



Stock price prediction using Machine Learning

Demetris Mesimeris

C1646577

Project supervisor: Dr. Daniela Tsaneva

Abstract

The result of a trader's decision-making process, to buy or sell a stock, has a direct impact on the complex behaviour of financial markets and data. The use of machine learning in the finance domain can be seen as a significant change through enabling the prediction of stock prices and patterns with the use of algorithms. The aim of this project is to consider different elements that impact the financial market and by implementing algorithms, explore the effectiveness of these tools to help traders in trying to predict public companies' stock prices.


I hereby declare:

that except where reference has clearly been made to work by others, all the work presented in this report is my own work.

that it has not previously been submitted for assessment.

that I have not knowingly allowed any of it to be copied by another student.

I understand that deceiving or attempting to deceive examiners by passing off the work of another as my own is plagiarism. I also understand that plagiarising the work of another or knowingly allowing another student to plagiarise from my work is against the University regulations and that doing so will result in loss of marks and possible disciplinary proceedings against me.

Signed 

Date04/11/2021.....

Contents

1. Introduction	4
2. Aims and Objectives.....	7
3. Background	8
4. Theory	11
4.1. Time Series	11
4.2. Model Training	11
4.3. Algorithms.....	13
4.3.1. Prophet	13
4.3.2. LSTM.....	16
4.3.3. Random forest	23
5. Implementation	29
5.1. Dataset acquisition	29
5.2. Prophet	30
5.3. LSTM.....	33
5.4. Random Forest Classifier	39
5.4.1. Sentiment analysis	39
5.4.2. Merge datasets	42
5.4.3. Price prediction	43
6. Results and Discussion	50
6.1. Prophet	50
6.2. LSTM.....	55
6.3. Random Forest Classifier	62
7. Conclusion.....	71
8. Further Work.....	73
9. References	75

1. Introduction

Stock markets influence both private investor's lives, and the relevant financial institutions that have the sole purpose of trading stocks. Ranging from economic upheaval to sustainable growth within a community, or in some cases, whole continents, the behavior of such markets can be vastly complex and difficult to predict. Due to the ever-increasing regulations and the constant 'watching eye' of the world-renowned regulations around such infrastructure, research around the movements of stock markets has been, considering the relation to prior years, easier with very passing day. As such, with each passing day, an increasing amount of people have been trying to analyze this very behavior.

Data provided from financial transactions may in turn affect a trader's decision-making, especially when it comes to the initialization of buying or selling stocks. Through modern means, and with ease of access to the Internet and the relevant data that may come with it, such information may influence a retail trader's decision.

The biggest incentive that drives retail investors to the stock market is profit maximization. On the stock market, individuals can buy and/ or sell shares of publicly traded companies, in a regulated environment. Many different tactics, ideologies, views, and ideas exist, no matter the scenario. The "push and pull" scenario where are people buying and selling stocks, is exactly what drives the relevant Demand & Supply of the specified markets, as well as setting its price. For one to maintain a profitable portfolio, he/she must be highly experienced and practiced, and have great knowledge in calculating futuristic outcomes.

Further from Demand & Supply, many other factors may affect the price of stocks, especially in the modern day. One of the most trending matters of this era is social responsibility and how

companies choose to “behave” in certain circumstances. For instance, when Activision-Blizzard faced huge internal HR (Human Resources) issues, their stock price was seen to plummet, as it was made quite clear that many investors did not want to be affiliated with a company that has such a demeaning attitude towards its own personnel. (GmbH, 2021) Similarly, a company that may opt into certain ‘green’ strategies may find the public on its side and experience an unexpected increase in price due to the rising demand from their proper actions.

Additionally, it has been found that political moves, natural disasters, and statements made by the market (i.e., an increase in regulations) may affect the price of certain indexes. (Wealth Advisors, 2021) The plethora of the matters, when complied into one, brings about the actual price of a stock. However, inflations may take place for short period of times, but the market will almost always set back itself into price equilibrium. Consequently, traders should always study the market and the specific share they are interested in extensively, and not be led by impulse-buying/ selling.

Furthermore, it can be considered unquestionable that market investors would dwell towards AI (artificial intelligence) and ML (machine learning) to predict outcomes, make their lives easier and hope for a deeper understanding of the markets. Examples of machine learning models include the Prophet, LSTM and Random Forest Classifier.

It is important to highlight the fact that AI and ML have an immense impact on the financial sector including, the speeding of the underwriting process, portfolio composition and optimization. (Thomas, 2021) Additionally, many firms in the Fintech industry, albeit a new industry, but mostly composed of experienced code-writers, are using the strength of AI/ ML to

automate tasks and processes that were seen in the past as time consuming and mundane.

Such actions were seen to be taken on by many banks and trading firms as well.

In relation to other investing matters, investing in stocks can be considered to have a lower risk than opening your own business, or chasing a high- salary career. Investors have a chance to realize gains through owning actual stocks of publicly traded companies through an exchange or over-the-counter market. Furthermore, traders can explore currency trading, equities, and derivatives over actual platforms. However, traders (especially ones of retail nature) can be impacted by the high volatility that exists in the market. Automated Trading Systems and their relevant regulatory control, 'Best Execution' (where brokers are required by law to make the most advantageous order execution for their clients without considering prospective profits/ losses for the company) can perform trades much faster and much more efficiently than actual humans. Risk strategies and risk-nullification (which can almost always, not be completely nullified) must be applied, which can only be derived from human judgement.

2. Aims and Objectives

The aim of this project is to test different algorithms and methods towards stock price prediction and compare their usability when trading stocks.

The first approach considers the prediction of stock prices by using only historical data. To achieve this, three different machine learning algorithms will be developed and trained using the same data. The algorithms will then be used to predict actual stock values to evaluate their accuracy.

The second approach will take into consideration twitter sentiment in combination with historical prices, to find whether the user sentiment affects the predictions positively. Datasets containing millions of user tweets throughout the years have been obtained, which will be processed and prepared to be used in a machine learning algorithm along with the historical data.

3. Background

Two types of research were proposed in the initial stages. One of Fama, E.F., relating to the Efficient Market Hypothesis (“EMH”), and one of Horne, J.C., & Parker, G.G, that of the Random Walk Theory. Both concepts have theorized that market price cannot be predicted since prices are affected by information that does not relate to historical prices.

EMH relies on the concept that newly released market information will impact the price of a stock, albeit it in a positive or negative matter. Additionally, it is stated that trading is done on the fair value of the instrument. Due to that, only by increasing the risk can a trader increase the yield, as nobody can sell the instruments at a price that is either undervalued or inflated.

There are three scenarios which affect market price:

1. Weak Form – Exclusively based on historical data.
2. Semi-Strong Form – Current public data along with historical data is used to create a basis.
3. Strong Form – Includes confidential data.

The movement of the prices are stated to be “either a result of new released information or a random move that would prevent prediction models from success.”

On the other hand, Horne, J.C., & Parker, G. G. have presented the Random Walk Hypothesis. This hypothesis provides a view that instrument prices change randomly and makes a point towards past price movement being independent of current movements. Up to a certain point, this viewpoint varies from EMH, as the emphasis is on the short-term patterns of the stock market. Based on the hypotheses, the stock market may indeed follow an unplanned and

irregular movement, where the accuracy of attempting to predict and speculate on such movements may be impaired by up to 50%.

In opposition to the theories, studies conducted in the recent years have shown that instrument market price movements can be predicted, up to a certain point. Two types of varying financial analysis predict these instrument prices:

1. Fundamental Analysis – Based on the health of the company in question, including both qualitative and quantitative factors through accounting formulas/ ratios (i.e., interest rate, return on assets, revenues, expenses, and price to earnings, amongst others). The specific target of this analysis was to verify the long-term sustainability and momentum of the specified company, for the purposes of a long-term investment.
2. Technical analysis – Based on time-series data, where both retail and professional traders analyse historical data on price movements and patterns, considering time as perhaps the most important framework in the prognosis of the future price.

Additionally, this analysis depends on three (3) decisive factors: (1) stock price movement [albeit seeming random at most times], (2) historical trends expected to repeat themselves during the progression of time, and (3) all relevant information regarding the instrument in question.

Through most studies, varying machine learning techniques have been utilized to forecast stock prices. Machine learning has demonstrated it is a valuable tool utilized in the task of prediction, as many techniques have been employed in the analysis of speculating a prevailing diagram/ pattern. Varying machine learning models and risk strategies have been utilized to predict at

the very least, the direction of the price for varying timeframes, rather than using diversified components that may affect the market in question.

4. Theory

4.1. Time Series

A time series is a data set that looks at a certain metric over a period of time. (Dinevski et al., 2021) An easy example of this would be the stock market. If we want to predict what a stock's next price will be, we will have to analyse historical prices data over a period of time to learn the patterns and estimate what will happen. Certain variables that could influence stock's price are the time of year, recurring or single events, investor's sentiment levels and more. A time series analysis looks at the relationship between the dependent variables in comparison to the independent variables to determine the impact of each. A time series analysis can also look at how timing impacts the dependent variable in the form of seasonality or overall upward or downward trends. (Dinevski et al., 2021)

Seasonality are trends within the data that occur at specific times. A time series analysis should be able to find that trend and incorporate it when forecasting the price. A time series should also be able to consider macro trends. (Dinevski et al., 2021) In the stock price, a macro trend would be if a given stock has been seeing an increase or decrease in price over time because of inflation.

Several machine learning algorithms can be implemented to help analyse time series data and be then used to predict the actual values.

4.2. Model Training

Machine learning models are not able to distinguish what is what when they are created. At first, the models are not able to do anything specific therefore they cannot provide a useful output. To make them learn how to do specific tasks, or predict an output, training takes place.

First, splitting the dataset in subsets is a good practice to do for each subset to be used for a different purpose while training. Some of the data is used for training the model while the rest is used to evaluate the trained model. The splitting task can be performed either by manually splitting the dataset or with the aid of Python libraries like Keras. When the training is complete and the data is evaluated, the programmer decides whether the model fit is ideal. What should be avoided, is overfitting or underfitting the model because these outcomes affect the predictability of the model.

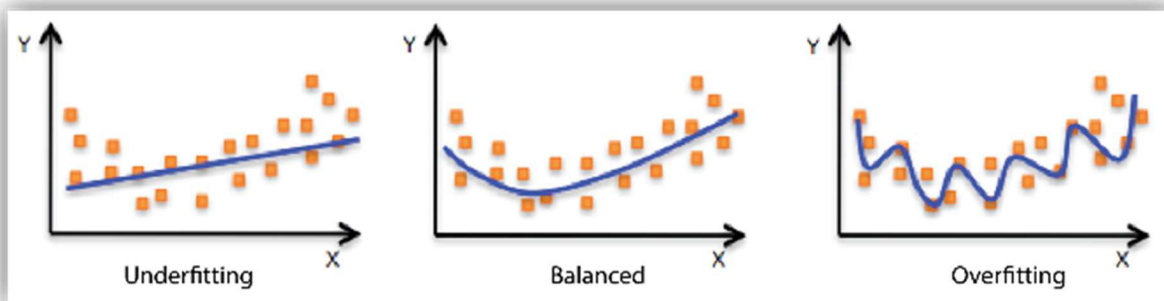


Figure 1. Training fitments (Amazon Learning, 2020)

Overfitting

When a model overfits, it means that the training was too accurate. In theory this should be a good thing since the target of a machine learning model is to achieve a high accuracy but having too high accuracy, or overfitting the model, results in a bad generalization. This will make the model perform well only on the training dataset given but it will have a substandard performance on the testing dataset, or any other data given. When overfitting exists the model instead of getting trained to do what it is meant to do, it focuses on learning the “noise” in the training data. In the case of stock price prediction, a model that is overfitted, it will only predict

patterns identical or like the training patterns, but it will perform poorly on the testing dataset or future predictions.

Underfitting

Underfitting a model is the opposite of overfitting. It means that the model's predictions do not match the training dataset and will thus not have a satisfactory performance on any other data given. This is usually the outcome when the model is not configured correctly, not enough training was performed or if there is not enough data. A model that underfits will not perform well in predicting testing data or future value predictions.

Balanced fitting

The best machine learning model fit is where the model's prediction trend is close to the training data but does not try to match every datapoint. To achieve this fit, the programmer should use good fitting practices and review the model after every training iteration with the help of metrics such as the mean squared error and plots. The sweet spot is the point just before the error on the test dataset starts to increase where the model has good skill on both the training dataset and the unseen test dataset. (Brownlee, 2021)

4.3. Algorithms

4.3.1. Prophet

Facebook's Prophet is an open-source algorithm for creating time-series models. It works exceptionally well at modelling time series that have multiple seasonalities and at its core the Prophet procedure is an additive regression model with four main components. (Facebook Research, 2021) It is the sum of three functions of time plus an error term: growth $g(t)$, seasonality $s(t)$, holidays $h(t)$, and error e_t :

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

Figure 2. Taylor SJ, Letham B. 2017

The Prophet algorithm was designed to automatically find the correct set of hyperparameters for the model and make predictions using the data. The Prophet’s default settings produce forecasts that are often accurate as those produced by skilled forecasters, with much less effort. (Facebook Research, 2021) In combination with an analyst, even one that has no experience or training in time series methods, the model can be further improved. See Figure 3 for the ideal learning loop.

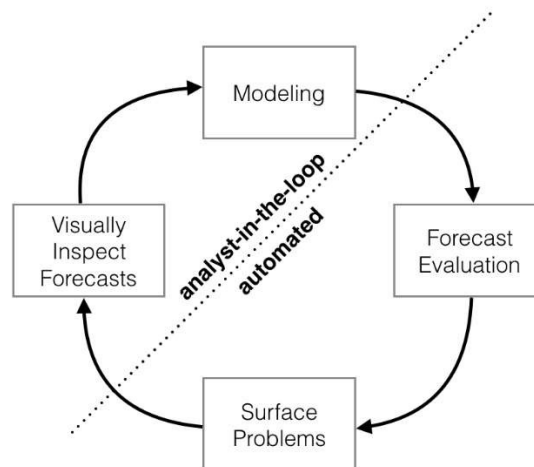


Figure 3. Most efficient forecasting process using the Prophet (Facebook Research, 2021)

4.3.1.1. The Growth Function

The growth function shapes the general trend of the data. The Facebook Prophet is created on the idea that the growth trend can be present at all points in the data or can be altered at what Prophet calls “change points.”

The changepoints are defined as the times where data changes direction and the Prophet can detect them automatically or the programmer can set them manually. The amount of data considered when changepoints are detected can also be adjusted.

The growth function is consisted of three options:

- Linear Growth: It is the default parameter for the algorithm which uses a set of piecewise linear equations with differing slopes between the changepoints. (Krieger, 2021) When the growth is linear, the growth term looks similar to the typical $y = mx + c$ equation for the straight line, with the difference that the slope(m) and offset(c) will change at each changepoint.
- Logistic Growth: It is used when the time series dataset has a cap or a floor which results in values becoming saturated because they cannot surpass a maximum or minimum value. When the growth is logistic, the growth term looks similar to the typical equation of a logistic curve (see Figure 4) with the difference that the carrying capacity (C) will vary as a function of time and the growth rate (k) and the offset(m) are variable and will change value at each change point. (Krieger, 2021)

$$g(t) = \frac{C(t)}{1 + e^{-k(t-m)}}$$

Figure 4. Prophet's growth function (Krieger, 2021)

- Flat: Finally, a flat trend can be selected where there is no growth over time. When a flat growth function is selected, the growth will be a constant value.

4.3.1.2. *The Seasonality Function*

The seasonality function is simply a Fourier Series as a function of time. (Krieger, 2021)

The Fourier series sum can estimate almost any cyclical pattern or in the case of Facebook

Prophet, the seasonality in the data. The seasonality function mathematical equation is defined below:

$$s(t) = \sum_{n=1}^N (a_n \cos(\frac{2\pi nt}{P}) + b_n \sin(\frac{2\pi nt}{P}))$$

Figure 5. Prophet's seasonality function equation (Taylor SJ, Letham B. 2017)

If the analyst has the required understanding of the Fourier Series, they can choose the number of terms in the series, or otherwise known as the Fourier order. The Facebook Prophet can be used even with no knowledge on the Fourier series though since it can automatically find the optimum Fourier order.

4.3.1.3. *The Holiday/Event Function*

The Facebook Prophet is able to alter the forecasting when there is a holiday or an important event. This is possible with the use of lists of dates containing the holidays or events, and when the specific dates are present in the dataset it alters the value based on historical movements on the same holiday or event. The algorithm can also recognize a range of days around the aforementioned dates.

4.3.2. LSTM

Neural networks are a type of machine learning which are related to biology and neuroscience, which were created to mimic how the human brain works. The human brain is consisted out of

approximately eighty-six (86) billion neurons that are interconnected with approximately 10^{14} - 10^{15} synapses (Karpathy, n.d.), or in other words connections, and together they put together the human nervous system.

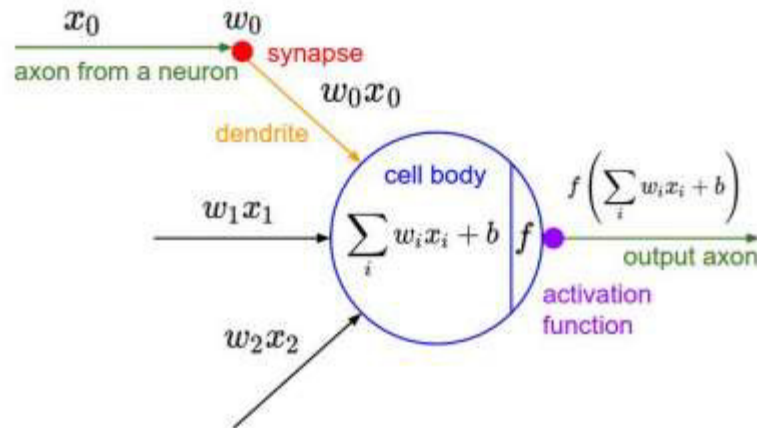


Figure 6. Mathematical representation of a neuron (Karpathy, n.d.)

In mathematical terms, the neuron is consisted out of the axons of a previous neuron which carry the signals represented by 'x' and are then multiplied by the weights represented by 'w,' to give the input dendrite ' w_0x_0 '. Every input dendrite is added up and their sum make the cell body of the new neuron. The aim of this process is that the weights are adjustable and manipulate the influence of one neuron to another. When the sum in the cell body is bigger than a threshold, the neuron is activated and sends a signal along its output axon. Only the frequency of the activation gives out information in the mathematical model instead of taking into consideration precise timings of activations. Based on this rate code interpretation, we model the firing rate of the neuron with an activation function f , which represents the frequency of the spikes along the axon. (Karpathy, n.d.)

The LSTM is type of neural networks called Recurrent Neural Networks. The acronym stands for Long short-term memory (LSTM) and it is a popular type of network used in the field of deep learning. Because RNNs can learn the result from the previous round, this gives RNN a time series context. However, RNNs can only read the results from the previous round. For stock price data, there are thousands of points of data. Therefore, only remembering the previous round result is not enough, because the stock price might also be influenced by the price weeks ago or months ago. LSTM is introduced into this report because LSTM can address this problem. The LSTM architecture fixes the lack of ability to learn long-term dependencies by introducing a memory cell that can preserve states over long periods of time.

Normally neural networks operate via feedforwarding, but the LSTM has feedback connections instead. Other than being able to process single data points (e.g., images), it can also process whole series of data (such as speech or video inputs). The LSTM networks can save information over a period or in other words they have a memory capacity which is especially useful when dealing with Time-Series.

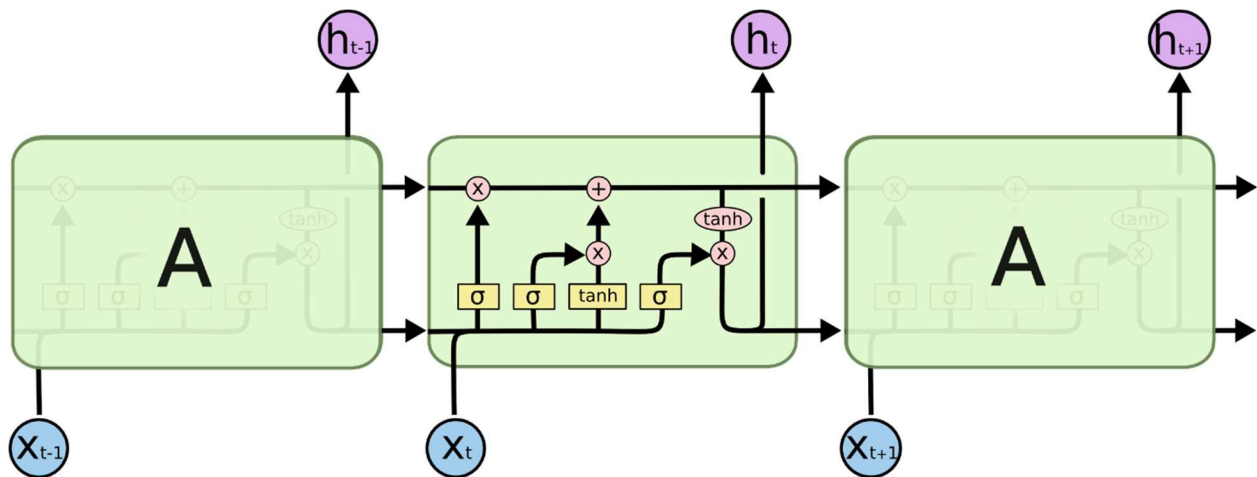


Figure 7. LSTM network (Olah, 2021)

Looking at Figure above, the output of an LSTM at a particular point in time is dependent on three things:

- The current long-term memory of the network - known as the cell state which is the horizontal line at the top of the Figure.
- The output at the previous point in time - known as the previous hidden state which is the h_{t-1} . (“LSTM Networks | A Detailed Explanation | Towards Data Science”)
- The input data at the current time step which is x_t .

When using an LSTM model, the information that will be stored and discarded can be decided. This is done with the use of gates which can be thought of as filters. Each gate is their own neural network layer.

Gates in LSTM are the sigmoid activation functions. This means they output a value between 0 or 1 and in most of the cases it is either 0 or 1. (Thakur, 2021) The sigmoid function is used for the LSTM gates because only positive values should be output between 0 and 1 to help decide on which features should be kept or dropped. ‘0’ means that the gates are blocking everything and ‘1’ means that the gates are allowing everything to pass through it.

Tanh is a non-linear activation function which adjusts the values going across the network, making sure the values remain between -1 and 1. For information to stay intact, it is required to use a function with a second derivative that can last longer.

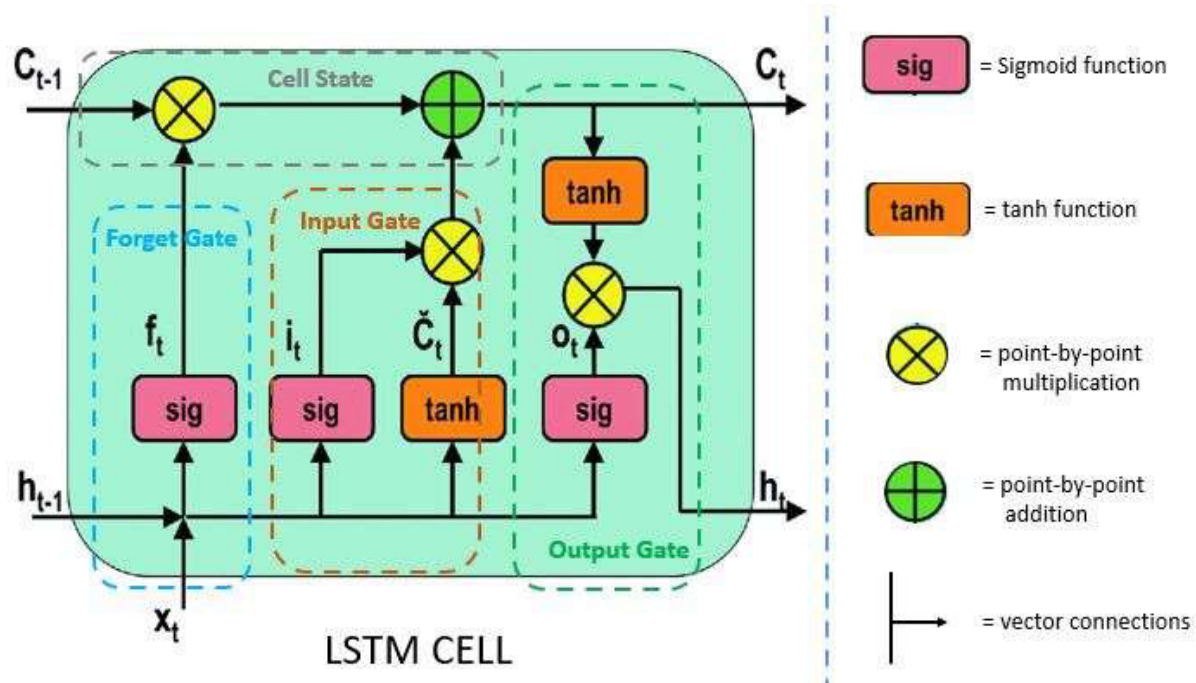
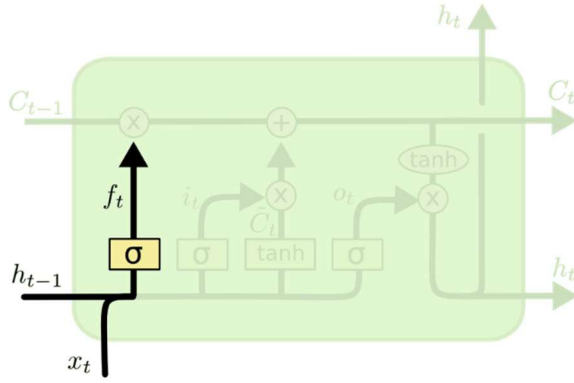


Figure 8. A detailed view of an LSTM Cell (Singhal and RNN, 2021)

Three types of gates engage in each LSTM model with the goal of controlling the state of each cell.

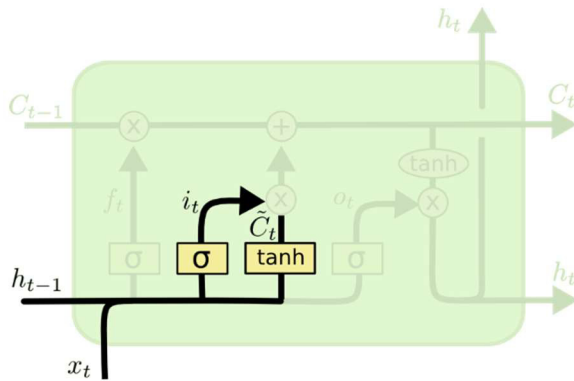
Forget Gate: It decides the information that will be kept or dropped by looking at the output of the previous hidden state, h_{t-1} , and the input at the current timestep, x_t . It then provides a number between 0 and 1 for every number in the cell state C_{t-1} , where 1 means ‘keep all the information’ and 0 means “ignore all this information”.



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

Figure 9. LSTM cells forget gate (Olah, 2021)

Input Gate: A sigmoid layer that is followed by a tanh layer decide the new data that is going to be stored in the cell. The sigmoid layer, named as ‘input gate layer’ selects which values will be modified. The tanh layer that follows, creates a vector, \tilde{C}_t with the new possible values that could be added to the cell state. (Saiful and Hossain, 2020)



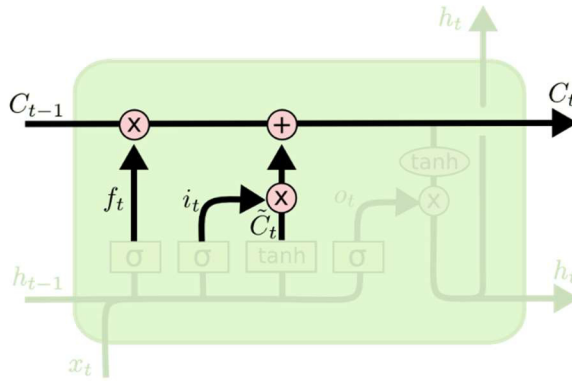
$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Figure 10. LSTM cell input gate (Olah, 2021)

After processing the information through the Forget and Input gates, the new cell state C_t is updated. To do that, the old state is multiplied by f_t and discards the processed information.

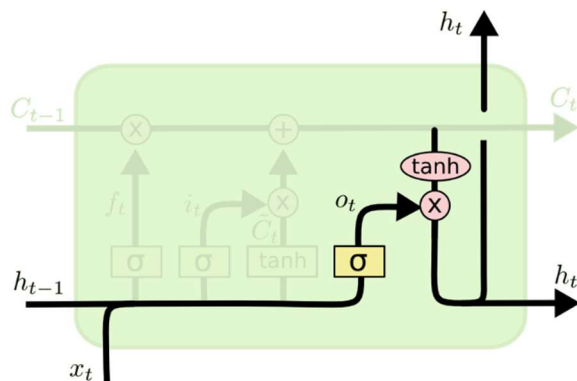
Then the new possible values in vector \tilde{C}_t , are scaled based on how much it was decided to update each value, i_t .



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 11. LSTM cell, cell state update (Olah, 2021)

Output Gate: Decides the final output. "The yielded value will be based on the cell state along with the filtered and newly added data. (Goel and Bajpai, 2020) A sigmoid layer is run first to decide the parts of the cell state that are going to be output. Then the cell state goes through a tanh function and is multiplied by the output of the sigmoid function to output only the parts that were decided.



$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Figure 12. LSTM cell output gate (Olah, 2021)

4.3.3. Random forest

The Random Forest algorithm is a supervised machine learning algorithm which is primarily used to solve regression and classification problems. It utilizes ensemble methods, and it is applied in several industries such as banking to predict behaviours and outcomes.

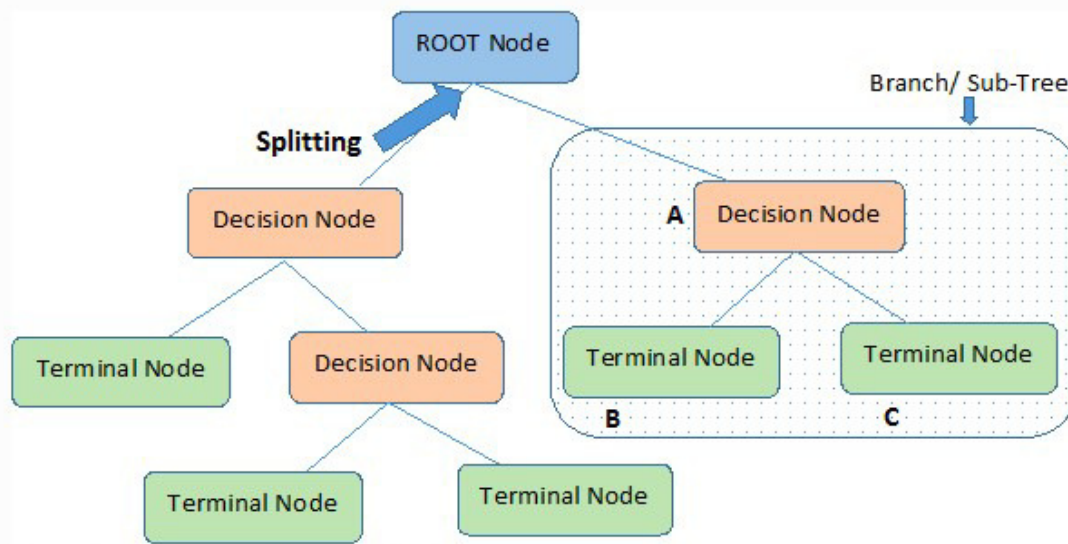
Ensemble methods that were mentioned above, is a machine learning method that combines many models to create a better predictive model. Ensemble means a group or a set and in this situation, a group of decision trees and grouped together they are called 'Random Forest.'

Individual models have less accuracy compared to ensemble models since they compile the results from the individual models and provide an outcome.

In simpler terms, a random forest is consisted out of several decision trees which helps to deal with the issue of overfitting in decision trees. Decision trees are a nonparametric machine learning algorithm that is very flexible and is subject to overfitting training data. (Brownlee, 2021) The reason a random forest deals with the overfitting issue is that takes the average of all the decision trees' predictions, and it cancels out the biases. These decision trees are built randomly by choosing random features from the given dataset.

4.3.3.1. Decision trees

Decision trees, have a hierarchical structure that looks like a tree with branches. The branches of a decision tree function as nodes and a certain decision is finally taken by going through these nodes which are based on the responses garnered from to the parameters related to the nodes.



Note:- A is parent node of B and C.

Figure 13. Decision Tree terminology (Chauhan, 2021)

- Root Node: This is the whole dataset that will be divided into smaller subsets.
- Splitting: it is the procedure of breaking a node into sub-nodes.
- Decision Node: This is a sub-node that will be further split in other sub-nodes.
- Leaf / Terminal Node: A terminal node is a sub-node that does not split any further.
- Branch / Sub-Tree: This is a subdivision of the entire tree which is therefore called a branch or sub-tree.
- Parent and Child Node: When a node is split in sub-nodes, it is called a parent node while the sub-nodes it split into are called child nodes.

As a simple example, the classic Dog or Cat problem will be used to explain how a decision tree without branches works.

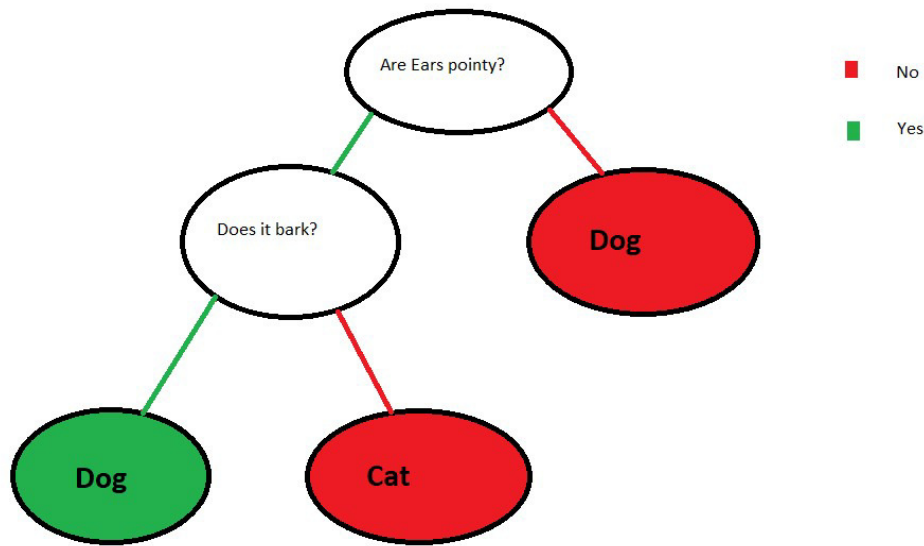


Figure 14. Decision Tree Dog/Cat example (Shastri, 2021)

In this example, the decision tree's root node is the one that asks if the animal's ears are pointy. The root node is split in a decision and a terminal node. If the answer to the root node is 'No' the algorithm decides that the animal is a dog, if the answer is 'Yes' it goes on to ask another question if the animal barks to help the algorithm decide. If the answer is 'Yes' then the decision tree's decision is that the animal is a dog but if the answer is 'No' then the decision is cat.

4.3.3.2. *Ensemble Learning*

Ensemble learning gives credence to the idea of the "wisdom of crowds," which suggests that the decision-making of a larger group of people is typically better than that of an individual expert. (Education, 2021) Similarly, ensemble learning refers to several learning models working together to achieve a better prediction compared to one. When a single model, or also known as a weak learner, is used it is susceptible to high bias which leads to problems when trying to achieve a good prediction in machine learning. Yet, if several weak learners are aggregated,

they can yield a better model performance because this combination reduces the bias. Decision trees are the most common example of ensemble learning because this algorithm is prone to overfitting and underfitting, so several decision trees are combined to form a stronger algorithm, also known as the Random Forest algorithm.

There are several types of ensemble learning and BAGGing, or Bootstrap AGGregating is one of them, it gets its name because it combines Bootstrapping and Aggregation to form one ensemble model (Lutins, 2021). BAGGing is a method that is used to deal with a noisy dataset by reducing variance. It works by taking several random samples of data from the training set, with the selected datapoints being able to be chosen more than once. When the samples are generated, the multiple weak learners are trained individually and when they all produce a prediction, the average value of these predictions is taken to produce a more accurate final value. A classification example of BAAGing, is a sample dataset with five (5) bagged decision trees, predicting between three (3) colours: Blue, Red, Green. If three (3) or more trees' output is Blue, then the final prediction of the BAGGed model will be Blue. This is illustrated in Figure 15.

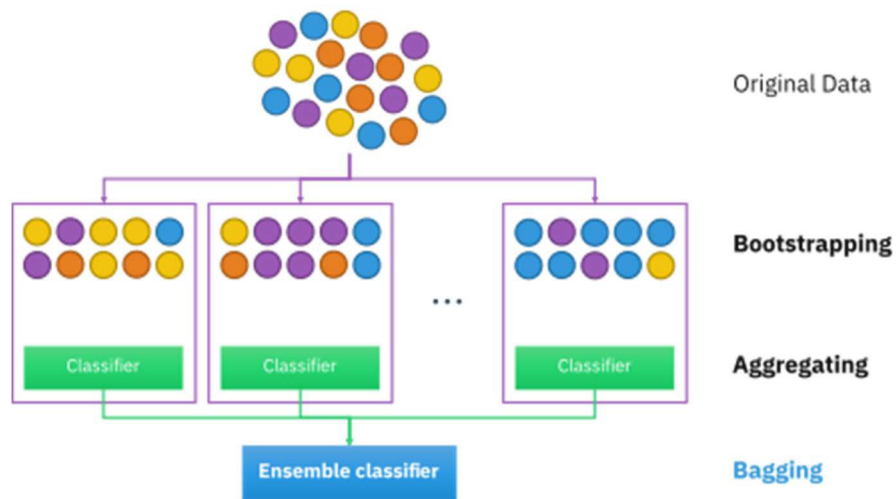


Figure 15. BAGGING illustration (Bootstrap aggregating - Wikipedia, 2021)

Random Forest models are like BAGGING with a slight tweak. BAGGED decision trees can choose between the full range of features when deciding where to split and how to make decisions. (Lutins, E., 2021) This can lead to decision trees breaking off the data to subsamples similar to each other because they attempt to minimize error using an algorithm which chooses which variables to split on. This is where Random Forest differs to BAGGED decision trees since this model decides how to break the data in subsamples based on a random selection of features. Rather than leaving each tree to decide on which features to use and end up having similar features throughout each node, the Random Forest model creates a structure with differentiated trees with distinctive features. This technique provides a better accuracy when predicting values since it leads to a greater ensemble to aggregate over. An illustration of a Random Forest model can be seen in Figure 16.

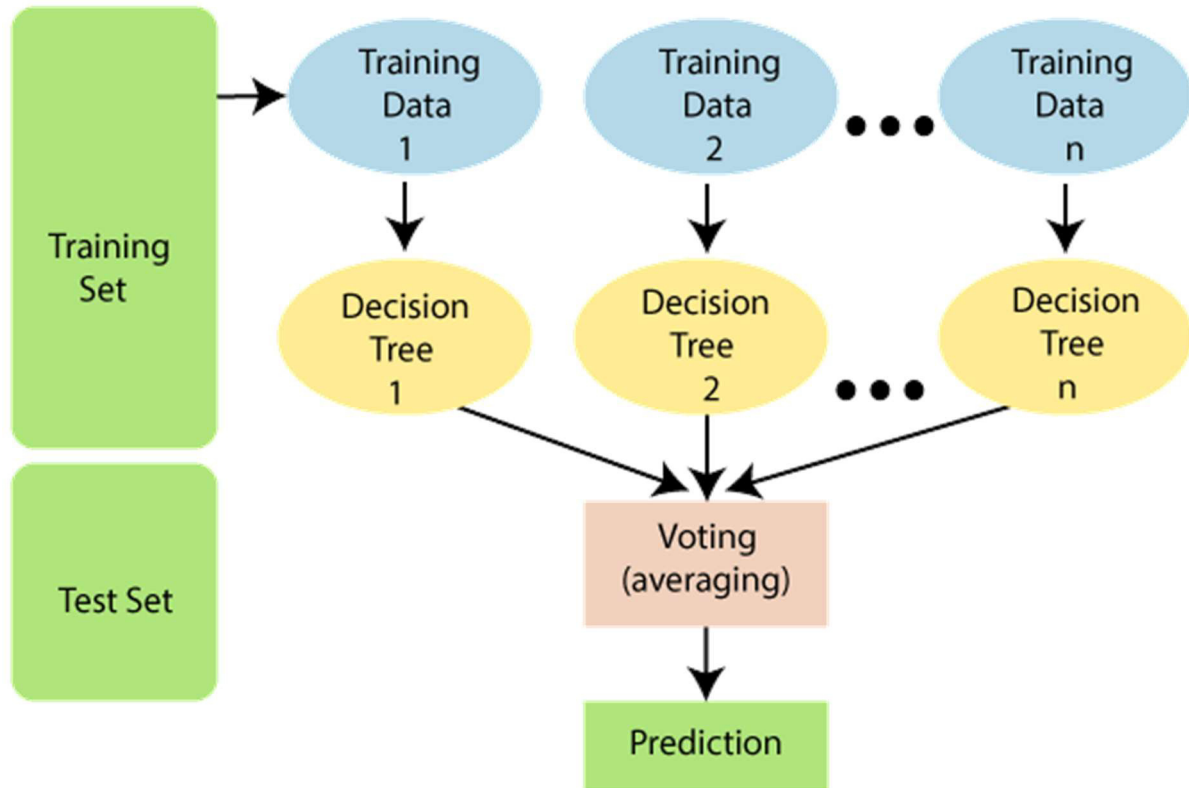


Figure 16. Illustration of a Random Forest algorithm (Javatpoint, 2021)

5. Implementation

5.1. Dataset acquisition

Both approaches to the project required datasets of stocks historical prices. The datasets were acquired from the Yahoo Finance website by searching for the specific stock and accessing the 'Historical Data' tab. (See Figure 17)

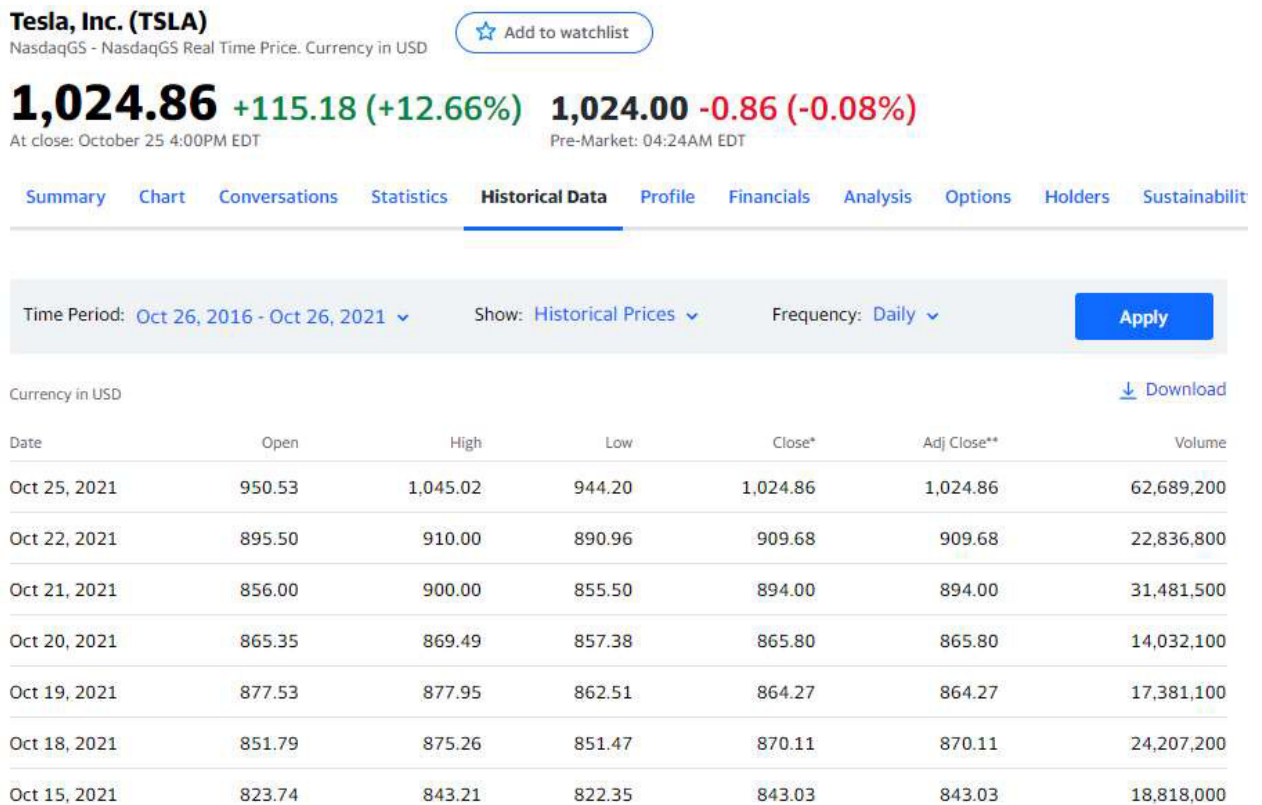


Figure 17. Yahoo finance's historical data tab

The datasets contained the Date, Open price, High price, Low price, Close price, and Adjusted Close price as well as the Volume. Regarding the columns, the only useful data is the Date, adjusted close price which is the close price, adjusted for splits and dividend and/or capital gain distributions and the Volume.

In this project, the datasets used was Apple Inc. past 5-year historical prices. This range has been used because of the explosion of prices in technology related stocks in the

past 5 years. Also, for the purpose of consistency and comparability, the historical price dataset used was only for Apple Inc., but any stock can be used simply by changing the first cell of code where the file is loaded.

5.2. Prophet

The first and most simple implementation was the Prophet algorithm. To use Prophet for predicting values, the algorithm needs the data to be in the form of a Pandas data frame. After the data is loaded in this form, a Prophet object is defined and configured. Then the 'fit' function is called to pass time series data and fit the dataset. The arguments taken by the object that are used to configure the model are type of growth, type of seasonality and more.

Also, for the data frame to be passed in the model, a specific format must be set. The first column has to be named 'ds' and include the date-times. It is also required to be converted to date-time objects. The second column must be named 'y' and include the observations.

The 'fbprophet' library was installed and after importing all the required packages the .csv file containing the historical data for the Apple Inc. stock were loaded using Pandas. Then the Date and Adjusted Close columns were filtered to be renamed as 'ds' and 'y' respectively as the algorithm requires. (See Figure 18)

```
df = pd.read_csv("AAPL.csv")
df
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2016-10-28	28.467501	28.802500	28.362499	28.430000	26.662807	151446800
1	2016-10-31	28.412500	28.557501	28.299999	28.385000	26.620605	105677600
2	2016-11-01	28.365000	28.442499	27.632500	27.872499	26.139956	175303200
3	2016-11-02	27.850000	28.087500	27.807501	27.897499	26.163403	113326800
4	2016-11-03	27.745001	27.865000	27.387501	27.457500	25.882967	107730400
...
1253	2021-10-21	148.809998	149.639999	147.869995	149.479996	149.479996	61421000
1254	2021-10-22	149.690002	150.179993	148.639999	148.690002	148.690002	58839600
1255	2021-10-25	148.679993	149.369995	147.619995	148.639999	148.639999	50720600
1256	2021-10-26	149.330002	150.839996	149.009995	149.320007	149.320007	60893400
1257	2021-10-27	149.360001	149.729996	148.490005	148.850006	148.850006	55951800

1258 rows × 7 columns

```
# Select only the date and price
df = df[["Date", "Adj Close"]] # select Date and Price
# Rename the columns to "ds" and "y" for model fitting
df = df.rename(columns = {"Date": "ds", "Adj Close": "y"})
df.head(5)
```

	ds	y
0	2016-10-28	26.662807
1	2016-10-31	26.620605
2	2016-11-01	26.139956
3	2016-11-02	26.163403
4	2016-11-03	25.882967

Figure 18. Inspection of the loaded data and filtered/renamed columns while using the Prophet.

Now the data frame is ready to be fitted in the Prophet algorithm. Since the Prophet was not explored in depth, only the default settings were used and since the dataset contained daily data, the 'daily_seasonality' setting was set to True. (See Figure 19)

```

from fbprophet import Prophet
model = Prophet(daily_seasonality = True)
# fit the model using the dataframe
model.fit(df)

```

Figure 19. Prophet model

The next step was to plot the model predictions by indicating the days in the future to be predicted. (See Figure 20)

```

future = model.make_future_dataframe(periods=365) #number of days in future
prediction = model.predict(future)
model.plot(prediction)
plt.title("Prediction of the Stock Price using the Prophet")
plt.xlabel("Date")
plt.ylabel("Close Stock Price")
plt.legend(['Train', 'Val', 'Predictions'], loc = 'lower right')
plt.show()

```

Figure 20. Prophet's results visualization

Then, from the predicted output of the Prophet, the range of predictions for the historical prices was gathered, called 'yhat_lower' and 'yhat_upper', filtered and renamed appropriately. (See Figure 21)

```

predictions = prediction[['ds', 'yhat_lower', 'yhat_upper']]
predictions = predictions.rename(columns = {'ds': 'Date',
                                             'yhat_lower': 'Low Prediction',
                                             'yhat_upper': 'High Prediction'})
predictions[:1258]

```

Figure 21. Prophet's range of predictions of the historical prices

Next, by using a similar approach, the future predictions are shown. (See Figure 22)

```

futurepred = prediction[['ds', 'yhat_lower', 'yhat_upper']]
futurepred = predictions.rename(columns = {'ds': 'date',
                                             'yhat_lower': 'Low Prediction',
                                             'yhat_upper': 'High Prediction'})
futurepred[1258:]

```

Figure 22. Prophet's range of predictions of the future prices

Finally, an inspection of the predicted values, called 'yhat' in the models' predictions, against the actual values was performed by creating a data frame comparing them in two columns called 'Predictions' for the predicted values against 'Actual Values' for the Adjusted Closing values from the dataset provided. (See Figure 22)

```
valid = prediction.yhat[:1258]
valid = pd.DataFrame(valid)
valid['Adj Close'] = df['y']
valid = valid.rename(columns = {'yhat' : 'Predictions',
                                'Adj Close' : 'Actual Values'})
valid
```

Figure 22. Prophet's predictions against the actual values

5.3. LSTM

To begin the implementation of the LSTM model, a file with the stock's historical prices was loaded and inspected as a Pandas data frame. (See Figure 23)


```
#Read the data from .csv file
df = pd.read_csv("AAPL.csv")
df
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2016-10-28	28.467501	28.802500	28.362499	28.430000	26.662807	151446800
1	2016-10-31	28.412500	28.557501	28.299999	28.385000	26.620605	105677600
2	2016-11-01	28.365000	28.442499	27.632500	27.872499	26.139956	175303200
3	2016-11-02	27.850000	28.087500	27.807501	27.897499	26.163403	113326800
4	2016-11-03	27.745001	27.865000	27.387501	27.457500	25.882967	107730400
...
1253	2021-10-21	148.809998	149.639999	147.869995	149.479996	149.479996	61421000
1254	2021-10-22	149.690002	150.179993	148.639999	148.690002	148.690002	58839600
1255	2021-10-25	148.679993	149.369995	147.619995	148.639999	148.639999	50720600
1256	2021-10-26	149.330002	150.839996	149.009995	149.320007	149.320007	60893400
1257	2021-10-27	149.360001	149.729996	148.490005	148.850006	148.850006	55951800

1258 rows × 7 columns

Figure 23. Inspection of the loaded data while using the LSTM

Then a new dataset that contained only the Adjusted Closing price column was created since that is the only data needed in the file. Next, using the dataset, by convention 80% of all the data was taken to be used later as the training data, with the use of the math package and the 'ceil' and 'len' functions. Furthermore, the data was scaled and transformed using the 'MinMaxScaler' to make all the data values range from 0 to 1. (See Figure 24)

```
#Number of training data
training = math.ceil(len(dataset) * 0.8)

#Scale the data to become values from 0 to 1
scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(dataset)
```

Figure 24. Getting the training dataset and scaling the data.

Then using the training dataset, the data is split in in X and y which are the dependent and independent training variables, respectively. Using a 'for' loop, until the script reaches the end of the data, sixty (60) values are appended in the 'X_train' dataset while the 61st value is appended in the 'y_train' dataset. (See Figure 25)

```
#Create and split the training dataset
train_data = scaled_data[0:training, :]
X_train = []
y_train = []

for i in range(60, len(train_data)):
    X_train.append(train_data[i-60:i])
    y_train.append(train_data[i,0])
```

Figure 25. Splitting and appending the training dataset to the X and y variables

The training datasets are then converted to 'Numpy' arrays so they could then be reshaped to become 3-dimensional to fit the required LSTM model dimensions. (See Figure 26)

```
#Convert to numpy arrays
X_train, y_train = np.array(X_train), np.array(y_train)

#Reshape data to 3 dimensions
X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))
X_train.shape

(948, 60, 1)
```

Figure 26. Data is converted to numpy arrays and reshaped to become 3D

The next step was building the LSTM network which was done using the 'Sequential' function from Keras models and the Dense, Dropout and LSTM layers from the Keras layers package. Four LSTM layers with fifty (50) units were added with Dropout to avoid overfitting. Finally, the output layer is added using Dense. (See Figure 27) This model has over 71,000 trainable parameters and the final shape is in one (1) dimension. (See Figure 28)

```

from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout

#Build the LSTM model

model = Sequential()
#Adding the first LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True, input_shape = (X_train.shape[1], 1)))
model.add(Dropout(0.2))
# Adding a second LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
# Adding a third LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True))
model.add(Dropout(0.2))
# Adding a fourth LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50))
model.add(Dropout(0.2))
# Adding the output layer
model.add(Dense(units = 1))
model.summary()

```

Figure 27. LSTM model creation

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 60, 50)	10400
dropout (Dropout)	(None, 60, 50)	0
lstm_1 (LSTM)	(None, 60, 50)	20200
dropout_1 (Dropout)	(None, 60, 50)	0
lstm_2 (LSTM)	(None, 60, 50)	20200
dropout_2 (Dropout)	(None, 60, 50)	0
lstm_3 (LSTM)	(None, 50)	20200
dropout_3 (Dropout)	(None, 50)	0
dense (Dense)	(None, 1)	51
Total params: 71,051		
Trainable params: 71,051		
Non-trainable params: 0		

Figure 28. LSTM model summary

Now that the LSTM model was built, it was compiled using the the “Adam” optimizer which is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. (Kingma and Ba, 2014) Also the loss function was set which is the parameter that the model would attempt to reduce. In this case, mean squared error (MSE) function was selected. (See Figure 29)

```
#Compile the model  
model.compile(optimizer='adam', loss='mse')
```

Figure 29. LSTM model compilation

Next was the model training which was defined as ‘history’ to be accessed later. That used the ‘X’ and ‘y’ training datasets that were created in the previous steps along with the ‘batch_size’ and ‘epoch’ parameters. (See figure 30)

The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. (Brownlee, 2021) Training the model for a high number of epochs usually means that the model will be trained better.

The batch size is the number of sub samples given to the network after which parameter update happens. (Keshari, 2019) A very low amount of batch size would increase the training time and could lead to overfit, while a very high amount would lead to lower training times but also underfitting and lower quality of training.

```
#Train the model  
history = model.fit(X_train, y_train, batch_size=1, epochs=1)
```

Figure 30. LSTM model training

With the model trained, the next step was to create a ‘test_data’ array containing the rest 20% of the scaled values that were not used for training. These are the values that the model has never seen before. To do this, a similar approach to the training dataset was used with a ‘for’ loop appending the past sixty (60) scaled values in ‘X_test’. For ‘y_test’ though, the values used were not scaled since these are the actual values that the model will attempt to predict. Next, the ‘X_test’ data was converted to a NumPy array and reshaped to be used in the LSTM model. (See Figure 31)

```

test_data = scaled_data[training - 60: , :]
X_test = []
y_test = dataset[training:, :]

for i in range(60, len(test_data)):
    X_test.append(test_data[i-60:i, 0])

```

```

X_test = np.array(X_test)

```

```

X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

```

Figure 31. Testing dataset creation, conversion to numpy array and reshape

With the model built and trained, the only thing that was left to do was predicting the actual values. This was done by using the 'predict' function and then inverse transforming the scaled data to turn it back to the actual values. Also, the root mean squared error is found to evaluate the model, the lower the root mean squared, the better the model fit. (See Figure 32)

```

rmse = np.sqrt(np.mean(((predictions - y_test)**2)))
rmse

```

Figure 32. Root mean squared error

Finally, a plot is done to visualize the effectiveness of the model by plotting the training, testing, and predicted values against the dates. (See Figure 33)

```

train = data[:training]
valid = data[training:]
valid['Predictions'] = predictions

plt.figure(figsize=(16,8))
plt.title("Prediction of the Stock Price using LSTM")
plt.xticks(np.arange(0,1259, 200), df['Date'][0:1259:200])
plt.xlabel('Date')
plt.ylabel('Adjusted Price USD ($)', fontsize=18)
plt.plot(train['Adj Close'])
plt.plot(valid[['Adj Close', 'Predictions']])
plt.legend(['Train', 'Testing', 'Predictions'], loc = 'upper left')
plt.show()

```

Figure 33. LSTM results visualization

5.4. Random Forest Classifier

In contrast to the previous algorithms, the Random Forest algorithm used twitter sentiment analysis in combination with historical data. This was done to explore the potential effect tweets have on the stock prices.

To begin, an attempt was made to scrape twitter sentiment manually with the use of 'tweepy' package. After creating a Twitter Developer account and getting the required keys, the program could scan Twitter for a word and return specified number of tweets. This was deemed ineffective though because of the limits Twitter has imposed on data scraping. Only one thousand tweets could be gathered every 15 minutes which is not enough.

Eventually, it was decided to use datasets that were available online and contained millions of tweets from 2015 to 2019 for the big five (5) companies: Apple, Tesla, Amazon, Google, and Microsoft. This would provide better results since Machine Learning is a data hungry method and it would save countless of hours waiting for the created script to scrape this data manually.

5.4.1. Sentiment analysis

To begin, the twitter sentiment should be processed in an appropriate .csv file that will then be merged with another .csv file containing the historical data of the specific stock. Since we had three (3) files containing different data, all the files were loaded. (See Figure 34)

```
company = pd.read_csv('Company.csv')
company_tweets = pd.read_csv('Company_Tweet.csv')
user_tweets = pd.read_csv('Tweet.csv')
```

Figure 34. Loading files containing tweets

Then a new data frame was created called 'tweets' with the user tweets merged with the company tweets using the 'tweet_id' column that was common. (See Figure 35)

```
tweets = pd.merge(user_tweets, company_tweets, on='tweet_id', how='inner')
tweets
```

Figure 35. Merging the files

	tweet_id	writer	post_date	body	comment_num	retweet_num	like_num	ticker_symbol
0	550441509175443456	VisualStockRSRC	1420070457	lx21 made 10,008onAAPL -Check it out! htt...	0	0	1	AAPL

Figure 36. Inspection of merged datasets

The next step was to create a new column in the data frame called 'Date' to contain a readable date format since the date in 'post_date' was unreadable. This was done with the use of pandas 'to_datetime' and lambda functions. (See Figure 37)

```
tweets['post_date'] = pd.to_datetime(tweets['post_date'], unit='s')
tweets['Date'] = pd.to_datetime(tweets['post_date'].apply(lambda date: date.date()))
tweets['Date'].head()

0    2015-01-01
```

Figure 37. Converting Date to readable format

The data frame was then tidied up by dropping the columns that were of no use for the remaining of the script. (See Figure 38)

```
tweets = tweets.drop(['tweet_id', 'writer', 'post_date', 'comment_num', 'retweet_num', 'like_num'], axis=1)
tweets.head()
```

	body	ticker_symbol	Date
0	lx21 made 10,008onAAPL -Check it out! htt...	AAPL	2015-01-01

Figure 38. Dropping of irrelevant columns

Then with the use of the 'ticker_symbol' column, each company was labeled accordingly and the number of tweets for each were counted. (See Figure 39)

```
msft = tweets[tweets['ticker_symbol'] == 'MSFT']  
aapl = tweets[tweets['ticker_symbol'] == 'AAPL']  
tsla = tweets[tweets['ticker_symbol'] == 'TSLA']  
amzn = tweets[tweets['ticker_symbol'] == 'AMZN']  
goog = tweets[tweets['ticker_symbol'] == 'GOOG']
```

```
tweets.ticker_symbol.value_counts()
```

Figure 39. Count of tweets related to each company

For the next step, the Afinn package was used to score the tweets sentiment. (See Figure 40)

Afinn is a sentiment analysis package that contains 3300+ words with a polarity score associated with each word. (Wings, 2021) Due to the large amount of data, this step took a lot of time to run.

```
from afinn import Afinn  
  
afinn = Afinn()  
msft['ts_polarity'] = msft['body'].apply(lambda tweet: afinn.score(tweet))  
aapl['ts_polarity'] = aapl['body'].apply(lambda tweet: afinn.score(tweet))  
tsla['ts_polarity'] = tsla['body'].apply(lambda tweet: afinn.score(tweet))  
amzn['ts_polarity'] = amzn['body'].apply(lambda tweet: afinn.score(tweet))
```

Figure 40. Sentiment analysis of tweets using Afinn

The next task was to create a calendar using Pandas that would only take into consideration the sentiment on days that the stock market was open, thus US federal holidays and weekends should be excluded from the data frame. Also, the appropriate time zone should be used. (See Figure 41)

```
import pandas_market_calendars as mcal

# Create a calendar
nyse = mcal.get_calendar('NYSE')
nyse.tz.zone

early = nyse.schedule(start_date=min(msft['Date']).date(), end_date=max(msft['Date']).date())
early
```

Figure 41. Calendars containing weekends and US Federal holidays

Since there were multiple tweets in a single day, the tweets were grouped by date and a new column was created denoting the volume of tweets on the specific date. Then the data that is not in the calendar specified before is dropped. The last step was to export the data frame to a .csv file. (See Figure 42)

```
msft['Holiday'] = msft['Date'].isin(early.index)
msft['twitter_volume'] = msft.groupby('Date')['body'].transform(len)
new_msft = msft.drop(msft[msft.Holiday==False].index).groupby('Date').mean().drop(['Holiday'], axis=1)
new_msft.to_csv('msft-twitter.csv')
```

Figure 42. Preparing data to be exported to .csv format

5.4.2. Merge datasets

To merge the datasets of sentiment analysis and historical prices, the files were first loaded and inspected. After processing the twitter sentiment file, there was now an equal number of rows in both files and since the 'Date' columns were the same, the files were merged using the date. A new data frame called 'merged' now contained all the data required to proceed with the price prediction script and it was finally exported to a csv file. (See Figure 43)

```
merged = stock_prices.merge(twitter_sentiment, on = "Date")
merged
```

	Date	Open	High	Low	Close	Volume	ts_polarity	twitter_volume
0	02-01-15	41.180947	41.851703	41.075039	41.269203	27913900	0.289720	107

Figure 43. Merging twitter sentiment with historical stock prices

5.4.3. Price prediction

Now that the required file is available, the algorithm can be implemented. After the file is loaded, a new data frame is created where the 'High' and 'Low' price columns are dropped since only the close price will be used and the date column is set as the index. (See Figure 44)

```
dataframe = merged_dataframe[["Date", "Close", "Volume", "ts_polarity", "twitter_volume"]]  
dataframe.set_index("Date", inplace = True)  
  
dataframe
```

	Close	Volume	ts_polarity	twitter_volume
Date				
02-01-15	41.269203	27913900	0.289720	107

Figure 44. Dropping irrelevant columns

The next step was to visualize the sentiment polarity which was done with the help of a histogram. (See Figure 45)

```
dataframe['ts_polarity'].plot(kind='hist',range=(-2,3),bins=40,edgecolor='black');
```

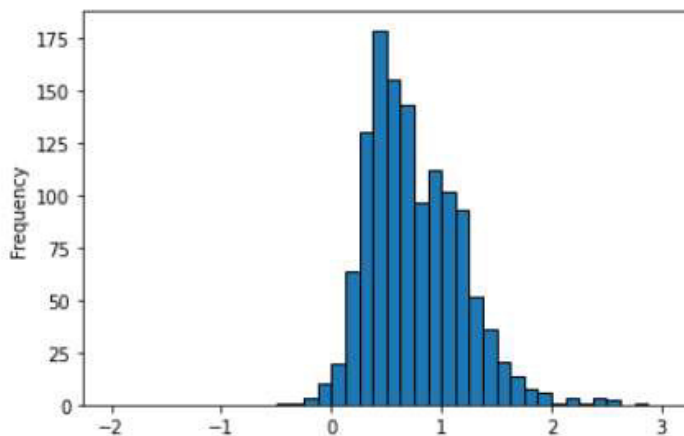


Figure 45. Visualization of tweet polarities with the help of a histogram

The visualization was important before proceeding to the next step where the sentiment would manually be classified as 'Positive', 'Negative' or 'Neutral'. In the Microsoft example, the most

frequent polarity was around 0.4, therefore the 'Neutral' threshold was considered anything from 0.3 to 0.75 while anything lower or more than these values was considered 'Negative' or 'Positive' respectively. The upper and lower thresholds were set after running the final script several times until the highest accuracy was achieved. Finally for this step, a new column was added to the data frame called 'Sentiment' labeling each day's sentiment accordingly. (See Figure 46)

```

POSITIVE_SENTIMENT_THRESHOLD = 0.75
NEGATIVE_SENTIMENT_THRESHOLD = 0.3
sentiments = []
for sentiment_score in dataframe["ts_polarity"]:
    if sentiment_score >= POSITIVE_SENTIMENT_THRESHOLD:
        sentiments.append("Positive")
    elif sentiment_score <= NEGATIVE_SENTIMENT_THRESHOLD:
        sentiments.append("Negative")
    else:
        sentiments.append("Neutral")
dataframe["Sentiment"] = sentiments

```

Figure 46. Setting sentiment thresholds and labeling them accordingly in a new column

	Close	Volume	ts_polarity	twitter_volume	Sentiment
Date					
02-01-15	41.269203	27913900	0.289720	107	Negative

Figure 47. Inspection of the updated dataframe

Another visualization was carried out in the form of a pie chart to inspect the sentiment analysis for the stock. (See Figure 48)

```
dataframe["Sentiment"].value_counts()

Neutral      565
Positive     552
Negative     141
Name: Sentiment, dtype: int64

dataframe.Sentiment.value_counts().plot(kind='pie', autopct='%1.1f%%')
```

Figure 48. Counting sentiments in each category and plotting a pie chart with the data

The next task was to create a stock trend column that would indicate whether the price has risen or fallen compared to the previous day. To do this, the price difference was calculated between the two days and then using NumPy this was converted to 1 for 'Rise' and 0 for 'Fall'. (See Figure 50) The first row of the data frame must be dropped since there is no previous value to be compared to. (See Figure 49)

```
dataframe["Price Difference"] = dataframe["Close"].diff()
dataframe.dropna(inplace = True)
```

Figure 49. Creating a new column called "Price Difference" with the price difference compared to the previous day and dropping the first row

```
import numpy

RISE = 1

FALL = 0

dataframe["Stock Trend"] = numpy.where(dataframe["Price Difference"] > 0, RISE, FALL)
```

Figure 50. Creating a new column called “Stock Trend” saying if the price Rose or Fall compared to the previous day

	Close	Volume	ts_polarity	twitter_volume	Sentiment	Price Difference	Stock Trend
Date							
05-01-15	40.889698	39673900	0.625000	112	Neutral	-0.379505	0

Figure 51. Inspection of the updated dataframe

The next step was to perform binary encoding of sentiment which was done so the machine learning model would be able to distinguish what ‘Positive’, ‘Negative’ and ‘Neutral’ sentiment is. This was done by creating a new data frame, splitting the ‘Sentiment’ column in three other columns using the Pandas function ‘get_dummies’. Each column contained only 0 and 1 where 0 means False and 1 means True. (See Figure 52)

```
new_dataframe = dataframe[["Close",
                           "Volume",
                           "twitter_volume",
                           "Sentiment",
                           "Stock Trend"]]

new_dataframe = pd.get_dummies(new_dataframe, columns = ["Sentiment"])
```

Figure 52. Creating a new data frame containing dummy columns of “Sentiment”

	Close	Volume	twitter_volume	Stock Trend	Sentiment_Negative	Sentiment_Neutral	Sentiment_Positive
Date							
05-01-15	40.889698	39673900	112	0	0	1	0
06-01-15	40.289543	36447900	78	0	0	0	1
07-01-15	40.801445	29114100	88	1	0	1	0

Figure 53. Inspecting the new dataframe

Now that the sentiment was binary encoded, the data could be labeled as 'X' which is the input and independent variable and 'y' which is the dependent variable. Therefore, X was just a copy of the 'new_dataframe' that contained all the information required to make the prediction just without the 'Stock Trend' column since this is what the algorithm needs to predict. (See Figure 54)

```
X = new_dataframe.copy()
X.drop("Stock Trend", axis = 1, inplace = True)
```

Figure 54. Creating variable X with a copy of the new data frame

Next was 'y' which will contain the aim of the algorithm which in this case is to predict the stock trend in 0 or 1 for Fall and Rise respectively. Therefore, 'y' will contain the values of 'Stock trend' column of 'new_dataframe' reshaped from -1 to 1. (See Figure 55)

```
y = new_dataframe["Stock Trend"].values.reshape(-1, 1)
```

Figure 55. Creating variable y using the "Stock Trend" column and reshaping the values

The datasets will then be split into training and testing. This means that some data will be used to train the machine learning algorithm while the rest will be used to test the accuracy of it with

data that will be omitted from the training and the program has never seen it before. By convention, 80% of the data is used for training and 20% is used for testing. (See Figure 56)

```
SPLIT = int(0.8 * len(X))  
  
X_train = X[:SPLIT]  
X_test  = X[SPLIT:]  
  
y_train = y[:SPLIT]  
y_test  = y[SPLIT:]
```

Figure 56. Splitting the variable values to training and testing datasets.

Now that the data is split, the next step was to scale the data in 'X' since there is a large difference in the values as the price is in the tens while volume is in the millions, and this would cause a bias towards volume rather than the stock price. To do that, the 'StandardScaler' class was used from the Sci-Kit package to transform the data to be on the same scale without affecting the distribution. (See Figure 57)

```
from sklearn.preprocessing import StandardScaler  
  
scaler = StandardScaler()  
  
X_scaler = scaler.fit(X_train)  
  
X_train_scaled = X_scaler.transform(X_train)  
  
X_test_scaled  = X_scaler.transform(X_test)
```

Figure 57. Scaling and transforming the training and test values using 'StandardScaler'

Finally, the data processing is complete and next is the building of the machine learning model.

The Random Forest Classifier was provided by the Sci-Kit package and only the number of

estimators and random state arguments had to be set manually. Then, with the use of the 'fit' function to train the model, the 'X_train_scaled' was input along with 'y_train' followed by the ravel function to flatten the array since the model requires a 1-dimensional list. (See Figure 58) The final arguments for number of estimators and random stage were decided after acquiring the best accuracy from several runs of the model.

```
from sklearn.ensemble import RandomForestClassifier

NUMBER_OF_ESTIMATORS = 1500

RANDOM_STATE = 80

classifier = RandomForestClassifier(n_estimators = NUMBER_OF_ESTIMATORS,
                                  random_state = RANDOM_STATE)

classifier = classifier.fit(X_train_scaled,
                           y_train.ravel())

classifier

RandomForestClassifier(n_estimators=1500, random_state=80)
```

Figure 58. Building the Random Forest model

Then the model attempted to predict the stock trend values that were omitted from the training phase, with the use of the 'predict' function and 'X_test_scaled' data. The predictions were then compared to the actual values in a binary format where 0 meant Fall in price and 1 meant Rise in price. (See Figure 59)

```
predictions = classifier.predict(X_test_scaled)
```

```
pd.DataFrame({  
    "Prediction": predictions,  
    "Actual": y_test.ravel()  
})
```

	Prediction	Actual
0	0	0
1	0	0
2	1	1

Figure 59. Making the built Random Forest model predict the testing data set and comparing results

Finally, with the use of the 'accuracy_score' class from the Sci-Kit package, the accuracy of the model was found. (See Figure 60)

```
from sklearn.metrics import accuracy_score  
  
accuracy = accuracy_score(y_test, predictions)
```

Figure 60. Finding the accuracy of the model

6. Results and Discussion

6.1. Prophet

For the Prophet algorithm only the default settings were used, it was the simplest and easiest algorithm to implement whilst also providing high accuracy results. There was no data processing required other than renaming the two columns while the model configuration only took one argument. Then by using the 'make_future_dataframe' function of the library, the Prophet could predict specified number of days in the future using the historical data. The

predictions were provided as a range of values rather than exact values. This was illustrated while plotting the results and in the data frame of the predictions. (See Figure 61 & Figure 62)

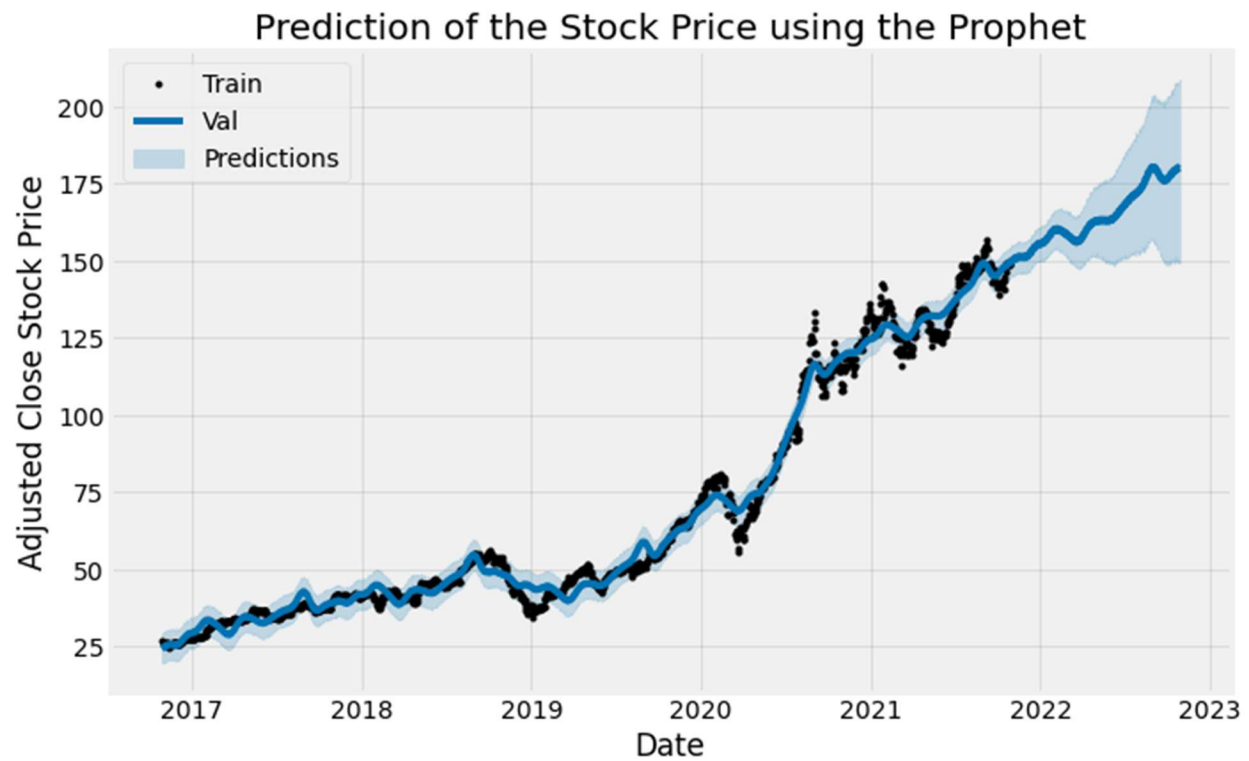


Figure 61. Plotting the results of the Prophet

	Date	Low Prediction	High Prediction
0	2016-10-28	19.352831	29.380554
1	2016-10-31	19.645757	29.376943
2	2016-11-01	19.568309	29.381132
3	2016-11-02	20.006414	29.750708
4	2016-11-03	19.589357	29.521987
...
1253	2021-10-21	144.155649	154.091566
1254	2021-10-22	143.999396	153.836296
1255	2021-10-25	143.926442	153.818502
1256	2021-10-26	144.217232	154.235808
1257	2021-10-27	144.474397	154.552506

1258 rows × 3 columns

Figure 62. Inspecting the low and high predictions of the model on each day

The range of predictions for the dates that the data was provided remains constant to ten (10) dollars difference from high to low predictions and is very close to the validation data. When the actual values are inspected against the predicted values of the model, the predictions are always within a range of two (2) dollars difference to the actual values. (See Figure 63)

	Predictions	Actual Values
0	24.291092	26.662807
1	24.616990	26.620605
2	24.851589	26.139956
3	24.926376	26.163403
4	24.978207	25.882967
...
1253	148.839686	149.479996
1254	148.810213	148.690002
1255	149.142966	148.639999
1256	149.377252	149.320007
1257	149.452850	148.850006

Figure 63. Models' predictions compared to actual values

For the future predictions though, this range increased proportionally to the number of days in the future. This means, the more distant future, the bigger the range of predicted values which made these predictions become useless.

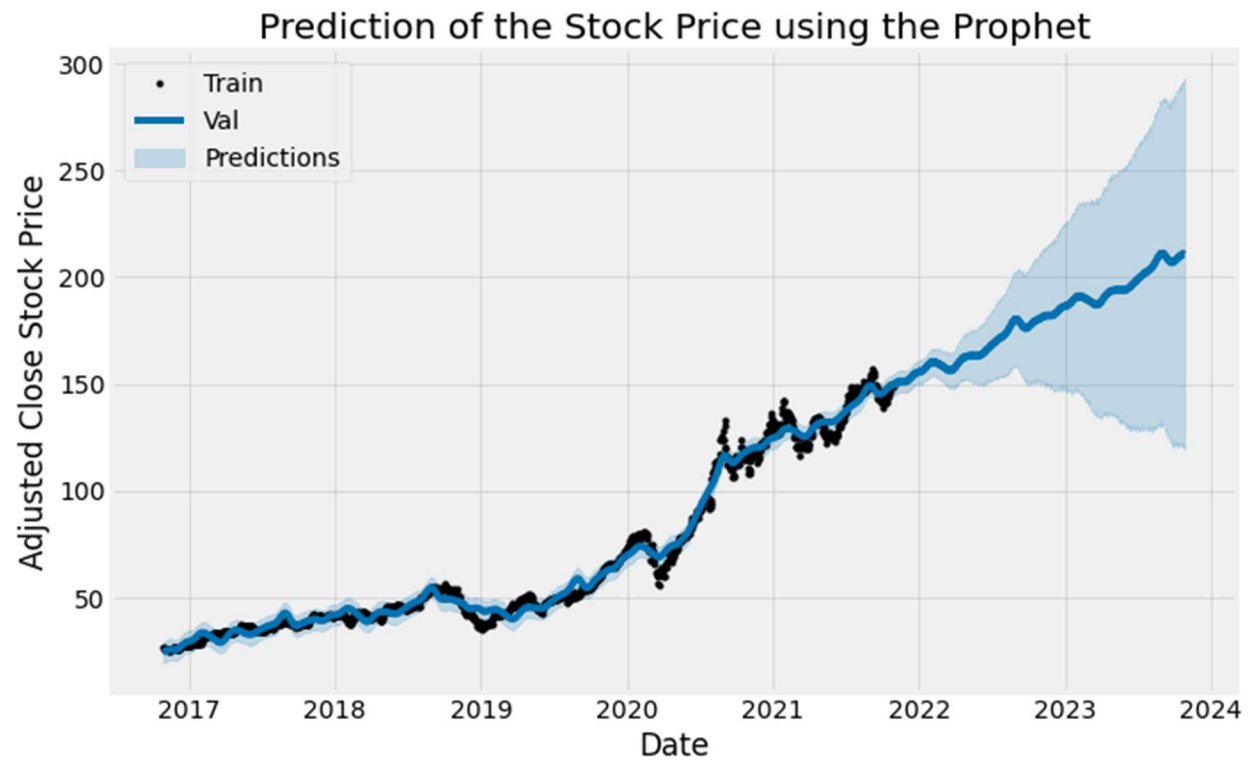


Figure 64. Attempting to predict prices in more distant future

	Date	Low Prediction	High Prediction
1258	2021-10-28	144.828669	154.674990
1259	2021-10-29	144.441807	154.454232
1260	2021-10-30	145.271690	155.058912
1261	2021-10-31	145.041385	155.241555
1262	2021-11-01	144.749237	154.639312
...
1618	2022-10-23	152.416870	209.004047
1619	2022-10-24	150.691337	209.615251
1620	2022-10-25	151.046117	209.792587
1621	2022-10-26	150.964336	209.248567
1622	2022-10-27	151.404475	211.127064

365 rows × 3 columns

Figure 65. Range of predictions in more distant future

As it is seen from the data frame, the first days of the future predictions have a difference of 10 dollars from the low to the high prediction, but this difference increases to 60 dollars from low to high predictions a year from now. (See Figures 64 & 65)

Therefore, it can be deduced that the Prophet algorithm is a useful tool since it is easy to implement and provides good short-term predictions, but it cannot be trusted for long-term predictions.

6.2. LSTM

The LSTM implementation required much more work compared to the previous two methods.

The dataset had to be split to training and testing datasets, scaled, and transformed based on

the requirements of the model. Also, even though the layers were provided from relevant libraries, the network had to be built layer by layer rather than using a preset model.

After processing and splitting the data, while using the 5-year dataset for Apple Inc., the model was given 947 rows of training data. (See Figure 66)

```
#Reshape data to 3 dimensions
X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))
X_train.shape

(947, 60, 1)
```

Figure 66. Inspecting number of training rows

On the first iteration of the model with hyperparameters of 1 epoch and batch size 1, the loss function was at 0.075 and the root mean square error was found to be around 14 which did not seem bad at first. (See Figure 67)

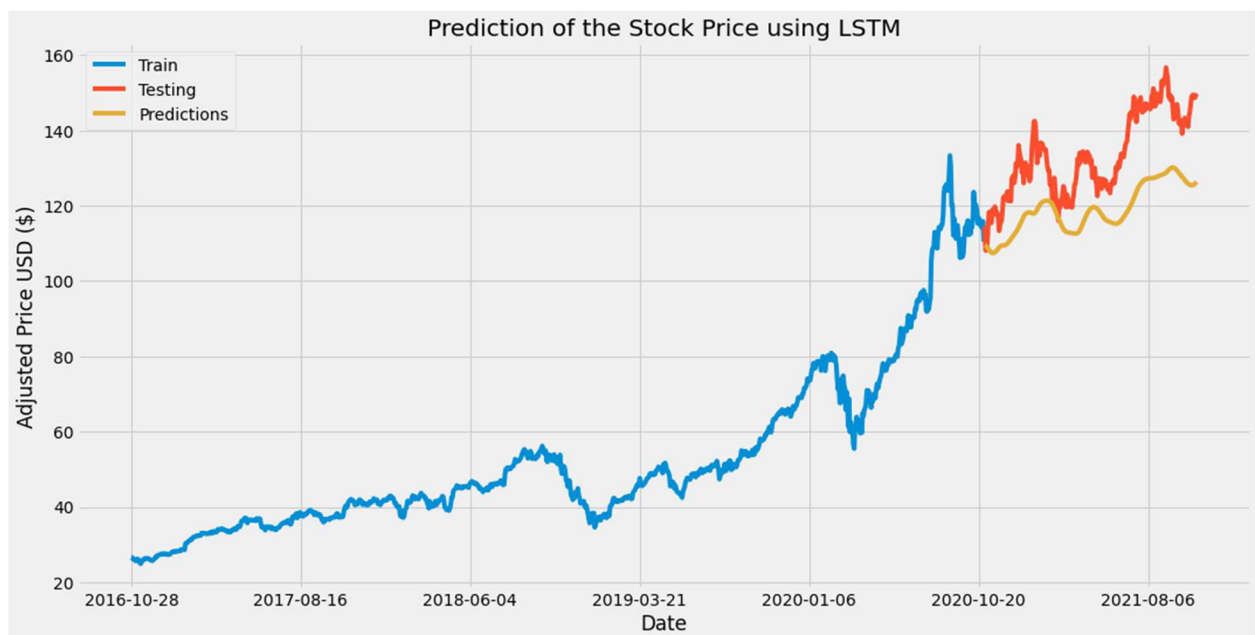


Figure 67. Visualization of model training with one (1) epoch and batch size one (1)

	Adj Close	Predictions
1007	114.583023	109.704071
1008	108.164307	109.656235
1009	108.074883	109.533798
1010	109.734215	109.329536
1011	114.215385	109.048920
...
1253	149.479996	129.944366
1254	148.690002	130.038300
1255	148.639999	130.201157
1256	149.320007	130.420013
1257	148.850006	130.681717

251 rows × 2 columns

Figure 68. Comparing Actual values to the model's predictions

After visualizing the data though, it was observed that the model roughly followed the testing pattern, but the predicted values were significantly lower than the actual values, thus the training was not enough. (See Figure 68)

As an attempt to make the model more accurate, the hyperparameters were changed. At first the batch size number was increased to 10 which resulted in a better result and a closer fit to the validation data. The loss dropped to 0.0013 while the root mean squared error was down to 5.5. (see Figure 69)

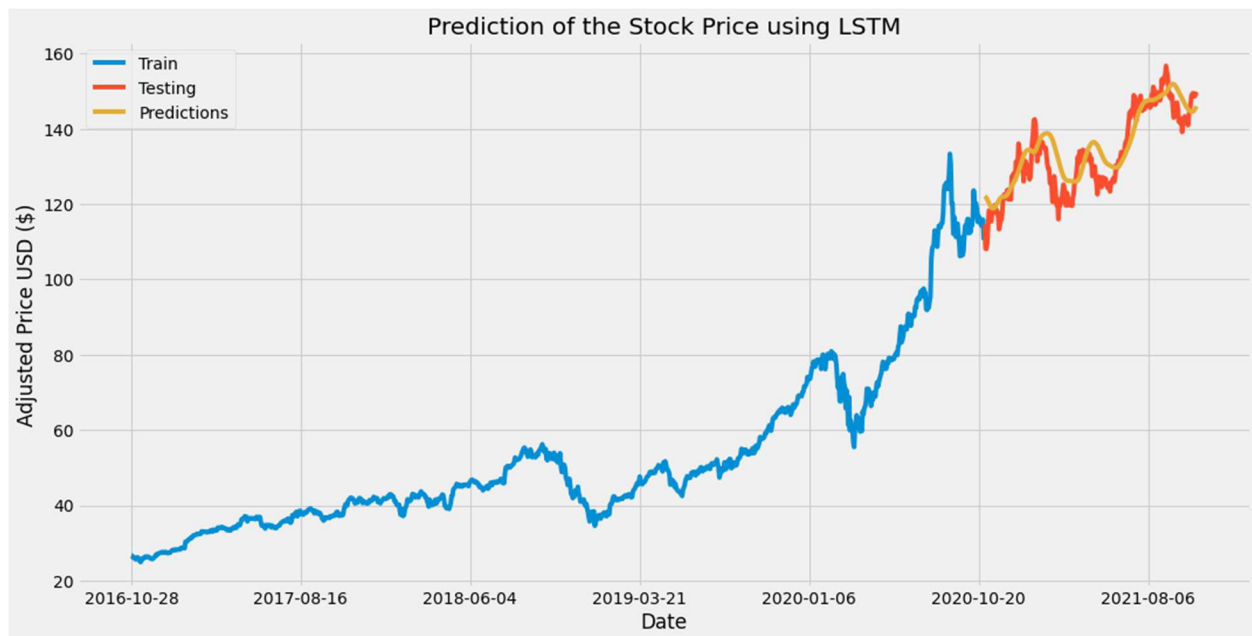


Figure 69. Visualization of model training with one (1) epoch and batch size ten (10)

	Adj Close	Predictions
1007	114.583023	122.068878
1008	108.164307	121.894569
1009	108.074883	121.622726
1010	109.734215	121.245506
1011	114.215385	120.773903
...
1253	149.479996	144.641769
1254	148.690002	144.860184
1255	148.639999	145.167435
1256	149.320007	145.541595
1257	148.850006	145.961197

251 rows × 2 columns

Figure 70. Comparing actual values to the model's new predictions

The predicted values were also much closer to the actual values but there was still space for improvement. Therefore, the number of epochs was increased to 20 resulting in a significant reduction of the loss function down to 0.0008. (See Figure 71)

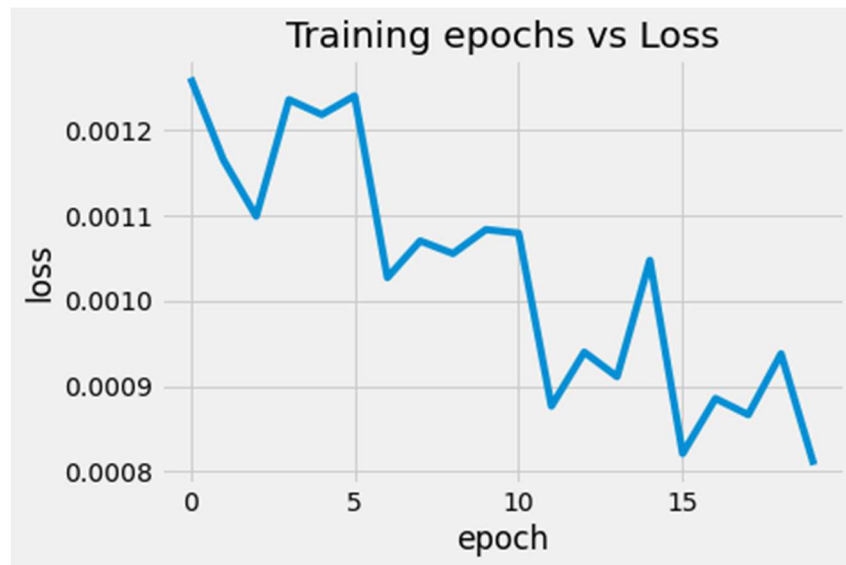


Figure 71. Visualization of loss value after each epoch of training

The root mean squared error was also reduced to around four (4) and after plotting the results, it was observed the results were improved even further. (See Figure 72)

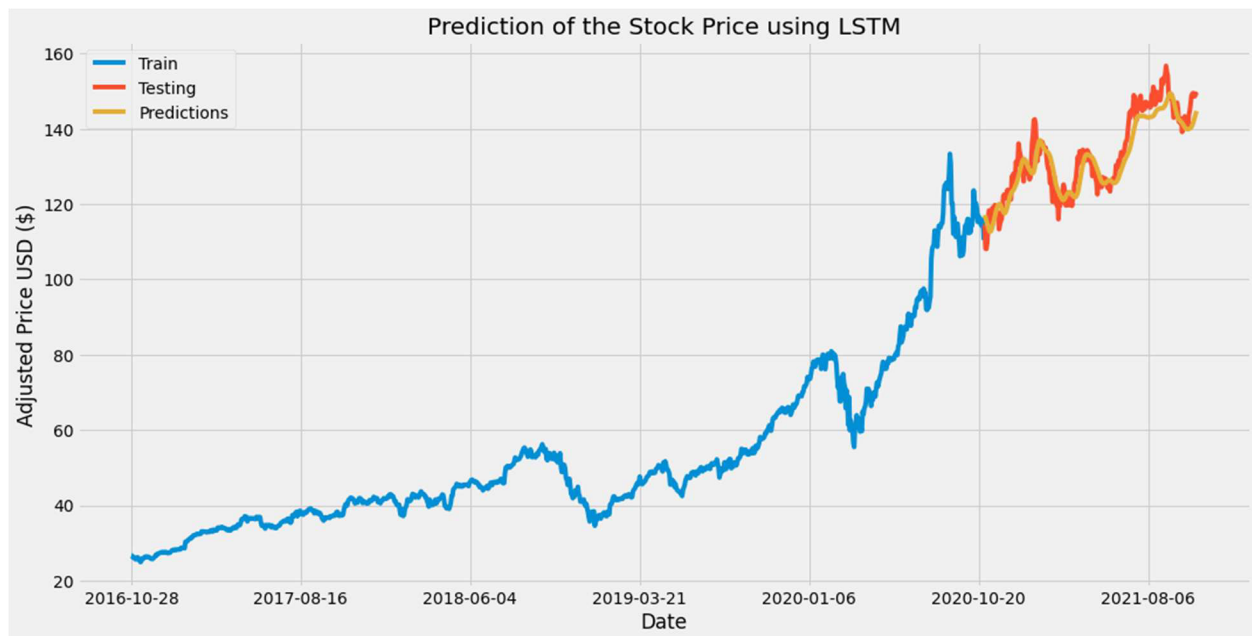


Figure 72. Visualization of model training with twenty (20) epoch and batch size ten (10)

	Adj Close	Predictions
1007	114.583023	117.120155
1008	108.164307	116.387619
1009	108.074883	115.621849
1010	109.734215	114.781288
1011	114.215385	113.900391
...
1253	149.479996	141.578369
1254	148.690002	142.361832
1255	148.639999	143.203369
1256	149.320007	144.009750
1257	148.850006	144.720520

251 rows × 2 columns

Figure 73. Comparing actual values to the model's new predictions

Increasing the epochs even more, resulted in a decrease of the loss function but an increase of root mean squared error which meant that the fitment was worse than the previous attempt.

(See Figure 74)

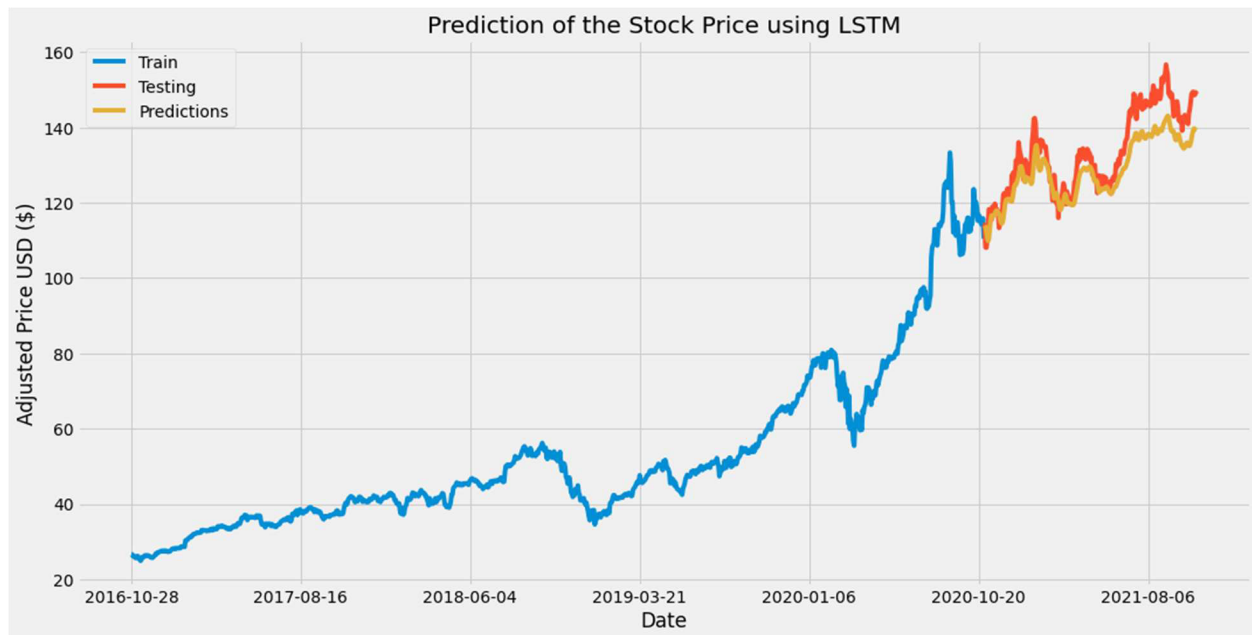


Figure 74. Visualization of model training with fifty (50) epoch and batch size ten (10)

Therefore, it can be deduced that too much training does not result in better accuracy. In the last attempt which made the model train over fifty (50) epochs, the model appeared to be overfitting since the pattern was almost the same as the actual testing dataset, but the height of the curve was less accurate than with 20 epochs, leading to less accurate predicted values. This demonstrates that the epochs of training for the LSTM models have a random effect on the performance when predicting time-series datasets.

To conclude, it can be said that the LSTM model is a very useful tool that when built, tuned, and trained correctly it can produce accurate predictions even on the long-term. The same model

though cannot be used to predict other stocks' prices without training it again on the specific dataset.

6.3. Random Forest Classifier

For this approach, instead of trying to predict actual values, the algorithm was programmed to predict whether the stock rose or fall on the specific day using binary encoding. Therefore, instead of evaluating the accuracy using root mean squared or a plot, the algorithm was evaluated by the times it accurately predicted the rise or fall on the specific day.

Beginning from the sentiment analysis, the datasets provided 4,336,445 user tweets to be processed from 01/01/2015 until 31/12/2019. (See Figure 74)

	body	ticker_symbol	Date
0	lx21 made 10,008onAAPL -Check it out! htt...	AAPL	2015-01-01
1	Insanity of today weirdo massive selling. \$aap...	AAPL	2015-01-01
2	S&P100 #Stocks Performance <i>HDLOW SBU XTGT...</i>	AMZN	2015-01-01
3	<i>GMTSLA</i> : Volkswagen Pushes 2014 Record Recal...	TSLA	2015-01-01
4	Swing Trading: Up To 8.91% Return In 14 Days h...	AAPL	2015-01-01
...
4336440	In 2020 I may start Tweeting out positive news...	TSLA	2019-12-31
4336441	Patiently Waiting for the no twitter sitter tw...	TSLA	2019-12-31
4336442	I don't discriminate. I own both <i>aapl</i> andms...	AAPL	2019-12-31
4336443	I don't discriminate. I own both <i>aapl</i> andms...	MSFT	2019-12-31
4336444	\$AAPL #patent 10,522,475 Vertical interconnect...	AAPL	2019-12-31

4336445 rows × 3 columns

Figure 74. Inspecting the tweets data frame

As it can be seen from the 'ticker_symbol' column, were not exclusively for one company. By filtering the data using their symbols, the tweets for each company were counted with Apple Inc. having the largest chunk of the data. (See Figure 75)

```
AAPL      1425013
TSLA      1096868
AMZN       718715
GOOG       392569
MSFT       375711
GOOGL      327569
Name: ticker_symbol, dtype: int64
```

Figure 75. Number of tweets for each company

Since Apple Inc. data was analyzed in the previous algorithms, this will be the case here as well but with the matching dates to the tweets dataset. It should be noted that the data still covers 5 years but instead of the past 5 years from now, it is from 2015 until 2019. After the sentiment was calculated using the 'Afinn' library, the data was merged with the historical data. (See Figure 76)

	Close	Volume	ts_polarity	twitter_volume
Date				
2015-01-02	27.332500	212818400	0.419839	867.0
2015-01-05	26.562500	257142000	0.625109	1147.0
2015-01-06	26.565001	263188400	0.679832	1190.0
2015-01-07	26.937500	160423600	0.582267	1094.0
2015-01-08	27.972500	237458000	0.761873	1495.0
...
2019-12-24	71.067497	48478800	1.244318	352.0
2019-12-26	72.477501	93121200	1.091172	691.0
2019-12-27	72.449997	146266000	0.916010	381.0
2019-12-30	72.879997	144114400	1.393443	732.0
2019-12-31	73.412498	100805600	1.004392	683.0

1255 rows × 4 columns

Figure 76. Merged historical prices and volume to tweets polarity and volume on each day

By plotting a histogram of the polarity, it was noticed that most of the tweets had a polarity of 0.5 and more but the most frequent polarities were 0.4 and 0.5 followed by 0.3. (See Figure 77)

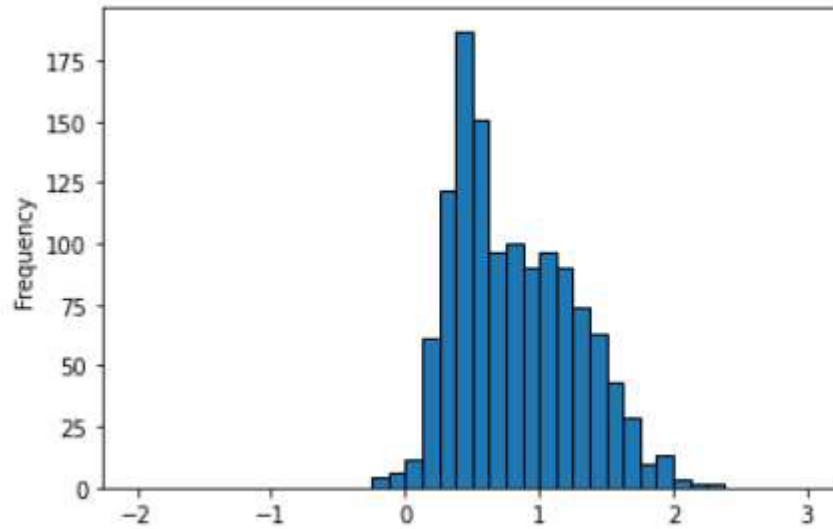


Figure 77. Tweet polarities visualized with the help of a histogram

Based on this data, the sentiment thresholds had to be altered and to choose the appropriate thresholds, the price chart of the stock was inspected.



Figure 78. Visualization of Apple Inc. trend from 2015 to 2019

Since the trend was going upwards (See Figure 78), most of the sentiment should be positive therefore as a starting point the positive threshold was set to 0.7 and the negative threshold to 0.4.

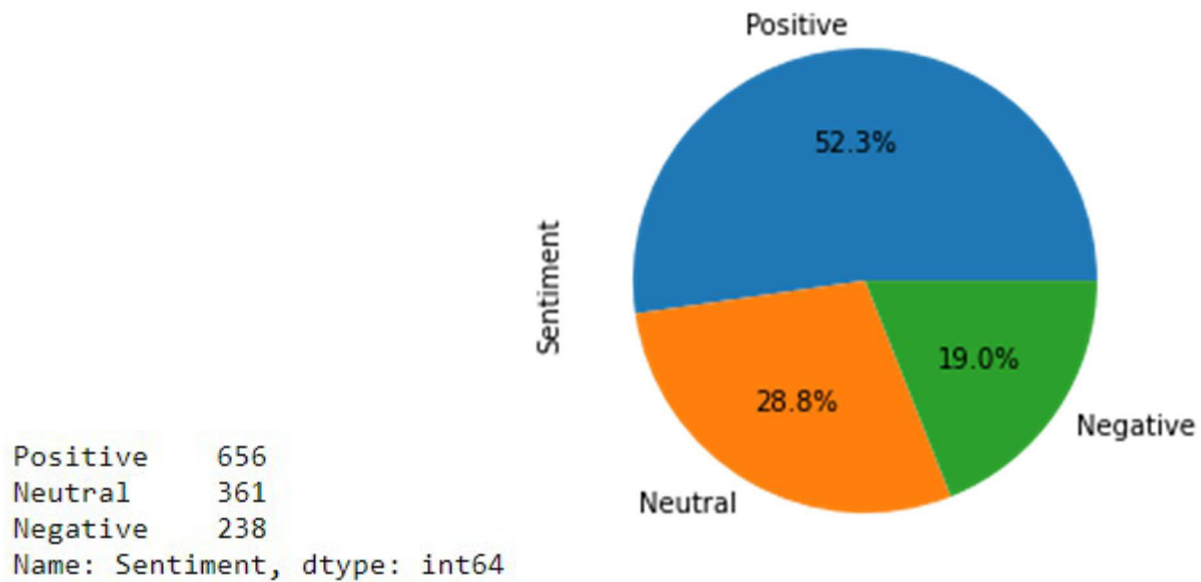


Figure 79. Pie chart of the sentiment analysis based on the chosen thresholds

With the thresholds set, the positive sentiment was 52.3%, neutral was 28.8% and the negative sentiment was 19% of the data indicating a positive sentiment towards the stock. (See Figure 79)

After processing and preparing the data, the Random Forest Classifier algorithm was built with the hyperparameters 'number of estimators' and 'random state' set to 100 and 10, respectively. (See Figure 80)

```

#Create the Random Forest model
from sklearn.ensemble import RandomForestClassifier

NUMBER_OF_ESTIMATORS = 100
#50

RANDOM_STATE = 10
#10

classifier = RandomForestClassifier(n_estimators = NUMBER_OF_ESTIMATORS,
                                  random_state = RANDOM_STATE)

classifier = classifier.fit(X_train_scaled,
                           y_train.ravel())

classifier

```

```

RandomForestClassifier(random_state=10)

```

Figure 80. Random Forest hyperparameters set to one hundred (100) estimators and random state to ten (10)

Out of the 1255 rows of the original data, since the validation dataset was only 20%, the algorithm was left with 251 rows to predict which is a low amount of data. (See Figure 81)

	Prediction	Actual
0	1	0
1	0	1
2	0	0
3	0	1
4	0	1
...
246	1	1
247	1	1
248	1	0
249	0	1
250	1	1

251 rows × 2 columns

Figure 81. Random Forest predictions compared to Actual values in binary form

And with the current set up the accuracy was only at 53.8% which just a little over 50% that indicates a random guessing. (See Figure 82)

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, predictions)
accuracy
0.5378486055776892
```

Figure 82. Model's accuracy with current hyperparameters

As a first attempt to improve the accuracy of the model the number of estimators in the model was changed from 100 to 50 resulting in an accuracy of 57.3%.


```

#Create the Random Forest model
from sklearn.ensemble import RandomForestClassifier

NUMBER_OF_ESTIMATORS = 50
#50

RANDOM_STATE = 10
#10

classifier = RandomForestClassifier(n_estimators = NUMBER_OF_ESTIMATORS,
                                  random_state = RANDOM_STATE)

classifier = classifier.fit(X_train_scaled,
                           y_train.ravel())

classifier

RandomForestClassifier(n_estimators=50, random_state=10)

```

Figure 83. Random Forest hyperparameters set to fifty (50) estimators and random state to ten

(10)

```

#Find the model's accuracy
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, predictions)

accuracy

0.5737051792828686

```

Figure 84. Model's accuracy with current hyperparameters

The next thing to change in an attempt improve the model's predictions accuracy was the sentiment thresholds since the day's sentiment affect the model's predictions. Eventually, the thresholds that were found to provide the best accuracy were the ones used from the start, 0.7 and 0.4 for positive and negative respectively which meant that the logic used when selecting them was correct.

The final and best accuracy that was found by the Random Forest Classifier was 57.3% which is not a reliable accuracy to use for trading since it is only a little better than guessing randomly.

This approach has proven that twitter user sentiment does not affect the stock price significantly and that it is not a suitable method to use when using machine learning to aid with trading stocks. Therefore alternative approaches should be explored.

7. Conclusion

Reflecting on the results of all the tested methods, each algorithm had its strengths and weaknesses.

The Facebook Prophet algorithm was efficient in the task of predicting stock prices because of easier implementation without compromising on prediction accuracy. This makes it ideal for situations where the analyst does not have a lot of experience or background knowledge on complex machine learning algorithms. It was also very easy to configure the parameter to perform future predictions rather than just splitting the dataset in training and testing subsets. The downsides of the algorithm though, is that it is limited to the seasonality and special events features and it won't be able to help in multi-level product hierarchy challenges.

Moving on to the LSTM, the algorithm is a more advanced model and required more experience and background knowledge to implement. In the hands of an experienced analyst though, it can be optimized to perform better than the default Prophet settings. This is because the LSTM has many tunable hyperparameters which can be adjusted to make the model perform better on each specific dataset. As LSTMs are trained to learn long term correlations in a series, they can model complex multivariate sequences without the need to specify any time window. On the other hand, even though LSTM networks can be made to perform exceptionally well on a specific time-series dataset, the same network will perform poorly on a different dataset.

Finally, the Random Forest algorithm was the most complex and advanced method. Even though the data preprocessing and model creation was like the LSTM, the twitter user sentiment added the extra complexity which made this method not suitable for inexperienced people. The predictions of this model were portrayed differently than the previous two models,

instead of plotting graphs and inspecting predicted values against the actual values, the Random Forest method was evaluated by trying to predict rise or fall to the stock's prices. The result of this method was disappointing since it was expected that the extra work you also lead to better results but after tuning the model to find the ideal hyperparameters, the best it could do was around 60% which is just 10% above 50%. Therefore, the Random Forest only performed a little better than random guessing.

In general, as mentioned in the introduction, predicting the stock market prices is believed to be impossible because the stock market can be random and closely correlated with real time events. Sometimes the stocks can follow specific patterns that can, at a certain degree, be predicted with the help of machine learning algorithms, but other times the same stock's movements can be totally random compared to the past. The aim of this report was to indicate possible methods and algorithms that can be used to help human decision making when investing in stocks but from the tests that have been done and the results received, people cannot solely rely on the predictions of the models when investing. The results can be used though, cooperated with human experiential knowledge to make financial decisions. "Machine learning and deep learning methods do not yet deliver on their promise for univariate time series forecasting and there is much research left to be done." ("ARIMA/SARIMA vs LSTM with Ensemble learning Insights for ...")

8. Further Work

Something that could be done to further improve the work in this project, was to increase the amount of data that was fed in the models to find if accuracy would be enhanced. The idea of using only the past 5 years as mentioned before was based on the fact that the market behavior was significantly different from 2015 onwards, compared to the previous years, especially in technology related stocks, but other timelines could be used to test the results like 1990-2000.

Also, the sentiment approach could be further explored to find if there was a better relationship between the news articles or professional posts sentiment rather than only using user tweets sentiment and volume which were proven to have no effect.

Another task that could be carried out to test the accuracy of the algorithms, was the use of more volatile and unregulated market stocks instead of public companies like cryptocurrencies such as the Bitcoin or Ethereum. Or on the opposite side, commodities like Gold, Silver and Oil could be used which have must more stable prices and could produce better predictions.

Finally, some theories related to the algorithms could be tested. For example, instead of training the algorithms on every stock separately, they could be trained on one stock and be then implemented on a different one to see how accurate the trained model predictions are. This would explore the generalization of the model's fitment and test the theories that networks such as the LSTM cannot perform well on different stocks other than the specific one it was trained on.

9. Reflection

Our society becomes increasingly dependable on machine learning and artificial intelligence in every imaginable way. This project and the experience I gained while working on it will help me in the future where the use of machine learning and artificial intelligence in all sectors are inevitable.

Through the research and work I have performed during this project I expanded my knowledge and experience on Machine Learning. I learnt the theory behind some of the most important machine learning algorithms such as the LSTM how they work when given a time series dataset. Also, I learnt how to implement these algorithms by preprocessing time series datasets and developing models that can use this data for predicting values, in this case stock prices.

10. References

1. Facebook Research. 2021. Prophet: forecasting at scale - Facebook Research. [online] Available at: <<https://research.fb.com/blog/2017/02/prophet-forecasting-at-scale/>> [Accessed 27 October 2021].
2. Medium. 2021. *Time Series Analysis with Facebook Prophet: How it works and How to use it*. [online] Available at: <<https://towardsdatascience.com/time-series-analysis-with-facebook-prophet-how-it-works-and-how-to-use-it-f15ecf2c0e3a>> [Accessed 31 October 2021].
3. Dinevski, E., Blizzard, J., Shelton, M., Events, A. and Battani, A., 2021. *Facebook Prophet Tutorial: How to Use Time Series Forecasting | Tessellation*. [online] Tessellation. Available at: <<https://tessellationtech.io/facebook-prophet-tutorial-time-series-forecasting/>> [Accessed 31 October 2021].
4. Lib.dr.iastate.edu. 2021. [online] Available at: <<https://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=1437&context=creativecomponents>> [Accessed 31 October 2021].
5. Lutins, E., 2021. *Ensemble Methods in Machine Learning: What are They and Why Use Them?*. [online] Medium. Available at: <<https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f>> [Accessed 3 November 2021].
6. Cs231n.github.io. 2021. *CS231n Convolutional Neural Networks for Visual Recognition*. [online] Available at: <<https://cs231n.github.io/neural-networks-1/>> [Accessed 1 November 2021].
7. Singhal, G. and RNN, I., 2021. *Introduction to LSTM Units in RNN | Pluralsight*. [online] Pluralsight.com. Available at: <<https://www.pluralsight.com/guides/introduction-to-lstm-units-in-rnn>> [Accessed 1 November 2021].
8. Thakur, D., 2021. *LSTM and its equations*. [online] Medium. Available at: <<https://medium.com/@divyanshu132/lstm-and-its-equations-5ee9246d04af>> [Accessed 1 November 2021].
9. Medium. 2021. *LSTM Networks | A Detailed Explanation*. [online] Available at: <<https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9#:~:text=LSTMs%20use%20a%20series%20of,each%20their%20own%20neural%20network.>>> [Accessed 1 November 2021].
10. www.javatpoint.com. 2021. *Machine Learning Random Forest Algorithm - Javatpoint*. [online] Available at: <<https://www.javatpoint.com/machine-learning-random-forest-algorithm>> [Accessed 2 November 2021].
11. Chauhan, N., 2021. *Decision Tree Algorithm, Explained - KDnuggets*. [online] KDnuggets. Available at: <<https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html>> [Accessed 2 November 2021].

12. Shastri, A., 2021. *3 decision tree-based algorithms for Machine Learning*. [online] Medium. Available at: <<https://towardsdatascience.com/3-decision-tree-based-algorithms-for-machine-learning-75528a0f03d1>> [Accessed 2 November 2021].
13. Brownlee, J., 2021. *Overfitting and Underfitting With Machine Learning Algorithms*. [online] Machine Learning Mastery. Available at: <<https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>> [Accessed 2 November 2021].
14. Docs.aws.amazon.com. 2021. *Model Fit: Underfitting vs. Overfitting - Amazon Machine Learning*. [online] Available at: <<https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html>> [Accessed 2 November 2021].
15. Taylor SJ, Letham B. 2017. Forecasting at scale. PeerJ Preprints 5:e3190v2 <https://doi.org/10.7287/peerj.preprints.3190v2> [Accessed 2 November 2021].
16. Goel, S. and Bajpai, R., 2020. Impact of Uncertainty in the Input Variables and Model Parameters on Predictions of a Long Short Term Memory (LSTM) Based Sales Forecasting Model. *Machine Learning and Knowledge Extraction*, 2(3), pp.256-270.
17. Olah, C., 2021. *Understanding LSTM Networks -- colah's blog*. [online] Colah.github.io. Available at: <<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>> [Accessed 3 November 2021].
18. Education, I., 2021. *What is Bagging?*. [online] Ibm.com. Available at: <<https://www.ibm.com/cloud/learn/bagging>> [Accessed 3 November 2021].
19. Singhal, G. and Boosting, E., 2021. *Ensemble Methods in Machine Learning: Bagging Versus Boosting | Pluralsight*. [online] Pluralsight.com. Available at: <<https://www.pluralsight.com/guides/ensemble-methods:-bagging-versus-boosting>> [Accessed 3 November 2021].
20. En.wikipedia.org. 2021. *Bootstrap aggregating - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Bootstrap_aggregating> [Accessed 3 November 2021].
21. GmbH, f., 2021. *Activision Blizzard has lost nearly \$8 billion in market value amid the growing fallout from a workplace discrimination lawsuit*. [online] markets.businessinsider.com. Available at: <<https://markets.businessinsider.com/news/stocks/activision-blizzard-stock-lost-8-billion-market-value-discrimination-lawsuit-2021-7>> [Accessed 4 November 2021].
22. Wealth Advisors, A., 2021. *How Is the Stock Market Impacted By Politics? - ARQ Wealth Advisors*. [online] ARQ Wealth Advisors. Available at: <<https://arqwealth.com/how-is-the-stock-market-impacted-by-politics/>> [Accessed 4 November 2021].
23. Thomas, M., 2021. *How AI Trading Technology Is Making Stock Market Investors Smarter*. [online] Built In. Available at: <<https://builtin.com/artificial-intelligence/ai-trading-stock-market-tech>> [Accessed 4 November 2021].

24. Brownlee, J., 2021. *Difference Between a Batch and an Epoch in a Neural Network*. [online] Machine Learning Mastery. Available at: <<https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/#:~:text=The%20number%20of%20epochs%20is%20a%20hyperparameter%20that%20defines%20the,update%20the%20internal%20model%20parameters.>> [Accessed 5 November 2021].
25. Wings, T., 2021. *Python - Sentiment Analysis using Affin - GeeksforGeeks*. [online] GeeksforGeeks. Available at: <<https://www.geeksforgeeks.org/python-sentiment-analysis-using-affin/>> [Accessed 5 November 2021].
26. Dinevski, E., Bernard, C., Events, A., Pedersen, G. and Blizzard, J., 2021. *Facebook Prophet Tutorial: How to Use Time Series Forecasting | Tessellation*. [online] Tessellation. Available at: <<https://www.tessellationtech.io/facebook-prophet-tutorial-time-series-forecasting/>> [Accessed 5 November 2021].
27. Saiful, I., Emam, H. *Foreign Exchange Currency Rate Prediction using a GRU-LSTM Hybrid Network*. [online] ScienceDirect. Available at: <https://www.sciencedirect.com/science/article/pii/S2666222120300083> [Accessed 5 November 2021].