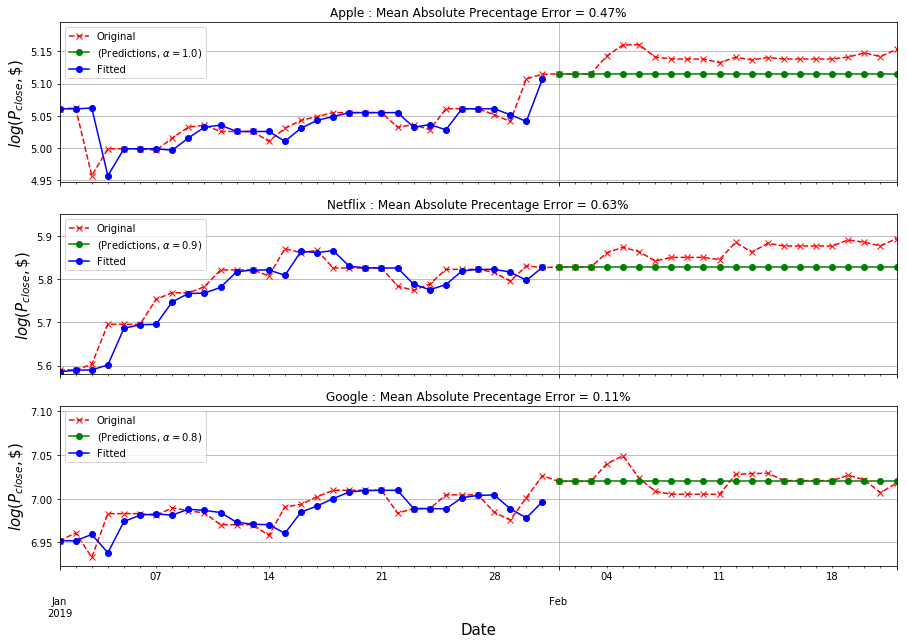


Figure 2.12 SSE model fit and predictions.

### Holt’s Linear Trend Method

In the financial data, the trend is often strongest among the three-time series components (section 2.1). Analysis models with constant prediction function are not reliable, especially for long-term forecasting. Models that incorporate the effect of the trend in forecasting are required. Holt’s linear trend method (Holt, 2004) is a modification of the SES method with an extra step to compute/predict the time series trend. The modified forecast system has three components:

Level Equation, Equation 2‑11

Trend Equation, Equation 2‑12

Forecast Equation, Equation 2‑13

Where is the estimate of time series trend at time *t*. This system has four unknown parameters: two initial conditions (), smoothing parameter for level (α), and smoothing parameters for trend (*β*). The final forecast is a linear function of *h*-future-steps (Equation 2-13).

Figure 2.13 shows the actual observations, the fitted values, and model forecasts. The level equations were the same as the stock time histories due to the unit value of the smoothing level parameter. Smoothing trend parameter (*β*) for Netflix and Google was zero, and the trend equation for these two stocks was constant function*.* Table 2.3 compares the test MAPE measure of Holt's method with the naïve model. A linear trend equation reduced the error by 13% and 35% for Apple and Netflix, respectively, but the error increased by 57% for Google. The test data for Google was more-or-less fluctuating around a horizontal line, and prediction function with linear trend resulted in an improved error.

Table 2.3 MAPE comparison.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Naïve Method | SES Method | Holt’s Linear Trend Method |
| Apple | 0.47 | 0.47 | 0.41 |
| Netflix | 0.63 | 0.63 | 0.41 |
| Google | 0.14 | 0.11 | 0.22 |

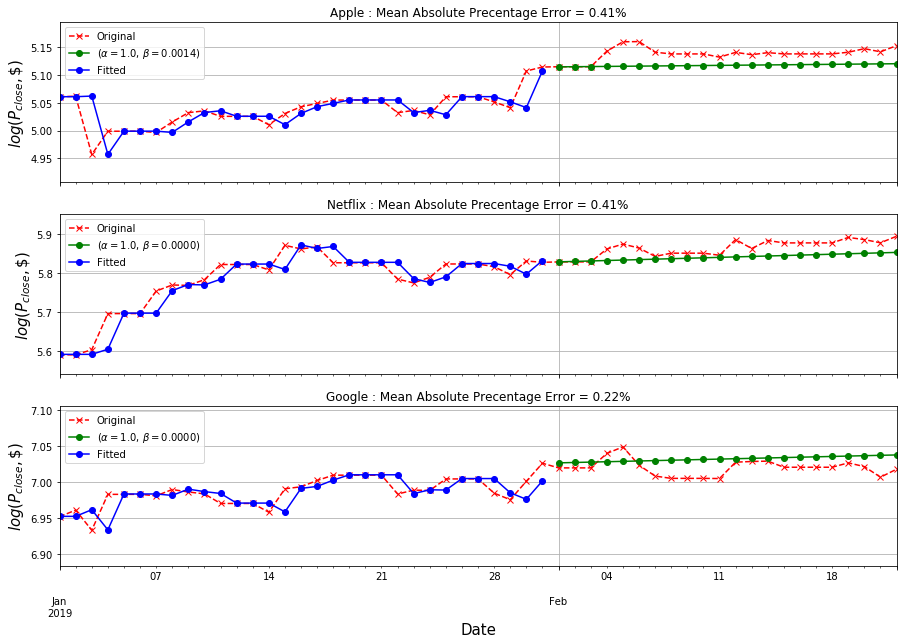


Figure 2.13 Holt’s linear trend method: fitted and predicted values.

### Holt’s Damped Trend Method

Holt's damped trend method introduces a new unknown damping parameter () that serves to reduce the linear trend in the final forecast. It is helpful, especially in long-term predictions, as the stock prices trend is not always constantly increasing or decreasing. The system of equations is updated as follows,

Level Equation, Equation 2‑14

Trend Equation, Equation 2‑15

Forecast Equation, Equation 2‑16

Figure 2.14 shows the model fits along with the original time series. Holt's damping model did not have any forecast improvements over the naïve method (Table 2.5). The damping parameter value for all three-time series was less than 0.2. In practice, the damping parameter is usually not less than 0.8. Figure 2.15 plots the time histories and model fit with damping parameter magnitude of 0.8. Due to the relatively small size of test data (22 days), the forecast accuracy was not sensitive to the magnitude of damping parameters. Compared to Holt's linear model, the damped model error was higher for Apple and Netflix. Due to the reduced slope, the Google MAPE measure improved as the prediction function close to a horizontal line.

Table 2.4 MAPE comparison.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Naïve Method | SES Method | Holt’s Method | Holt’s Damped Model |
| Apple | 0.47 | 0.47 | 0.41 | 0.44 |
| Netflix | 0.63 | 0.63 | 0.41 | 0.63 |
| Google | 0.14 | 0.11 | 0.22 | 0.15 |

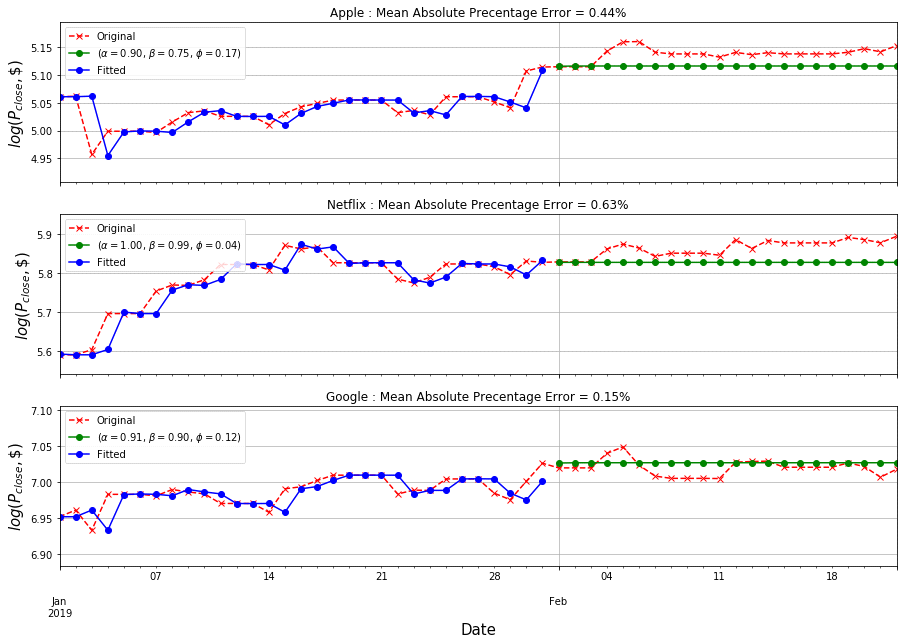


Figure 2.14 Holt’s damped trend method: fitted and predicted values.

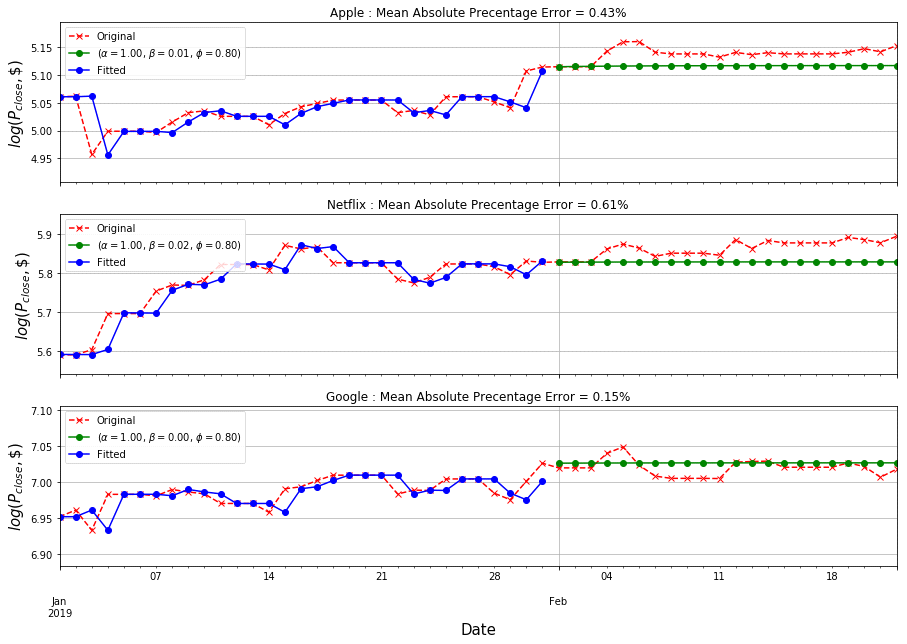


Figure 2.15 Holt’s damped trend method: fitted and predicted values.

## Autoregressive Integrated Moving Average Models

Autoregressive Integrated Moving Average () models are regression model that linearly relates the current value in the time series to past data and errors. Unlike smoothing methods that rely on trend and seasonality, models predictions rely on the covariance in the time series. Depending on the type of time series, model may include an autoregressive () term, a moving average () term, and integrated/differencing () operation. Like most statistical models, condition of data normality is crucial for reliable predictions using models. Stationarity of time series is another necessary condition for implementing the ARIMA model (section 2.4.3). The following sections discuss the individual components of model.

### Autoregressive (AR) Model

Autoregressive models predict the present value of time series as a weighted sum of past observations (say p),

Equation 2‑17

The order of the model can be assessed using the ACF plots. The time series must be stationary and ϕp cannot be negligible. If the mean of is constant but non-zero,

Equation 2‑18

With the backshift notation, can be re-written as,

Equation 2‑19

### Moving Average (MA) Model

The model assumes the linear combinations of past values can estimate the current value of the time series. Moving Average () model linearly combines white noise () form the currently observed value as follows,

Equation 2‑20

Unlike process, the process is stationary for parameters. model is expressed as follows,

Equation 2‑21

### Stationary Time Series

A time series is stationary if probabilistic behavior of a sample data set () is identical to the time-shifted set ().

Equation 2‑22

The above definition of stationarity is too strong for most time series in practice. Thus, weaker stationarity conditions are used:

1. The mean of the time series is constant and does not depend on the time, and
2. The covariance of two points in the time series (also called autocovariance) only depends on the lag between them.

One example of a stationary time series is white Gaussian noise. For a stationary time series with a constant mean (*μ*), autocovariance is the second-moment product,

… Equation 2‑23

In practice, the Autocorrelation function is more useful than autocovariance as it is more convenient to deal with correlation measures between plus/minus one.

### Autocorrelation Function

Autocorrelation function (ACF) measures the linear predictability of using only the value of as,

Equation 2‑24

For a stationary time series,

Equation 2‑25

For a large sample of white-noise data, the sample ACF has a normal distribution with zero mean and finite standard deviation where *T* is the length of time series sample. A time series lag is significant if the value of ACF at that lag exceeds the interval of plus/minus two standard deviations. For a time series to be stationary, approximately 95% of sample ACFs should lie within the boundary limits. Figure 2.16 shows the ACF of the three stock for the first 48 lags (for visualization). Due to the high trend, the ACF values at small lags were large, positive, and slowly decreased as the lag increased, because observations nearby in time are also nearby in size and their magnitude.

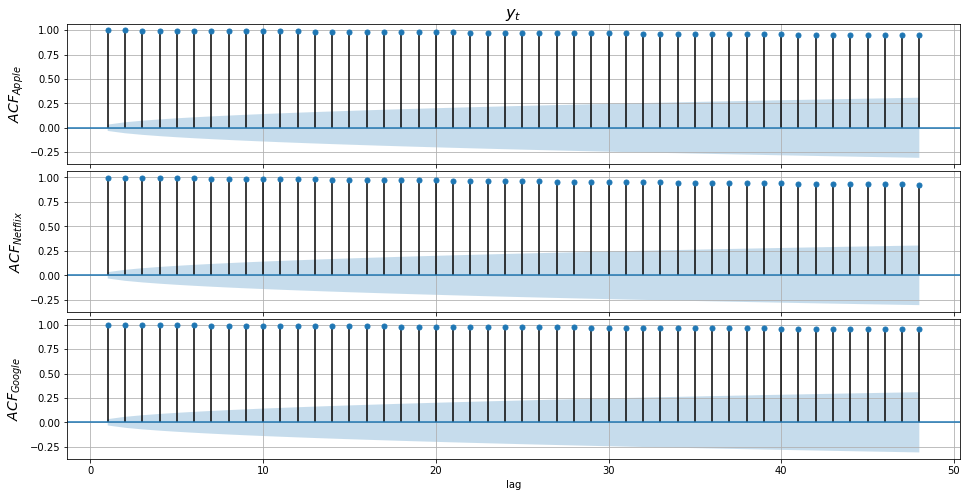


Figure 2.16 Autocorrelation function plots of original data.

The stock times series were log-transformed to reduce the variance. Figure 2.17 plots the *annual* standard deviation of three stocks over the last decade. After the logarithmic operation, the variance of the time series was much lower compared to the original data. Figure 2.18 plots the ACF of the log-transformed data, and it was the same as the ACF plots of the original series in Figure 2.16. The log transformation was effective in reducing the variance, but the time series remained non-stationary due to the relatively higher trend component (Figure 2.8).

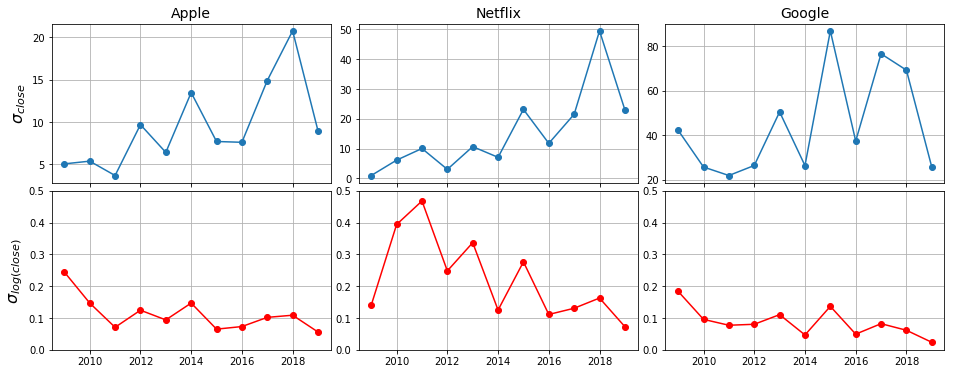


Figure 2.17 Annual variance of three stocks.

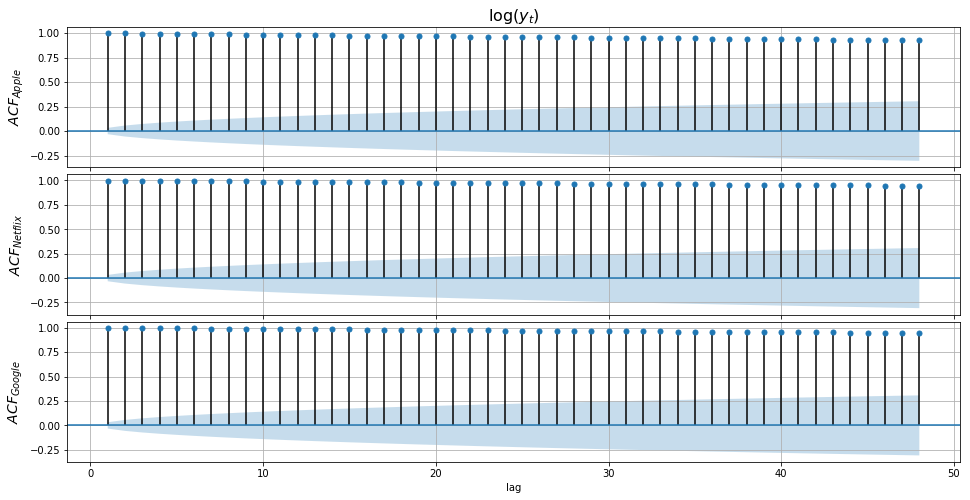


Figure 2.18 Autocorrelation function plots of log-transformed data.

The time series were transformed once again to remove the dominating trend by differencing the observations between consecutive time steps.

Equation 2‑26

With backshift notation, the above equation is rewritten,

Equation 2‑27

The first differenced log series (also called log returns), had roughly all of the peaks within the two standard deviation limits (unit-root test in section 2.4.6).

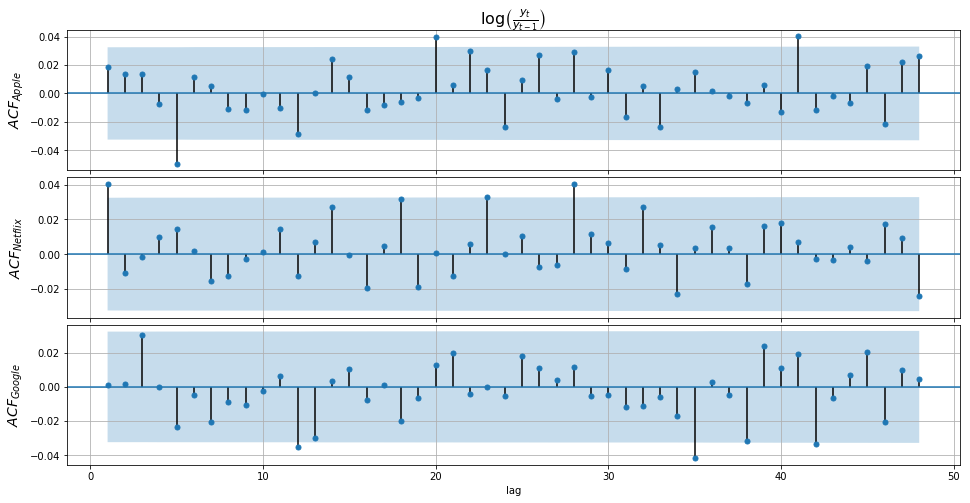


Figure 2.19 Autocorrelation function plots of log returns.

### Partial Autocorrelation Function

Partial Autocorrelation Function (PACF) measures the dependence of any two points in the time series by removing the direct effect of all points in-between. PACF correlated portion of and that is not explained by.

Equation 2‑28

For a stationary time series PACF can be written as,

… Equation 2‑28

PACF is relatively more reliable than ACF for selecting the order of AR model.

### Unit Root Test

Unit root tests are used to test the stationarity of time series. A time series is non-stationary if it has a unit-root. For process with non-zero mean, the regression equation is,

Equation 2‑30

In the above equation, the noise term does not necessarily have to be white noise. For models, is assumed to have zero mean. If,

Equation 2‑31

Equation 2‑21 is a non-stationary time series with a linear trend (Equation 2‑32) and non-zero variance (Equation 2‑33) as the noise was assumed to have zero mean but non-zero variance.

Equation 2‑32

Equation 2‑33

For the existence of unit root, must be unity (). Several statistical tests have been developed to check if a time series has a unit root. Augmented Dickey-Fuller (ADF) unit root test is the most common test used to test the presence of unit-root in a financial time series. The test considers that errors are serially correlated, and enough number of lags must be added to the equation (say *p*) to transform the final error term into white noise.

If the unit root exists, then from Equation 2‑19,

The above equations represent the building block of ADF unit-root test. ADF test is simply a t-test for the null hypothesis () of against the one-sided left tailed (stationary) alternative hypothesis () of. The test statistic is,

Where is the least square estimate and is the standard error estimate. Python’s *statsmodel* library provides a simple one-line code to compute ADF unit-root test parameters. The model provides three critical level values for acceptable test statistic with 1%, 5%, and 20% limits (Table 2.5). The critical value of 1% was chosen to reject the null hypothesis, i.e., the ADF statistic must be smaller than the -3.4321 to reject the null hypothesis.

Table 2.5 Critical t-statistic values for ADF unit-root test.

|  |  |
| --- | --- |
| Confidence Level | Critical t-statistic |
| 99% | **-3.4321** |
| 95% | -2.8623 |
| 90% | -2.5672 |

Table 2.6 ADF unit-root test statistics for the three stocks.

|  |  |  |  |
| --- | --- | --- | --- |
| Stocks | ADF Statistic | | |
| Original | Log Transformed Series | Log-returns |
| Apple | -1.0191 | -3.3515 | -28.0529 |
| Netflix | 0.8750 | -0.8553 | -58.0202 |
| Google | -0.0911 | -1.3457 | -60.4180 |

Table 2.6 lists the ADF t-statistic measured for the original, log-transformed, and log-return (differenced) time series. All original and log-transformed series had unit roots, as the ADF statistics were higher than the 1% critical value (Table 2.5). After differencing the log-transformed series, the ADF statistics were much lower than the critical values, and the log-return time series was considered stationary.

### ARIMA Model

Previous sections discussed the important transformations of the time series to have them ready for the ARIMA ARIMA model. can be written as,

Equation 2‑34

Equation 2‑35

Python package *pmdarima* provides function *auto\_arima* that can fit the model to the time series. The *auto\_arima* method tries different combinations for the models. Best fit model is the one with minimum . Akaike's Information Criterion () is a *model selection measure* that helps determine the order of the model. For a given likelihood () of the training data, is given as,

Equation 2‑36

Table 2.7 provides a model summary after fitting the model on the training data of three stocks. Random walk model with a constant was the best fit for Apple and Google. Equation 2‑34 suggests that the following final models for the three series:

All three best fits were essentially linear equations. Figure 2.20 confirms this as all model predictions lie on straight lines. The red-highlighted zone represents a 95% confidence interval for the ARIMA model predictions.

Table 2.8 compares the MAPE measures of the three-time series based on the different prediction models discussed so far. Overall, the model performed better for Apple and Netflix compared to the naïve model. In the case of Google, the ARIMA model performed similarly to Holt's linear trend mode.

Table 2.7 ARIMA model fit summary.

|  |
| --- |
| A screenshot of a social media post  Description automatically generated |
|  |
| A screenshot of text  Description automatically generated |
|  |
| A screenshot of a social media post  Description automatically generated |

Table 2.8 MAPE comparison of different models.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Naïve Method | SES Method | Holt’s Linear Trend Method | Holt’s Dampened Trend Method | ARIMA Models |
| Apple | 0.47 | 0.47 | 0.41 | 0.44 | 0.31 |
| Netflix | 0.63 | 0.63 | 0.41 | 0.63 | 0.41 |
| Google | 0.14 | 0.11 | 0.22 | 0.15 | 0.22 |

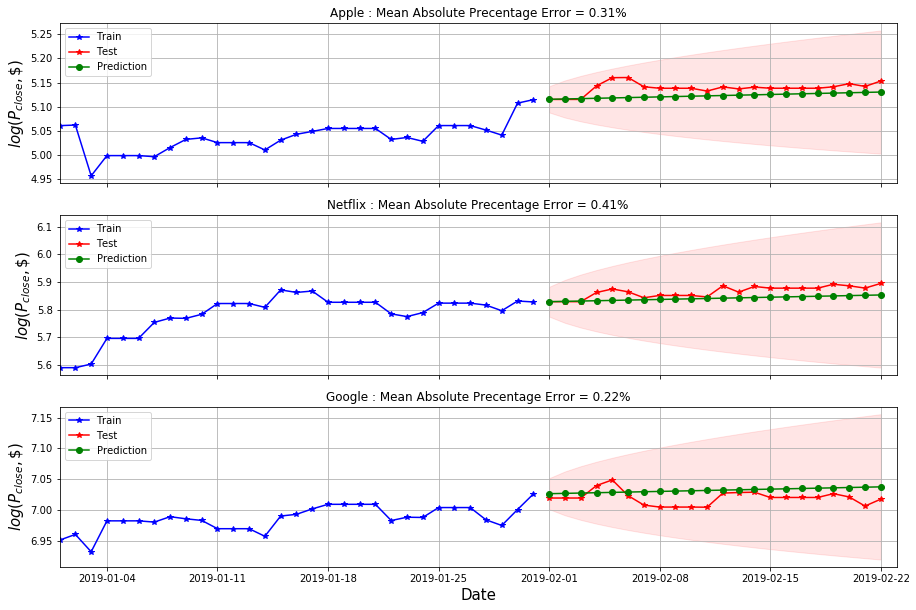


Figure 2.20 ARIMA model predictions for three stocks.

# Decision Trees

## Introduction

Decision trees are supervised learning methods that are intuitive and provide excellent visualization of the modeling process. Decision trees can model both regression and classification problems. The discussion in this chapter is mostly regarding regression models. In the simplest case, prediction space is divided to grow a large tree until a threshold has reached (e.g., the minimum size of the terminal node). Predictions are mean node values of the terminal region. Just like other regression methods, the decision trees also suffer from overfitting i.e., training error of model can be lower than the test error. Various methods, like random forests and boosting, have been developed to tackle this issue of overfitting. All these algorithms rely on the number of observations and the size of the feature space.

## Features Space

The main aim of an investor is to understand the market and invest wisely, maximizing the gains, and reducing the losses. If the market is going long, the stock price is expected to rise, and it would be beneficial to buy the stock. On the other hand, it is wise to sell shares that are predicted to fall in price going short. For technical analysis of the market, several financial indicators have been developed to help investors understand the market trends. These indicators fall into two main categories: Trend-Following (TF) indicators and Momentum-Following (MF) indicators, also called the oscillators. There are more than 100 such indicators, and investors use whichever help them invest profitably. For this study, following ten financial indicators selected

1. 5- days simple moving average
2. 10-days simple moving average
3. 5-days exponential moving average
4. 10-days exponential moving average
5. Moving average convergence divergence line
6. Moving average convergence divergence signal
7. Relative strength index
8. Slow stochastic indicator %K
9. Fast stochastic indicator %D
10. Commodity channel index

All these features capture the impact of past data points on recent and future prices. The number of previous observations chosen for computing each indicator is called the window. The size of window impacted each technical indicator and was assigned the values recommended in the literature. The decision trees were grown in the 10-dimensional feature space to predict the closing price of three stocks. The final performance was measured by comparing the mean absolute percentage error with the Naïve model.

## Simple Moving Averages (TF)

Simple Moving Average (SMA) is a smoothing technique widely used to (approximately) predict the stock market. For a rolling window of size *n*, SMA at any time *t* in the time series is,

Equation ‑

SMA assigns equal weights , to all past data points in the rolling window. For this study, *n* value of 5- and 10-days were selected. Figure 3.1 shows the 5- and 10-days SMAs for the first half of 2018, along with the original observations. SMA plots were smoother than the original time series and slightly shifted to the right with the shift being higher for the 10-days SMA series. SMAs computed for all three stocks followed the original trend closely.

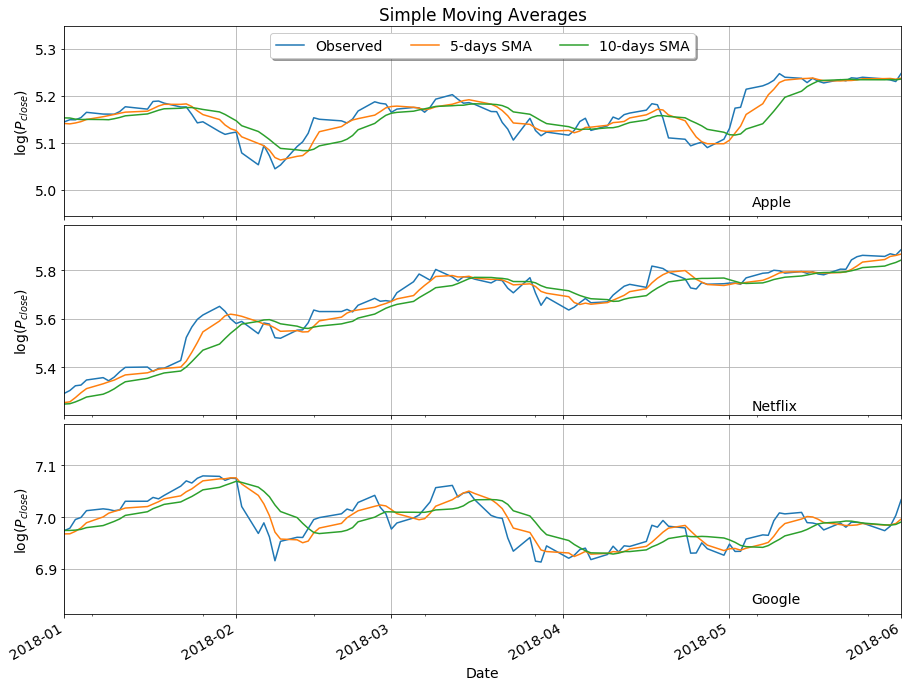


Figure 3.1 Simple Moving Averages from January 2018 to June 2018.

## Exponential Moving Averages (TF)

Often time in time series, recent observations are more important than the past ones. It makes more sense to assign relatively higher weights to recent data points. Exponential Moving Averages (EMA) does exactly this with the higher weight assigned to the most recent observation and exponentially decreasing weights for the rest of the past data points in the window. EMA at any time *t* can is as follows,

Equation ‑

Figure 3.2 shows, 5- and 10-days EMA plotted alongside the stock time histories for the first half of 2018. Similar to SMA, EMA plots were smoother than the actual observation plot, but unlike SMA, the plots did not shift to the right after smoothing. In this report, EMA and WMA (weighted moving average) used interchangeably.

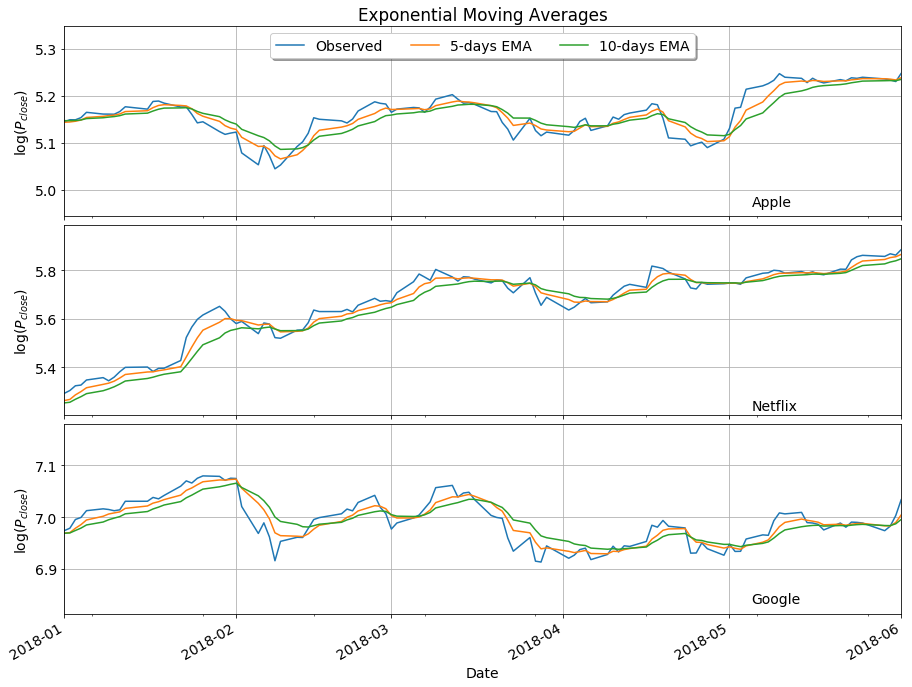


Figure 3.2 Exponential Moving Averages from January 2018 to June 2018.

## Moving Average Convergence Divergence (TF/MF)

Moving Average Convergence Divergence (MACD) belongs to both the trend-following and the oscillator categories. It helps in detecting the early stages of trend and computed by taking the difference between long and short EMA.

Equation ‑

Short- and long- EMAs were respectively computed for the window of size 12 and 26 days. The time series obtained using the above equation is called the MACD line. MACD signal (or trigger line) is 9-days EMA of the MACD line. Anytime the line crosses the signal, a trade signal is triggered. A ‘sell' signal is triggered when the MACD line crosses below the signal and ‘buy' signal when MACD line crosses above the MACD signal. The MACD oscillator for Google followed trend reversal expected at the beginning of February (sell signal) and in the first week of May (buy signal). Same was true for the other two stocks (Figure 3.3).

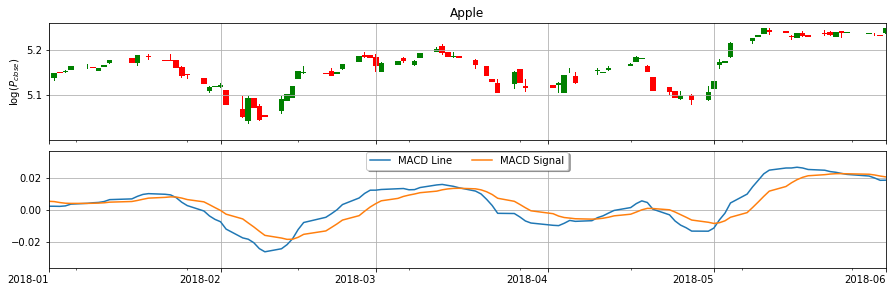




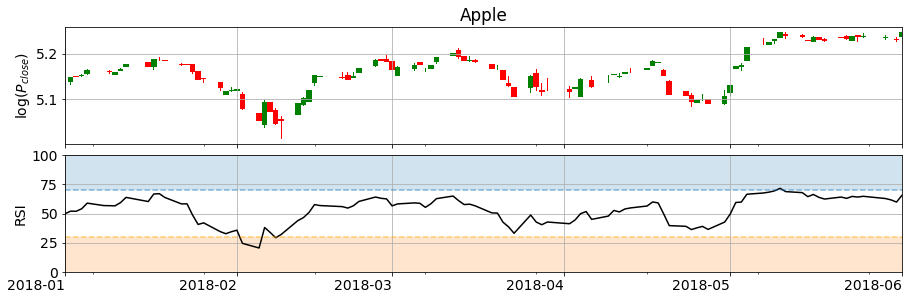
Figure 3.3 Moving Average Convergence Divergence from January-June 2018.

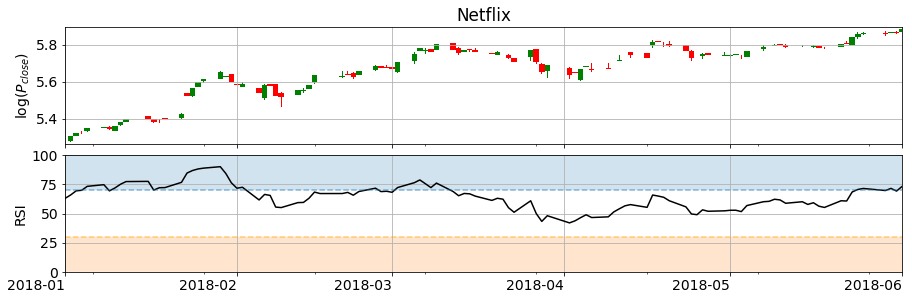
## Relative Strength Index (MF)

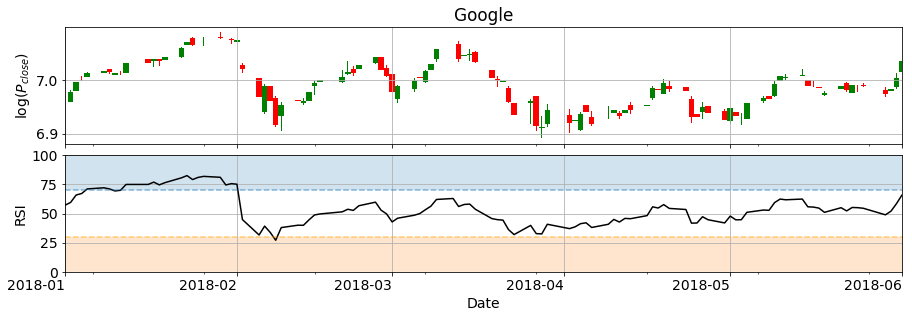
Relative Strength Index (RSI) is a momentum indicator that measures the recent changes in stock price or the strength of a stock. RSI computed for a rolling window size of 14 days is written as,

Equation ‑

RSI oscillated between value 0 and 100. RSI value above 70 indicates that stock is overvalued and may be ready for a trend reversal. RSI less than 30 suggests an undervalued stock. Figure 3.4 shows the stock prices and RSI in the first six months of 2018. The overall RSI magnitude increased with the number and size of the green candlesticks, and as the number of red candlesticks increased, the RSI value fell. For the first month of 2018, Netflix and Google stocks were overpriced (RSI > 70).





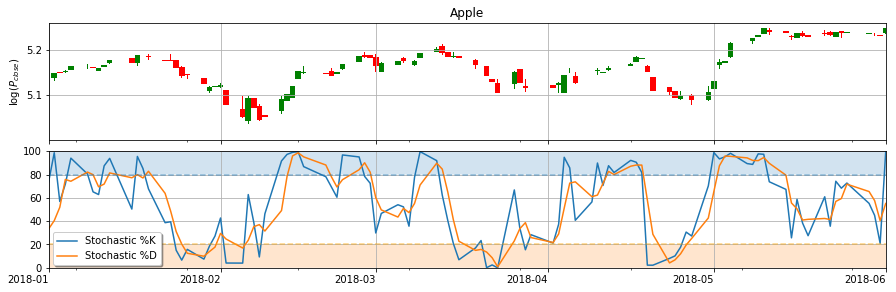
Figure 3.4 Relative Strength Index from January-June 2018.

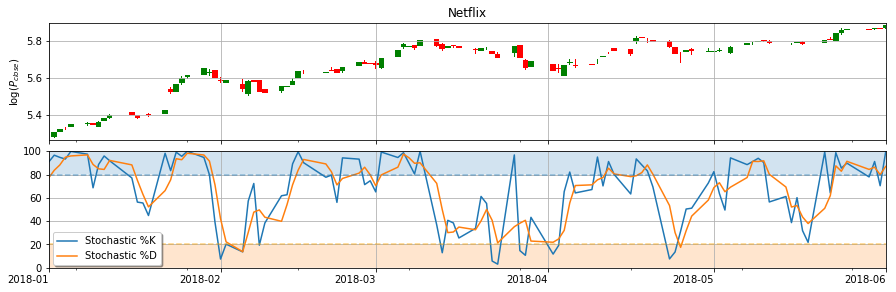
## Stochastic Oscillator

Stochastic Oscillator is a momentum indicator that compares the most recent closing price (also the stochastic in this case) to the highest high () and lowest low () stock price in a given rolling window (*n =* 14 days). Similar to MACD, stochastic oscillator has two-lines: a slow-stochastic indicator (*%K*) and fast-stochastic indicator (*%D*).

Equation ‑

Fast-stochastic indicator (*%D*) is 3-days SMA of slow-stochastic indicator (*%K*). Stochastic oscillator lies between 0 to 100. Values above 80 and below 20 indicate an overpriced and underpriced stock, respectively. Netflix showed a relatively stronger uptrend and had longer instances of being overbought with stochastic exceeding magnitude 80 (Figure 3.5).





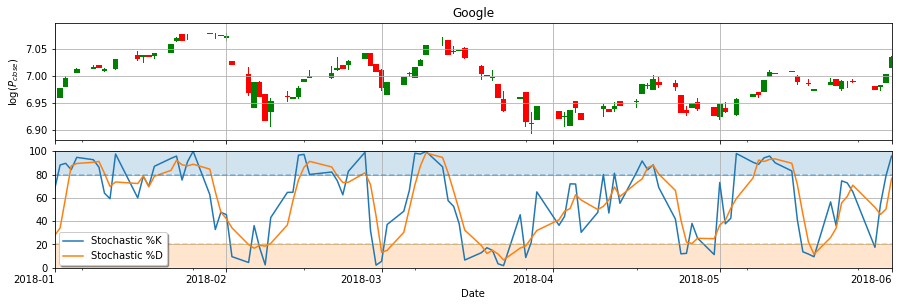


Figure 3.5 Stochastic Oscillators from January-June 2018.

## Commodity Channel Index (TF/MF)

Commodity Channel Index (CCI) compares the recent stock prices with the past averages. To compute CCI first ‘typical price’ is computed for all days in the rolling window (*n = 14 days)*,

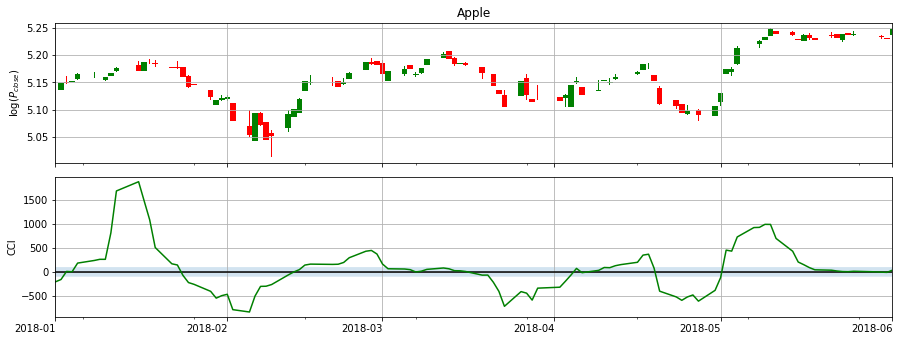
Equation ‑

The simple moving average of the typical price is computed along with the average standard deviation. To compute the (average) standard deviation, the moving average is subtracted from the typical price on a given day. The absolute values of these differences are added and divided by the window size. All computed values are then assembled per the formula below to compute the CCI,

Equation ‑

CCI helps spot new trends and early signal if the stock is reaching the extent of being oversold or overbought, which helps traders to decide whether to buy, sell, or refrain from both. CCI does not have any bounds but fluctuates above and below zero. High (>100) positive or negative value of CCI indicates that stock is well above or below the historical average, respectively.

Figure 3.6 shows the CCI plots for the first half of 2018. Of the three stocks, Netflix had a relatively stronger uptrend, and the CCI was in the positive zone for the most part. Netflix also showed divergence in the first month where the over trend was upward, but the CCI fell, which suggested a weak trade signal and a ‘possibility' of price reversal. Google has high volatility in the first half of 2018. Its CCI stayed near the range [-100,100] (blue zone in the figure). For all three stocks, at almost each trend reversal, the CCI values changed sign, and the magnitudes jumped drastically (more than 200 points).



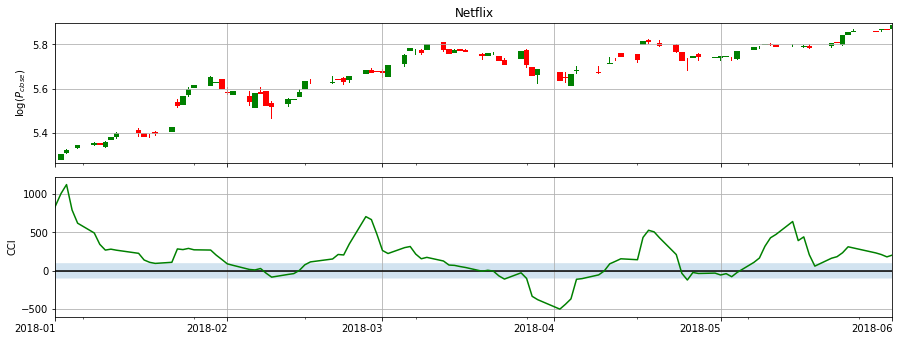


Figure 3.6 Commodity Channel Index from January-June 2018.

## Random Forest

Random forest is an ensemble learning algorithm that combines results from random decision trees to improve the model predictability. Random forests decorrelate the decision trees by restricting the number of features it can consider during each split. By random selection, the algorithms give a chance to weaker predictors and avoiding picking of stronger ones at every split. Otherwise, the strong predictor will be selected at each split, resulting in similar decision trees that are highly correlated. Taking the mean of correlated quantities does not reduce the variance resulting in weak model predictions. Random Forests the final results by averaging ‘random' trees, reducing the variance and increasing the prediction reliability.

Python has an in-built random forest regression function. The function has many different input arguments. The function arguments have default values which do not fit every regression problem. Hyperparameter tuning is often an essential step before running the final model.

### Hyperparameter Tuning

Hyper-parameter tuning is vital for obtaining a well-optimized model. One size fits all analogy does not work well in statistics. For random forests algorithm arguments like the optimal number is trees and number of features considered at each split, are extremely hard to find in the first attempt. Optimization involves trial-and-errors with different magnitudes of input features. Sample data is split for cross-validation to avoid overfitting. Hyperparameter tuning was carried out in two steps in python:

1. Randomized Search Cross-validation
2. Grid Search Cross-Validation

Both search methods used a range of hyperparameter values. RandomizedSearchCV function in python picks random values of each parameter and try to find a combination with least cross-validation error.

On the other hand, GridSearchCV lays out all the elements and runs the best model search on every possible combination of the parameter set. It may seem like overkill, but random forests often perform poorly on the time series, so both steps were carried out to find the best parameter set for the model fitting. The best hyperparameters suggest by the RandomizedSearchCV were selected and then fed into the GridSearchCV along with one value in the neighborhood of each parameter.

### Model Fit and Predictions

After finding the best parameters, the model was fit on the training dataset. The first row of Figure 3.7 shows the relative importance of each feature for the three stocks. SMA and EMA were the best predictors for the time series. Figure 3.7 also shows the model fit on the training data and the test predictions. Compared to the naïve model, random forest models performed exceptionally well. The models were able to predict the stock ups and down quite well without any lags (compared to ARIMA).

Table 3.1 shows the mean absolute percentage error (MAPE) measured on the test data from the Naïve and Random Forest. MAPE values obtained from random forest models were about three times less than the Naïve model, which was a significant improvement in model optimization.

Table 3.1 Random Forest MAPE comparison with Naïve method

|  |  |  |  |
| --- | --- | --- | --- |
|  | Apple | Netflix | Google |
| Naïve Model MAPE | 0.47 | 0.63 | 0.14 |
| Random Forest MAPE | 0.16 | 0.21 | 0.05 |

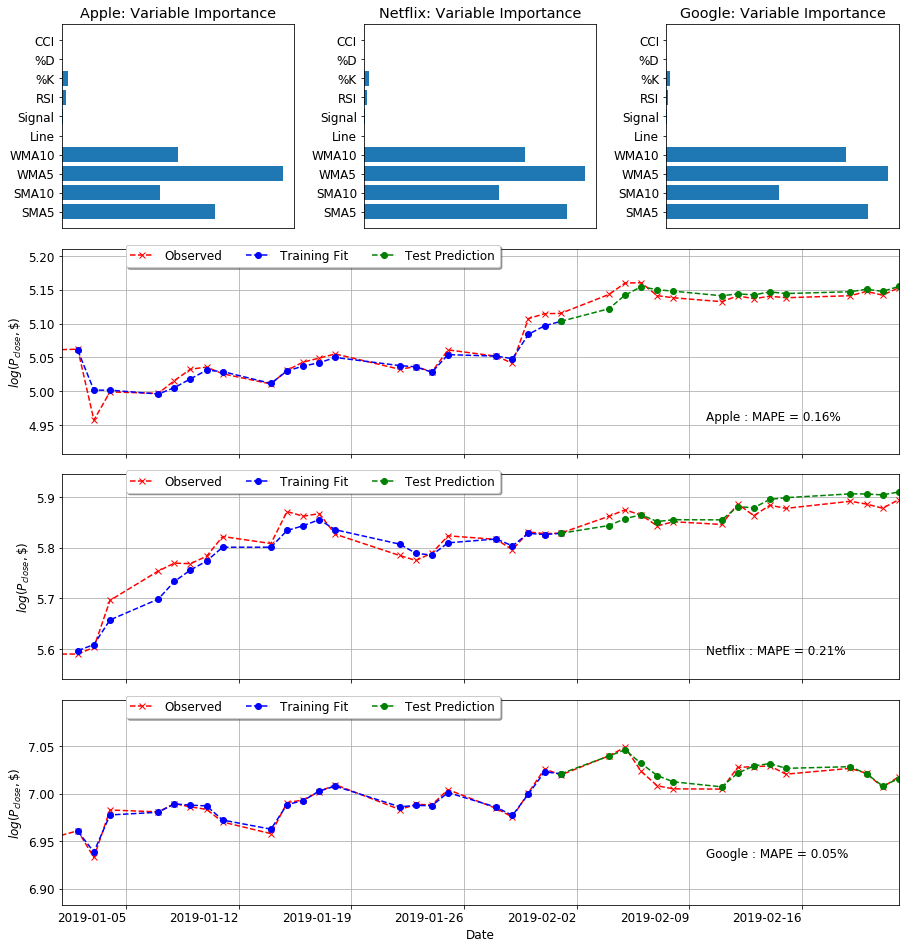


Figure 3.7 Random Forest Model: Relative variable importance and model fits.

## Boosting

In boosting, trees are grown sequentially. Rather than fitting the data hard by growing a whole tree, boosting learns slowly. The algorithm adjusts the decision trees on model-fit residuals. The fitted trees are added back to the original model residual is updated (reduced). The process is repeated with different hyper-parameters like the number of trees, learning rate, maximum depth of each tree, etc. Boosting works on optimization skills by reducing both bias and variance.

*Python* has an in-built library dedicated to boosting *xgb*. *XGBRegressor* was used for fitting and predicting the stock closing prices. Hyper-parameter tuning steps (section 3.9.1) were repeated for *XGBRegressor*. Figure 3.8 shows the final model fits and predictions along with the relative importance of features. Compared to the random forest, XGBoost SMA and EMA measured with 5-days window were the strongest predictors for Apple and Google. In the case of Netflix, SMA at 5-days, and both EMAs were strong predictors.

MAPE measure of XGBoost was one-third that of the naïve model (Table 3.2). Thus, decision tree methods performed better than the benchmark model. The prediction accuracy of both random forest and XGBoost was almost the same. The number of predictors needed in case of XGBoost was nearly half the number of predictors for Random Forest.

Table 3.2 MAPE comparison of Naïve and Ensemble methods

|  |  |  |  |
| --- | --- | --- | --- |
|  | Apple | Netflix | Google |
| Naïve Model MAPE | 0.47 | 0.63 | 0.14 |
| Random Forest MAPE | 0.16 | 0.21 | 0.05 |
| XGBoost MAPE | 0.15 | 0.18 | 0.06 |

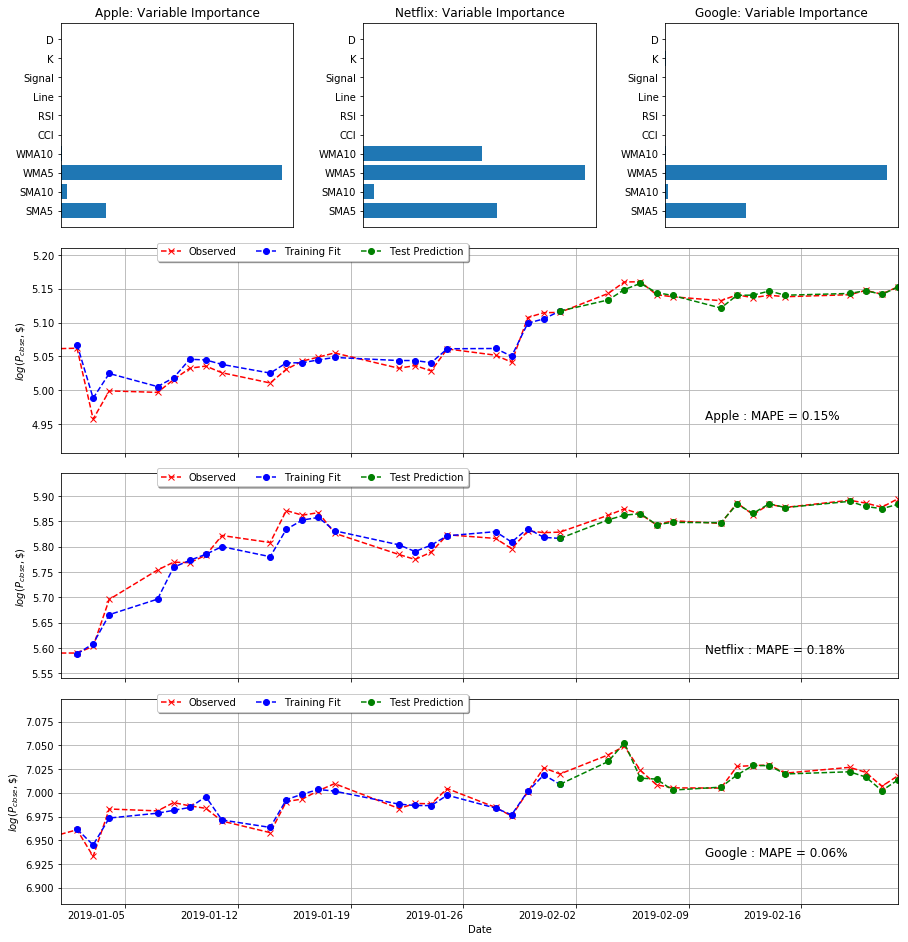


Figure 3.8 XGBoosting: Relative variable importance and model fits.

# Long Short-Term Memory Method

## Introduction

Artificial Neural Networks (ANN) are a simplified version of the brain architect. ANN can be trained for both supervised and unsupervised problems. ANN comprises of various components like nodes, weights, activation functions, etc. Nodes form the primary connection hubs for the network. A node accepts the weighted information from other nodes and sums it up before passing it on to the activation unit. Assigned weights control the influence of a node on the nodes in the next layer. An activation unit acts as a filter for ANN, which manages the amount and type of information transferred down the pipeline. Different activation functions are available in the literature, and the choice of a particular function depends on the problem at hand. Nodes and activation units form the layers of an ANN (hidden layers to be specific). Assuming information is flowing from left to right, the two outermost layers are called the input layer (leftmost) and the output layer (rightmost). The number of nodes in each of these layers depends on the available data. Any layers in-between the input and output layer are called hidden layers. The user decided the number and size (number of nodes) of a layer which can fine-tune for improving the model accuracy. Long-Short Term Memory (LSTM) models are often most effective in time series forecasting. The model can retain long-term data and removing the information that is not crucial for the predictions (Gers, 2001). LSTM is a recurrent neural network, numerous articles, papers, and reports have been published to verify its effectiveness. For the current project, most basic implementation of LSTM was done in *Python* using *Keras* (Herta, Christian, n.d.).

## Pre-Processing the Data

Before using the ANN model, the univariate time series, in the training and test set, was transformed into predictors and variable space. The regression equation for the LSTM model was,

Equation 4‑1

The number of predictors () was the same as the number of neurons in the input layer of the LSTM model. For this project, 20 past data points were used to predict the stock closing price on the 21st day, and so on. The data were scaled using

*Python’s Keras* package has in-built libraries for running an LSTM model. The number of layers and neurons in each layer are two crucial hyper-parameters. The model had two hidden layers with 50 neurons in each. The output unit had only one neuron, the next-day stock closing price prediction. The LSTM model was compiled using Adam’s optimizer (Diederik P. Kingma, 2015) criteria to reduce the mean squared errors.

The LSTM model fitted values and predictions are shown in Figure 4.1 and Figure 4.2, respectively. Based on the plots in Figure 4.1, the LSTM model fits were similar to a smoothed and shifted version of actual observations. In terms of MAPE measure, the LSTM model performed similarly to the ones discussed in Chapter 2 (naïve, ARIMA, etc.). Unlike time series analysis, the LSTM model predictions were not constant or linear, which is crucial for long-term forecasting as linear or constant functions often fail to capture the market movements.

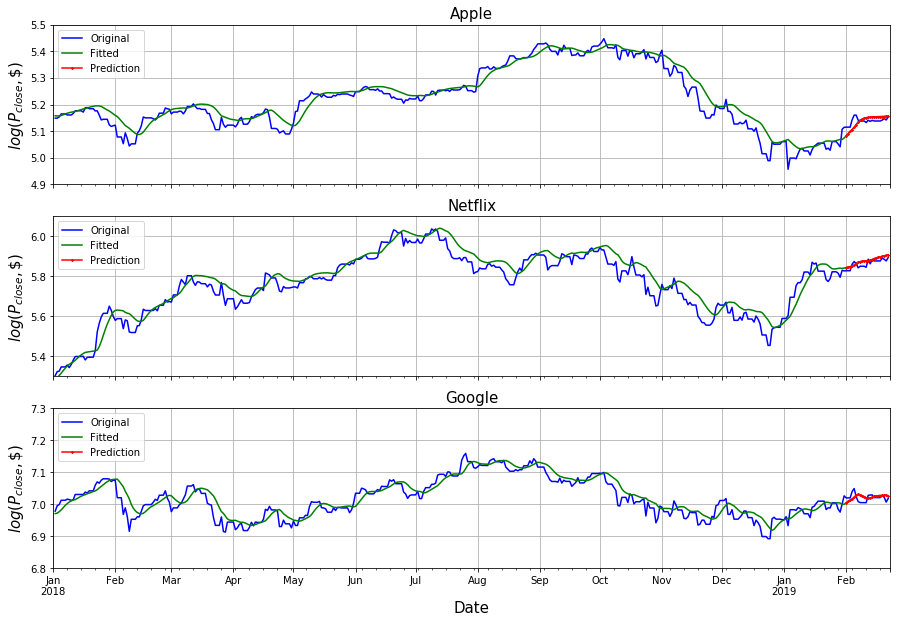


Figure 4.1 LSTM model: fitted values

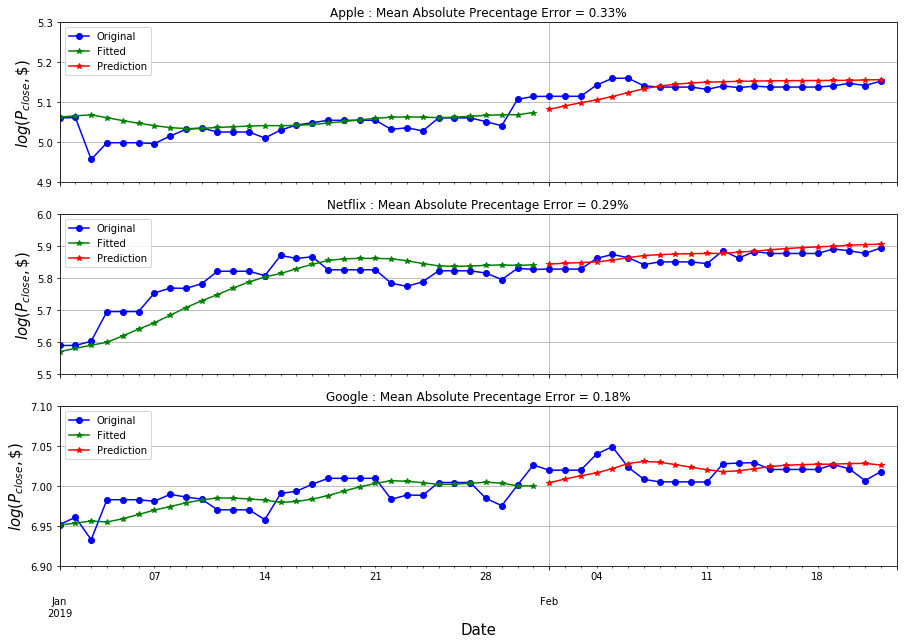


Figure . LSTM model: predictions.

Table 4.1 MAPE comparison of different models.

|  |  |  |
| --- | --- | --- |
|  | Naïve Method | LSTM Method |
| Apple | 0.47 | 0.33 |
| Netflix | 0.63 | 0.29 |
| Google | 0.14 | 0.18 |

# Conclusion

In this project, various forecasting models were studied, and their relative accuracy was compared using *mean absolute percentage error* of test data. A decade long time history of three technology stocks: Apple, Netflix, and Google were obtained from Nasdaq website (Nasdaq, 2019). Univariate closing price time series (measured daily) were selected as the variable of interest for this study. All results and visualizations were performed using *Python 3.7*, which has in-built libraries for time series analysis and predictions. The training and test set was the same for all models. The time series was log transformed to reduce the variance, and the data was pre-processed to fit the coding requirements of individual *Python* function.

The naïve method is one of the most straightforward and often time relatively accurate forecasting technique. The stock closing price of next day is assumed to equal to the one on previous trading day.

Figure 5.1 compares the MAPE histograms of the three stocks. The MAPE of Netflix was higher than Apple and Google for all methods, except LSTM due to the relatively continuous and linear trend of Apple and Google compared to Netflix (Figure 2.8). Figure 5.2 compares the performance of the forecasting method for individual stocks. The variance of MAPE using different ways was lowest for Google and highest for Netflix.

Table 5.1 tabulates the test MAPE measured for all models. Ensemble learning methods (Random Forest and XG Boost) outperformed all other models with minimum MAPE for all three stocks. Decision trees models were able to reduce the MAPE measure by 68%, 71%, and 64% for Apple, Netflix, and Google, respectively, compared to the benchmark model. The processing time for decision tree models was, however, more than Naïve model due to the number of hyper-parameters involved.

Exponential smoothing models (SES and Holt's trend methods) had the least improvement in MAPE measure over the benchmark model. ARIMA model that relies on the covariance between the past and present values performed slightly better than exponential smoothing methods. Due to the number of statistical restrictions (stationarity, normality, etc.) on input data, most time series needs transformations to obtain reliable ARIMA outputs. In practice, it is quite challenging to satisfy all ARIMA model requirements as it is hard to control the nature of real-world data. Forecast of exponential smoothing methods and ARIMA models was either (almost constant) or linear.

In comparison, the forecast plots of LSTM model was able to provide and smoother polynomial fit to the data, which is beneficial in long-term forecasting. With small test data (22 points), LSTM model performed similarly to time-series-analysis models in the current study.

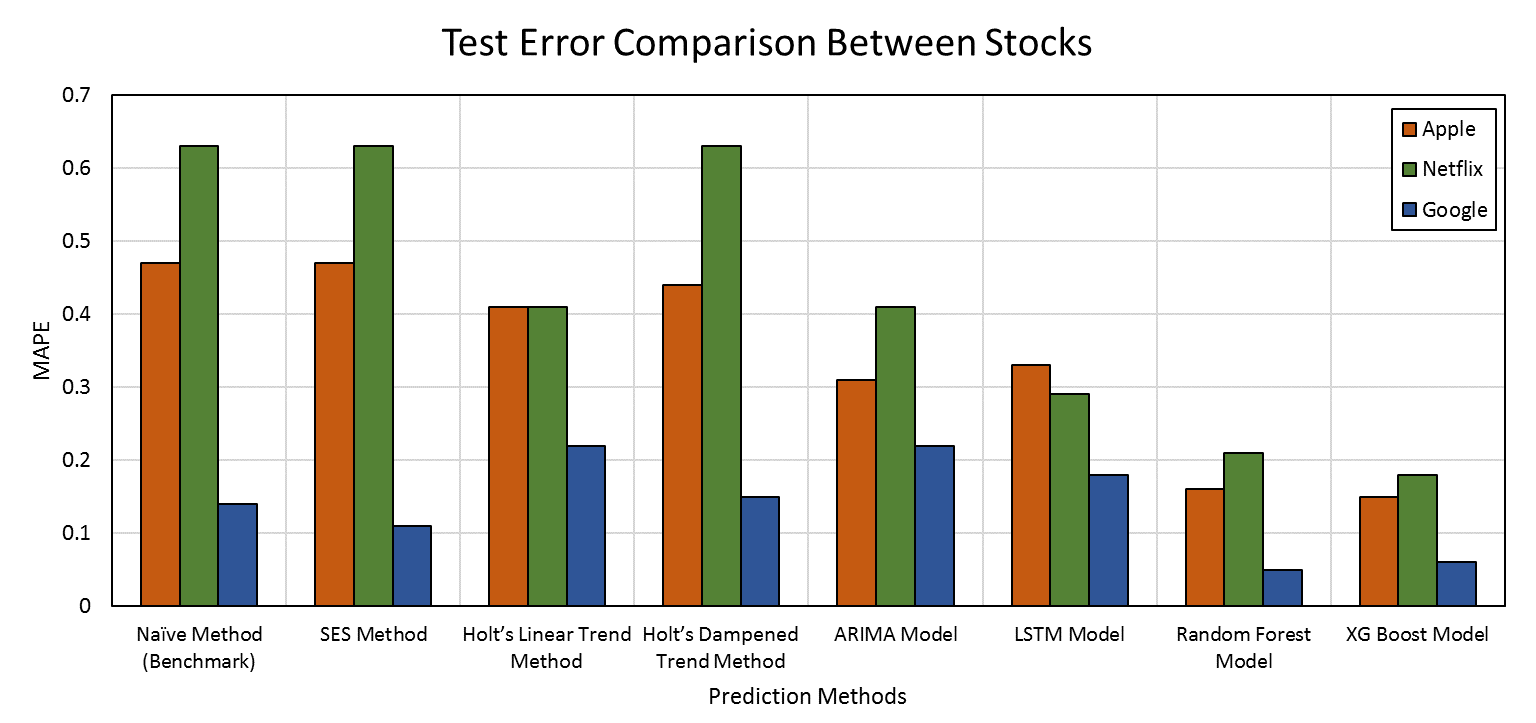


Figure . MAPE Comparison between Apple, Netflix, and Google.

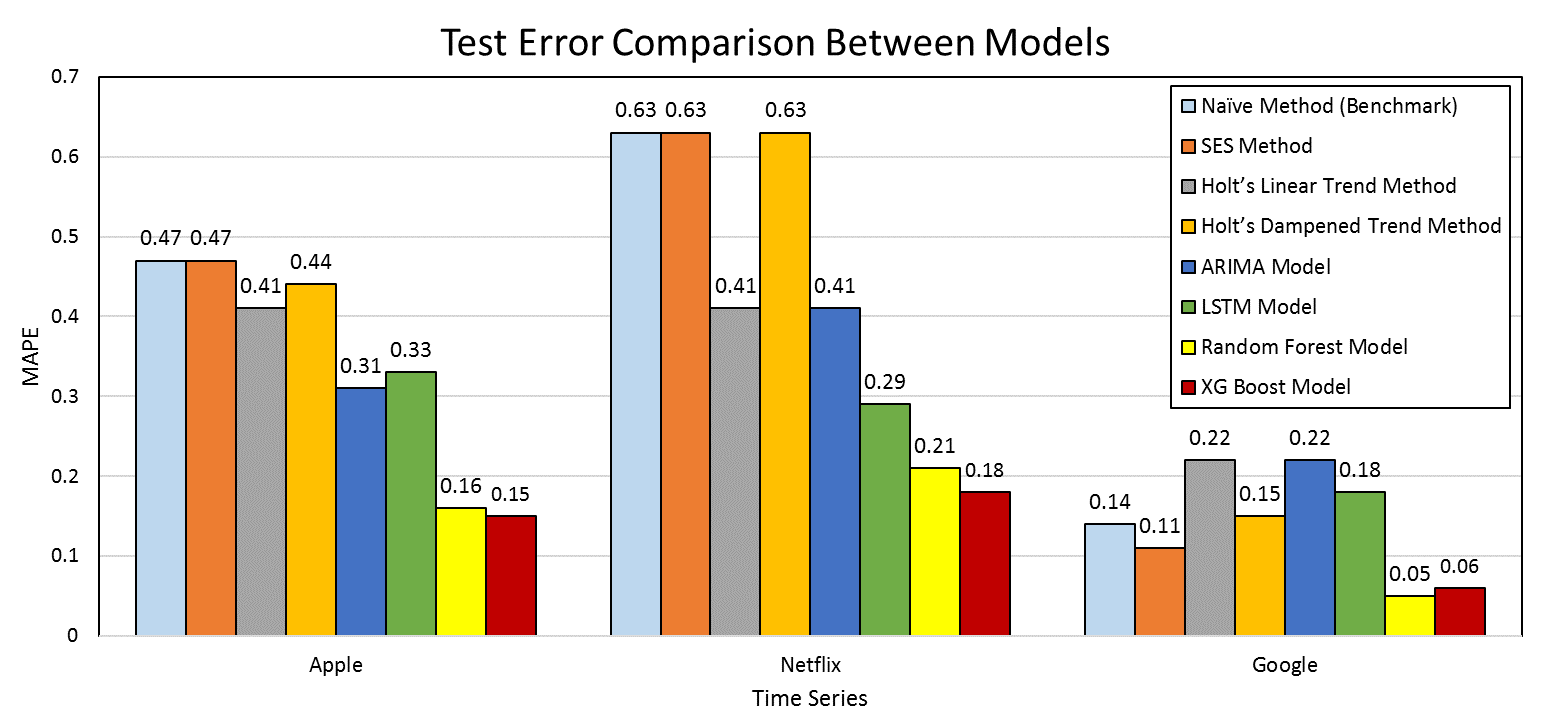


Figure . Prediction model performance for individual stocks.

Table 5.1 MAPE comparison of different models.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Apple | Netflix | Google |
| Naïve Method (*Benchmark*) | 0.47 | 0.63 | 0.14 |
| SES Method | 0.47 | 0.63 | 0.11 |
| Holt’s Linear Trend Method | 0.41 | 0.41 | 0.22 |
| Holt’s Dampened Trend Method | 0.44 | 0.63 | 0.15 |
| ARIMA Model | 0.31 | 0.41 | 0.22 |
| LSTM Model | 0.33 | 0.29 | 0.18 |
| Random Forest Model | 0.16 | 0.21 | **0.05** |
| XG Boost Model | **0.15** | **0.18** | 0.06 |
| Average MAPE | 0.34 | 0.42 | 0.14 |
| Standard Deviation | 0.12 | 0.19 | 0.07 |

# Appendix A

## Quandl API

As mentioned in chapter 2, due to the limited amount of data, the stock histories from Quandl API were not used for the final model. This appendix was prepared in the course work and submitted as one of the milestone projects. Data wrangling process was the first step of the project.

Quandl offers access to stock time histories. Project API key was obtained by signing up for a free online accounting. Code Section 1‑1 shows the code snippet used for the data access.

Code Section 1‑1

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import quandl  # *API key for data access*  quandl.ApiConfig.api\_key = "AAAAAAAAAAAA”  # *Accessing the APPl Stock histories, ticker = AAPL*  df = quandl.get("WIKI/AAPL")  # *Display the first five rows of the data frame*  df.head()  https://lh6.googleusercontent.com/hebLpkPBmsHt24wcQ1zvvcwlMceyAQILEXfbImEXgXK5_f6sdI_9qNy8j8Ag1fwiR0kTfez8RgjJLQOVkRRardIlczazat7Ohxqm6elM_Lkq2uyT243Sum_FMPopiprqquxP80SA |

Note the column 4 and 11, with column names ‘Close' and ‘Adj. Close'. The adjusted closing price is the closing price of a stock adjusted to include any changes that occurred on the day, for example, stock splits, dividend adjustments, right offerings, etc. Figure 1.3.1 shows the adjusted and actual closing price of Apple.

Apple went public on 12-12-1980 at $22.00 per share. Since then the stock has split 4 times: 7-for-1 basis on 06-09-2014, 2-for-1 basis on 02-28-2005, 06-21-2000 and 06-16-1987. These splits caused a relatively sharp decline in closing price afterward. A stock split does not affect the company's market capitalization, but it does affect the stock price. The adjusted closing price is thus a better analysis tool since it provides an accurate representation of the company's equity value. The closing price values obtained from the Nasdaq website were actually ‘adjusted closing price.'

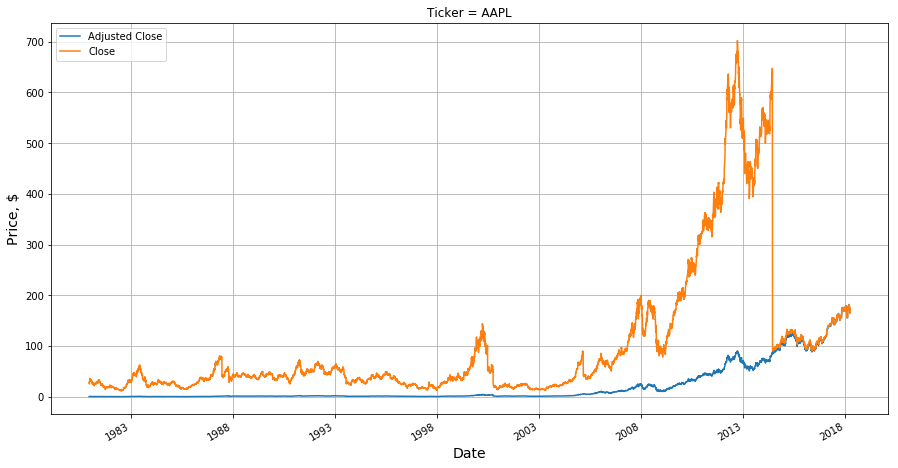


Figure A.1 Adjusted and actual closing price histories of Apple.

# Bibliography

Diederik P. Kingma, J. B. (2015). Adam: A Method for Stochastic Optimization. *3rd International Conference for Learning Representations*.

Gers, F. (2001). *Long Short-Term Memory in Recurrent Neural Networks.* Ecole Polytechnique Federale De Lausanne.

Herta, Christian. (n.d.). *How to Implement LSTM in Python with Theano*. Retrieved from http://christianherta.de: http://christianherta.de/lehre/dataScience/machineLearning/neuralNetworks/LSTM.php

Holt, C. C. (2004). Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 5-10.

Nasdaq. (2019, 02 23). Retrieved from Nasdaq Business: https://www.nasdaq.com/

Quandl. (2019, 02 20). Retrieved from quandl.com

Springboard. (2019). Retrieved from springboard.com

# 