10 April 2019

AMAZON STOCK PRICE PREDICTION

National University of Singapore

Prepared For:

NUS Fintech Society

Machine Learning Dept

Done By:

Samuel Tan Jun Jie

Bjorn Jee

Kwok Kuin Ek Jeremy

Soh Rui Min

Table of Contents

1.0 Introduction

The aim of this paper serves to examine, mainly, 2 machine learning models; ARIMA (AutoRegressive Integrated Moving Average) and RNN (Recurrent Neural Networks) models as a basis to forecast seasonal Time Series in the S&P 500 Index, i.e. leveraging on Python libraries to provide forecasting for time series data. Amongst the 500 large companies as listed in the document, we will be analysing only Amazon's (Ticker: AMZN) stock prices to forecast the prices in the upcoming period.

The S&P 500 Index is an American stock market index based on the market capitalizations of 500 large companies having common stock listed on the NYSE, NASDAQ, or the Cboe BZX Exchange. However, where the S&P 500 Index exists over a continuous time interval with equal spacing between every two consecutive measurements, it is a time series, and cannot be simply analysed with traditional linear regression methods. As such, utilising the ARIMA and RNN models, we can then make use of the average price of stocks, which is in consideration of the lowest and highest prices, to ultimately return a prediction on the movement of the stock prices for Amazon using the aforementioned models.

By following the steps in time series modelling, i.e. visualising the time series, stationarising the series, plotting ACF/PACF charts whilst finding optimal parameters, and then building the ARIMA/RNN models, we can return predictions on the stock prices, and further evaluate forecast accuracy using error metrics. Upon which, an evaluation of both ARIMA and RNN models will be performed.

2.0 Data Analysis

Upon analysing the .csv file "all_stocks_5yr", it is evident in the dataframe that the dates, open price, close price, high price, low price, volume and company name are available. The open price is defined as price at which a security first trades upon the opening of an exchange on a trading day, while the closing price is the final price at which a security is traded on a given trading day. The 'high price' and 'low price' are the highest and lowest price traded for the stock in a day, respectively. Lastly, volume is defined as the number of shares traded during a given day. Generally as examined from the data, it is rather clean. However, a few steps in data cleaning will be performed as shown in the next segment.

2.1 Data Cleaning

In this task, we imported the necessary pandas library, numpy library, seaborn library, matplotlib, and datetime module as required, as to be used for future steps. By using df.head(5) we returned the top 5 rows of our data series for general analysis. Upon then, missing values was examined for using dataset.isnull().sum as per normal data preprocessing. Where the rows for 'open', 'high', 'low' had missing values, it must be handled. Although deleting a row is a possible option, we did not choose to do so as it potentially leads to a loss of information which will not give the expected results whilst predicting output. As such, to negate the loss of data, we replaced missing values in features 'high', 'low', 'open' with the means of the respective features, adding variance to the dataset. Where thereupon, we examined for missing values again.

Thereafter, for our upcoming ARIMA/RNN analysis, we created a new column for the average of stock prices, i.e. (price of high+price of low)/2, to obtain the central value i.e. average of stock price, which we believe is representative of the data set, as our target variable. Where the closing price is the last trading price of the previous day and the opening price the first trading price of the day, transactions are happening and shifting prices even after hours. The option of utilising the closing and opening price of stocks was not our choice as such, where we believe using the average price of the stock as obtained from the highest and lowest stock price is a better gauge.

Using pandas' query method, we chose to analyse Amazon's stock prices, 'AMZN' only. Where the 'date' in our dataframe is still in string form, we converted it to datetime format for easier analysis in our upcoming time series chart, facilitating future steps for data visualisation.

Where the data as provided in our .csv file is rather clean e.g. no categorical data, no lack in certain behaviour or trends, there is no necessity to do any further data preprocessing. As such, the prepared data is now available for application to the ARIMA and RNN algorithms.

2.2 ARIMA Analysis

ARIMA modelling is used quite frequently in the industry when it comes to analyzing time series based data. There are three parameters in the model, namely p, d and q, which represents the autoregressive factor, differentiation factor, and the moving average factor respectively. Hence, the generic form of an ARIMA model can be expressed as ARIMA(p,d,q), which is a clear expression of stating the various factors present in the three parameters. In order to kickstart the analytical process in finding out which particular ARIMA model is suitable in being utilized to fit Amazon's time series data, the first step would be to perform a Stationarity Test to check if the data is Stationary. This is a vital first procedure as in order to perform ARIMA modelling, or in general, any model as a matter of fact, the data has to be Stationary in nature (i.e. getting rid of any Seasonality trends) before any modelling analysis could be implemented.

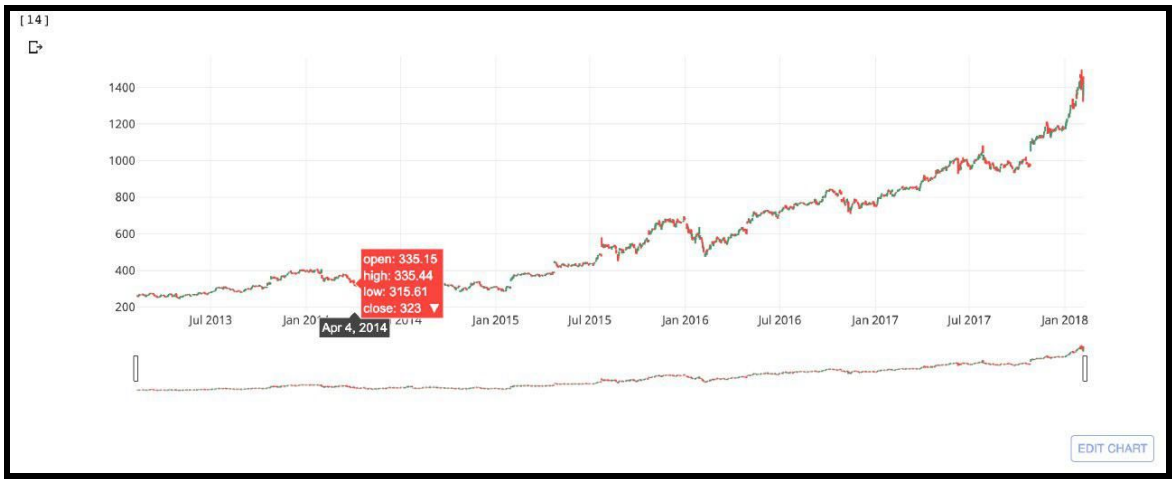

Figure 1: Time Series Chart of Amazon's Average Share Prices via Candlesticks

We first perform an introductory visualization of Amazon's Average Share Prices by churning out a time series chart via candlesticks to gain an initial sensing of its stationarity through simple eye-balling. From Figure 1, it is evident that there is a general upward trend in the prices, illustrating that there might not be a presence of Stationarity.

```
Augmented Dickey-Fuller Test Result
ADF Test Statistic          2.769535
P-Value                     1.000000
#Lags Used                  3.000000
#Observations Used       1255.000000
Critical Value (1%)        -3.435571
Critical Value (5%)        -2.863846
Critical Value (10%)       -2.567998
dtype: float64
isStationary: (False)
```

Figure 2: Augmented Dickey-Fuller Test Result for Amazon's Average Share Prices

We then proceed to do an in-depth analysis by using a statistical test called Augmented-Dicker Fuller (ADF) Test to prove the presence of Stationarity. This test is a specialized one which is designed for such a purpose. Looking at Figure 2, it can be seen that the time series was indeed not Stationary given the high ADF Test Statistic of 2.77 (5% critical value at -2.86 and 1% critical value at -3.44) and a high P-value of 1.00, thus giving us a false result for Stationarity as shown.

Therefore, in order to make the data Stationary, we would have to perform First Order Differencing to the Average Share Prices to see if that would make it Stationary. Differencing, which is also a key parameter in the ARIMA model, is essentially the action of taking the data at each reference point and subtract it with its corresponding lag values. To illustrate, a First Order Differencing is done by taking all data found at time T and subtracting it by the data found at T-1. This is simple to be understood. However, a Differencing with Second Order and higher is

not as simple as taking all the data found at time T and subtracting it by the data found at T-N, where N is the number of orders. For Second Order Differencing, it is in fact taking the data churned out by First Order Differencing and subtracting it from the output which is the difference of data at T-1 and the data at T-2. Following this logic and intuition, we can then understand the derivations for the higher orders of differencing. For this step, Orders of Differencing would be applied sequentially at an ascending order to check for Stationarity at each order. Once the data has been rendered Stationary, the differencing process would stop and the corresponding Order of Differencing which caused the data to be first Stationary would be taken as the order that is required before any model could be applied.

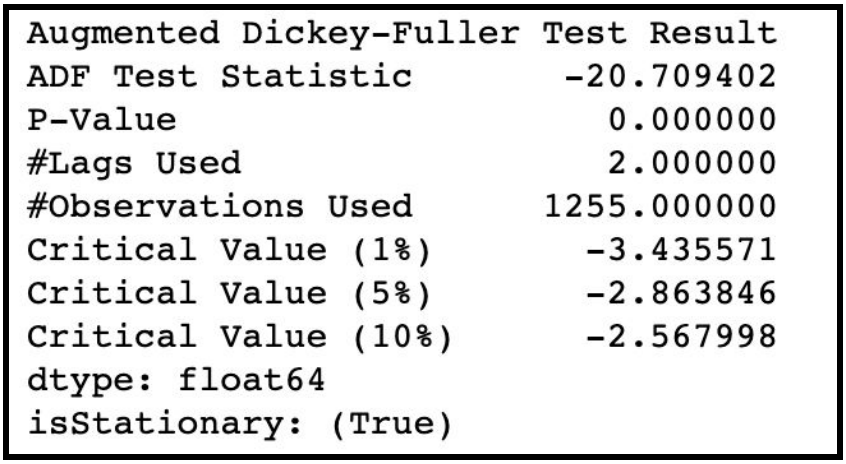

Figure 3: Augmented Dickey-Fuller Test after First Order of Differencing

Hence, in our case, we applied the First Order of Differencing first to check for its Stationarity. In addition, we also proceeded to drop any resulting null values which arose from this differencing so that the ADF test could be performed. The end result which we have gotten, as shown in Figure 3, is a True result for Stationarity, as supported by its very negative ADF Test Statistic of -20.7 (5% critical value at -2.86 and 1% critical value at -3.44), and a low P-value of 0.00. Therefore, we can stop our differencing process here, and take the First Order of Differencing to be inputted in our ARIMA model, with d taking a value of 1.

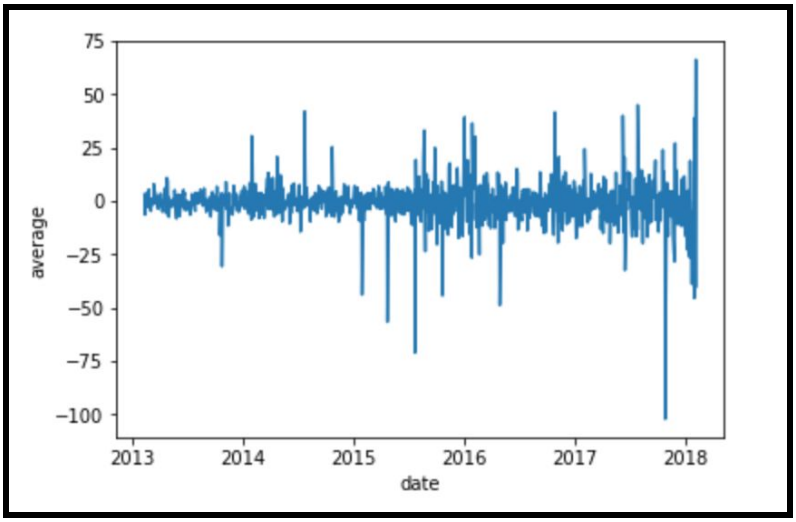

Figure 4: Time Series Chart for Dataset after First Order of Differencing

To provide a visualization for this, we have also plotted a time series chart (as shown in Figure 4) for this differenced data to show how the data is seen to be fluctuating around the value of 0, eradicating any seasonality trends present.

The next step would involve several feature selection methods which we undertook in order to choose the appropriate corresponding orders for p and q in our ARIMA(p,1,q) model. Three types of methods have been chosen for this analysis for comparison.

Firstly, the commonly used method would be to investigate the plots of both the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF). This two plots are strong indicators in telling us what p and q should be. To provide some background knowledge, an ACF plot showcases the coefficient of correlation between a data and a lagged data at the respective lag values, whereas a PACF plot showcases the partial correlation between a data and the lagged data that is not explained by the correlations at all lower-order-lags.

If ACF plot is decaying exponentially while PACF is not decaying but cuts off sharply (i.e. sudden drop in significance as we go down the lagged values in ascending order) at some x lagged value, it would mean that the data follows an AR(x) process, translating it to be an

ARIMA(x,1,0) model. However, if the PACF plot is decaying exponentially while the ACF plot is not decaying but cuts off sharply at some y lagged value, it would mean that the data follows an MA(y) process, translating it to be an ARIMA (0,1,y) model. If, however, both ACF and PACF plots are decaying exponentially, it would mean that the factors of p and q would be at least one for each, translating to be an ARIMA(1,1,1) model. This is not the end unfortunately as the new ACF and PACF graphs would have to be plotted to see if higher orders of AR or MA should be added in using the same processes as explained earlier.

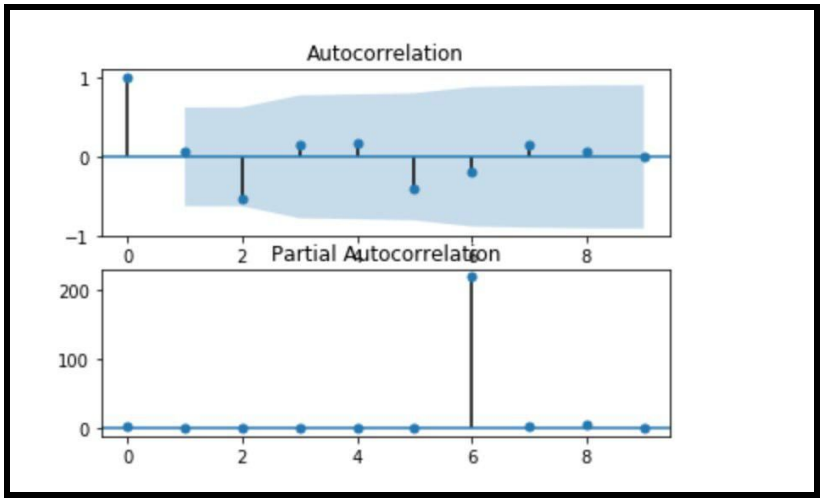

Figure 5: ACF and PACF Plots for Dataset after First Order of Differencing

Based on the plots churned out as illustrated in Figure 5 (ACF plot is at the top, PACF plot is at the bottom), it can be deduced that there are no presence of decaying processes in both ACF and PACF plots. The blue field as evident in the ACF plot denotes the significance level, and it can be seen that there is no sharp cutoff in any of its lag values (ignore the value at lag 0 as it is redundant). Furthermore, the values in PACF seem to fluctuate very closely to 0 as well with no sudden cutoff. Therefore, these evidences seem to point that an ARIMA(0,1,0) model is the appropriate model to be used. This particular model corresponds to a simple random walk model, in which the next price at time T+1 is very largely dependent on the price at Time T, plus an error term.

```
import warnings
from sklearn.metrics import mean_squared_error
from math import sqrt
from statsmodels.tsa.arima_model import ARIMA
# evaluate combinations of p, d and q values for an ARIMA model
def evaluate_models(dataset, p_values, d_values, q_values):
  dataset = dataset.astype('float32')
  best_score, best_cfg = np.inf, None
  n = len(dataset)
  for p in p_values:
    for d in d_values:
      for q in q_values:
        if p==0 and d==0 and q==0:
          continue
        order = (p,d,q)
        try:
          model = ARIMA(dataset, order)
          model_fit = model.fit(disp=0)
          y_pred = model_fit.predict(n-50,n-1)
          y_true = dataset[n-50:]
          rmse = sqrt(mean_squared_error(y_true,y_pred))
          if rmse < best_score:
            best_score, best_cfg = rmse, order
            print('ARIMA%s RMSE=%.3f' % (order,rmse))
        except:
          continue
  print('Best ARIMA%s RMSE=%.3f' % (best_cfg, best_score))

# evaluate parameters
p_values = range(0, 5)
d_values = range(1, 3)
q_values = range(0, 5)
warnings.filterwarnings("ignore")
evaluate_models(df_amzn['average'].values, p_values, d_values, q_values)
```

```
ARIMA(0, 1, 0) RMSE=19.837
Best ARIMA(0, 1, 0) RMSE=19.837
```

Figure 6: Creation of Algorithm to Choose Best ARIMA Model Based On Lowest RMSE

The second method which we performed to choose an appropriate p and q would be to create and run a specific algorithm (as shown in Figure 6) that can return us with the best combination of p and q parameters with the lowest Root Mean Square Error (RMSE). With the help of several packages to devise this algorithm, namely warnings, sklearn, math and statsmodels, the best model returned to us was also ARIMA(0,1,0) with RMSE of 19.837, where we tested p from a factor of 0 to 5, d from a factor of 1 to 3, and q from a factor of 0 to 5.

The third method performed to choose appropriate p and q parameters would be via the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC). Both criterions are forms of penalized-likelihood criteria. In terms of definition, AIC would be an estimate of a constant with the addition of the relative distance between the unknown true likelihood function of the data and the fitted likelihood function of the model, whereas BIC is an estimate of a function of the posterior probability of a model being true, under the pre-specified Bayesian setup. In both criterions, lower values would suggest that the fitted model would be closer to the truth, indicating that this particular fitted model would be the most justified model to be utilized in the analysis. Given that using the first two methods, ARIMA(0,1,0) seems to be the most plausible ARIMA model, we would investigate the AIC and BIC criterions for p and q parameters where it is small, i.e. considering only the situation where it is 1. In addition, another rationale for only considering p and q to be only 1 is due to the nature of the initial PACF plot. Upon close examination of PACF, it could be seen that for the first few lags, some PACF values are seen to be slightly below zero. Thus, it would also seem to suggest that there might be a need to add in an additional factor for the MA term, spurring us to also consider the MA(1) process. Therefore in this analysis, we considered 4 scenarios, where ARIMA is (0,1,0), (1,1,0), (0,1,1) and (1,1,1).

ARIMA Model Results

Dep. Variable:	D.average	No. Observations:	1257
Model:	ARIMA(0, 1, 0)	Log Likelihood	-4946.347
Method:	css	S.D. of innovations	12.380
Date:	Wed, 10 Apr 2019	AIC	9896.694
Time:	13:39:23	BIC	9906.967
Sample:	1	HQIC	9900.555

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0343	0.349	-0.098	0.922	-0.719	0.650

ARIMA Model Results

Dep. Variable:	D.average	No. Observations:	1257
Model:	ARIMA(0, 1, 1)	Log Likelihood	-4591.406
Method:	css-mle	S.D. of innovations	9.308
Date:	Wed, 10 Apr 2019	AIC	9188.811
Time:	13:39:23	BIC	9204.221
Sample:	1	HQIC	9194.602

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0021	0.001	-2.876	0.004	-0.003	-0.001
ma.L1.D.average	-1.0000	0.003	-336.408	0.000	-1.006	-0.994

Roots

	Real	Imaginary	Modulus	Frequency
MA.1	1.0000	+0.0000j	1.0000	0.0000

ARIMA Model Results

Dep. Variable:	D.average	No. Observations:	1257
Model:	ARIMA(1, 1, 0)	Log Likelihood	-4812.694
Method:	css-mle	S.D. of innovations	11.131
Date:	Wed, 10 Apr 2019	AIC	9631.387
Time:	13:39:23	BIC	9646.797
Sample:	1	HQIC	9637.179

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0262	0.218	-0.120	0.905	-0.454	0.402
ar.L1.D.average	-0.4387	0.025	-17.263	0.000	-0.488	-0.389

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-2.2796	+0.0000j	2.2796	0.5000

ARIMA Model Results

Dep. Variable:	D.average	No. Observations:	1257
Model:	ARIMA(1, 1, 1)	Log Likelihood	-4583.702
Method:	css-mle	S.D. of innovations	9.252
Date:	Wed, 10 Apr 2019	AIC	9175.404
Time:	13:39:23	BIC	9195.949
Sample:	1	HQIC	9183.125

	coef	std err	z	P>|z|	[0.025	0.975]
const	-0.0021	0.001	-2.597	0.010	-0.004	-0.001
ar.L1.D.average	0.1111	0.028	3.937	0.000	0.056	0.166
ma.L1.D.average	-1.0000	0.003	-374.179	0.000	-1.005	-0.995

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	8.9991	+0.0000j	8.9991	0.0000
MA.1	1.0000	+0.0000j	1.0000	0.0000

Figure 7: ARIMA Model Summary Statistics for ARIMA(0,1,0), ARIMA(0,1,1), ARIMA(1,1,0) and ARIMA(1,1,1)

Based on the output generated as shown in Figure 7, we could see that ARIMA (1,1,1) performed the best as compared to the rest under AIC and BIC, having the lowest value of 9175 and 9196 respectively.

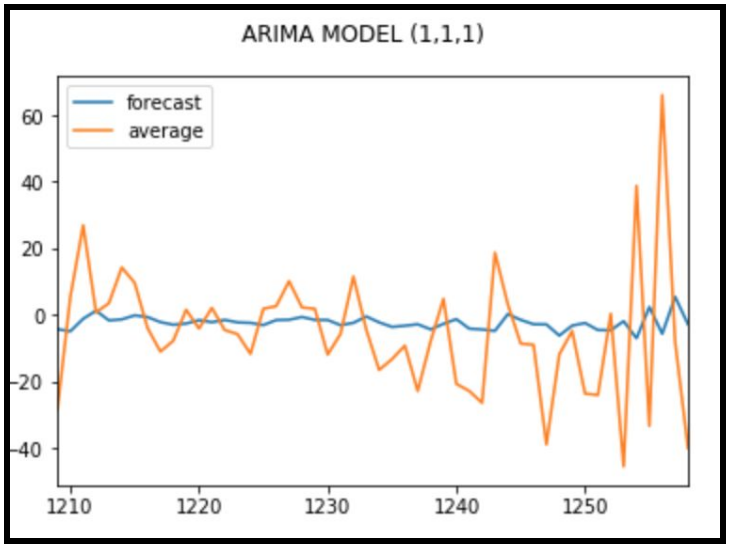

Figure 8: Time Series Chart for Forecasted Amazon's Average Share Price and Actual Amazon's Average Share Price after First Order Differencing

However, upon plotting the time series chart for ARIMA(1,1,1) as our fitted model together with the actual values, as shown in Figure 8, we can see that there is indeed a drastic difference between the two plots. ARIMA(1,1,1) showcases a model which is relatively flat, fluctuating slightly around the 0 value, whereas the actual values indicate a much bigger fluctuation around 0 instead. Hence, given this significant difference, we have decided that this might not be an appropriate model to be utilized since a high amount of information has been eradicated from using ARIMA(1,1,1).

Thus, from these 3 methods performed, ARIMA(0,1,0) would be our preferred model choice for ARIMA given its high performance in ACF/PACF plots and the RMSE test.

2.3 RNN Analysis

A Recurrent Neural Network (RNN) model is a type of artificial neural network where connections between nodes form a temporal sequence. RNN has a fundamental error of information loss, and often suffers from the limitation of disappearing gradient while adjusting the weights through backpropagation. To overcome this limitation, we chose to use a Long Short Term Memory(LSTM) model for this analysis. The difference between an LSTM model from the rudimentary RNN model is the usage of 'gates' to regulate the flow of information.These gates are fundamentally mathematical functions which decides which information to keep and throw, by the importance of the node with respect to the prediction of the target value.

Neural networks are classifier models which works well on discrete solutions. To use a RNN for a continuous time series data, we have to format the data in a way which it reduces our problem to a classification problem, and it was done in a few steps.

Firstly, we performed various feature selection methods to choose the best features to be used for training the model. We used Pearson's correlation matrix and did univariate plots for each feature and identified that the volume of the Amazon stock is highly correlated to the average price of the stock.

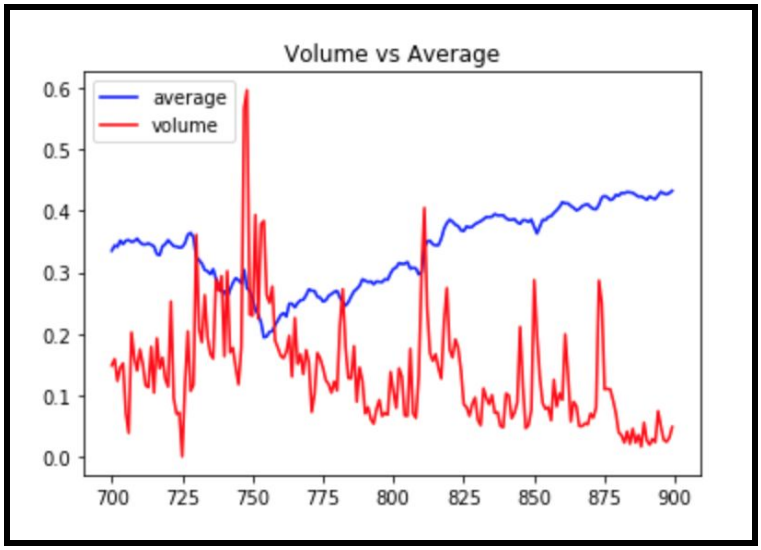

Figure 9: Graph of Normalized Values of Volume and Average Share Prices for Amazon

With reference to Figure 9, By plotting normalized volume and average price on an overlapped graph, it is observed that at every spike in volume, the average price of the stock significantly changes. After these tests, we decided to use the past data of volume and average price as features for training.

```
[ ]  lookback = 50
     X=[]
     y=[]
     for i in range(len(dataset['open'])-lookback-1):
         t=input_data[i:i+lookback,0:2]
         X.append(t)
         y.append(input_data[i+ lookback,1])
```

Creation of the train and test datasets. We need to reshape the train data into LSTM expected structure of (samples,timestamp,features)

```
[ ]  train_size=int(.7 * len(X))
     X, y= np.array(X), np.array(y)
     X_train = X[:train_size]
     Y_train = y[:train_size]
     X_test = X[train_size+lookback+1:]
     Y_test = y[train_size+lookback+1:]
     X_train = X_train.reshape(X_train.shape[0],lookback, 2)
     X_test = X_test.reshape(X_test.shape[0],lookback, 2)
     print(X_train.shape)
     print(X_test.shape)

     (845, 50, 2)
     (312, 50, 2)
```

Figure 10: Generation of the New Input Data and Target Values

Next, we needed to format the dataset in a way which it reduces to classification problem, yet retaining the information as a time-series data. We did that by using a sliding window with a length of 50 days with values of (volume, average price) as our X input. For each window, the value of the average price for the first day outside of the window will be appended into the target_input (Y) array. The code is illustrated in Figure 10. After which, we generated a train-test split with a 7:3 ratio from the newly formatted dataset.

The LSTM model was trained on the train sets with parameters epoch=200 and batch-size = 32, with 18,631 total parameters and performed decently with a r2_score of 0.872484 and mean_squared_error score of 0.002138.

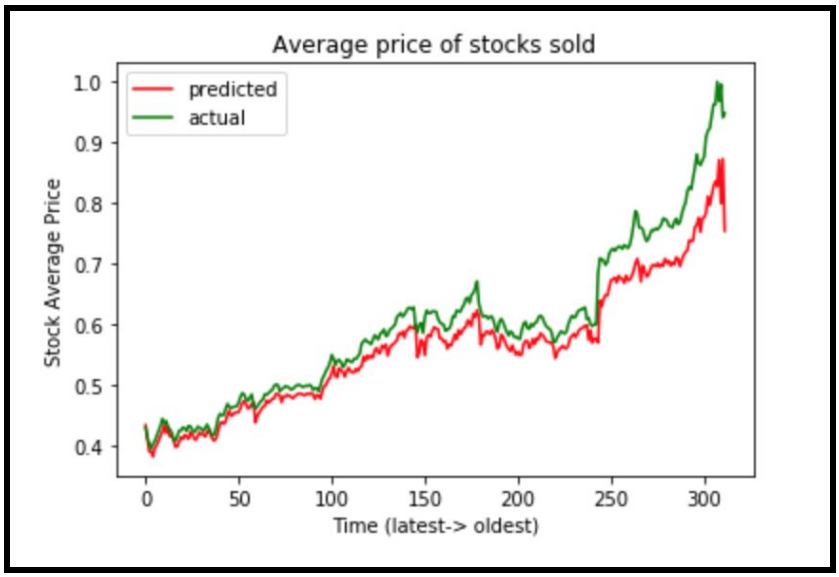

Figure 11: Graph of Predicted Amazon's Average Share Prices and True Average Prices from the Test Set

From Figure 11, we observe that the predicted value is close to the actual value of the Amazon's average price at a time period closer to the trained dataset and differs more significantly at later time periods. This could be a result of a few factors - the size of our dataset (1259 rows), the size of our sliding window (50), and the number of epochs (200). These hyperparameters can be optimized through trial and error. Also, we note that the difference in the predicted price and

actual price could be the result of the usage of MinMaxScaler to normalize the data, which bounds our predicted price to be between the Minimum and Maximum values of our data set, as the transformation by MinMaxScaler is given by :

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min.
```

Figure 12: Plot of Amazon's Average Share Prices Against Time

With reference to figure 12, it is evident that the average price of the Amazon stock has an increasing trend, and possible that values data in the future will exceed the current Maximum value and hence MinMaxScaler may not be the best choice to normalize the data in this context.

3.0 Evaluation of Models

To determine which model is more suitable for analysing the time series data, we decided to use mean squared error (MSE) and root mean square error (RMSE). MSE, is calculated as the average of the squared forecast error values. Squaring the forecast error values forces them to be positive; it also has the effect of putting more weight on large errors. Given large errors in predictions for the stock market can be costly, it is important for more attention to be given to these errors. RMSE, is the root of the MSE, in the units of the price.

Metric/Model	**ARIMA**	**RNN**
MSE	393.5	0.002138
RMSE	19.837	0.04623

The result above shows that the RNN model has the best prediction efficiency and accuracy with a RMSE value of 0.04623. This means it only differs from the actual price by 4 cents on average.

Using the RNN model, the last date provided in the dataset is 7 Feb 2018. The central price of 'AMZN' that day is 1438.07. The model was inaccurate with a predicted central price of 1268.38. However, the predicted trend was similar to that of the actual data.

Hence, in predicting the S&P 500 data, using a RNN model is better compared to an ARIMA model. RNN model will have a better prediction of stock price, for its memory of time. This is shown as while the predicted trend modelled the actual trend, the RNN model was only close to actual values for time periods near the training set and was less accurate further away. This could also be due to overfitting of the data which results in greater instability in predicting the data. However, both ARIMA model and RNN model are more suitable for short-term predictions, so only one-step forward prediction was done.