Department of Informatics,

University of Leicester

CO4015 Computer Science Project

Dissertation

for

Stock Market Prediction using python

Author: Osato Osagie (1690 11647)

# Introduction

The term ‘stock market’ does not refer to a single market. It refers to several stock exchanges dispersed around the world. In these stock exchanges, traders and investors can purchase and sell shares of companies that are publicly traded. The share prices of these companies constantly fluctuate in response to the law of supply and demand.

A share is a small possession stake in a publicly traded company. The price of a stock portrays the expectation of stock investors and market analysts on the company’s future earnings.

When traders believe a company will perform well, they bid the price up by creating demand for stocks in that company. On the other hand, those traders who don’t believe in a company’s future will bid the price down by selling their holdings, thus creating an excess supply. Sellers aim to receive a high price for each share hopefully more than their initial payment, whilst buyers aim to acquire shares at their lowest price possible so that they can make a profit when selling the stock.

Investing in stocks is deemed as a reliable method to achieve profits that beat inflation over time. The returns, on average do better than those of other investments, such as bonds and commodities. According to research, as of February 2021, the total market value of all companies trading on the London Stock Exchange stood at 3.67 trillion British pounds [1].

There are two ways to make a profit on the stock market. Investors can either trade stocks or decide to hold them. Trading requires investors to buy and sell stocks frequently, taking advantage of little ticks in cost. Investors who purchase and lean toward to let their stocks appreciate in esteem over time. In some cases, investors who hold shares get rewarded with regular payments of dividends.

Investors have previously found ways to obtain insight about the businesses listed on the market for as long as markets have existed in order to increase their investment returns. However, owning to the size market and the pace at which transactions are conducted, this is not possible today. Simple statistical analysis of financial data can reveal certain trends, but in recent years, investment firms have increasingly turned to Artificial Intelligence (AI) systems to search for patterns in vast quantities of real-time equity and economic data.

AI is the ability of machines to behave like humans and learn autonomously. For instance, a machine might display learning and problem-solving abilities without the use of hard-coded software containing detailed instructions [1].

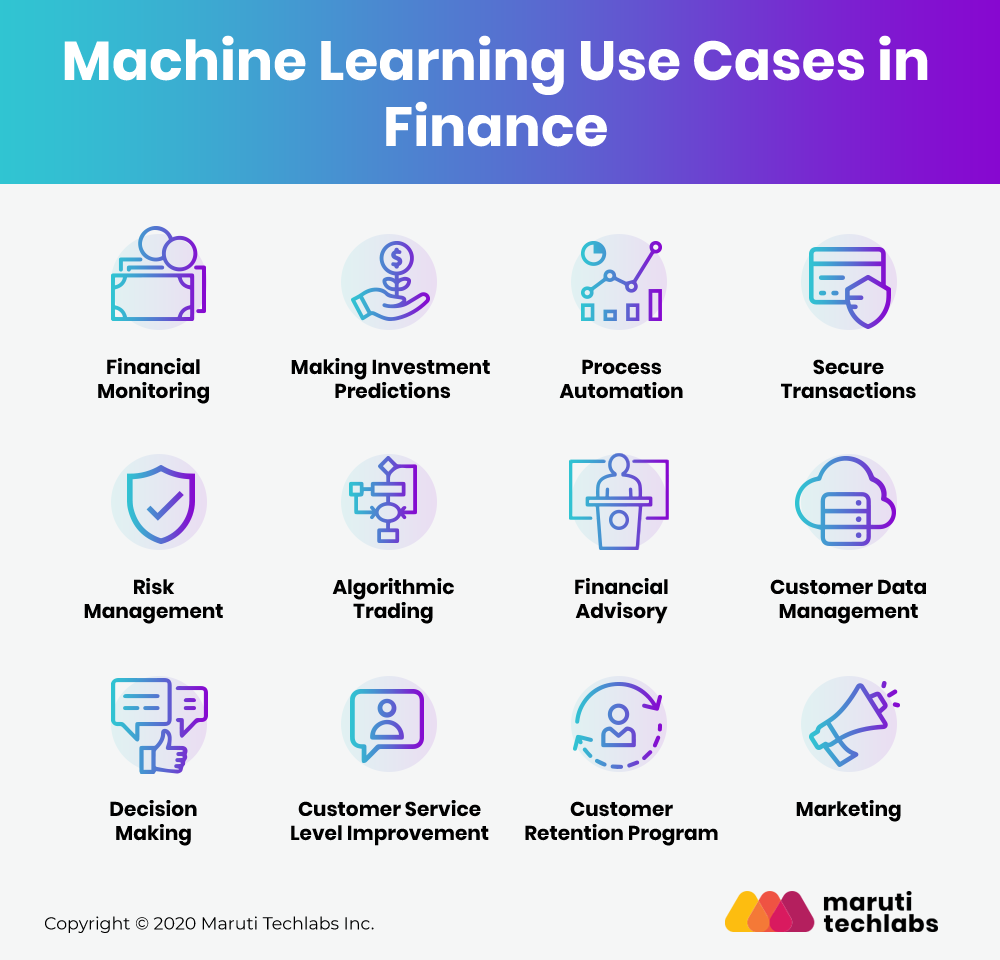


Figure 1- An image showing the main use cases of Machine Learning in Finance [2]

Machine Learning (ML) is a subfield of AI, that enables machines to learn from historic data or experiences without being explicitly programmed. Figure 1 shows the different use cases for Machine Learning in finance. The project I will be building focusing on using ML to make investment predictions. Using ML to make investment predictions is advantageous as it can lead to better predictions of stock prices, fewer errors, and greater efficiency for the investor. To do this, ML algorithms extract key insights from the dataset, learns from it then apply several techniques to accurately predict the result.

However, it is important to take into consideration the other factors that might affect the price of a company’s stock. The stock market is very volatile, thus meaning no system can accurately predict it.

Deep Learning (DL) is a subfield of ML. It teaches computes to learn by example in the same way that humans do. A prominent example of DL is self-driving vehicles, allowing them to identify a stop sign or differentiate between a pedestrian and a lamppost. DL models are able to achieve cutting-edge precision, often even outperforming humans. A wide collection of labelled data and neural network architectures with several layers are used to train models.

## 1.1 Aims

Predicting markets has become an increasing priority for investors. The primary goal of an investor is to buy a stock when its value is low and sell when the value of the stock is high. However, this can be daunting for financial investors as they are unaware of the stocks that will return maximum profits. Using Machine learning to predict the long-term value of a stock makes this process somewhat easier. This project aims to predict the stock market price of a company using supervised machine learning algorithms.

## 1.2 Objectives

To meet the aims this project sets out to achieve, I have broken it down into several tasks. These objectives are as follows:

* Obtain real-time and historical equity data from Alpha Vantage API.
* Clean data and form data sets with the obtained data.
* Build python functions to calculate technical indicators from the obtained dataset.
* Train a Decision Tree and SVM model to predict the S&P 500.
* Test various models to find which one works best for predicting Standard and Poor’s 500 (S&P 500).
* Fine tune model parameters to have as low bias as possible while also having low variance on the training data.
* Split datasets into training and test data, and train models with data.
* Tests the models with data and measure accuracy using metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R2).
* Evaluate which features have the most impact on stock prices.
* Visualise results using a line chart showing the predicted prices versus the actual prices of the stock.
* Implement a user-friendly interface for the prediction tool.

## 1.3. Changed objectives

Following the feedback from my interim report to ‘add more up to date technologies, some objectives have been changed. These include:

* Train a Ridge Regression, Random Forest and Long Short-Term Memory models to predict the S&P 500.

## 1.4. Resources & Tools

This project is built using Python programming language. I chose to use Python because it is easy and flexible to use. Python is also versatile and has a robust collection of libraries that make machine learning tools easily available to use. Also, Python has a diverse pack of visualisation options available which makes it ideal for creating graphs and charts.

The Python code will be written using Jupiter notebook. It is a web application that allows users to create documents containing live code and visualizations. In addition, I have used various Python libraries to access tools that have enabled me to build my project.

The main library used in my project is scikit-learn. It offers a consistent Python framework for a variety of supervised and unsupervised algorithms. This library's stack contains the following items:

* NumPy: Base n-dimensional package
* Matplotlib: Comprehensive 2D/3D plotting
* Pandas: Data structures and analysis

Another library that plays a major role within my project is Streamlit. It is an open-source Python library that makes it easy to create and share beautiful, custom web apps for ML and data science.

# Literature Review

This chapter will discuss current literature that will be used to set the stage for this project. Such literature are based on subjects surrounding the current technology development of AI in trading and investing,

## 2.1. Impact of AI on Trading and Investing

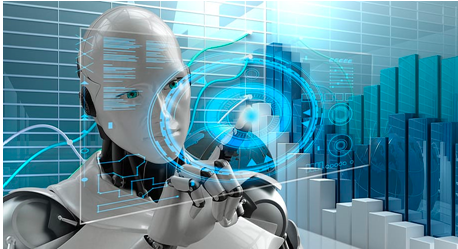


Figure 2-Photo credit: [3]

AI, ML, and DL have been transforming finance and investing. Although humans remain an important part of the trading equation, AI is becoming increasingly important. Electronic trades account for approximately 45 percent of cash equities trading revenues, according to a new report by Coalition [4], a U.K research company. Though hedge funds are wary of automation, many of them use AI-powered analysis to generate investment ideas and build portfolios.

“Machine Learning is progressing much faster, and financial institutions are among the early adopters.” Intelenet Global Service's vice president of global business growth, Anthony Antenucci, recently said [5]. Globally companies are developing new products and technology that use AI to make trading and investing more data-driven and effective. Bloomberg announced in September 2017 that Japan’s third largest lender will use AI in the equities sector through algorithm-based services for institutional clients [6]. In the United States, Merrill Lynch is currently testing a stock platform to find value in small-cap stocks that analysts would otherwise ignore [7].

Currently, there are companies already using AI for smarter trading. Aquan, a UK data science company [8] uses its platform to democratize trading by encouraging data scientists of all backgrounds to build algorithmic trading strategies that assist in the resolution of investment problems. As a result, investment clients will benefit from data science without having to invest in costly in-house expertise. Aquan has had significant industry impact, they graduated from Techstars in 2018 and was recently named the 2019 Europa Awards’ Hottest Fintech n Europe.

Similarly, EquBot’s[9] proprietary investment technology, which is affiliated with IBM, blends AI with an active exchange-traded fund (ETF). The business centralizes the investment process by gathering and processing data from different sources (news articles, social media postings, financial statements) from around the world to “build a cause-and-effect understanding of economies, businesses, and management”. The impact EquBot has had in the industry has also been significant, recently they launched the AI-powered Foreign Equity ETF, which aims to invest in established international markets outside of the United States.

## 2.2 Stock Market Prediction

There are three separate trading schools of thinking, all driven by the need to forecast market fluctuations and profit: fundamental, analytical and quantitative technical research.

### 2.2.1 Fundamental Analysis

The study of economic factors that affect the price of a stock is known as fundamental analysis. A balance sheet and an income statement are examples of such causes. Throughout the year, these reports are published quarterly. Fundamental analysis is commonly used to predict long-term market fluctuations because it depends on forecasts that are published on the basis of a longer time period.

### 2.2.2 Technical Analysis

Chart, histogram

Description automatically generated

Figure 3- Relative Strength Index

Technical research attempts to predict what other stakeholders are thinking based on available knowledge about stock prices and volume. To forecast future prices, technical analysts use a range of metrics derived from the past experience of stock price and volume. Overall, the main concept technical analysis is trend. According to technical analysts, stock price trends are triggered by a discrepancy between supply and demand for stocks, which is reflected in the bid and ask prices. Scientific analysts strive to derive trends from the chaotic data of stock prices. Since it depends on visual analysis, technical analysis is mostly qualitative. Figure 3 shows an example of such stock charts. It represents one year’s worth of historical stock price data for the IT sector in 2016.

### 2.2.3 Quantitative Analysis

Quantitative Analysis is a method of determining the value of a financial asset, such as a stock. It emphasises mathematical and statistical analysis. Investors use the data produced by these computer models to evaluate investment opportunities and create what they believe will be a profitable trading strategy. On various types of stock markets around the world, different types of datasets and machine learning algorithms are used to achieve these goals. After researching a few of them, I've written about them below, reviewing the datasets used, algorithms used, and results obtained.

In this proposed project, I will be implementing three prediction models: Linear Regression, Random Forest, and Long Short-Term Memory. Each will be evaluated using regression metrics.

Machine learning is a technique that has been used in a variety of areas, including science and medical research, such as using machine learning models to determine whether or not someone has cancer, and financial research, such as forecasting stock market movements.

The first series of papers focuses on experiments that use artificial neural networks to forecast stock market movements (ANNs).

Using data from several global stock markets, Jasic and Wood (2004) created an artificial neural network to predict regular stock market index returns. The main goal is to encourage efficient trading. Short-term stock market index return forecasts are provided using a framework focused on univariate neural networks with untransformed data inputs. The Standard and Poor’s 500 Index (S&P 500), the German DAX index, the Japanese TOPIX Index, and London’s Financial Times Stock Exchange Index are all include in the analysis (FTSE All Share). The S&P 500, DAX, and FTSE Index samples range from January 1, 1965 to November 11, 1999. Since data from years was not available, the TOPIX sample spans the years from January 1, 1969 to November 11, 1999. The neural network’s prediction efficiency is compared to that of a benchmark linear autoregressive model, and the prediction improvement is verified when applied to the S&P 500 and DAX indices.

Enke and Thawornwong (2005) test the predictive relationships for a variety of financial and economic variables using a machine learning knowledge gain technique. A ranking of the variables is obtained by calculating the information gain for each model variable. Only the most important variables are held in the forecasting models after a threshold is calculated. The capacity of neural network models for level estimation and classification to provide an accurate forecast of future values is investigated. The generalizability of multiple models is also improved using a cross-validation technique. S&P data from March 1976 to December 1999 were used to compare the models. The findings show that trading strategies led by classification models produce higher risk-adjusted profits than buy-and-hold, other neural network models, and linear regression models.

The following research employs a stochastic time efficient neural network model to discover the predictive relationships between a variety of financial and economic variables (Liao and Wang, 2010). Investors are assumed to make investment decisions based on historical stock market data, which is weighted according to how close it is to the present. The greater the effect of historical data on the predictive model, the closer the data is to the present. The model's efficacy is assessed utilizing data of each trading day over an 18-year period, spanning December 19, 1990 to June 7, 2008.

On disadvantage of using ANN is that uses an optimization algorithm for finding local optima called Gradient descent. However, this often gets suck in the local maxima, thus making it a challenge to find global minima and maxima. My project is different because it uses Long Short-Term Memory (LSTM). Since they avoid the vanishing gradient problem, LSTM networks outperform traditional RNNs.

# Methodology

The methodology that will be used to build accurate models to predict the stock price of companies in the S&P 500 will be discussed in this chapter. There are several components of the methodology used to create a functional model that will be discussed in depth. The following sections must be completed in order to create an efficient model for this project:

* Technical Indicators
* Stock Dataset
* Train Mode
* Test Model
* Evaluate Model

Diagram

Description automatically generated

Figure 4-Training Structure of Model

## 3.1 Technical Indicators

To get better results I have decided to use Technical Indicators. Technical indicators are mathematical calculations based on the price, volume, or open interest of a security [5]. Whilst these indicators are designed to analyze short-term price movements, they are also useful to long-term investors who want to identify entry and exit points.

In particular, I will be using momentum technical indicators. These indicators measure the rate at which a security’s price moves, and there are a number of metrics that can be used to do so including:

* The Relative Strength Index (RSI) is a common momentum indicator for evaluating whether a stock is overbought or oversold. When demand unjustifiably drives the price of a stock upwards, it is said to be overbought. This is commonly considered an indicator that the stock is overvalued and is likely to decline in price. When a stock’s price falls dramatically below its true worth, it is considered to be oversold. As a consequence of panic sale, this is the outcome. The RSI scale runs from 0 to 100, and a reading of more than 70 suggests that the stock is overbought, whereas a reading of less than 30 implies that the stock is oversold.

**Formula:**



Figure 5- RSI Formula

**Code:**

The code for RSI calculation is shown in appendix A. I started off by identifying the up days (days where the stock price went up) and the down days (days where the stock price went down). I created a column that identifies the change in price, then used a condition that set the value of up and down days based on the change in price. In addition, I ensured the values for down days were absolute by modifying the column and calculating the Exponential Moving Average of both the UP and Down columns. The final step was to calculate the Relative Strength metric and pass that through to the RSI calculation.

* Stochastic Oscillator tracks the price’s momentum. In certain cases, momentum shifts before the price shifts. It calculates the closing price in relation to the low-high range for a given time span.

**Formula:**

Graphical user interface, text, application

Description automatically generated

Figure 6- Stochastic Oscillator Formula

**Code:**

The code I used to calculate the Stochastic Oscillator is shown in appendix B. To calculate Stochastic Oscillator, I use the rolling lambda function. With this function, I specify a 14-period window, and what measurement to apply to each window. After the maximum and minimum values are obtained, I then pass it through my formula and apply the results to the main data frame.

* Williams %R varies from -100 to 0. It shows a sell signal when its value is greater than -20, and a buy signal when the value is less than -80.

**Formula:**

Text

Description automatically generated

Figure 7- Williams %R Formula []

**Code:**

The code I used to calculate the Stochastic Oscillator is shown in appendix C.

* Price Rate of Change (ROC) is a momentum-based technical indicator which compares the present price to the price from a certain number of periods ago. When price changes are to the upside, the ROC indicator moves upwards into positive territory, and when price changes are to the downside, the indicator moves downwards into negative territory.

**Formula:**

Graphical user interface, text, application

Description automatically generated

Figure 8-ROC Formula []

**Code:**

The code I used to calculate the Stochastic Oscillator is shown in appendix E.

* Moving Average Convergence Divergence (MACD) depicts the connection between two security price moving averages. By subtracting the 26-period exponential moving average (EMA) from the 12-period EMA the MACD is estimated. The "signal line," a nine-day EMA of the MACD, is then plotted on top of the MACD line. A sell signal is indicated when the MACD falls below the SingalLine. It indicates a buy signal as it rises above the SignalLine.

**Formula:**



Figure 9-MACD Formula []

**Code:**

The code I used to calculate the MACD is shown in appendix D. To calculate the MACD, I used the column in the data frame containing the closing price of the stock. After obtaining this column, I then applied the transform method along with the specified Lambda function. Then I proceeded to calculate the EMA by calling the exponential moving weight (emw) function and specifying how many periods to look back. I used the default definition provided by the formula and specified 26 & 12.

After calculating the 26-period EMA and the 12-period EMA, I took the difference between both to obtain the MACD. In addition to the MACD, I also wanted to calculate the EMA of the MACD, so I applied the same ewm function, but in this case I specified a span of 9. Finally, both the MACD and MACD\_EMA were added to the main data frame.

## 3.2 Stock Dataset

Table

Description automatically generated

Figure 10- real-time and historical time series data

The figure above shows data was obtained from Alphavantage API. The columns represent the following:

* Open: the starting price of the stock for each day
* Close: the final price of the stock for that particular trading day
* High: highest price for a particular date.
* Low: lowest price for a particular date.
* Adjusted Close: amended close price that reflects the value of a stock after accounting for any corporate actions.
* Volume: the number of shares of a stock exchanged on a stock exchange in that day.
* Dividend amount: the amount of dividend paid in that day.
* Split coefficient:

A picture containing graphical user interface

Description automatically generated

Figure 11-Correlation heatmap of obtained data

For the modelling I used Open and High columns combined with technical indicators as they have the highest correlation with the closing price which is the variable we are trying to predict. Although there are several more highly correlated columns such as Low and Adjusted close, adding them to the model will not improve the prediction accuracy.

## 3.3 Data Pre-processing

Data pre-processing is an important step in ML as real-world data is often incomplete, unreliable, and/or deficient in specific habits or patterns, as well as containing numerous errors. Pre-processing data is a tried-and-true way of addressing such problems. It entails converting raw data into a format that can be understood.

### 3.3.1 Missing values

A picture containing text

Description automatically generated

Figure 12- check for missing values in dataset

Although my datasets is from a reliable source it is important to handle missing values in the data. To deal with missing values, I either delete a specific row if it has a null value for a specific feature and a specific column if it has more than 75% missing values.

This method is only recommended when the data set contains a sufficient number of samples. It is critical to ensure that there is no additional bias after the data has been deleted. Removing the data will result in information loss, which will result in the predicted output not yielding the expected results.

### 3.3.2 Feature Scaling

The method of limiting the range of variables so that they can be compared on common grounds is known as feature scaling. For example, in the dataset pictured above the ‘Open and Volume columns do not have the same scale, and this will cause some issues on the ML models.

Text

Description automatically generated with medium confidence

Figure 13- Euclidean Distance (ED) Formula

Majority of ML models are based on ED, so therefore it is easy to see that the Open column will be dominated in ED, and we must try to avoid this. Although not all ML models are based on Euclidean Distances, it is still important to do feature scaling because the algorithm will converge much faster. For example, Decision Trees are not based on ED but, without feature scaling then it will run for a very long time. to improve the performance of the model, I scaled the data using Scikit-Learn’s MinMaxScaler and scaled the dataset to numbers between zero and one.

### 3.3.3 Splitting the dataset into Training and Test set

Text

Description automatically generated

Figure 14-model split code snippet

Before building any ML model, the data set needs to be split into two separate sets:

1. Training Set- the data sample used to fit the model. The model observes, learns, and optimises its parameters based on this data.
2. Test Set- a data sample used to provide an unbiased evaluation of a final model fit on the training dataset. It is used only after the model has been fully trained using the training and validation sets. As a result, the test set is used to simulate the type of scenario that will occur once the model is dispatched for real-time use.

Splitting the dataset is important because the model we use are nothing more than estimation techniques that learn the statistical trends in the data. As a result, it is critical that the data used to learn and that used to test the model have a similar statistical distribution as possible in order to avoid overfitting. One method for achieving this as precisely as possible to choose the subsets at random. However, due to the nature of our data, it is not possible to randomly select subsets of the data, so I split the data into a 65:35 ration. This means that 65% of the data is used to train the model, and 35 percent is used for testing as shown above in figure 13.

The variables shown in figure 13 stand for the following:

* X\_train- the training part of the matrix of features.
* X\_test- the test part of the matrix of features.
* y\_train- the training part of the dependent variable that is associated X\_train.
* y\_test- the test park of the dependent variable that is associated to X\_train.

## 

# 3.4 Models

## 3.4.1 Supervised Learning

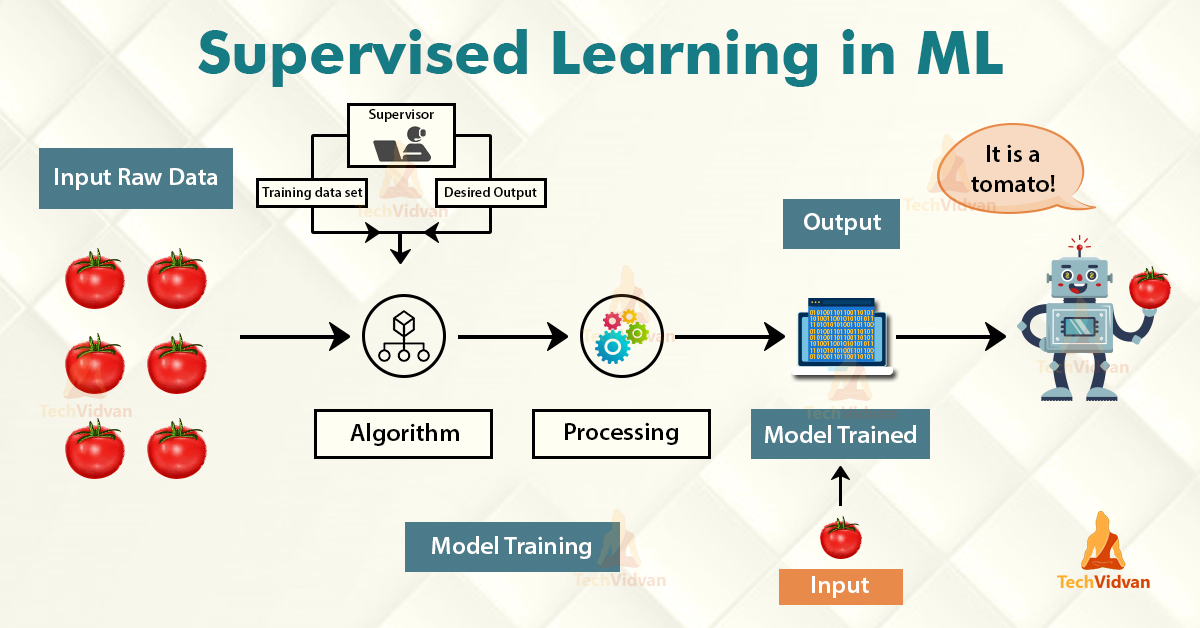


Figure 15- Supervised learning in ML

There are two main types of learning in ML: supervised and unsupervised. In unsupervised learning the model isn’t supervised, instead it is allowed to explore knowledge on its own. We accomplish this by giving the model an “unlabelled” dataset that does not specify which category or value is the “right” response. When we use supervised learning, we give the model a “labelled” dataset that tells it what the “right” value is.

### 3.4.1.1 Regression

Regression is a supervised learning technique that aids in the discovery of variable correlations and allows one to forecast a continuous output variable using one or more feature variables. Prediction, forecasting, time series modelling, and evaluating the cause-effect relationship between variables are all common applications. In my project, a single output value is produced for the price of a stock using the training data provided. This a probabilistic interpretation that is determined by taking into account the strength of association between the input variables.

## 3.4.2 Random Forest

Random Forest is a ML ensemble approach that is common due to its versatility, simplicity, and often high-quality performance. In this project, I use Random Forest algorithm to build a regression model that will help me predict the closing price of a stock based on a range of technical indicators.

Random Forest is made up of Decision Trees, which are the basic building blocks. In essence, Decision Trees are a flowchart framework in which each node tests a different attribute of an entity. Consider the following scenario: I have an individual who will represent our object. We then put this person’s qualities to the test. One test would be to determine whether they are male or female. In our tree, the test will be a “Decision Node” and each of the potential outcomes, “Male” or Female” will be a leaf node. Out “Root Node” will be the first “Decision Node” in our Decision Tree.

**Key Words**

* **Root Node**- The entire population or sample is represented by this node, which is then divided into two or more homogeneous sets. This is where we begin out Decision Tree.
* **Splitting**- The process of breaking down a node into two or more sub-nodes, such as gender.
* **Decision Node**- A decision Node is Formed when a sub-node splits into more sub-nodes.
* **Leaf/Terminal Node**- This refers to nodes that do not split.
* **Pruning-** this is a method of removing sub-nodes from a decision node. The splitting process can be defined as the polar opposite of splitting.
* **Branch/Sub-Tree**- This is a part of a tree that is smaller than the entire tree.
* **Parent and Child Node**- A parent node of sub-nodes is a node that is divided into sub-nodes, while sub-nodes are the children of the parent node.

Diagram

Description automatically generated

Figure 16-Bagging (Bootstrap Aggregation) Flow

Ensemble Learning models are models that uses decisions from multiple models combined to increase the model’s overall efficiency. Ensemble Learning Is perfectly summed up by the old adage that two heads are better than one. To get a better understanding of what the true response is, we combine the effects of the several models. Bagging is used for decision trees to increase model consistency in terms of reducing variance and improving accuracy, which removes the problem of overfitting.

In ensemble ML, bagging takes multiple poor models and aggregates the predictions to find the strongest one. The weak models specialise in specific areas of the feature space, allowing bagging leverage predictions to come from any model to achieve the highest level of accuracy.

Random Forest are typically more accurate than single decision trees for several reasons:

1. Instability: Even minor changes to the input data can have a significant impact on the decision tree’s overall structure.
2. They are always insufficiently reliable. With similar data, several other predictors perform better.
3. Knowledge benefit in decision trees is skewed in favour of attributes with more levels when data contains categorical variables of different number of levels.
4. Calculations can become extremely complicated, particularly when multiple values are unknown and/or multiple values are unknown and/or multiple outcomes are related.

Aforementioned are some of the reasons why Random Forest is superior to Decision Tree because it can help solve some of their flaws. There is no ideal model, as there is for everything. Just because something has flaws doesn’t mean it’s worthless; it simply means we need to be aware of them and keep an eye out for them while we use it.

### 3.4.2.1 Drawbacks

Although Random Forest comes with its advantages, are also some drawbacks which are discussed below:

* Random forest models are similar to black boxes, they lack model interpretability.
* It isn’t advantageous to use Random Forest on large datasets as the trees the size of the trees can consume a large amount of memory.
* Because Random Forest has a tendency to overfit, you should tune the hyperparameters.

## 3.4.3 Ridge Regression

### 3.4.3.1 Simple Linear Regression

Chart

Description automatically generated

Figure 17-Simple linear regression example

Simple linear Regression is a mathematical technique for extracting a formula to predict the

values of one Y variable from the values of another variable X when both variables have a causal relationship. X is called the independent variable and Y is called the dependant variable. It is called ‘simple’ because it only examines the relationship between two variables. It is linear because when the independent variable increases (or decreases), the dependent variable increases (or decreases) in a linear fashion.

Chart, scatter chart

Description automatically generated

Figure 18-Simple linear regression formula

As shown in figure 10, the goal is to obtain a relationship (model) between the X (Number of Years of Experience) and the Y (Salary) variable. Once the coefficients m and b are obtained, we will have obtained a simple linear model. This trained model can be later used to predict any salary based on the number of years of experience an employee has.

A picture containing diagram

Description automatically generated

Figure 19-Least Sum of Squares

### We need to find the Least Sum of Squares to get our model parameters. Least squares fitting is a method for determining the curve or line that best fits a range of points. The best fit curve or line is calculated using the sum of the squares of the offsets (residuals). The coefficients m and b are obtained using the least squares process, as shown in figure 12.

### 3.4.3.2 Regularisation

Regularisation techniques are used to avoid networks overfitting. Overfitting occurs when the model provides great results on. The training data but performs poorly on the testing dataset. Overfitted models generally provide high accuracy on the training dataset but low accuracy on the testing and validation (evaluation) datasets.

### 3.4.3.3 Ridge Regression

Diagram, schematic

Description automatically generated

Figure 20-Overfitting Data Example

Ridge regression is advantageous because it avoids overfitting. Ultimately, we want a model that can generalise patterns. works best on the training and testing data. Ridge regression works by applying a penalizing term (reducing the weights and biases) to overcome overfitting. As shown in figure 13, least sum of squares is applied to obtain the best fit line. Since the line passes through 3 training dataset points, the sum of squared residuals = 0. However, for the testing dataset, the sum of residuals is large, so the line has a high variance. Variance means that there is a difference in fir (or variability) between the training dataset and the testing dataset. This regression model is overfitting the training dataset.

Diagram, schematic

Description automatically generated

Figure 21-Ridge Regression Example

Ridge regression works by attempting to increase the bias to improve variance (generalisation capability). This works by changing the slope of the line as shown in figure 14. Although the model performance might be slightly poorer on the training set, but it will perform consistently well on both the training and testing datasets. Due to the slope being reduced with ridge regression penealty, the model becomes less sensitive to changes in the independent variable.

A picture containing line chart

Description automatically generated

Figure 22- L2 regularisation

## Ridge regression applies a factor of sum of squares of coefficients to the optimisation goal, which is known as L2 regularisation as shown in figure 15. Here, α (alpha) is the parameter that balances the importance of minimising RSS (Residual Sum of Squares) vs Minimising number of square coefficients. α can have a variety of values.

As alpha increases, the regression line's slope decreases, and the line becomes more horizontal. Furthermore, the model becomes less susceptible to changes in the independent variable.

### 3.4.3.4 Drawbacks

* Ridge regression increases bias.
* When building a Ridge regression model, you need to pick the perfect alpha (hyper parameter).
* The model's interpretability is poor.

### 3.4.3.5 Training

The code for the implementation of this model can be found in appendix F. The following steps were taken to create the model:

1. Obtain stock data from Alpha Vantage.

Table

Description automatically generated

Figure 23- Vanguard 500 Index Fund ETF

In this step the relevant stock data is obtained from Alpha Vantage. The dataset is already clean, so therefore no data cleansing is required at this stage.

1. Calculate Technical indicators

Table

Description automatically generated

Figure 24- Technical Indicators

In this step, the following Technical indicators are calculated:

* Stochastic Oscillator
* Williams %R
* Moving Average Convergence Divergence (MACD)
* Price Rate of Change

The data is cleaned after the Technical indicators are calculated, and any null values are removed.

1. Set the trading window we are trying to predict.

Because the primary goal of this project is to forecast future stock prices, we must shift the data so that the target stock price today is tomorrow's price, depending on how far into the future we want to forecast. This will be the target variable for the ML model.

1. Creating Feature

We only use two feature variables to build the ML model: the stock's opening price and its highest price. These, along with the technical indicators calculated in step 2, are chosen from the dataset. Every other column in the dataset is removed.

1. Create the training and test datasets

We will train our model on the training set and then test it on the test set. This is referred to as the holdout-validation method. Because it is not possible to select subsets of our data at random, I divided it in a 65:35 ratio. As shown in Figure 13, 65 percent of the data is used to train the model, while 35 percent is used for testing.

1. Build and evaluate regression model

## 3.4.4 Long Short-Term Memory (LSTM)

Neural networks are a series of algorithms based on how the brain functions. When you open your eyes, the data you see is processed by the Neurons (data processing cells) in your brain, which recognises what's around you. That's how close Neural Networks are to each other. They take a large amount of data, process it (drawing out patterns from it), and then output it.

Since they are not natural like neurons in your brain, neural networks are often referred to as Artificial Neural Networks (ANNs). They are designed to look and function like a neural network. An artificial neural network (ANN) is made up of a large number of highly interconnected computing elements (neurones) that work together to solve a specific problem.

ANNs, like adults and children, learn by example. Via a learning process, an ANN is optimised for a particular application, such as pattern recognition or data classification, image recognition, or voice recognition.

A picture containing pencil

Description automatically generated

Figure 25-Feed Forward ANN

Icon

Description automatically generated

Figure 26-Recurrent Neural Network [12]

Vanilla networks (feedforward neural networks) map a fixed size input (such as an image) to a fixed size output (classes or probabilities). Feedforward networks have the disadvantage of having no time dependence or memory impact. A recurrent neural network (RNN) is a type of artificial neural network (ANN) that is designed to understand the temporal dimension by providing a memory (feedback loop).

A picture containing text, clock

Description automatically generated

Figure 27-RNN architecture [13]

RNNs have a temporal loop in which the secret layer not only outputs anything, but also feeds itself. Time has been added as an extra. RNN will remember what happened in the previous time stamp, so it's ideal for text sequences.

Diagram

Description automatically generated

Figure 28-RNN math

A RNN takes an input of x and produces an output of o. The performance o is independent. The input x by itself, on the other hand, is dependent on the entire background of inputs fed to the network in previous time steps. Figure 19 depicts the two equations that control the RNN.

An LSTM is a form of recurrent neural network (RNN). In several ways, LSTMs outperform traditional feed-forward neural networks and RNNs. This is due to their ability to recall patterns selectively over long periods of time. This is due to the fact that LSTMs store information in a memory similar to that of a machine. The LSTM has the ability to read, write, and erase data from its memory. The memory can be thought of as a gated cell, this means that the cell determines whether or not to store or erase information (i.e., whether to open or close the gates) depending on the value it assigns to the information. Weights, which are also learned by the algorithm, are used to assign importance. This simply means that it learns what data is useful over time and what data is not.

Diagram, schematic

Description automatically generated

Figure 29- an illustration of RNN with three gates [10]

As shown in figure 16, the input, forget, and output gates are all present in an LSTM. These gates decide whether new input should be allowed (input gate), whether it should be deleted (forget gate), or whether it should have an effect on the output at the current timestep (output gate). Each gate consists of a sigmoid neural net layer along with a pointwise multiplication operation. Sigmoid output ranges from 0 to 1, where 0 does allow data to flow and 1 allows everything to flow.

Diagram

Description automatically generated

Figure 30-Vanishing gradient problem

# Since they avoid the vanishing gradient problem, LSTM networks outperform traditional RNNs. When an error must propagate across all previous layers, a vanishing gradient is the product shown in figure 21. The network weights are no longer changed as the gradient decreases. The gradients of the loss function approach zero as more layers are added, making the network difficult to practise.

# With the recent advances in data science, it has been discovered that Long Short Term Memory networks, also known as LSTMs, are the most powerful solution for almost all of these sequence prediction problems. LSTM network has even been trained to write movies [14]. The LSTM network was trained with corpus of dozens of sc-fi screenplays and movies dating back from the 1980s and 90s.

### 3.4.4.1 Drawbacks

* Because they could solve the problem of vanishing gradients, LSTMs became popular. However, they fail to completely remove it.
* They necessitate a significant investment of resources and time in order to be properly trained and prepared for real-world applications.
* Overfitting is a problem with LSTMs, and the dropout algorithm is difficult to use to combat it. Dropout refers to a regularisation technique that leaves out input and recurrent connections to LSTM units from activation and weight updates while training a network.

## 3.5 Model Evaluation

Text

Description automatically generated

Figure 31- Example Evaluation Metrics

In data science, model evaluation is critical. It assists you in understanding the quality of each model, thus allowing you to determine which works best with the current and future data. Various models can be used to evaluate regression model, I will use the following for this project:

## 3.5.1 R Square/Adjusted R Square

R Square is a metric of how much of the variability in the dependent variable can be explained by the model. It is called R Square because It is the square of the Correlation Coefficient (R).

Text

Description automatically generated with medium confidence

Figure 32-R Square formula [16]

To calculate R Square, we start by squaring the prediction error and dividing it by the total sum of squares that replace the calculated prediction with mean. The R Square value ranges from 0 to 1, with higher value indicating a better fit between prediction and actual value.

Although R Square is a useful metric for determining how well a model fits the dependant variables, it fails to account for the overfitting problem. For example, if there is a regression model that has many independent variables, it may fit really well to the training data but perform poorly on the testing data because the model is too complicated. Therefore, Adjusted R Square is introduced; it penalises the addition of additional independent variables to the model and it adjusts the metric to avoid overfitting.

From the sample model shown in figure 29, the model can explain 61 percent of the dependent variability, and Adjusted R Square is roughly the same as R Square, indicating that the model is quite robust.

## 3.5.2 Mean Square Error (MSE)/Root Mean Square Error (RMSE)

In this project, we include the Mean Square Error to assess the absolute goodness for the fit.

A white background with black text

Description automatically generated with low confidence

Figure 33-Mean Square Error Formula [16]

MSE is calculated by adding the sum of the squares of the prediction error, which is the difference between the real and predicted output, and afterwards dividing by the number of data points. It provides an absolute number indicating the amount of your predicted results differ from the true amount.

The square root of MSE is the Root Mean Square Error (RMSE). It is the preferred metric of evaluation in this project for two reasons. For starters, the MSE value may be too large to compare. Second, because MSE is calculated by the square error, taking the square root returns it to the same level of prediction and makes it easier to interpret.

## 3.5.3 Mean Absolute Error (MAE)

 MAE is comparable to MSE. MAE, on the other hand, takes the sum of absolute value of error rather than the sum of square of error as MSE does.

A white background with black text

Description automatically generated with medium confidence

Figure 34-MAE Formula [16]

# 4 Implementation and Results

In this project I have implemented three different ML models to predict the price of a company’s stock. In this chapter, I will give an in depth description of how each of them were implemented.

## 4.1 Modules and Packages

The ML models in this project have been implemented using Python 3. Various Python packages were used to implement this project, including the following:

* NLTK[17] - The Natural Language Toolkit (NLTK) is a leading platform for developing Python programmes that work with human language data. It offers simple interfaces to over 50 corpora and lexical resources, including WordNet, as well as a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, as well as wrappers for industrial-strength NLP libraries.
* Scikit-Learn[18] - Scikit-learn is a free and open source machine learning library that can perform both supervised and unsupervised learning. It also includes tools for model fitting, data pre-processing, model selection and evaluation, and a variety of other utilities.
* Matplotlib[19] - Matplotlib is a Python library that allows you to create static, animated, and interactive visualisations.
* Pandas[20] - Pandas is an open source data analysis and manipulation tool built on top of the Python programming language that is fast, powerful, flexible, and simple to use.
* Numpy[21] - NumPy is a Python library that is used to work with arrays. It also includes functions for working with linear algebra, the Fourier transform, and matrices.
* Seaborn[ 22] – Seaborn is a Python library for creating statistical graphics. It is built on top of matplotlib and closely integrates with Panda’s data structures. Seaborn assists you in exploring and comprehending your data. Its plotting functions work on data frames and arrays containing entire datasets, performing the necessary semantic mapping and statistical aggregation domestically to produce informative plots. Its dataset-oriented, declarative API enables us to focus on what the different elements of your plots signify instead of how to draw them.
* AlphaVantage[ 22]- Alpha Vantage API provides access to historical and real-time data for a variety of markets. The API allows me to access the data directly in python, from there I can manipulate the data or store it for later use. Alpha Vantage provides its service at no fee. They permit 5 requests per minute and 500 requests per day.
* Yfinance[ 22]- Yfinance provides a dependable, threaded, and Pythonic method for downloading historical market data from Yahoo! Finance.
* Plotly[24]- Plotly is a Python graphing library that is interactive, open-source, and browser-based. It is a high-level, declarative charting library built on top of plotly.js, containing over 30 chart types, including Scientific charts and 3D graphs.
* Tensorflow [25]- TensorFlow is a complete open-source ML platform. It has an extensive, adaptable ecosystem of tools, libraries, and community resources that allow researchers to push the boundaries of ML and developers to quickly create and deliver ML-powered applications.
* Math [26]- Math Provides access to basic math functions ad constants in Python, these can be combined throughout our code to perform mathematical computations of a complex nature.
* Streamlit [27]- It is an open-source Python library that makes it easy to create and share beautiful, custom web apps for ML and data science.

## 4.2 Streamlit User Interface (UI)

In this project, I use Streamlit to create a custom web application for the machine learning models I created.

### 4.2.1 Query Parameters

Graphical user interface, application, Teams

Description automatically generated

Figure 35-Example Query Parameters

Query Parameters are used to obtain data from Alpha Vantage and build the ML models. The user inputs the following parameters before any calculations are done within the application:

* **Start Date**- the data obtained from the Alpha Vantage API's beginning period
* **End Date**- the data obtained from the Alpha Vantage API's end period
* **Stock Symbol**- a string of characters (usually letters) that represents publicly traded companies on an exchange.
* **Number of Days**- the trading window we are trying to predict.
* **Model**- ML model used to train data.

### 4.2.2 Company Description

Graphical user interface, text

Description automatically generated

Figure 36-Example Company Description

Based on the stock symbol selected by the user, the company name and description are retrieved from the yfinance library using Python. Because the description is often long and contains unnecessary information NLTK- the Natural Language Toolkit is applied to summarize and shorten the text [28]. This is done in the following steps:

Step 1: Import required libraries

For this feature, there two NLTK libraries that are necessary for summarising this text.

1. from nltk.corpus import stopwords- a data set containing pre-determined stop words
2. from nltk.tokenize import word\_tokenize, sent\_tokenize - tokenizer for words and sentences.

Step 2: Stop Words are being removed and stored in a separate array of words. Any word such as (is, a, an, the, for) that adds nothing to the meaning of a sentence. For example,

“Osato is one of the best programmers for freelance work “, a few words can be removed to reduce the word count whilst preserve the meaning as follows: [“Osato”, “one”, “best”, “programmers”, “freelance”, “work”].

Step 3: A frequency table of words created. After removing the stop words, a Python dictionary will keep track of how many times each word appears in the feedback.

* 1. stopWords = set(stopwords.words("english"))
  2. words = word\_tokenize(text)
  3. frequencyTable = dict()

Step 4: A score is assigned to each sentence based on the words it contains and the frequency table. Secondly, the score for each sentence is tracked using a dictionary. Later on, we iterate through the dictionary to generate the summary.

1. sentences = sent\_tokenize(text)
2. sentenceValue = dict()

Step 5: To compare the sentences in the feedback, a score is assigned. Finding the average can be a good criterion for comparing scores.

1. sumOfValues = 0
2. for sentence in sentenceValue:

    sumOfValues += sentenceValue[sentence]

1. average = int(sumOfValues / len(sentenceValue))

Code: Text Summarizer is fully implemented in Python is shown in appendix I.

### 4.2.3 Closing Price

Chart

Description automatically generated

Figure 37- Example Closing Price Chart

In the web application, the selected stocks closing price is also analysed. Figure 35 shows trendline used to indicated whether the behaviour of a stock has been positive or negative overtime. Although stocks generally fluctuate over the course of the day, the overall behaviour persists over time. So therefore, understanding trends enables predictive analysis.

I used NumPy's polynomial fitting function, polyfit(), to identify the best adjusting first degree polynomial and its fitting error in order to identify patterns. The slope and offset of a first-degree polynomial are its two parameters. If the slope is greater than zero, there is a positive trend in the data, otherwise vice versa.

### 4.2.4 Data Correlation

Graphical user interface, application

Description automatically generated with medium confidence

Figure 38-Correlation Matrix using Seaborn’s heatmap functionality

Identifying and reducing highly correlated features in a dataset is one of the quickest ways to strengthen a model. Correlated features will add noise and inaccuracy to a model, making it more difficult to achieve the anticipated result.

Figure 36 may appear a little perplexing at first glance. It is, however, relatively simple. On both axes, each feature (variable) is listed, and their relationships with other variables are coloured. The darker the colour, the more highly correlated those variables are and should not be paired in the same model. For this project, only variables that are highly correlated with Closing Price are considered.

The opening price of the stock and its highest price are the two features used to build the ML models in this project. When selecting features, the principle of Occam's razor was used, because after two features were chosen, the number of features had no effect on the model's performance.

### 4.2.5 Descriptive Statistics

A picture containing text, monitor, black, screen

Description automatically generated

Figure 39-Descriptive Statistics

Descriptive statistics are an important part of machine learning because they provide us with a deeper understanding of the data; ignoring these insights often leads to incorrect conclusions. It also presents the data in a meaningful way, as illustrated in figure 37, allowing for easier interpretation of the data.

Measures of central tenancy (mean) are used in the web application to describe the dataset by identifying a central position. Furthermore, variability measures (interquartile range) are used to quantify the amount of spread or variability within the data.

### 4.2.6 Models

### 4.2.6 Models

#### 4.2.6.1 Pre-processing

I used Scikit-Learn’s MinMax Scaler to pre-process my data in order to make it more simplified for the model. It transforms features by scaling all of the features to a certain range. The data in this project was scaled to numbers between zero and one.

To utilise this in my project, I used the following lines of code:

1. from sklearn.preprocessing import MinMaxScaler
2. sc = MinMaxScaler(feature\_range=(0, 1))
3. stock\_df\_targeted\_scaled=sc.fit\_transform(stock\_df\_targeted\_scaled.drop(columns=['date']))

#### 4.2.6.2 Ridge Regression

The first model built in this project was the

# References

[[1] https://www.statista.com/statistics/324578/market-value-of-companies-on-the-london-stock-exchange/](%5b1%5d%20https://www.statista.com/statistics/324578/market-value-of-companies-on-the-london-stock-exchange/)

[2] Marutitech.com (-) 12 Use Cases of AI and Machine Learning In Finance [ONLINE]. Available at: <https://marutitech.com/ai-and-ml-in-finance/>

[3] <https://www.pxfuel.com/en/free-photo-qualj>

[4] <https://www.crisil.com/en/home/our-businesses/crisil-coalition.html>

[5] <https://www.itprotoday.com/machine-learning/how-ai-trading-systems-will-shake-wall-street>

[6] <https://www.bloomberg.com/news/articles/2017-09-20/mizuho-is-said-to-offer-ai-trading-service-before-mifid-overhaul>

[7] <https://www.cybersecobservatory.com/2017/06/06/artificial-intelligence-transforming-investment-strategies/>

[8] <https://www.auquan.com/>

[9] <https://equbot.com/>

[10] <https://builtin.com/data-science/recurrent-neural-networks-and-lstm>

[11] <https://commons.wikimedia.org/wiki/File:RecurrentLayerNeuralNetwork_english.png>

[12] <https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg>

[13] <https://fr.wikipedia.org/wiki/Fichier:Recurrent_neural_network_unfold.svg>

[14] <https://arstechnica.com/gaming/2016/06/an-ai-wrote-this-movie-and-its-strangely-moving/>

[15] <https://hackernoon.com/what-steps-should-one-take-while-doing-data-preprocessing-502c993e1caa>

[16] <https://towardsdatascience.com/what-are-the-best-metrics-to-evaluate-your-regression-model-418ca481755b>

[17] NLTK 3.5 Documentation [online] https://www.nltk.org/

[18] Scikit-learn 0.21.3 Documentation [online] https://scikit-learn.org/0.21/ documentation.html

[19] Matplotlib 3.3.2 Documentation [online] https://matplotlib.org/users/ index.html

[20] Pandas Documentation [online] https://pandas.pydata.org/docs/ user\_guide/index.html

[21] Numpy 1.19 Documentation [online] https://numpy.org/doc/stable/

[22] Seaborn [online] <https://seaborn.pydata.org/introduction.html>

[23] <https://pypi.org/project/yfinance/>

[24] <https://pypi.org/project/plotly/>

[25] <https://www.tensorflow.org>

[26] <https://docs.python.org/3/library/math.html>

[27] <https://streamlit.io>

[28] <https://www.geeksforgeeks.org/python-text-summarizer/>

[29] https://www.emilkhatib.com/analyzing-trends-in-data-with-pandas/

[?]<https://github.com/tthustla/efficient_frontier/blob/master/Efficient%20_Frontier_implementation.ipynb>

[?] <https://towardsdatascience.com/efficient-frontier-portfolio-optimisation-in-python-e7844051e7f>

# Appendix A

## Indicator Calculation: Relative Strength Index (RSI)

### Code Explanation:

1. Copy the desired columns and store them in new variables.
2. To calculate the indicator, group the columns by symbol, pick the column we want to transform, and use the transform method along with a lambda function.
3. Save the values in the main data frame

### Code:

def calc\_RSI(data\_dated):

# Calculate the 14 day RSI

n = 14

# First make a copy of the data frame twice

up\_df, down\_df = data\_dated[['Ticker','change\_in\_price']].copy(), data\_dated[['Ticker','change\_in\_price']].copy()

# For up days, if the change is less than 0 set to 0.

up\_df.loc['change\_in\_price'] = up\_df.loc[(up\_df['change\_in\_price'] < 0), 'change\_in\_price'] = 0

# For down days, if the change is greater than 0 set to 0.

down\_df.loc['change\_in\_price'] = down\_df.loc[(down\_df['change\_in\_price'] > 0), 'change\_in\_price'] = 0

# We need change in price to be absolute.

down\_df['change\_in\_price'] = down\_df['change\_in\_price'].abs()

# Calculate the EWMA (Exponential Weighted Moving Average), meaning older values are given less weight compared to newer values.

ewma\_up = up\_df.groupby('Ticker')['change\_in\_price'].transform(lambda x: x.ewm(span = n).mean())

ewma\_down = down\_df.groupby('Ticker')['change\_in\_price'].transform(lambda x: x.ewm(span = n).mean())

# Calculate the Relative Strength

relative\_strength = ewma\_up / ewma\_down

# Calculate the Relative Strength Index

relative\_strength\_index = 100.0 - (100.0 / (1.0 + relative\_strength))

# Add the info to the data frame.

data\_dated['down\_days'] = down\_df['change\_in\_price']

data\_dated['up\_days'] = up\_df['change\_in\_price']

data\_dated['RSI'] = relative\_strength\_index

# Appendix B

## Indicator Calculation: Stochastic Oscillator

### Code Explanation:

1. Copy the desired columns and store them in new variables.
2. To calculate the indicator, group the columns by symbol, pick the column we want to transform, and use the transform method along with a lambda function.
3. Save the values in the main data frame

### Code:

def stochastic\_Oscillator(data\_dated):

# Calculate the Stochastic Oscillator

n = 14

# Make a copy of the high and low column.

low\_14, high\_14 = data\_dated[['Ticker','3. low']].copy(), data\_dated[['Ticker','2. high']].copy()

# Group by symbol, then apply the rolling function and grab the Min and Max.

low\_14 = low\_14.groupby('Ticker')['3. low'].transform(lambda x: x.rolling(window = n).min())

high\_14 = high\_14.groupby('Ticker')['2. high'].transform(lambda x: x.rolling(window = n).max())

# Calculate the Stochastic Oscillator.

k\_percent = 100 \* ((data\_dated['4. close'] - low\_14) / (high\_14 - low\_14))

# Add the info to the data frame.

data\_dated['low\_14'] = low\_14

data\_dated['high\_14'] = high\_14

data\_dated['k\_percent'] = k\_percent

# Appendix C

## Indicator Calculation: Williams %R

### Code Explanation:

1. Copy the desired columns and store them in new variables.
2. To calculate the indicator, group the columns by symbol, pick the column we want to transform, and use the transform method along with a lambda function.
3. Save the values in the main data frame

### Code:

def calc\_williams\_r(data\_dated):

# Calculate the Williams %R

n = 14

# Make a copy of the high and low column.

low\_14, high\_14 = data\_dated[['Ticker','3. low']].copy(), data\_dated[['Ticker','2. high']].copy()

# Group by symbol, then apply the rolling function and grab the Min and Max.

low\_14 = low\_14.groupby('Ticker')['3. low'].transform(lambda x: x.rolling(window = n).min())

high\_14 = high\_14.groupby('Ticker')['2. high'].transform(lambda x: x.rolling(window = n).max())

# Calculate William %R indicator.

r\_percent = ((high\_14 - data\_dated['4. close']) / (high\_14 - low\_14)) \* - 100

# Add the info to the data frame.

data\_dated['r\_percent'] = r\_percent

# Appendix D

## Indicator Calculation: Moving Average Convergence Divergence (MACD)

### Code Explanation:

1. Copy the desired columns and store them in new variables.
2. To calculate the indicator, group the columns by symbol, pick the column we want to transform, and use the transform method along with a lambda function.
3. Save the values in the main data frame

### Code:

def calc\_macd(data\_dated):

# Calculate the MACD

ema\_26 = data\_dated.groupby('Ticker')['4. close'].transform(lambda x: x.ewm(span = 26).mean())

ema\_12 = data\_dated.groupby('Ticker')['4. close'].transform(lambda x: x.ewm(span = 12).mean())

macd = ema\_12 - ema\_26

# Calculate the EMA

ema\_9\_macd = macd.ewm(span = 9).mean()

# Store the data in the data frame.

data\_dated['MACD'] = macd

data\_dated['MACD\_EMA'] = ema\_9\_macd

# Appendix E

## Indicator Calculation: Price Rate Of Change

### Code Explanation:

1. Copy the desired columns and store them in new variables.
2. To calculate the indicator, group the columns by symbol, pick the column we want to transform, and use the transform method along with a lambda function.
3. Save the values in the main data frame

### Code:

def calc\_price\_rate\_of\_change(data\_dated):

# Calculate the Price Rate of Change

n = 9

# Calculate the Rate of Change in the Price, and store it in the Data Frame.

data\_dated['Price\_Rate\_Of\_Change'] = data\_dated.groupby('Ticker')['4. close'].transform(lambda x: x.pct\_change(periods = n))

# Appendix F

## Ridge Regression Model

### Code Explanation:

1. Obtain live data about the given stock from Alpha vantage API.
2. Calculate technical indicators.
3. Set the trading window we are trying to predict.
4. Remove unwanted columns.
5. Feature scaling
6. Create feature and target variables
7. Create and train model
8. Make a prediction using features
9. Obtain evaluation metrics
10. Create a data frame with original closing prices vs predicted closing prices
11. Create an interactive plot of closing and predicted prices.

### Code:

def pricePrediction\_LR(symbol,days, starting\_date):

# obtain stock data from alpha vantage

stock\_df= stock\_data(symbol,starting\_date)

# obtaining technical indicators

stochastic\_Oscillator(stock\_df)

calc\_williams\_r(stock\_df)

calc\_macd(stock\_df)

calc\_price\_rate\_of\_change(stock\_df)

# set the trading window we are trying to predict

stock\_df\_targeted = trading\_window(stock\_df,days)

stock\_df\_targeted.reset\_index(inplace = True)

stock\_df\_targeted = stock\_df\_targeted.dropna()

stock\_df\_targeted\_scaled= stock\_df\_targeted

stock\_df\_targeted\_scaled.head(10)

# remove unwanted columns

stock\_df\_targeted\_scaled.drop(['Ticker','1. open','2. high','3. low', '5. adjusted close', '6. volume', '8. split coefficient', 'low\_14','high\_14','MACD\_EMA'], axis = 1, inplace=True)

stock\_df\_targeted\_scaled = sc.fit\_transform(stock\_df\_targeted\_scaled.drop(columns = ['date']))

# Creating Feature and Target

X = stock\_df\_targeted\_scaled[:,:6]

y = stock\_df\_targeted\_scaled[:,6:]

split = int(0.65 \* len(X))

X\_train = X[:split]

y\_train = y[:split]

X\_test = X[split:]

y\_test = y[split:]

show\_plot(X\_train, 'Training Data')

show\_plot(X\_test, 'Testing Data')

regression\_model = Ridge()

# fit model

regression\_model.fit(X\_train, y\_train)

# obtain model score

lr\_accuracy = regression\_model.score(X\_test, y\_test)

# get a list of predicted prices

predicted\_prices = regression\_model.predict(X)

# print out the evaluation metrics

print("Linear Regression Score: ", lr\_accuracy)

print('RMSE: ' + str(math.sqrt(mean\_squared\_error(y, predicted\_prices))))

print('Rsquared '+ str(r2\_score(y, predicted\_prices)))

print('MAE: ' + str(mean\_absolute\_error(y, predicted\_prices)))

# put the predicted values into a list

Predicted = []

for i in predicted\_prices:

Predicted.append(i[0])

# put the closing prices into a list

close = []

for i in stock\_df\_targeted\_scaled:

close.append(i[0])

# create a new df for only close and predicted prices

df\_predicted = stock\_df\_targeted[['date']]

df\_predicted['Close'] = close

df\_predicted['Prediction'] = Predicted

# create a plot of closing and predicted prices

interactive\_plot(df\_predicted, "Original Vs. Prediction")

### 4.2 Building the Model

The steps used to build this model are very similar to the ones used in to build the Random Forest model shown below.

# Appendix G

## Random Forest

def pricePrediction\_RandomForest(symbol,days, starting\_date):

p=0

mse=[]

rmse=[]

rsquared=[]

mae=[]

# obtain stock data

stock\_df= stock\_data(symbol,starting\_date)

# obtaining technical indicators

stochastic\_Oscillator(stock\_df)

calc\_williams\_r(stock\_df)

calc\_macd(stock\_df)

calc\_price\_rate\_of\_change(stock\_df)

# set the trading window we are trying to predict

stock\_df\_targeted = trading\_window(stock\_df,days)

stock\_df\_targeted.reset\_index(inplace = True)

stock\_df\_targeted = stock\_df\_targeted.dropna()

stock\_df\_targeted\_scaled= stock\_df\_targeted

# stock\_df\_targeted\_scaled.head(10)

stock\_df\_targeted\_scaled.drop(['Ticker','2. high','3. low', '5. adjusted close','7. dividend amount' ,'6. volume', '8. split coefficient', 'low\_14','high\_14','MACD\_EMA'], axis = 1, inplace=True)

stock\_df\_targeted\_scaled = sc.fit\_transform(stock\_df\_targeted\_scaled.drop(columns = ['date']))

# # Creating Feature and Target

X = stock\_df\_targeted\_scaled[:,:6]

y = stock\_df\_targeted\_scaled[:,6:]

split = int(0.65 \* len(X))

X\_train = X[:split]

y\_train = y[:split]

X\_test = X[split:]

y\_test = y[split:]

show\_plot(X\_train, 'Training Data')

show\_plot(X\_test, 'Testing Data')

rf = RandomForestRegressor()

rf.fit(X\_train, y\_train.ravel())

pred\_rf = rf.predict(X)

print('MSE: ' +str(mean\_squared\_error(y, pred\_rf)))

print('RMSE: ' + str(math.sqrt(mean\_squared\_error(y, pred\_rf))))

print('Rsquaed: '+ str(r2\_score(y, pred\_rf)))

print('MAE: ' + str(mean\_absolute\_error(y, pred\_rf)))

print('')

Predicted = []

for i in pred\_rf:

Predicted.append(i)

close = []

for i in stock\_df\_targeted\_scaled:

close.append(i[0])

df\_predicted = stock\_df\_targeted[['date']]

df\_predicted['Close'] = close

df\_predicted['Prediction'] = Predicted

interactive\_plot(df\_predicted, "Original Vs. Prediction for " )

### Code Explanation:

When building the Random Forest model, the first thing I did was to load the stock data from Alphavantage API. I then checked the head of the dataset to get a glimpse of the kind of dataset I am working with.

The Open column is the starting price of the stock for each day, while the Close column is the final price of the stock for that particular trading day. The High and Low columns represent the highest and lowest prices for each day. For our modelling we will use Open and High columns combined with technical indicators.

Next, the technical indicators referenced in appendixes A-E are calculated. Then we set the ‘trading window’ we are trying to predict. Note that the goal of this project is to predict future stock price, so for example if our trading window is 1 then the target stock price today will be tomorrow’s price. All unwanted columns are then removed, and to improve the performance of the model, I scaled the data using Scikit-Learn’s MinMaxScaler and scaled the dataset to numbers between zero and one. In addition, I build and train our model. After training the model, I then tried to predict the stock prices of the data.

Chart, line chart, histogram

Description automatically generated

Figure 40- Actual vs Predicted prices of APPL stock using a 1-day trading window

To evaluate the performance of our model, I then obtained the evaluation metrics using the scikit-learn library. Finally, I created an interactive plot of the predicted stock prices vs the original closing prices of the data shown above.

# Appendix H

## LSTM Model

### Code Explanation:

def pricePrediction\_LSTM(symbol,days, starting\_date):

# obtain stock data

stock\_df= stock\_data(symbol,starting\_date)

# obtaining technical indicators

stochastic\_Oscillator(stock\_df)

calc\_williams\_r(stock\_df)

calc\_macd(stock\_df)

calc\_price\_rate\_of\_change(stock\_df)

stock\_df.reset\_index(inplace = True)

# set the trading window we are trying to predict

stock\_df\_targeted = trading\_window(stock\_df,days)

stock\_df\_targeted.drop(['Ticker','4. close','7. dividend amount','3. low', '5. adjusted close', '6. volume', '8. split coefficient', 'low\_14','high\_14','MACD\_EMA'], axis = 1, inplace=True)

stock\_df\_targeted.dropna(inplace=True)

training\_data\_X = stock\_df\_targeted.iloc[:, 1:6].values

training\_data\_y = stock\_df\_targeted.iloc[:, 6:].values

stock\_df\_targeted\_scaled= sc.fit\_transform(stock\_df\_targeted.drop(columns = ['date']))

X = sc.fit\_transform(training\_data\_X)

y = sc.fit\_transform(training\_data\_y)

# Convert the data into array format

X = np.asarray(X)

y = np.asarray(y)

# Split the data

split = int(0.7 \* len(X))

X\_train = X[:split]

y\_train = y[:split]

X\_test = X[split:]

y\_test = y[split:]

# Reshape the 1D arrays to 3D arrays to feed in the model

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

X\_train.shape, X\_test.shape

# Create the model

inputs = keras.layers.Input(shape=(X\_train.shape[1], X\_train.shape[2]))

x = keras.layers.LSTM(150, return\_sequences= True)(inputs)

x = keras.layers.Dropout(0.3)(x)

x = keras.layers.LSTM(150, return\_sequences=True)(x)

x = keras.layers.Dropout(0.3)(x)

x = keras.layers.LSTM(150)(x)

outputs = keras.layers.Dense(1, activation='linear')(x)

model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(optimizer='adam', loss="mse", metrics=['mean\_squared\_error', 'mae'])

model.summary()

# Train the model

history = model.fit(

X\_train, y\_train,

epochs = 20,

batch\_size = 32,

validation\_split = 0.2

)

predicted = model.predict(X)

test\_predicted = []

for i in predicted:

test\_predicted.append(i[0])

close = []

for i in stock\_df\_targeted\_scaled:

close.append(i[0])

df\_predicted = stock\_df\_targeted[['date']]

df\_predicted['Close'] = close

df\_predicted['Prediction'] = predicted

# interactive\_plot(df\_predicted, "Original Vs. Prediction for " )

scores = model.evaluate(X, y, verbose=0)

print("MSE:" + str((scores[0])))

print("MAE:" + str((scores[1])))

print('R2 Score: ', r2\_score(y, predicted))

# Plot the data

interactive\_plot(df\_predicted, "Original Vs Prediction")

### Building the Model

The steps to build the LSTM model are slightly different from the two other models previously mentioned. This is mainly because LSTM expects the data fed into the model to be in a specific format, usually a 3D array.

Additionally, when building this model, I Keras which is Tensor Flow 2’s high level API. TensorFlow 2 is a complete machine learning framework that is open source. It's a differentiable programming infrastructure layer. It brings together four main abilities:

* Using the CPU, GPU, or TPU to efficiently perform low-level tensor operations.
* The gradient of arbitrary differentiable expressions can be efficiently computed.
* Scaling computing to a large number of devices.
* Programmes (“graphs”) can be exported to external runtimes such as servers, browsers, mobile and embedded devices.

Keras is a user-friendly, high-productive platform for solving machine learning problems, which has an emphasis on modern deep learning. It provides the required abstractions and building blocks for developing and shipping high-iteration-rate machine learning solutions.

The three main modules imported from Keras in my project are:

* Dense- this is used to add a densely connected neural network layer.
* LSTM- this is used for adding the Long Short-Term Memory layer.
* Dropout- this is used for adding dropout layers which prevent overfitting.

The LSTM layer is added with the following arguments:

* 150 units which is the dimensionality of the output space.
* Return\_sequences=True which decides if the last output in the output should be returned.
* Input which is the shape of the training set

Dropout layers are defined by specifying 0.3, which means that 30% of the layers will be dropped. When the model is compiled, adam optimizer is used which is very popular. In addition, the loss is set as ‘mse’. Thus, meaning the mean of the squared errors will be computed. Finally, the model is fitted to run on 20 epochs with a batch size of 32. The runtime id. The model will vary depending on the specs of the given computer.

# Appendix I

## NLTK Text Summarizer

def summarise\_text(string\_summary):

# Input text - to summarize

text = string\_summary

# Tokenizing the text

stopWords = set(stopwords.words("english"))

words = word\_tokenize(text)

# Creating a frequency table to keep the

# score of each word

frequencyTable = dict()

for word in words:

word = word.lower()

if word in stopWords:

continue

if word in frequencyTable:

frequencyTable[word] += 1

else:

frequencyTable[word] = 1

# Creating a dictionary to keep the score

# of each sentence

sentences = sent\_tokenize(text)

sentenceValue = dict()

for sentence in sentences:

for word, freq in frequencyTable.items():

if word in sentence.lower():

if sentence in sentenceValue:

sentenceValue[sentence] += freq

else:

sentenceValue[sentence] = freq

sumOfValues = 0

for sentence in sentenceValue:

sumOfValues += sentenceValue[sentence]

# Average value of a sentence from the original text

average = int(sumOfValues / len(sentenceValue))

# Storing sentences into our summary.

textsummary = ''

for sentence in sentences:

if (sentence in sentenceValue) and (sentenceValue[sentence] > (1.2 \* average)):

textsummary += " " + sentence

return textsummary