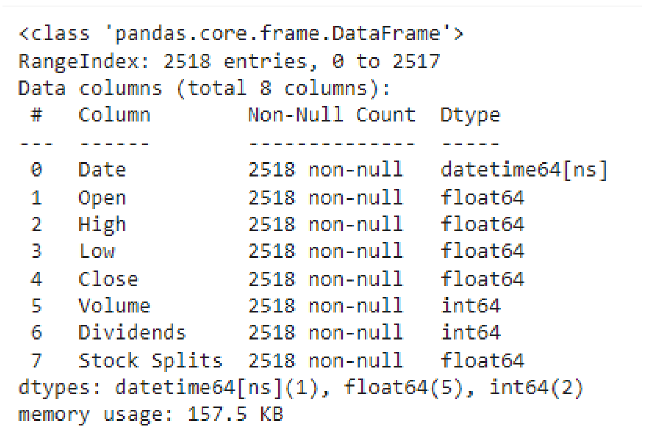
<u>Descriptive analysis of the dataset (Joe & Rob)</u>

The data we used was downloaded from Yahoo Finance using Python package yfinance.

The datasets we decided to use were the top 5 largest market capitalisation stocks in the USA: Apple, Microsoft, Amazon, Google and Tesla, with a focus on Tesla. (reference https://companiesmarketcap.com/usa/largest-companies-in-the-usa-by-market-cap/

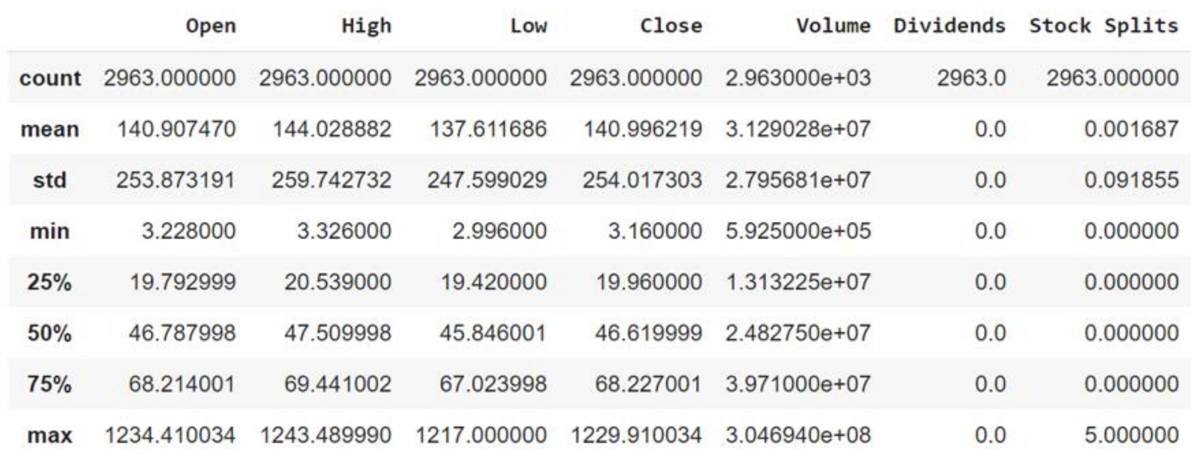
<u>Checking missing values:</u>



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2518 entries, 0 to 2517
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Date          2518 non-null   datetime64[ns]
 1   Open          2518 non-null   float64
 2   High          2518 non-null   float64
 3   Low           2518 non-null   float64
 4   Close         2518 non-null   float64
 5   Volume        2518 non-null   int64
 6   Dividends     2518 non-null   int64
 7   Stock Splits  2518 non-null   float64
dtypes: datetime64[ns](1), float64(5), int64(2)
memory usage: 157.5 KB
```

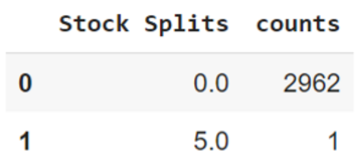
We can see above that for the Tesla values, there are 2518 entries and all 2518 of these values are non-null. The same was true for the four other stocks. Therefore we do not need to worry about replacing or dealing with missing values.

<u>Exploring summary statistics of Tesla dataset:</u>



	Open	High	Low	Close	Volume	Dividends	Stock Splits
count	2963.000000	2963.000000	2963.000000	2963.000000	2.963000e+03	2963.0	2963.000000
mean	140.907470	144.028882	137.611686	140.996219	3.129028e+07	0.0	0.001687
std	253.873191	259.742732	247.599029	254.017303	2.795681e+07	0.0	0.091855
min	3.228000	3.326000	2.996000	3.160000	5.925000e+05	0.0	0.000000
25%	19.792999	20.539000	19.420000	19.960000	1.313225e+07	0.0	0.000000
50%	46.787998	47.509998	45.846001	46.619999	2.482750e+07	0.0	0.000000
75%	68.214001	69.441002	67.023998	68.227001	3.971000e+07	0.0	0.000000
max	1234.410034	1243.489990	1217.000000	1229.910034	3.046940e+08	0.0	5.000000

The table above shows dividends as all zeros, so can omit this from any future analysis. The Stock Splits column looks like it contains a lot of zeros, but not all as there is a 5.0 value as the max, so this will need more exploration.



```
   Stock Splits  counts
0           0.0    2962
1           5.0       1
```

From the table on the left we can see that there is one occurrence of the 5.0 value, the rest are 0.0 values.

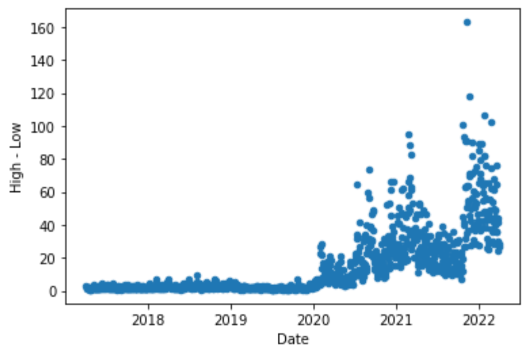
Date ranges:

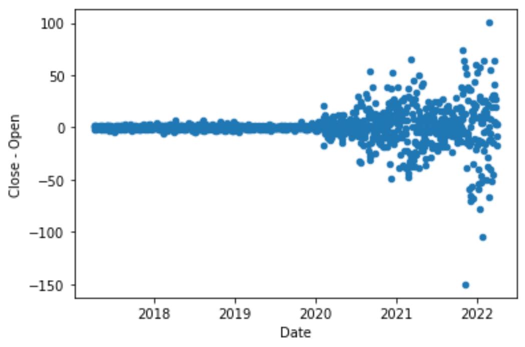
Stock	Earliest Date
Apple	1980-12-12
Microsoft	1986-03-13
Amazon	1997-05-15
Google	2004-08-19
Tesla	2010-06-29

Through exploring the datasets, we noticed that not all 5 of the largest stocks had been traded on the stock market for the same length of time. Below is a table detailing the earliest dates. All stocks' latest dates are the present date as the yfinance data is updated daily. This makes only the last 10 years as the date range where all 5 stocks are being traded.

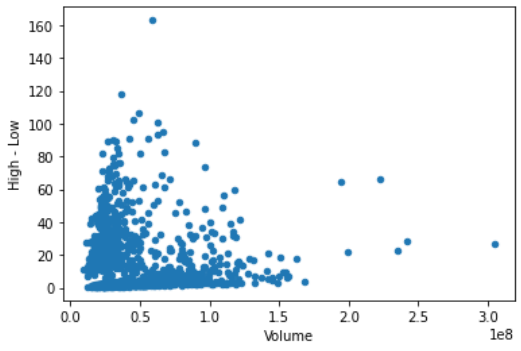
Data exploration with scatter plots and histograms:

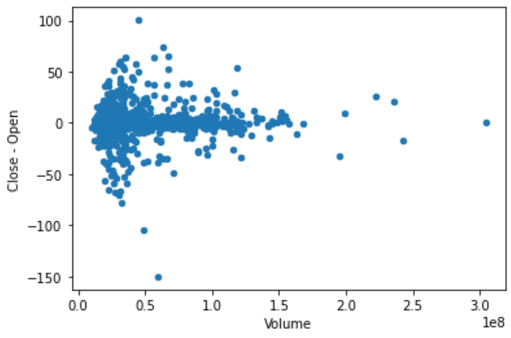
To help with the analysis and exploration of the data, we thought maybe it might be helpful to make two new columns for the range in the prices for the day (high - low) and the change in prices for the day (close - open). The plan for these two new columns was to see whether any additional useful information about the dataset could be gleaned from them.



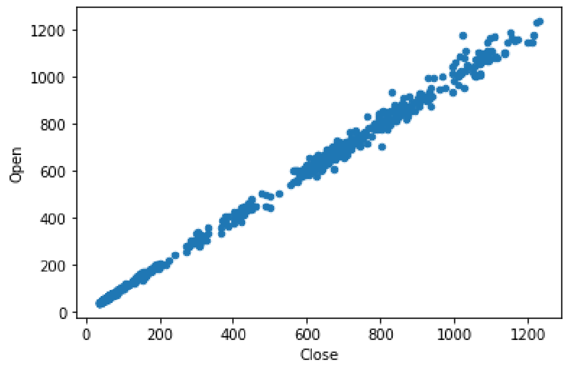


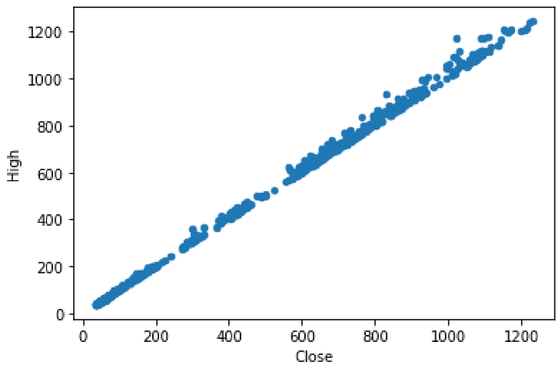
From the scatter plots above we can see that the range and volatility of stock prices for Tesla start drastically increasing from 2020 onwards.

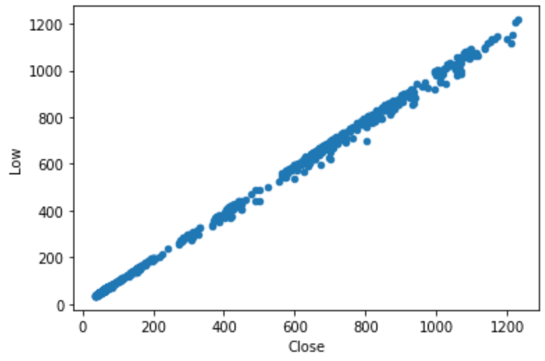




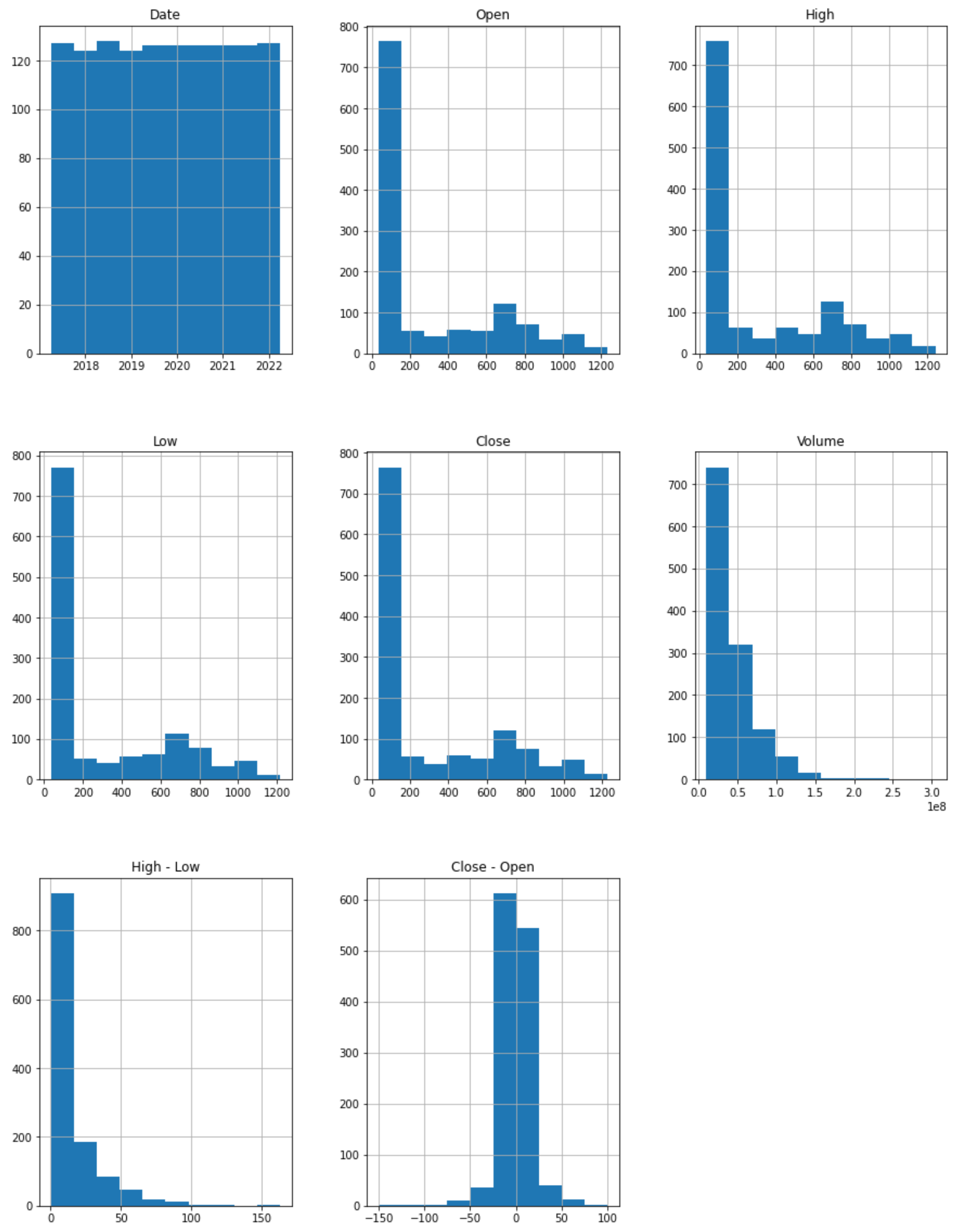
From the two scatter plots above we can see that higher amounts of Tesla stock are only traded when the range and volatility of the stock price is low.



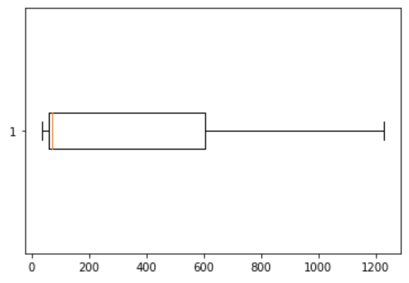




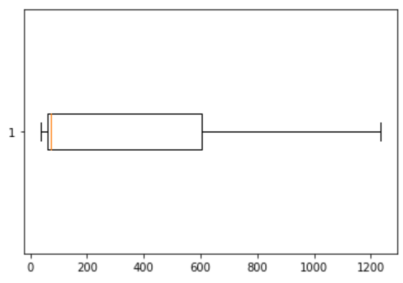
From the three plots above, we can see that open, high, and low are all basically linearly related to each other since, as expected, these four values will generally all be pretty close to each other on a given day.



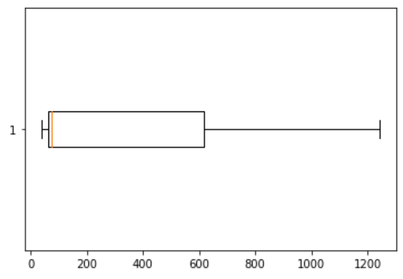
We can see from the histograms for the Tesla dataset that it has a very high distribution of smaller values. This corresponds to the sharp increase in Tesla stock closing prices from 2020 meaning that the majority of the 8 out of 10 years of Tesla stock data is for lower value, lower volatility stock prices. A point to consider from this is how effective these older eight years of data will be for the model to learn to predict Tesla stock prices considering how much different the most recent two years look.



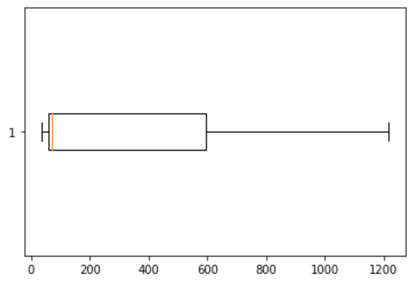
(Close)



(Low)



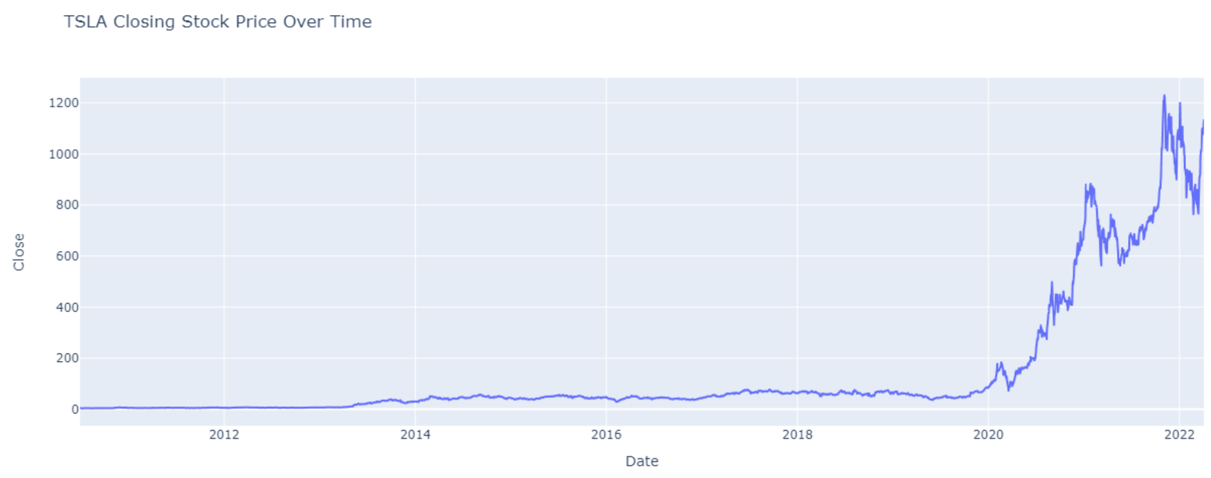
(Open)

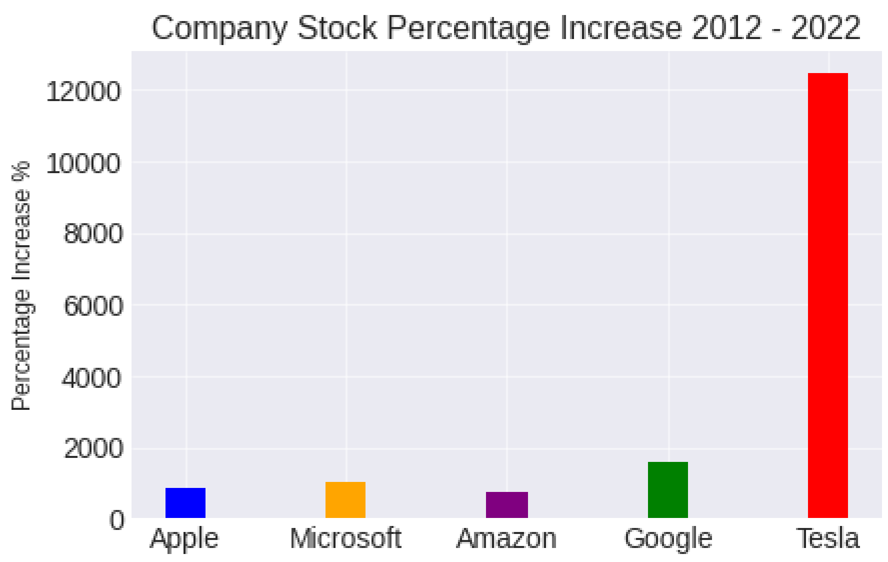


(High)

The boxplots for the Tesla stock shown above tells us that the vast majority of the Tesla data we have is where the stock price is very low as can be seen by how close the median line is to the left whisker of the boxplot. This corroborates with the previous point that a minority of the data points have a vastly higher value than the rest of the dataset which can be seen by the much larger right whisker of the boxplot. This also confirms the almost linear relationship between the Close, Open, Low and High columns shown by the scatterplots earlier as these four boxplots look almost identical.

<u>Visuals of stock performance over the last 10 years:</u>



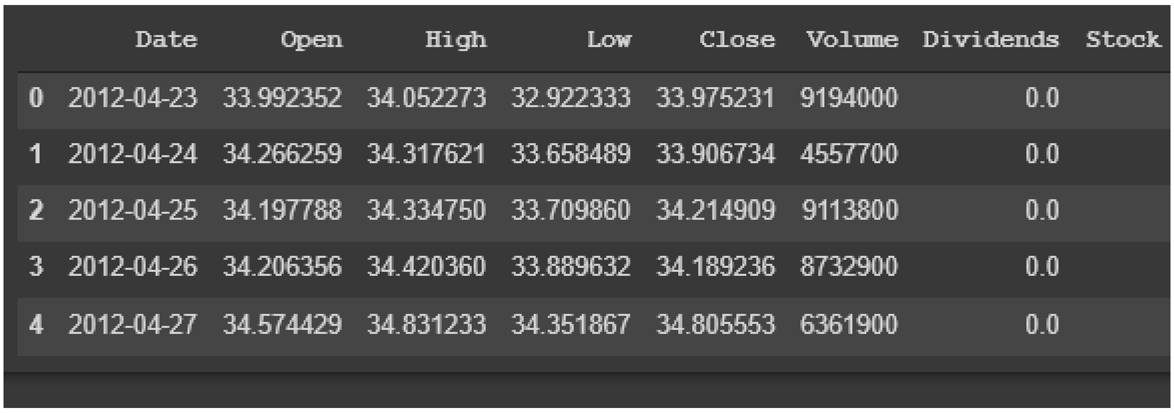


The bar chart above compares stock performance for the 5 largest stocks in the US. The comparison is with percentage increase because they each have very different absolute price values, especially considering Apple and Microsoft have been traded since the 80's. You can see that Tesla has massively outperformed the other 4 companies' stocks over the last 10 years.

Data exploration for Gold stock

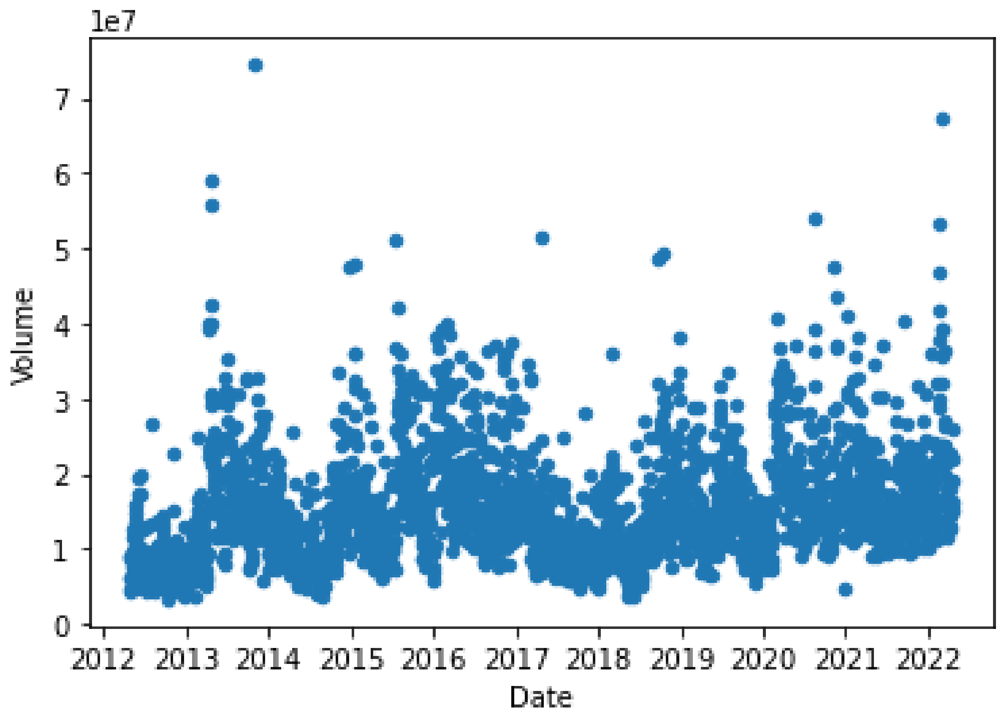
Unfortunately, the volatile nature of Tesla stock had made it impossible to predict with a good degree of accuracy. From this point on, we will shift focus to analyse GOLD stock price.

Gold stock trades at a much lower volume than the 5 companies discussed above:

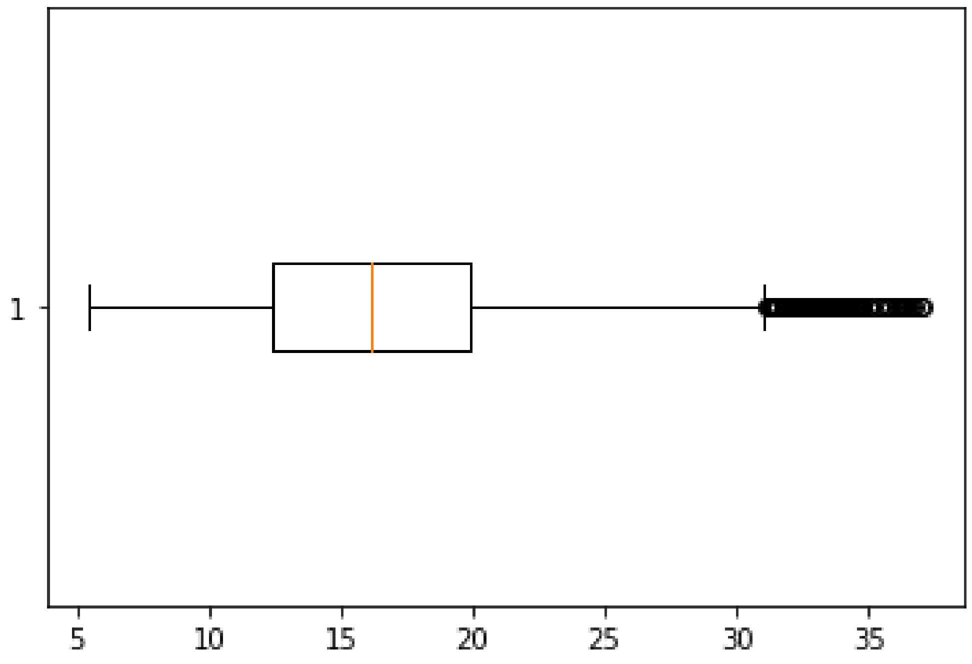


	Date	Open	High	Low	Close	Volume	Dividends	Stock
0	2012-04-23	33.992352	34.052273	32.922333	33.975231	9194000	0.0	
1	2012-04-24	34.266259	34.317621	33.658489	33.906734	4557700	0.0	
2	2012-04-25	34.197788	34.334750	33.709860	34.214909	9113800	0.0	
3	2012-04-26	34.206356	34.420360	33.889632	34.189236	8732900	0.0	
4	2012-04-27	34.574429	34.831233	34.351867	34.805553	6361900	0.0	

The volume of trades spikes and dips in the short term, but it has been steady over the years:



As a hedge against inflation, the volume of trades increased as a result of the pandemic. Its close value had a number of outliers on the high end:



However, despite this recent interest in the ticker, exponential behaviour in Gold Close is only seen over the short term, and its price has remained relatively steady (adjusting for inflation) over hundreds of years.

<u>Preprocessing + Literature review (Kundan, Zeye & Jian)</u>

DATA PREPROCESSING : Data pre-processing in Machine Learning refers to the technique of preparing (cleaning and organizing) the raw data to make it suitable for building and training Machine Learning models. In simple words, data preprocessing in Machine Learning is a data mining technique that transforms raw data into an understandable and readable format.

When it comes to creating a Machine Learning model, data preprocessing is the first step marking the initiation of the process. Typically, real-world data is incomplete, inconsistent, inaccurate (contains errors or outliers), and often lacks specific attribute values/trends. This is where data pre-processing enters the scenario – it helps to clean, format, and organize the raw data, thereby making it ready-to-go for Machine Learning models.

Introduction

As a kind of negotiable securities with no repayment period, stocks have become the first choice for many families' financial investments, due to their high returns. So many investors are highly interested in stock price prediction to get the maximum return. However, the forecast of the stock prices has been a difficult task for many of the researchers and analysts. To solve this situation, the model which can help investors and analysts forecast stock prices has been very popular. In this project, we have presented, modelled, and predicted the stock returns using LSTM. Several famous companies' stock prices data have been collected and then trained for the model. Finally, we got a broken line which represents the predicted prices and it is close to another broken line which represents the actual prices.

Literature review / Related work

For the prediction of the stock prices, there are 3 main methods which can (and often do) overlap: **fundamental analysis**, **technical analysis**, and **technological methods**.

Fundamental analysis, it's a **top-down analysis**, which first analyze the global economy, followed by country analysis and then sector analysis, and finally the company level analysis. The fundamental analysts assess a company's **past performance** and the **credibility of its accounts**. And they try to **find out the true value** of a stock and **compare it with** its value on the stock market to discover if its value is **undervalued** or not. This method is often used for long-term **strategies**.

Technical analysis only determines the stock prices by **using the trends of the past price**. It doesn't care about any aspect of the company. It uses many different patterns such as **candlestick patterns** and **head and shoulders patterns**. It is used more in short-term **strategies.**

Technological methods, which include using the Time Series to predict or using Text Mining together with Machine Learning algorithms …etc, are the most frequently used. This method attempts to accomplish predictions through technology such as artificial neural networks (ANNs) and Genetic Algorithms(GA). By using appropriate variables and suitable modelling we can predict the stock prices. In this regard, Sen and Datta Chaudhuri demonstrated a new approach to stock price prediction using the **decomposition of time series** [1–3]. Besides, Mehtab and Sen present a highly robust and reliable predictive framework for stock price prediction by combining the power of **text mining** and natural language processing in **machine learning** models like regression and classification [4]. By analyzing the sentiments in the social media and utilising the sentiment-related information in a non-linear multivariate regression model based on self-organising fuzzy neural networks (SOFNN), Mehtab and Sen have demonstrated a high level of accuracy in predicted values of NIFTY index values.

There are three camps broadly on this issue based on its methods and model: The **first camp** mainly consists of models that use bivariate or multivariate regression on cross-sectional data. Because of the simplicity and invalidity of the linearity assumptions, they cannot get the accurate results in many cases. The **second camp** makes use of the concepts of time series and other econometric techniques such as vector autoregression (VAR) and autoregressive integrated moving average (ARIMA).The **third camp** use the learning-based propositions using machine learning, deep learning, and natural language processing.

In this work, we made an attempt to address the problem by exploiting the machine learning and deep learning-based models, to build a very reliable and accurate framework for stock prices prediction. Apart from these, a **long-and-short-term memory (LSTM)** network-based deep learning model was used, and we studied its performance in predicting stock.

Refers:

1. Sen, J., Datta Chaudhuri, T.: An alternative framework for time series decomposition and forecasting and its relevance for portfolio choice - a comparative study of the Indian consumer durable and small cap sector. J. Econ. Libr. 3(2), 303–326 (2016)

2. Sen, J., Datta Chaudhuri, T.: An investigation of the structural characteristics of the Indian IT sector and the capital goods sector - an application of the R programming language in time series decomposition and forecasting. J. Insur. Finance. Manag. 1(4), 68–132 (2016)

3. Sen, J., Datta Chaudhuri, T.: Understanding the sectors of the Indian economy for portfolio choice. Int. J. Bus. Forecast. Mark. Intell. 4(2), 178–222 (2018)

4. Mehtab, S., Sen, J.: A robust predictive model for stock price prediction using deep learning and natural language processing. In: Proceedings of the 7th International Conference on Business Analytics and Intelligence, Bangalore, India, 5–7 December 2019 (2019)

Methodology

Let's explore various steps of data pre-processing in machine learning.

Step 1 : Identifying and handling the missing values : The problem of missing values is quite common in many real-life datasets. Missing value can bias the results of the machine learning models and/or reduce the accuracy of the model. Missing data is defined as the values or data that are not stored (or not present) for some variables in the given dataset.

Why Do We Need To Care About Handling Missing Value?

It is important to handle the missing values appropriately.

- Many machine learning algorithms fail if the dataset contains missing values. However, algorithms like K-nearest and Naive Bayes support data with missing values.
- You may end up building a biased machine learning model which will lead to incorrect results if the missing values are not handled properly.
- Missing data can lead to a lack of precision in the statistical analysis.

Figure Out How To Handle The Missing Data

Analyse each column with missing values carefully to understand the reasons behind the missing values as it is crucial to find out the strategy for handling the missing values.

There are 2 primary ways of handling missing values:

- Deleting the Missing values
- Imputing the Missing Values

Step 2 : Converting date field from string to Date format : Analysing datasets with dates and times is often very cumbersome. Months of different lengths, different distributions of weekdays and weekends, leap years, and the dreaded time zones are just a few things we may have to consider depending on our context. For this reason, Python has a data type specifically designed for dates and times called 'datetime'. Date time module provides classes to work with date and time. These classes provide a number of functions to deal with dates, times and time intervals. Date and datetime are objects in Python & when we manipulate them, we are actually manipulating objects and not string or timestamps.

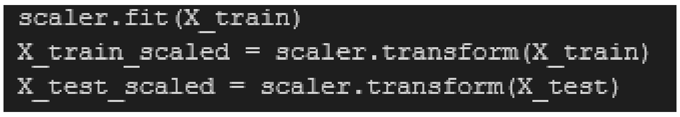
Step 3 : Feature Scaling : Feature scaling is about transforming the values of different numerical features to fall within a similar range like each other. The feature scaling is used to prevent the supervised learning models from getting biased toward a specific range of values. For example, if your model is based on linear regression and you do not scale features, then some features may have a higher impact than others which will affect the performance of predictions by giving undue advantage for some variables over others. This puts certain classes at a disadvantage in the training model. This is why it becomes important to use scaling algorithms so that you can standardise your feature values.

This process of feature scaling is done so that all features can share the same scale and hence avoid problems such as some of the following:

· Loss in accuracy.

· Increase in computational cost as data values vary widely over different orders of magnitude.

In this model we have used MinMaxScaler which is a type of scaler that scales the minimum and maximum values to be 0 and 1 respectively. This is appropriate for a stock with relatively reserved min/max values, such as gold. The scaler is fitted on the training data before being applied to both the train and scaled data, to avoid data skewness.



```
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 4 : Splitting the data : Data splitting is commonly used in machine learning to split data into a train, and test set. This approach allows us to find the model hyper-parameter and also estimate the generalisation performance. In this research, we conducted a comparative analysis of different data partitioning algorithms on both real and simulated data. Our main objective was to address the question of how the choice of data splitting algorithm can improve the estimation of the generalisation performance.

Each algorithm divided the data into two subset, training/test. The training set is used to develop models and feature sets; they are the substrate for estimating parameters, comparing models, and all of the other activities required to reach a final model. The test set is used only at the conclusion of these activities for estimating a final, unbiased assessment of the model’s performance. It is critical that the test set not be used prior to this point. Looking at the test sets results would bias the outcomes since the testing data will have become part of the model development process.

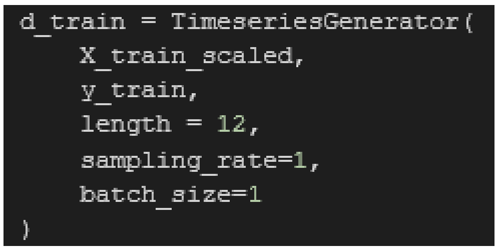
It is important to have a reasonable balance between the training/validation set sizes.

Now we have prepared our feature set & need to define X and Y. We'll now randomly split the data into a training and test dataset using the train_test_split() function. We'll assign 30% of the data to the test group, which will be held out of training, and we'll use 70% of data for training purposes.

Step 5 : Pre-Processing for LSTM : Although our LSTM model will be taking in a 3D array we will have to restructure our data in such a way so that it comes in 3D array format.

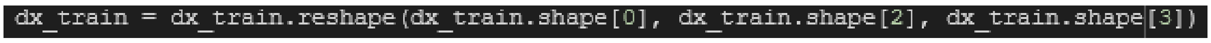
The process is nearly identical for the feature tensor preprocessing of the training and test set, so while we focus on the train set here, the process for the test set is analogous.

The training features are processed into a time series, where they are fed into the model in batches of 12 days.



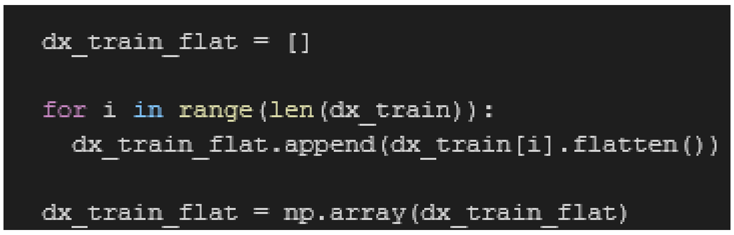
```
d_train = TimeseriesGenerator(
    X_train_scaled,
    y_train,
    length = 12,
    sampling_rate=1,
    batch_size=1
)
```

With 4 features for each day, this means each element of the feature tensor has 48 values. We then augment the tensor into the desired shape:



```
dx_train = dx_train.reshape(dx_train.shape[0], dx_train.shape[2], dx_train.shape[3])
```

Flatten it, so the 48 data point are in one single array, and convert it into a NumPy array, which is the desired format for input into an LSTM layer::

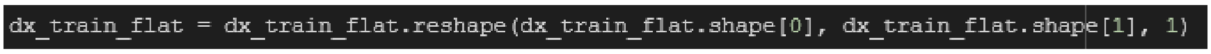


```
dx_train_flat = []

for i in range(len(dx_train)):
  dx_train_flat.append(dx_train[i].flatten())

dx_train_flat = np.array(dx_train_flat)
```

Finally, the NumPy array is reshaped once again to be the proper format for input:



```
dx_train_flat = dx_train_flat.reshape(dx_train_flat.shape[0], dx_train_flat.shape[1], 1)
```

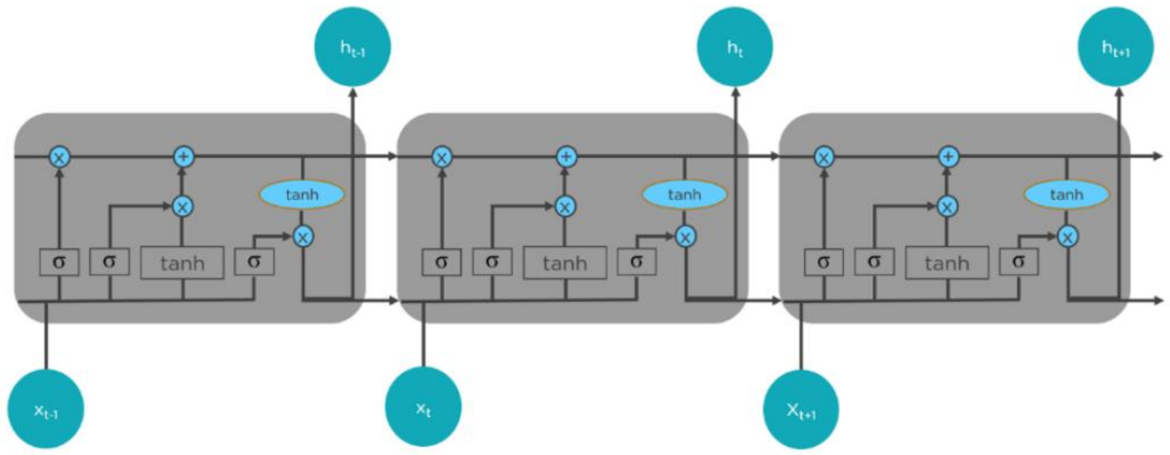
Implementation + Results (Kacper, Mostyn & Anthony)

LSTM models: LSTM algorithm is called Long short-term Memory, which is a special form of RNN. RNN (Recurrent neural network) is a general term for a series of neural networks capable of processing sequential data. RNN features that the connection between implicit elements is cyclic; If the input is a time series, you can expand it. Each of these units processes the output of the previous unit, in addition to the input data for the current point in time, and ultimately produces a single forecast.

Unlike standard feedforward neural networks, LSTM has feedback connections. It can process not only single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition and anomaly detection in network traffic.

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs.



From the image on the top, you can see LSTMs have a chain-like structure. General RNNs have a single neural network layer. LSTMs, on the other hand, have four interacting layers communicating extraordinarily.

LSTM models' properties: hidden, dense, dropout layers:

Hidden layer: In neural networks, a hidden layer is located between the input and output of the algorithm, in which the function applies weights to the inputs and directs them through an activation function as the output. In short, the hidden layers perform nonlinear transformations of the inputs entered into the network. Hidden layers vary depending on the function of the neural network, and similarly, the layers may vary depending on their associated weights.

Dense layer: The dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer. The dense layer is found to be the most commonly used layer in the models.In the background, the dense layer performs a matrix-vector multiplication. The values used in the matrix are actually parameters that can be trained and updated with the help of backpropagation.The output generated by the dense layer is an ‘m’ dimensional vector. Thus, a dense layer is basically used for changing the dimensions of the vector. Dense layers also apply operations like rotation, scaling, and translation on the vector.

Dropout layer: Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel. Overfitting is a serious problem in neural networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different thinned networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularisation methods.

Activation Function: An activation function is a function used in artificial neural networks which outputs a small value for small inputs, and a larger value if its inputs exceed a threshold. If the inputs are large enough, the activation function "fires", otherwise it does nothing. An activation function is like a gate that checks that an incoming value is greater than a critical number.

In this experiment, we used the relu function as the activation function. the function returns 0 if it receives a negative input, and if it receives a positive value, the function will return the same positive value. Since the stock price is greater than 0, it applies to this function.

Adam Optimization Algorithm: Adam is an alternative optimization algorithm that provides more efficient neural network weights by running repeated cycles of “adaptive moment estimation.” Adam extends on stochastic gradient descent to solve non-convex problems faster while using fewer resources than many other optimization programs. It’s most effective in extremely large data sets by keeping the gradients “tighter” over many learning iterations.

Adam combines the advantages of two other stochastic gradient techniques, Adaptive Gradients and Root Mean Square Propagation, to create a new learning approach to optimise a variety of neural networks.

Why use LSTM model for prediction

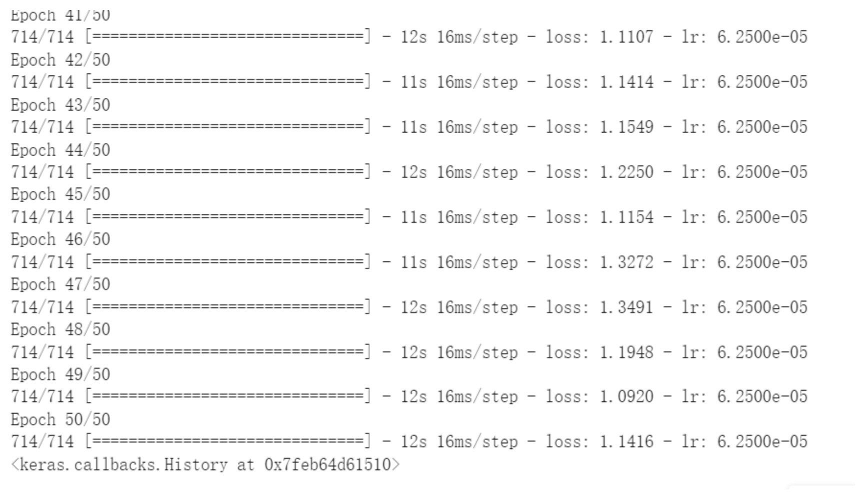
- Neural networks such as LSTM are good at dealing with multiple variables, which makes them helpful to solve the time series prediction problem.

- The LSTM rectifies a huge issue that recurrent neural networks suffer from: short-memory. Using a series of ‘gates,’ each with its own RNN, the LSTM manages to keep, forget or ignore data points based on a probabilistic model.
- LSTMs also help solve exploding and vanishing gradient problems.
- Demand data sets are featured with different seasonalities. The LSTM is capable of capturing the patterns of both long-term seasonalities such as a yearly pattern and short-term seasonalities such as weekly patterns.
- It is natural that events would impact demand on the day when it is happening as well as the days before and after the event is happening. The LSTM has the ability to triage the impact patterns from different categories of events
- The LSTM could take inputs with different lengths. This feature is especially useful when LSTM is used to build general forecasting models for specific customers or industries.
- The different gates inside LSTM boost its capability for capturing non-linear relationships for forecasting. Causal factors generally have a non-linear impact on demand. When these factors are used as part of the input variable, the LSTM can learn the nonlinear relationship for forecasting.

Model construction

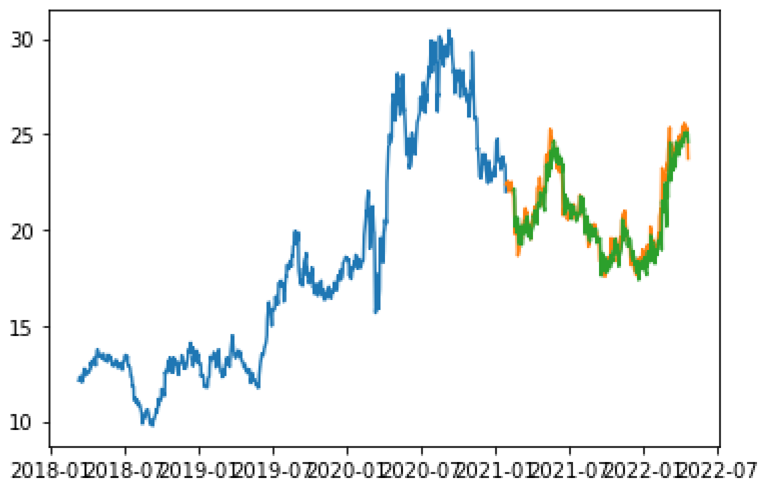
In this experiment, keras deep learning framework was used to build the model quickly. Established the Sequential model. The LSTM layer was added to it. The dropout layer is set to 0.3 (which means 30% of neurons will be randomly discarded during training), and the Dense layer is added to aggregate its dimension to 1. Relu was used as the activation function, MSE was used as the loss function(MAE is more robust to outliers because it does not square.), and Adam was used as the optimization algorithm. The model adopted 50 Epochs and each batch size was set to 1. Callbacks were set as "reduce_lr" (which means that if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.). There were 64 neurons in the hidden layer and one neuron in the output layer. The output variable is the feature of a time step.

After completing the training and testing, the loss values are as follows:

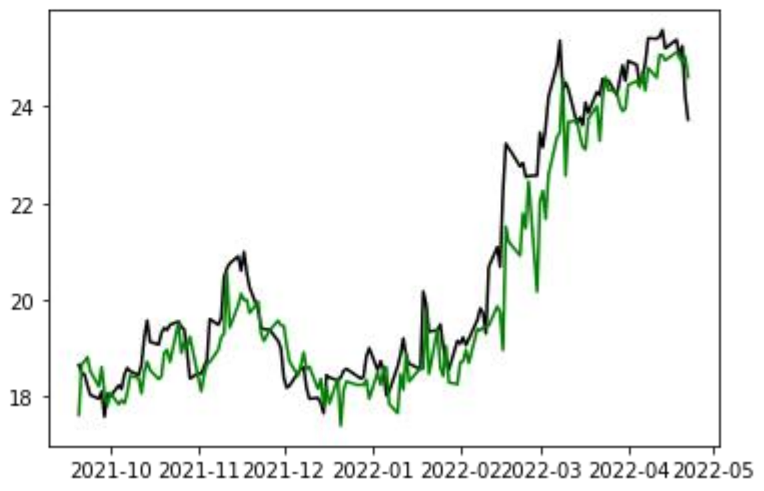


```
Epoch 41/50
714/714 [==============================] - 12s 16ms/step - loss: 1.1107 - lr: 6.2500e-05
Epoch 42/50
714/714 [==============================] - 11s 16ms/step - loss: 1.1414 - lr: 6.2500e-05
Epoch 43/50
714/714 [==============================] - 11s 16ms/step - loss: 1.1549 - lr: 6.2500e-05
Epoch 44/50
714/714 [==============================] - 12s 16ms/step - loss: 1.2250 - lr: 6.2500e-05
Epoch 45/50
714/714 [==============================] - 11s 16ms/step - loss: 1.1154 - lr: 6.2500e-05
Epoch 46/50
714/714 [==============================] - 11s 16ms/step - loss: 1.3272 - lr: 6.2500e-05
Epoch 47/50
714/714 [==============================] - 12s 16ms/step - loss: 1.3491 - lr: 6.2500e-05
Epoch 48/50
714/714 [==============================] - 12s 16ms/step - loss: 1.1948 - lr: 6.2500e-05
Epoch 49/50
714/714 [==============================] - 12s 16ms/step - loss: 1.0920 - lr: 6.2500e-05
Epoch 50/50
714/714 [==============================] - 12s 16ms/step - loss: 1.1416 - lr: 6.2500e-05
<keras.callbacks.History at 0x7feb64d61510>
```

It can be seen from the results that the final loss is about 1.14. The MSE was calculated to be 0.535 for a model with 50 epochs. The predictions for the test set are seen to match closely with the observed Close data:



This figure shows the training data (blue) as well as the test data and predictions (orange and green, respectively). Because this is daily data over 4 years, it is worth looking at a smaller section:



Here we can see the last 150 days of the training data and the predictions (in black and green, respectively).

Parameters such as the number of hidden layers, dense layers, dropout layers and neurons were all changed in the effort to reduce MSE as well as the number of epochs and batch size. The following hyperparameters were found to be close to optimal:

- Hidden Layers: 2
- Dense Layers: 2
- Dropout Layers: 3 at a weight of 0.2
- Neurons per layer = 60
- Epochs = 50
- Batch size = 1

Further systematic tuning was carried out in the form of a kerastuner Random Search tuner which uses random search to find optimal values for the hyper parameters however little improvement in MSE was found after using this method.

Alongside the visual evaluation of the graphs, the most important metric in evaluating the performance of the model was the MSE. We were able to reduce the mean-squared-error to 0.535 after 50 epochs which we felt was a good result taking into consideration the computing power required to train the model.

Could this model be used to successfully trade stock on the stock market to maximise ROI?

The stock market can be volatile and the changing of stock prices can appear random. A simple model such as this which only considers the daily Open, High, Low and Volume of a stock would not be sufficient in accurately predicting a stock's price. Past performance does not guarantee future results, and even with the help of technical analysis a neural network with numerical data alone cannot accurately predict a stock's price tomorrow, and much less so a stock's price next week or next month. The stock market is driven heavily by company success and its decisions as well as the consumers outlook on the company’s future. These factors cannot be taken into account without the use of NLP. In conclusion, this model is not to be relied on for monetary gain within the stock market.