COMPARISON ANALYSIS BETWEEN TIME SERIES FORECASTING MODELS FOR APPLE STOCK CLOSING PRICE PREDICTION

HO WENG TIM

ABSTRACT

Since the stock market is in a world of uncertainty, corporate finance depends heavily on projecting stock values. Recognizing patterns and using prediction technologies to assist them are crucial for stock traders when making decisions. Data science combines a number of techniques, strategies, and deep learning ideas with the intention of identifying underlying patterns in large amounts of data. Researchers can gain various information, including data on a company's economic condition by using data science and prediction techniques. Time series analysis is useful for prediction and usually employed in banking to predict financial indicators like stock prices. In this project, 3 kinds of time series forecasting models which are long-short-term memory (LSTM), bidirectional long-short-term memory (Bi-LSTM) and gated recurrent unit (GRU) models are built to predict Apple stock closing price. Different hyperparameters are tuned manually, which included the number of layers, number of neurons, number of epochs, dropout rate and types of optimizers. The tuned models are compared to filtered out the best model for stock closing price prediction. The result showed that LSTM model after tuning is the best model that can be used for Apple stock closing price prediction.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 General Introduction

The term "stock market" refers to a wide range of sites where investors purchase shares of publicly traded companies. These monetary dealings actually occur in over-the-counter (OTC) markets that follow a specified system of regulations as well as on recognized platforms. (Chen, 2022). When a firm becomes public, companies collaborate with financial institutions to determine the fundamental price level. The price is determined by institutional investors' desire and value (Price, 2022). The stock price refers to the amount that a stock's share is presently trading for on the marketplace. When stocks of a publicly traded company are created, their value is given a rate that reflects the company's value as a whole. The value of a stock may change in reaction to a range of factors, including changes in the general economy, changes in particular industries, political events, outright war, and ecological repercussions. (CFI, 2019).

The stock price reflects how investors think about a company's ability to make and create value for customers. If investors are happy and the company is operating well, as indicated by the stock price, directors are much more certain to remain in their current positions and receive bonuses. A high share price often deters potential takeovers as well (Chris, 2022). The goal of stock price forecasting is to foresee anticipated fluctuations in share price on a financial exchange. If share price swings can be properly forecast, shareholders will be able to earn more money. It is one of the toughest challenges to tackle due to the numerous elements involved in stock market forecasting, such as the rate of interest, administration, and job growth, which maintain the stock market uncertain and extremely hard to foresee properly (Alzazah & Cheng, 2021).

Since the stock market is in a world of uncertainty, corporate finance depends heavily on projecting stock values. Recognizing patterns and using prediction technologies to assist them are crucial for stock traders when making decisions. Data science combines a number of techniques, strategies, and deep learning ideas with the intention of identifying underlying

patterns in large amounts of data. Researchers can gain various information, including data on a company's economic condition by using data science and prediction techniques. Time series analysis is useful for prediction and usually employed in banking to predict financial indicators like stock prices, commodities, and assets (Budiharto, 2021). In artificial intelligence (AI) and machine learning, deep learning models represent a new paradigm of learning.

Deep learning is a subset of machine learning (ML), which processes data according to a predetermined logical framework in an effort to uncover patterns and correlations. Deep learning, also known as deep neural networks, employs a variety of hidden layers in the neural network as compared to conventional neural networks, which have a limited number of hidden layers. Different with the conventional machine learning approaches, deep learning is able to detrimental effects of overtraining (Biswas, n.d.). Due to its ability to accommodate massive datasets and data mapping with precise forecasting, its usage in stock market forecasting is receiving interest (Chandola, 2022).

Recurrent neural networks are artificial neural networks that use sequential data or time series information (RNN). They differ from other systems because of their "memory," which enables them to modify current input and outcomes by drawing on information from earlier input. Unlike normal deep neural networks, which assume that inputs and outcomes are unconnected to one another, recurrent neural networks' outputs are dependent on the earlier portions of the sequence (IBM, 2020). Every element of information is retained by an RNN over time. Only memorizing past inputs makes it helpful for time series forecasting (GeeksforGeeks, 2018). The main advantage of RNN over the conventional artificial neural network is that it can represent data sequences because each data can be presumed to rely on earlier samples (Gudikandula, 2019). Recurrent neural networks (RNNs) are proficient at processing sequential input, but due to an issue known as vanishing and exploding gradients, they struggle to capture the long-term relationships in the records. Long-and-Short-Term Memory (LSTM) networks, a type of RNN that successfully solves the issue, have been shown to be extremely efficient and precise at processing sequential input (Sen & Mehtab, 2022).

In economics and finance, predicting and stock price research are crucial. Any set of factors that fluctuates can be forecasted using a time series approach. Time series forecasting is frequently used to monitor a security 's cost changes with time for stock or stock prices (Hoare, n.d). Hence, building a model that can accurately predict the future stock price is

considered crucial to monitor the stock movement and understanding the company's potential. In this study, Apple's stock price from the year of 1980 to 2022 is utilized for building time series forecasting model. The models that were built in this research included Long-Short Term Memory (LSTM), bidirectional Long-Short Term Memory (biLSTM) and Gated Recurrent Unit (GRU) with different architecture. In the following session, the study will present the literature review in section II, methodology in section III, implementation and result in section IV, discussion and conclusion in section V.

1.2 Problem Statement

Although there is many research in stock price prediction, but there is still a lack of research that study the comparison analysis on the deep learning time series forecasting model which are the LSTM, bidirectional LSTM and GRU in different architecture. Moreover, most of the time series forecasting models in stock price prediction are machine learning model instead of deep learning. Even using deep learning model in time series forecasting, the studies are lacking investigation on a different type of architectures such as different epochs, number of layers, optimizer, dropout rate and number of neurons. Moreover, there is also a lack of studies that focus on Apple stock price prediction by comparing the long short-term memory model, bidirectional long short-term memory model and gated recurrent unit model with different architectures. Hence, there is a lack of knowledge to understand the best time series forecasting approach that can provide high performance across stock price prediction.

1.3 Research Question

1. What are the effective deep learning approaches to forecast Apple stock closing price?
2. How can the models be optimized to be the best fit for accurately predicting the Apple stock closing price?
3. How the best time series forecasting model for Apple stock price prediction be determined?

1.4 Aim and Objective

The main aim of this research is to develop an effective deep learning approach for time series forecasting in Apple stock price prediction.

For the purpose of this study, the following objectives are addressed:

1. To identify suitable deep learning approaches for Apple stock price prediction through literature review.
2. To optimize the deep learning approaches for Apple stock price prediction through hyperparameter tuning.
3. To evaluate the model's performance and compare with peer models for Apple stock price prediction through different evaluation matrix.

1.5 Research Scope

The dataset of this study is obtained from Kaggle. This is a data collection for forecasting the stock price of Apple Inc. Each observation brings information about the closing, opening, highest price, lowest price, adjusted closing and volume of trading for each day. The data was taken from Yahoo Finance. This work is limited to Apple stock price prediction by using long-short term memory, bidirectional long short-term memory and gated recurrent unit models in different architectures. The history stock price that was used are total 42 years which is from 1980 until 2022. The time period that was used in this study is 30 months. The architecture different include number of layer, number of neuron, number of epoch, dropout rate and type of optimizers. The evaluation matrix that was used are validation loss, root-mean-square error (RMSE), and R^2 score. Last but not least, the programming language and coding platform used in this project is Python and Jupyter Notebook.

1.6 Significant of Research

Since the stock market is a process full of unpredictability, anticipating stock prices is crucial in both finance and business. Recognizing patterns and having technologies to enable forecasts are crucial for stock brokers when making decisions. Investors will be able to earn more if share price movements can be predicted accurately. Traders can understand about the future value of business stocks as well as other capital instruments traded on the market by employing stock price forecasting powered by deep learning. Gaining substantial gains is the whole point of making stock price predictions. Moreover, carrying out this research, it enables the reader to understand the impact of different hyperparameters on different time series forecasting models. Moreover, the comparison analysis also improve the knowledge to understand the best time series forecasting approach that can provide high performance across stock price prediction.

CHAPTER 2

LITERATURE REVIEW

2.1 Related Work

The study of Roondiwala et al. (2017) showed simulated and forecasted NIFTY 50 stock market performance calculated using LSTM. They gathered 1312 sequences with 5 years period of NIFTY 50 past data from the National stock exchange. From this data, 1180 samples were utilized for the model's learning while 132 samples were used for testing. The based LSTM model that was used in this study are a sequential input layer, 2 LSTM layer, dense layer with ReLU activation and dense output layer with linear activation function. The output value produced by the LSTM's output nodes is verified to the target outcome. This study used Root Mean Square Error (RMSE) as evaluation matrix because RMSE is a great all-purpose error metric for numerical forecasts. The author used different epochs for training which are 250 epochs and 500 epochs. The result showed that the best results are obtained using 500 epochs, with 0.00983 training RMSE and 0.00859 testing RMSE.

Besides, the study of Ghosh et al. (2019) offers a tool for assessing and projecting stock price to understand firm's growth prospects utilizing the LSTM (Long Short-Term Memory) model and using this model as its net development computation method. Their framework's analysis of the ideal time period to forecast the share price of a firm in a specific industry is the main objective of this study. Their goal is to estimate the upcoming pricing and project future performance across various time horizons. Then, for each company in a separate industry, they examine the prediction error. Date and close price are used as the feature for training and validation. The based LSTM model that was used in this study is a sequential input layer, 3 LSTM layers, and a dense output layer with a linear activation function. From the results, the error level dramatically decreases for practically all sectors. Hence, researchers advise using this LSTM-based model to forecast stock prices based on extensive past data. The author suggested there is a need for more historical data for model training to obtain higher accuracy in stock price prediction.

In the study by Sunny et al. (2020), a framework for predicting stock prices is put out by the authors using two well-known models: the Long Short-Term Memory (LSTM) model and the Bi-Directional Long Short-Term Memory (BI-LSTM) model. The author used public available data from Yahoo finance with 4170 days of sequence data for stock price prediction. It is evident from the experimental findings that with the right hyper-parameter adjustment, their suggested scheme may accurately predict the upcoming direction of the stock market using LSTM and BI-LSTM. To develop a more accurate model that can be employed to predict upcoming stock values, the RMSE for both the LSTM and BI-LSTM models was assessed by adjusting the number of epochs, hidden layers, dense layers, and different units used in hidden layers. The authors pointed out that the effectiveness of test precision and training rises as the number of dense layers grows and the number of neurons in the hidden layers decreases. This research shows that the 100 epochs with 2 hidden layers including 64 units, and 1 dense layer training model has a smallest RMSE error than other models, giving the highest predictive performance. Moreover, it also proved that BI-LSTM has a better result than LSTM.

Next, in the study of Pramod et al. (2020), the TATAMOTORS share data is used by the authors' suggested LSTM model to perform a time series analysis and forecast the stock price throughout the specified time period. The time period that was used in this study included 1500 days and 300 days. In this study, a sequential input layer, 1 LSTM layer, 1 drop out layer, 1 output layer is built for the model training with 50 epoch size. Moreover, adam optimization is also implemented in this model. The model is using 60 timestamps and 1 output. The result of this study concluded that their predictive model is able to predict the stock price with a minimum rate of loss and error. The authors suggested that the forecast may be more accurate if the algorithm trains a larger amount of data sets utilizing more powerful computational resources, increasing layers, and LSTM modules.

Moreover, another study by Pawar et al. (2019) presents the use of LSTM and RNN for stock price forecasting in portfolio management in Yahoo Finance. Several LSTM architectures using LSTM and a combination of LSTM with RNN are tested such as LSTM, DLSTM, LSTMP, and DLSTMP by using hyperbolic tangent activation function in the hidden layer and rectified Linear Unit function in the output layer with 5 epochs and 10 epochs. 128, 256 and 512 layers are tested for the model building. The result pointed out that LSTM has the best result with a minimum mean square error. The findings demonstrate that when compared to conventional machine learning algorithms, the RNN-LSTM model is more likely to produce

reliable data. Upcoming research should take into account many market characteristics and elements to improve prediction accuracy.

In order to forecast stock price trends, the study by Wei, (2019) suggests an attention-based LSTM model to predict three representative stocks in China which are Shenzhen index, Shanghai and Shenzhen 300 index and shanghai composite index. The input layer, hidden layer, attention layer, and output layer are the four components of the model. In order to comply with the model's input specifications, the input layer helps clean the inputs. Through the LSTM unit, the hidden layer is linked to the line model network. The feature vector was weighted by the attention layer. The computed findings are sent to the output nodes. 12 months and 18 months of sequence data are trained by using this model. The author pointed out that the amount of the input data will affect the algorithm's impact in addition to the window of time of the data input. This is also consistent with deep neural networks' benefits over shallow neural networks. The deep neural network's model is more suited to locating the global minimum value since it has a larger number of input data sets, more model iterations, more input data used for training, more model parameters and a bigger number of hidden layers. Based on the visualization plotting, it was found that a longer time span gives a better result.

In addition, the study of Sethia & Raut (2018) is proposed suggest the best model for stock price forecasting, develop a daily exchange plan, and assess how well the LSTM and GRU models performs in comparison to SVM and ANN models. The Standard and Poor's 500 Index data has been used to develop the models for 12 years. In this study, the authors used RMSE, R2 score, return ratio, optimism ratio and pessimism ratio as evaluation matrix. A five-layered architecture has been employed, consisting of two recurrent layers with 64 and 128 units each, 2 dense layers with 256 and 512 units, and a layer with one unit as the output. All units have a dropout rate of 0.3 and employ the "linear" activation function. The "he normal" initialization function has initialized the dense layers. Root mean square error was utilized as the loss function, and the system was developed over 125 epochs using the "Adam" optimizer. The authors concluded that the LSTM and GRU models perform better than the ANN and SVM models. They are superior trend predictors because the method they produce has a high rate of return. Although the GRU model has a lower degree of prediction confidence than LSTM's, it has a greater return ratio, showing that it anticipates patterns better and has a more resilient approach.

In the study of Joo et al. (2021), the authors used the Google stock price to build LSTM and GRU neural network with 100 epoch, adam optimizer and 0.01 learning rate. They pointed out that LSTM with limited memory should be used inside an RNN. In this model building, the input layer consists of inputs, and 2 hidden layers consisting of LSTM cells and 1 output layer. Additionally, in order to address the problem of the propensity of the RNN to gain knowledge only according to the last sequence, the authors suggested the bidirectional LSTM stock price prediction model, in which the hidden layer is added in the opposite way of the flow of data. The mean square root error between the actual stock price and the forecasted stock price was calculated in order to assess the accuracy of the stock price prediction. As a consequence, bidirectional LSTM has better forecasting reliability than unidirectional LSTM.

Furthermore, in the study of Althelaya et al. (2018), the author research, assess, and contrast stacked and bidirectional LSTM frameworks for short- and long-term stock price forecasting. Additionally, they contrast the models' functionality with that of shallow neural networks and basic LSTM variants. The tests are carried out using stock market data for the S&P500 from Yahoo Finance for the time frame of the year 2010 to 2017. The models are trained with different neural network, which included 4 neural networks, 8 neural networks, 16 neural networks and 32 neural networks. The result was also divided into normalized and without normalized by using RMSE, MAE and R2 as evaluation matrix. The networks are developed over a number of runs with epochs ranging from 1 to 10 and depending on randomized beginnings for each epoch. The outcomes also demonstrated deep learning's advantage over shallow neural networks. Generally, for both short- and long-term predictions, Bidirectional lstm networks showed greater efficiency and performance than other models.

In the study of Dutta et al. (2020), the author research and contrast various time series forecasting methods in Bitocoin price prediction by using LSTM, simple neural network and GRU. A 15 days, 30 days, 45 days and 60 days sequence data are used to build the model with adam optimization and results are evaluated by the root mean squared error (RMSE). A number of hyperparameter are tuned in this model such as batch-size, drop-out ratio, number of input nodes, activation function, number of hidden layers, epochs and learning rate. Two dense layers with hidden nodes 25 and 1 were employed for the simple NN. One LSTM layer (50 nodes) and one dense layer (1 node) was used to simulate the LSTM layer. One GRU layer (50 nodes) and one dense layer with 1 node made up the structure of the simple GRU and the GRU with

recurrent dropout. Two GRU layers (50 nodes and 10 nodes) with a dropout and recurrent dropout of 0.1 made up the final GRU design. The neural network and RNN models' optimum batch sizes are found to be 125 and 100, respectively. Higher validation and training losses during the process of learning were correlated with larger batch sizes. According to the test finding, the result showed that the gated recurring unit (GRU) model with recurrent dropout outperforms popular current models. The authors also demonstrate the potential for basic trading techniques to generate profits when used in conjunction with their suggested GRU model and appropriate learning.

Based on the above literature review, most of the studies are using LSTM, Bidirectional-LSTM and GRU for stock price prediction in a different field. Some of the studies pointed out the importance of tuning the hyperparameter such as activation function, epoch, number of hidden layers, units of the neural network, choice of optimization, and regularization. The most commonly used optimization method is adam optimization while the most common regularization method is the dropout method. The summary of the above literature review is summarized in the table under section 2.2. After the literature review, 3 models are decided to build in this project to carry out comparison analysis which are LSTM, bidirectional-LSTM and GRU since there are fewer studies comparing these models together. Moreover, most of the study are using relu activation function in the hidden layer and linear activation function in the output layer. Moreover, there are different optimizers are used in this study which included RMSprop, SDG and adam. Hence, these parameters are used in this ongoing research. Furthermore, the time series forecasting models have carried out hyperparameter tunings such as different epochs, number of layers, dropout rates, number of neurons and optimizers.

2.2 Summary of Literature Review

Author	Dataset period	Field	Model	Neural network layer	Activation function	Epoch	Optimizer	Regularization	Batch normalization	Further tuning	Evaluation Matrix	Best result
Roondiwala et al. (2017)	5 years historical stock data	NIFTY 50 stock	LSTM	A sequential input layer, 2 LSTM layer, 1 output layer	Dense layer: ReLU Output layer: Linear	250 500	RMSprop	No	Yes	Epoch	RMSE	500 epochs
Ghosh et al. (2019)	3-month, 6-month, 1 year and 3 years	Different company in Indian stock market	LSTM	A sequential input layer, 3 LSTM layer, 1 output layer	Output layer: Linear	No mention	No	No	No	No	Error value	No comparison of model is done
Sunny et al. (2020)	4170 days	Stock data from Yahoo finance	LSTM BI-LSTM	I. 2 hidden layers for both	Hidden layer: RELU	10 20 50 100	No mentioned	Yes, by using Drop out	No	Epoch Hidden layer	RMSE	BI-LSTM better than LSTM

				LSTM and Bi-LSTM II. 4 hidden layers for both LSTM and Bi-LSTM III. BI-LSTM with 2 hidden layer with 128 unit and 1 dense layer 1 unit		250						100 epochs with 2 hidden layers including 64 units, and 1 dense layer

				IV. BI-LSTM RMSE with 2 hidden layer with 64 unit and 2 dense layers with 16 unit and 1 unit								
Promod et al. (2020)	1500 days 300 days	TATAMOTORS share	LSTM	A sequential input layer, 1 LSTM layer,1 drop out layer, 1 output layer	No mentioned	50	Yes, by using adam	Yes, by using Drop out	No	No	RMSE	1500 days has better result

Pawar et al. (2019)	300 days	Stock price forecasting in portfolio management in Yahoo Finance	RNN and LSTM	I. input-LSTM-output II. input-LSTM-LSTM-output III. input-LSTM-recurrent-output IV. input-LSTM-recurrent-LSTM-recurrent-output	Hidden layer: Hyperbolic tangent Output layer: Rectified Linear Unit function	5 10	Yes, by using adam	No	No	Epoch Hidden layer Cell units	RMSE	RNN-LSTM
Wei, (2019)	12 months	Three representative stocks in	Attention-LSTM	Input layer-LSTM	No mentioned	No mention	Yes, by using SGD	No mention	No	No	Train-test plot	Model with longer time

	18 months 24 months	China's stock market (Shanghai composite index, shenzhen index, the Shanghai and shenzhen 300 index)		layer-attention layer-output layer								span data (24 months)
Sethia & Raut (2018)	12 years	Standard's and Poor's 500 Index	LSTM GRU ANN SVM	2 recurrent layers with 64 and 128 units each, 2 dense layers with 256 and 512 units, and a layer with one	Linear activation function	125	Yes, by using adam	Yes, by using drop out	No	No	R^2 score	LSTM and GRU

				unit as the output								
Joo et al. (2021)	1000 days	Google stock price	LSTM BI-LSTM	inputs, and 2 hidden layers consisting of LSTM cells and 1 output layer	No	100	Yes, by using adam	No	No	No	RMSE	Bi-LSTM is better than conventional LSTM
Althelaya et al. (2018)	10 days for short term prediction 30 days for long term prediction	Historical data of the Standard & Poor 500 Index (S&P500)	LSTM BI-LSTM Stacked LSTM	Input layer – 2 hidden layer – 1 output layer	LSTM: hard sigmoid transfer function Dense layer: Linear activation function	Range from 1 to 10	No	No	No	Epochs And neural network	RMSE MAE R^2	BI-LSTM is the best model for short and long term prediction

					Output: Tanh transfer function							
Dutta et al. (2020)	15 days 30 days 45 days 60 days	Bitocon price prediction	Simple neural network LSTM GRU	Simple NN :input-hidden layer with nodes 25-hidden layer with nodes 1-output layer LSTM : input-LSTM layer (50 nodes)-dense	Hidden and dense layer: sigmoid function Output layer: tanh function	No mentioned	Yes, by using adam optimization	Yes, by using drop out rate of 0.1	No	learning rate alpha, number of iterations, number of hidden layers, choice of activation function, number of input nodes, drop-out	RMSE	GRU with drop out regularization and hyperbolic tangent (TanH) activation function is the best

				layer(1 nodes)-output layer GRU: input – GRU layer(50 nodes) – dense layer(1 nodes)-output layer						ratio, and batch-size		

CHAPTER 3

METHODOLOGY

3.1 General Flow Chart

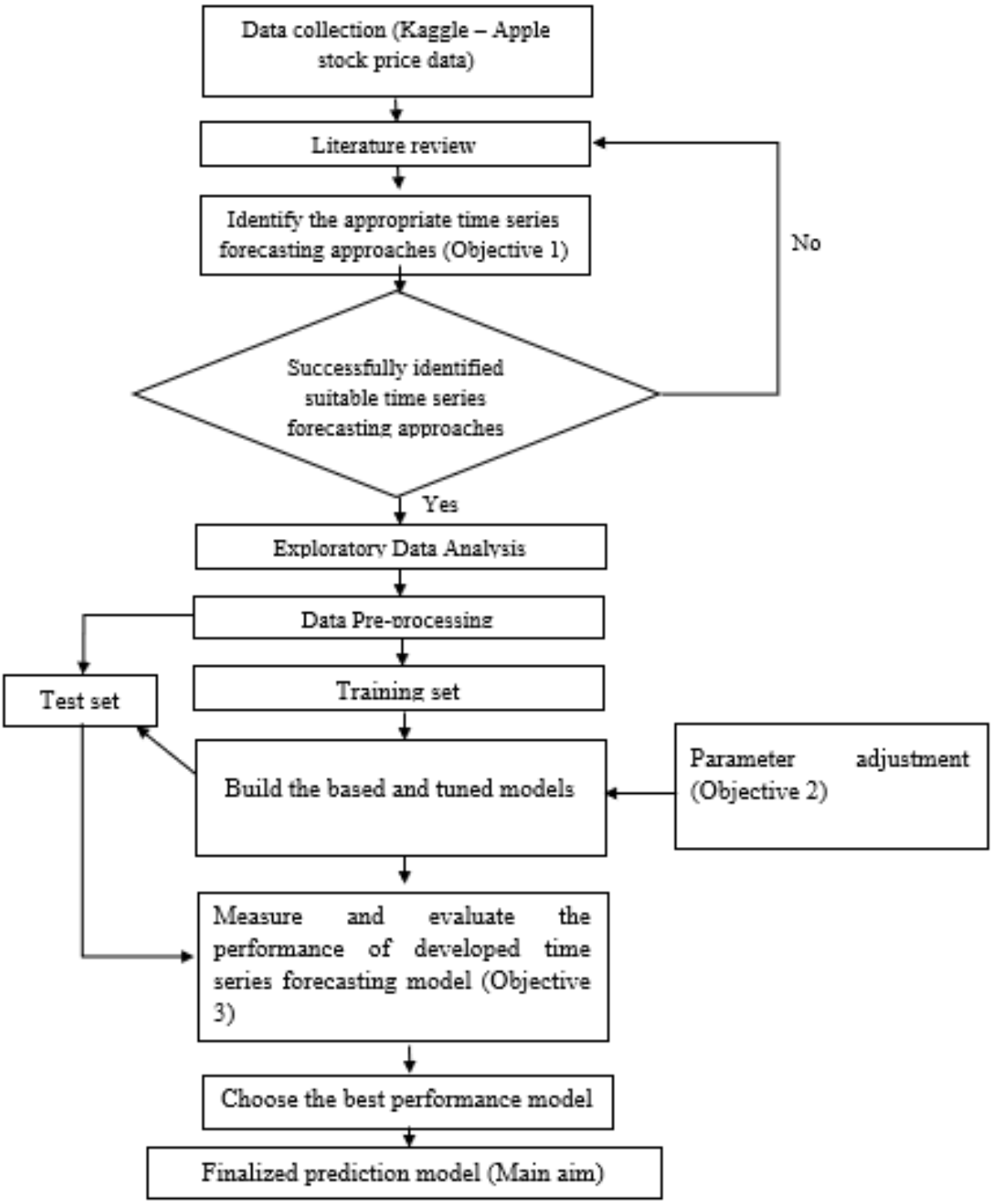


Figure 1: General flow chart

3.2 Data Collection

The data is an Apple stock price history data that collected from Kaggle. The dataset consists around 42 years history which is from the year of 1980 to 2022. There is total 7 variables and 10468 observations in this dataset. The variables included “Date”, “Open”, “High”, “Low”, “Close”, “Adj Close”, and “Volume”. Most of the variables are numerical variables. The target variable in this dataset is the “Close” variable which indicates the closing price of the day in Apple stock. In the below table, it showed the metadata for the dataset.

Variable	Description
Date	The date of the Apple stock trading
Open	The open price of the Apple stock
High	The highest price of Apple stock at the day
Low	The lowest price of Apple stock at the day
Close	The closing price of the Apple stock
Adj Close	The adjusted closing price of the apple stock
Volume	The volume of trading

3.3 Literature Review

A literature review is a thorough review of earlier studies on a subject. The literature review examines academic publications, journals, and other resources that are pertinent to a specific field of study. This prior work should be listed, described, summed up, impartially evaluated, and clarified in the review. It needs to provide a conceptual framework for the study and assist readers in defining its scope. By acknowledging the contributions of earlier researchers, the literature review reassures the reader that the work has been thoughtfully developed. When a past study on the subject is mentioned, it is believed that the researcher has studied, assessed, and incorporated that research into the current research. Based on the literature review, the types of time series forecasting models, activation functions and optimizers are filtered out.

3.4 Exploratory Data Analysis

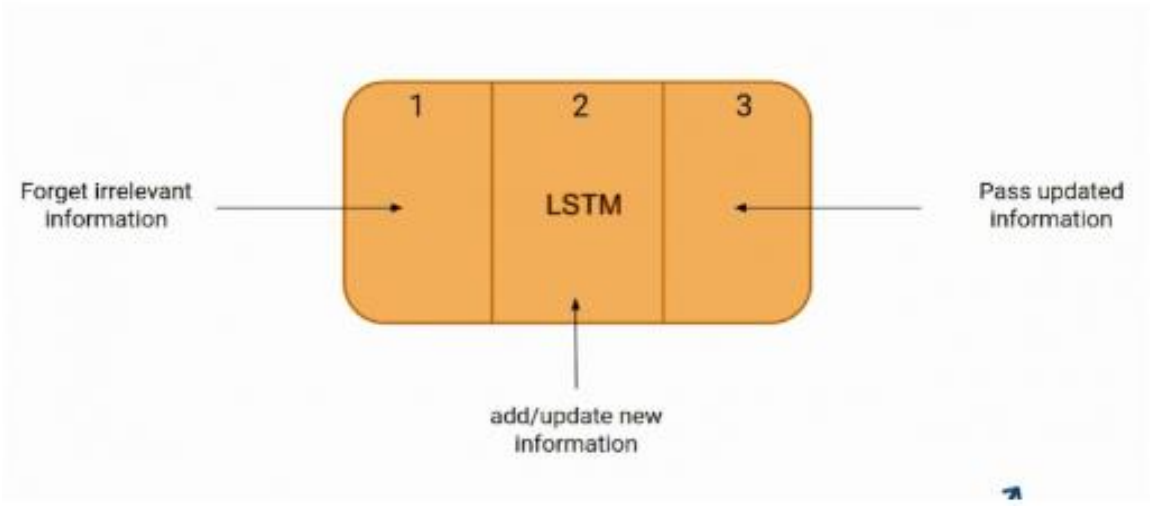
Exploratory data analysis is the crucial procedure of doing preliminary analyses of data in order to find patterns, identify abnormalities, validate hypotheses, and double-check assumptions with the aid of descriptive statistics and visualisations (Patil, 2018). In the exploratory data analysis part, the structure of data such as the data types, the number of variables and observations, the descriptive statistic and the length of stock period are explored. Besides, the missing value of the dataset is also examined. Moreover, the history of the stock price is explored through visualization techniques such as plotting, line chart and bar graph.

3.5 Data Pre-processing

As a part of data preparations, data pre-processing refers to any kind of manipulation done on original data to get it ready for another data handling technique. It has long been regarded as a crucial first stage in the data mining operation. The methods are typically applied early in the AI and machine learning technology process to guarantee the right output (Lawton, 2022). In this experiment, pre-processing such as normalizing the data by using mix max scaler, converting the data types of variables, converting array into dataset matrix, preparing the data for training and testing set and reshaping the data size are carried out.

3.6 Time Series Forecasting Model Building

3.6.1 Long-Short Term Memory Model (LSTM)

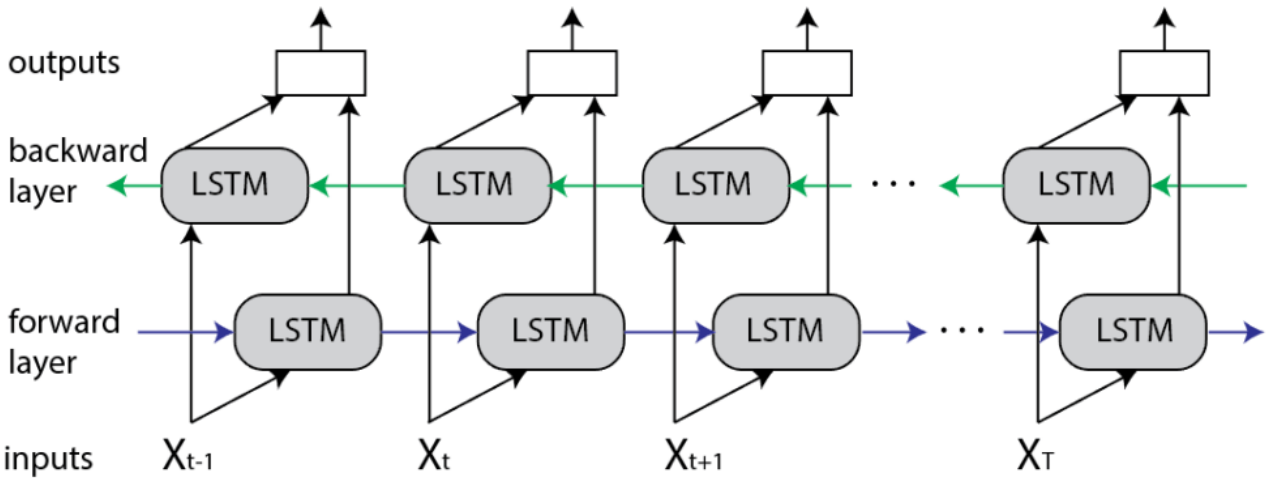


Recurrent neural networks that can learn order dependency in sequential forecasting tasks are known as Long Short-Term Memory (LSTM) networks. This architecture is primarily

developed to address the issue of vanishing and growing gradients. This kind of structure is also effective at sustaining long-distance links and understanding the relationship between numbers at the start and finish of a series (Zvornicanin, 2022). Based on the figure above, it showed that LSTM consists three parts. The first section determines if the data from the preceding timestamp needs to be preserved or can be ignored. The cell attempts to gain new data from the inputs to this cell in the second section. The cell finally transmits the revised data from the present timestamp to the following timestamp in the third section. Gates refers to these 3 LSTM cell components. The Forget gate, Input gate, and Output gate are the names of the three components, respectively (Saxena, 2021).

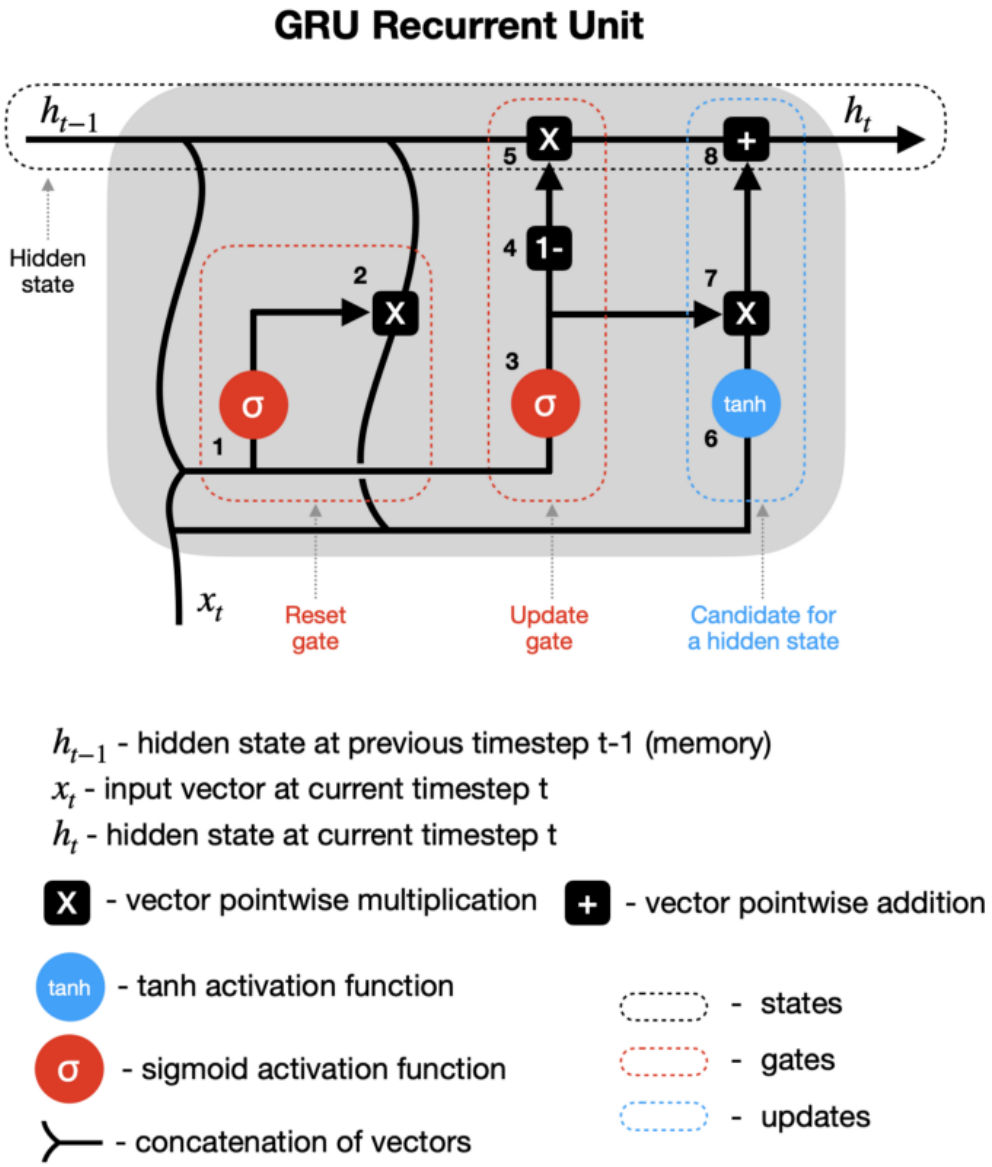
3.6.2 Bidirectional Long-Short-Term Memory Model (BI-LSTM)

Each training cycle is given both forward and backward in bidirectional LSTMs in order to differentiate recurrent nets. The same output layer is shared by both sequences. Bidirectional LSTMs are fully aware of every step in a particular pattern, as well as all that came previously and came afterwards it. It may use data from either side and, unlike regular LSTM, the data flows across both sides. On both sides of the sequence, it is a potent tool for simulating the serial relationships among phrases and words. In conclusion, BiLSTM changes the data flows by adding one extra LSTM layer. It simply means that in the extra LSTM layer, the input data flows backwards. The output from the two LSTM layers is then combined in a variety of ways, including mean, total, multiplied, and concatenated (Zvornicanin, 2022). Figure below showed the architecture of bidirectional LSTM.



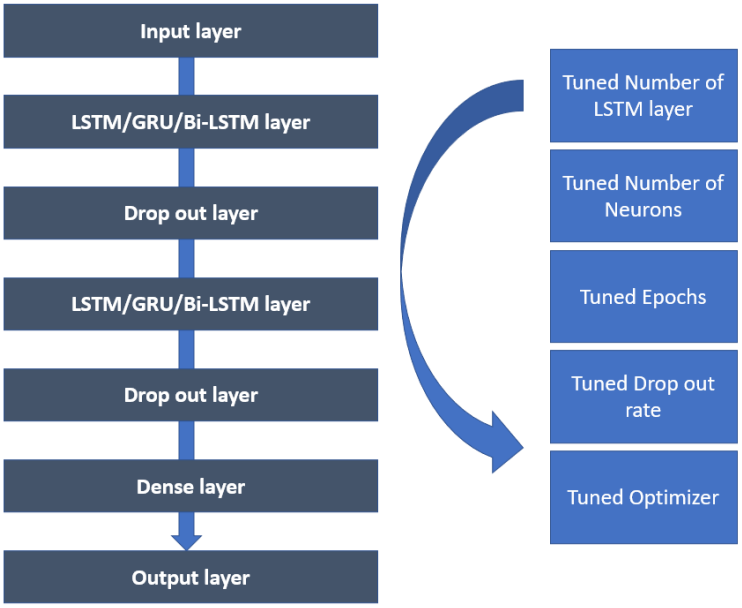
3.6.3 Gated Recurrent Unit (GRU)

GRUs able to handle the issue with vanishing gradients that affect regular recurrent neural networks (values used to update network weights). Grading could become too little to have an impact on training if it decreases over the period as it back propagates, rendering the neural network untrainable. RNNs can basically "forget" lengthier patterns if a layer in a neural net is unable to adapt. The update gate and reset gate are two gates that GRUs utilize to address this issue. The data sent to the output is determined by two vectors: the update gate and the reset gate. They can be taught to retain historical knowledge or to discard data that is unrelated to the forecast. (MarketMuse, n.d). Figure below showed the architecture of the gated recurrent unit (Dobilas, 2022):



3.7 Hyperparameter Tuning

Selecting the best set of hyperparameters for a training methodology is known as hyperparameter tuning. Hyperparameter tuning is essential for deep learning algorithms to work. The model's effectiveness is maximised by using that set of hyperparameters, which minimizes a predetermined loss function and results in better delivery with minimal mistakes. In this session, the hyperparameter tuning is carried out by using manual method to understand the effect of each parameter during the tuning process (Navas, n.d). The 5 main parameters that are tuned in this session included the number of layers, number of neurons, number of epochs, learning rate and dropout rate. Every best model in one session is used in the next session for further tuning until the end. Figure below showed the flow of hyperparameter tuning:

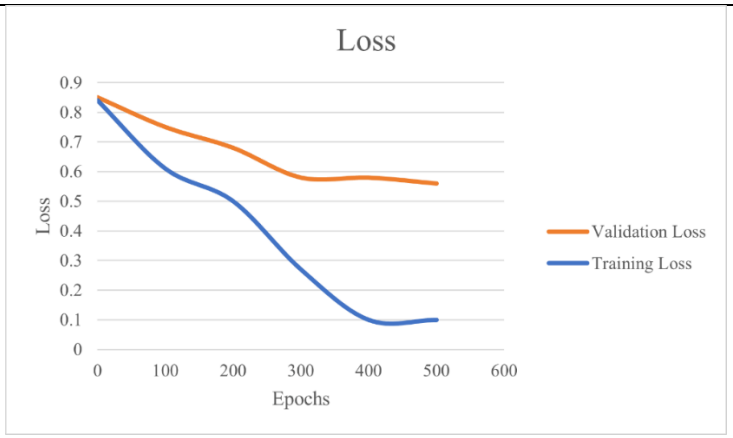
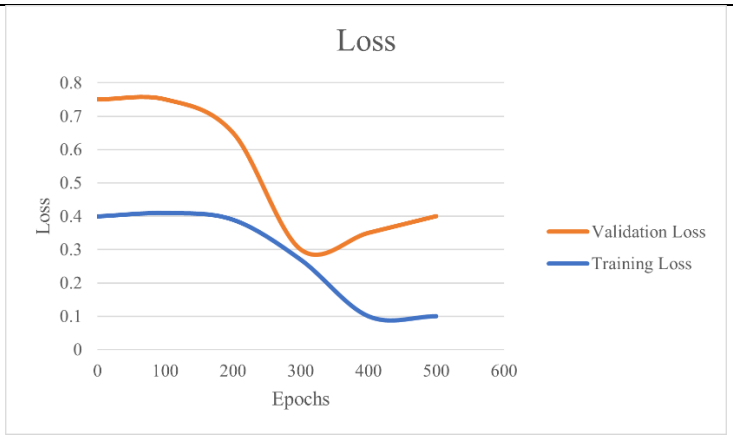
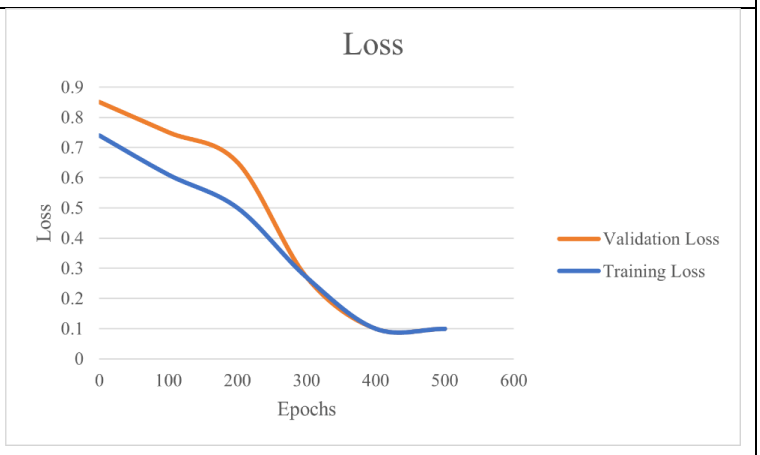


3.8 Evaluation Matrix

3.8.1 Training and Validation loss

An indicator of how well a deep learning algorithm matches the data for training is the training loss. In other words, it evaluates the model's performance on the training set. The total of mistakes for each sample in the training set is used to statistically determine the training loss (Baeldung, 2022). A deep learning model's function on the testing set is evaluated using the measure of validation loss. The dataset's validation data is a section set aside to check the model's efficacy. Similar to the training loss, the validation loss is determined by adding the errors for each sample in the test dataset. After every epoch, the validation loss is also recorded. This helps us determine whether the algorithm needs to be adjusted or tuned further (Baeldung,

n.d). Validation loss is chosen in this project as the evaluation matrix because by using the validation loss, we can indirectly look into the accuracy of the model. Moreover, by comparing the validation loss with the training loss, we can identify whether the model is good, underfitting or overfitting. Table below showed some examples of underfitting, overfitting and good model.

Underfitting model	
Overfitting model	
Good fit model	

3.8.2 Root Mean Square Error (RMSE)

The inaccuracy of a model in forecasting statistical data is typically measured using the Root Mean Square Error (RMSE). Square the quantity for each piece of data by multiplying it by the vertical distance between the point and the matching y value on the curve match. Positive values are not cancelled by negative ones according to the squaring process. The fit is more accurate to the data when the Mean Squared Error is lower. RMSE is used in this project because the RMSE is a better indicator of goodness of fit than the correlation coefficient because it can be immediately translated into measurement units (Padhma, 2021). Figure below showed the formula of RMSE:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

3.8.3 R-squared (R^2)

R-squared (R^2) is a quantitative metric that shows how much of a dependent factor's variance is described by one or more independent factors in a regression model. R-squared measures how well the variation of one variable accounts for the variance of the second, as opposed to correlations, which describes the degree of the interaction between the independent and dependent variables. Therefore, if a model's R2 is 0.50, its inputs can account for around half of the variance observed. This evaluation matrix is used in this project is to understand whether the model is biased. The formula below showed the R formula:

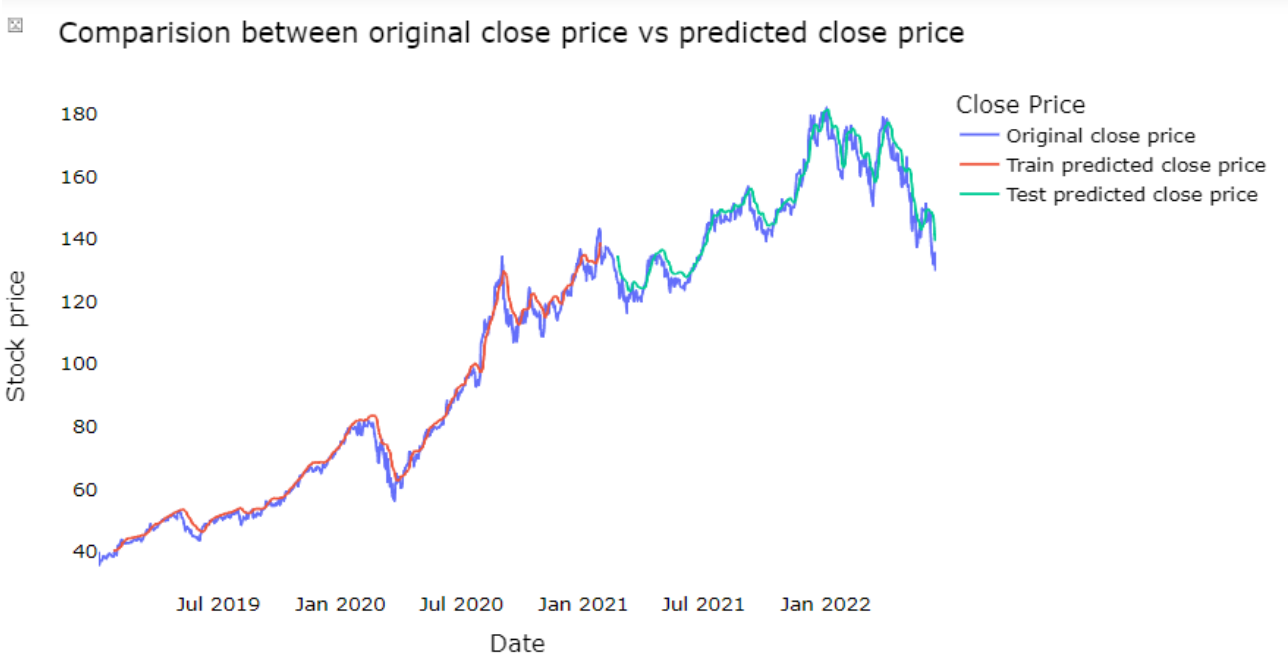
$$R = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}}$$

where,

n = Number of observations
Σx = Total value of the independent variable
Σy = Total value of the dependent variable
Σxy = Sum of the product of independent and dependent variable
Σx2 = Sum of the squares of the independent variable value
Σy2 = Sum of the squares of the dependent variable value

3.6.4 Prediction versus real stock visualization

After building the model, the visualization plot of comparison between the prediction result and the original results is built. In this plot, the training and prediction values is shown with different colour when comparing with the original data. By using this plot, researcher can easily identify the accuracy of the model and understand the model is good, underfitting or overfitting. The figure below showed the example of the visualization plot:



CHAPTER 4

IMPLEMENTATION

4.1 Exploratory Data Analysis

4.1.1 Structure of Data

```
In [3]: df.shape
Out[3]: (10468, 7)

In [4]: df.columns
Out[4]: Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')

In [5]: df.head()
```

Out[5]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1980-12-12	0.128348	0.128906	0.128348	0.128348	0.100178	469033600
1	1980-12-15	0.122210	0.122210	0.121652	0.121652	0.094952	175884800
2	1980-12-16	0.113281	0.113281	0.112723	0.112723	0.087983	105728000
3	1980-12-17	0.115513	0.116071	0.115513	0.115513	0.090160	86441600
4	1980-12-18	0.118862	0.119420	0.118862	0.118862	0.092774	73449600

Figure 4.1 Data Structure Exploration

Based on the figure 4.1, it is shown that the dataset consists 7 variables and 10468 observations. The 10468 observations indicate there are 10468 days of stock price in this dataset. The columns of the dataset and the first five rows are also explored in this section.

```
In [4]: df.dtypes
Out[4]: Date          object
        Open         float64
        High         float64
        Low          float64
        Close        float64
        Adj Close    float64
        Volume         int64
        dtype: object
```

Figure 4.2 Data types exploration

Based on Figure 4.2, it is shown that there are 5 ‘float64’, 1 ‘object’ and 1 ‘int64’. The date needs to be converted into datetime. This step will be carried out in the pre-processing step.

```
In [11]: df.describe()
```

Out[11]:

	Open	High	Low	Close	Adj Close	Volume
count	10468.000000	10468.000000	10468.000000	10468.000000	10468.000000	1.046800e+04
mean	14.757987	14.921491	14.594484	14.763533	14.130431	3.308489e+08
std	31.914174	32.289158	31.543959	31.929489	31.637275	3.388418e+08
min	0.049665	0.049665	0.049107	0.049107	0.038329	0.000000e+00
25%	0.283482	0.289286	0.276786	0.283482	0.235462	1.237768e+08
50%	0.474107	0.482768	0.465960	0.475446	0.392373	2.181592e+08
75%	14.953303	15.057143	14.692589	14.901964	12.835269	4.105794e+08
max	182.630005	182.940002	179.119995	182.009995	181.511703	7.421641e+09

Figure 4.3 Data description

Figure 4.3 showed the data description for the Apple stock dataset.

4.1.2 Missing Value Exploration

```
In [9]: df.isnull().sum()
```

```
Out[9]: Date         0
        Open         0
        High         0
        Low          0
        Close        0
        Adj Close    0
        Volume       0
        dtype: int64
```

Figure 4.4 Missing value exploration

Based on figure 4.4, it is shown that there is no missing value in the dataset. Hence, no missing value imputation need to be done in the pre-processing steps.

4.1.3 Apple Stock History Exploration

```
print("Starting date: ",df.iloc[0][0])
print("Ending date: ", df.iloc[-1][0])
print("Duration: ", df.iloc[-1][0]-df.iloc[0][0])
```

```
Starting date:  1980-12-12 00:00:00
Ending date:  2022-06-17 00:00:00
Duration:  15162 days 00:00:00
```

Figure 4.5 Stock Duration

Based on figure 4.5, it is shown that the apple stock history in the dataset start from 1980-12-12 until 2022-06-17 which are total 15162 days.

4.1.4 Apple Stock Monthly Exploration

```
monthvise= df.groupby(df['Date'].dt.strftime('%B'))[['Open','Close']].mean()
new_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August',
             'September', 'October', 'November', 'December']
monthvise = monthvise.reindex(new_order, axis=0)
monthvise
```

Date	Open	Close
January	14.850122	14.842134
February	15.339262	15.349828
March	15.340098	15.355185
April	15.566919	15.558388
May	15.229152	15.240276
June	14.417663	14.424042
July	13.289663	13.312471
August	14.046364	14.070591
September	14.525162	14.477264
October	14.327148	14.337092
November	14.823220	14.841222
December	15.381761	15.391395

Figure 4.6 Monthly comparison between Stock open and close price

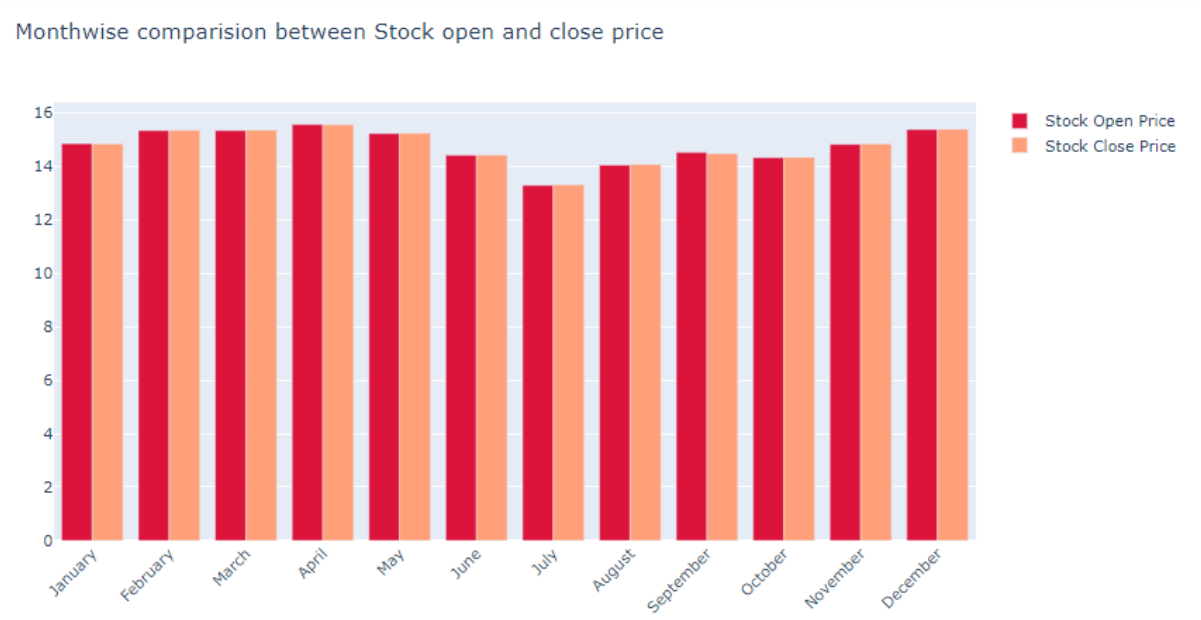


Figure 4.7 Monthly comparison between Stock open and close price bar chart

Figure 4.6 and figure 4.7 showed the monthly comparison between stock open and close price. Based on the figure, we can see that April has the highest monthly open price and closing price while July has the lowest open price and closing price.

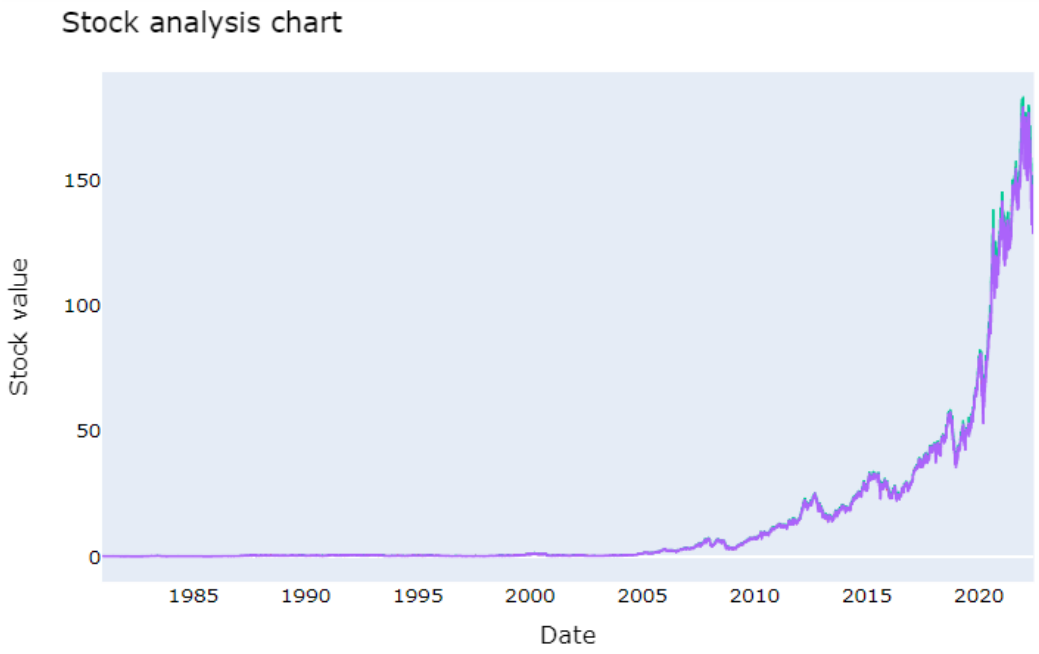


Figure 4.8 Stock analysis chart

Based on Figure 4.8, it showed that the Apple stock price has a significant rising from the year of 2010.

4.2 Data Pre-processing

Convert Date field into datetime format

```
# convert date field from string to Date format
import datetime as dt
df['Date'] = pd.to_datetime(df.Date)
df.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	1980-12-12	0.128348	0.128906	0.128348	0.128348	0.100178	469033600
1	1980-12-15	0.122210	0.122210	0.121652	0.121652	0.094952	175884800
2	1980-12-16	0.113281	0.113281	0.112723	0.112723	0.087983	105728000
3	1980-12-17	0.115513	0.116071	0.115513	0.115513	0.090160	86441600
4	1980-12-18	0.118862	0.119420	0.118862	0.118862	0.092774	73449600

Figure 4.9 Convert Date field into datetime format

Figure 4.9 showed that conversion of data field into datetime format.

```
In [89]: closedf = closedf[closedf['Date'] > '2019-01-01']
         close_stock = closedf.copy()
         print("Total data for prediction: ",closedf.shape[0])

         Total data for prediction:  873
```

Figure 4.10: Amount of last 30 months data

Based on figure 4.10, it shown tha the total amount of data after 2019-01-01 which is around 30 months of data. The output showed that there are total 873 data are used for prediction.

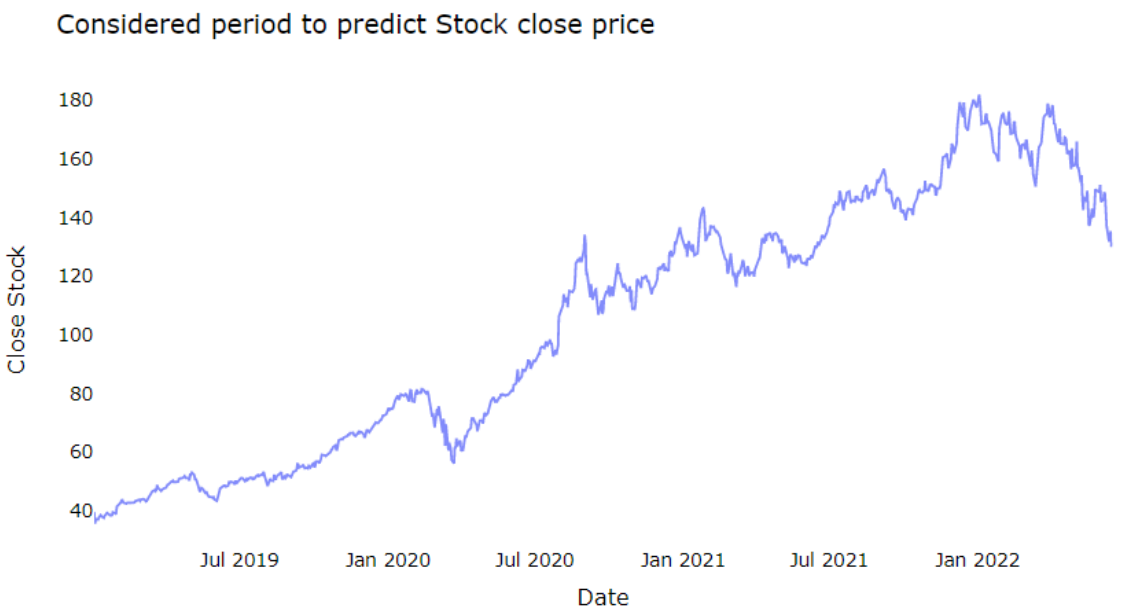


Figure 4.11 Period to predict stock close price plot

Based on figure 4.11, it showed that plot the explore the period to predict stock close price which is from 2019-01-01 onwards until the year of 2022. It showed that from 2019, there is gradually increase in the closing price while a significant increase is started from year 2020.

```
training_size=int(len(closedf)*0.60)
test_size=len(closedf)-training_size
train_data,test_data=closedf[0:training_size,:],closedf[training_size:len(closedf),:1]
print("train_data: ", train_data.shape)
print("test_data: ", test_data.shape)
```

```
train_data:  (523, 1)
test_data:  (350, 1)
```

```
# convert an array of values into a dataset matrix
def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
        a = dataset[i:(i+time_step), 0]   ###i=0, 0,1,2,3-----99   100
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)
```

```
time_step = 15
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

print("X_train: ", X_train.shape)
print("y_train: ", y_train.shape)
print("X_test: ", X_test.shape)
print("y_test", y_test.shape)
```

```
X_train:  (6264, 15)
y_train:  (6264,)
X_test:  (4172, 15)
y_test (4172,)
```

Figure 4.12 Train-test split

```
# reshape input to be [samples, time steps, features] which is required for LSTM
X_train =X_train.reshape(X_train.shape[0],X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0],X_test.shape[1] , 1)

print("X_train: ", X_train.shape)
print("X_test: ", X_test.shape)
```

```
X_train:  (6264, 15, 1)
X_test:  (4172, 15, 1)
```

Figure 4.13 Reshape input

Based on figure 4.12, it showed that the data is split into training and testing set with 15-time steps. In figure 4.13, the X_train and X_test data are reshaped into 3 dimension dataset which is required for LSTM model building.

4.3 Model Building Implementation and Result

Below showed the general flow of coding for the model building and evaluation process. In the next section, all the result of these coding for LSTM, GRU and BILSTM are displayed.

- **Step 1: Model Building**

```
]: # Initialising the RNN
lstm_model = Sequential()

# Adding the first LSTM layer
lstm_model.add(LSTM(units = 32, return_sequences = True, activation= 'relu', input_shape = (time_step, 1)))


# Adding a second LSTM layer and some Dropout regularisation
# This layer produces a single output
lstm_model.add(LSTM(units = 32,activation= 'relu'))


# Adding the output layer
# This is the final layer produces the required numerical prediction
lstm_model.add(Dense(units = 1,activation= 'linear'))
```

- **Step 2: Compile the model**

```
# Compiling the RNN
lstm_model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

- **Step 3: Fitting the model to the training set**

```
# Fitting the RNN to the Training set
history = lstm_model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=50,batch_size=32,verbose=1)
```

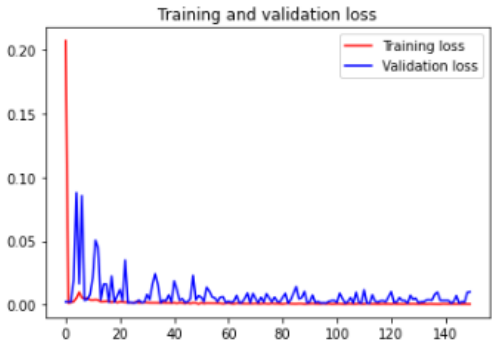
- **Step 4: Visualize the model summary**

```
lstm_model.summary()
```

```
Model: "sequential_51"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_111 (LSTM)              (None, 15, 256)           264192
_________________________________________________________________
dropout_12 (Dropout)         (None, 15, 256)           0
_________________________________________________________________
lstm_112 (LSTM)              (None, 512)               1574912
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_50 (Dense)             (None, 1)                 513
=================================================================
Total params: 1,839,617
Trainable params: 1,839,617
Non-trainable params: 0
_________________________________________________________________
```

- **Step 5: Plot the training and validation loss**

```
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(loss))

plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend(loc=0)
plt.figure()


plt.show()
```



- **Step 6: Do prediction and check performance metrics**

```
### Lets Do the prediction and check performance metrics
train_predict=lstm_model.predict(X_train)
test_predict=lstm_model.predict(X_test)
train_predict.shape, test_predict.shape
```

```
((507, 1), (334, 1))
```

- **Step 7: Transform back to original form**

```
# Transform back to original form

train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
original_ytrain = scaler.inverse_transform(y_train.reshape(-1,1))
original_ytest = scaler.inverse_transform(y_test.reshape(-1,1))
```

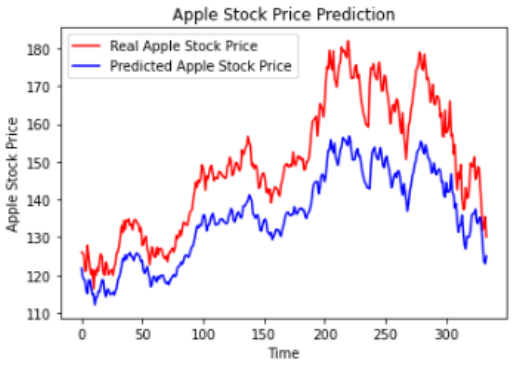
- **Step 8: Evaluate RMSE and R2 score**

```
# Evaluation metrices RMSE and R2 score
print("Train data RMSE: ", math.sqrt(mean_squared_error(original_ytrain,train_predict)))
print("Train data R2 score:", r2_score(original_ytrain, train_predict))
print("----------------------------------------------------------------------------------------------")
print("Test data RMSE: ", math.sqrt(mean_squared_error(original_ytest,test_predict)))
print("Test data R2 score:", r2_score(original_ytest, test_predict))
```

```
Train data RMSE:  4.212041890443418
Train data R2 score: 0.9781274199705463
----------------------------------------------------------------------------------------------
Test data RMSE:  14.687730023229546
Test data R2 score: 0.25653663147084416
```

- **Step 9: Visualizing the result**

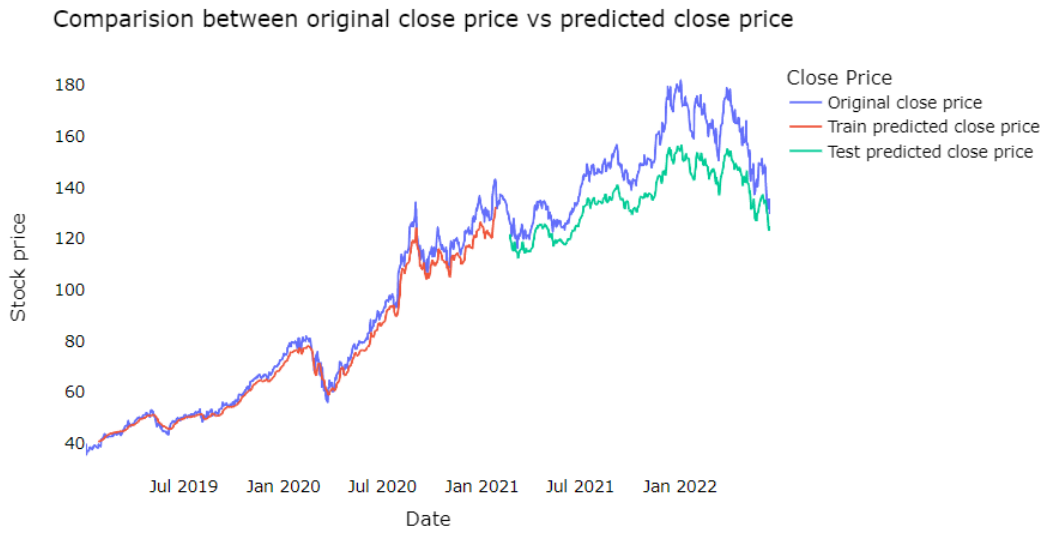
```
# Visualising the results
plt.plot(original_ytest, color='Red', label='Real Apple Stock Price')
plt.plot(test_predict, color='Blue', label='Predicted Apple Stock Price')
plt.title('Apple Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Apple Stock Price')
plt.legend()
plt.show()
```



- **Step 10: Shift train and test prediction for plotting**

```
# shift train predictions for plotting

look_back=time_step
trainPredictPlot = np.empty_like(closedf)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
print("Train predicted data: ", trainPredictPlot.shape)

# shift test predictions for plotting
testPredictPlot = np.empty_like(closedf)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(closedf)-1, :] = test_predict
print("Test predicted data: ", testPredictPlot.shape)

names = cycle(['Original close price','Train predicted close price','Test predicted close price'])


plotdf = pd.DataFrame({'Date': close_stock['Date'],
                       'original_close': close_stock['Close'],
                      'train_predicted_close': trainPredictPlot.reshape(1,-1)[0].tolist(),
                      'test_predicted_close': testPredictPlot.reshape(1,-1)[0].tolist()})

fig = px.line(plotdf,x=plotdf['Date'], y=[plotdf['original_close'],plotdf['train_predicted_close'],
                                          plotdf['test_predicted_close']],
              labels={'value':'Stock price','date': 'Date'})
fig.update_layout(title_text='Comparision between original close price vs predicted close price',
                  plot_bgcolor='white', font_size=15, font_color='black', legend_title_text='Close Price')
fig.for_each_trace(lambda t:  t.update(name = next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

```
Train predicted data:  (873, 1)
Test predicted data:  (873, 1)
```



4.3.1 Long Short-Term Memory

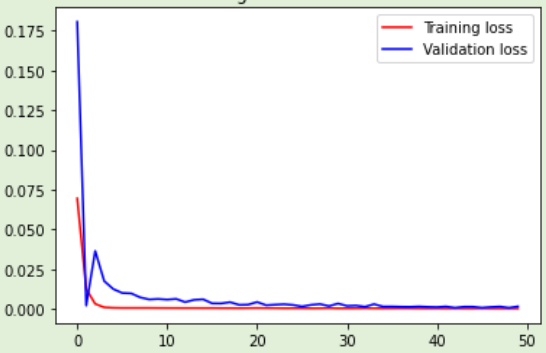
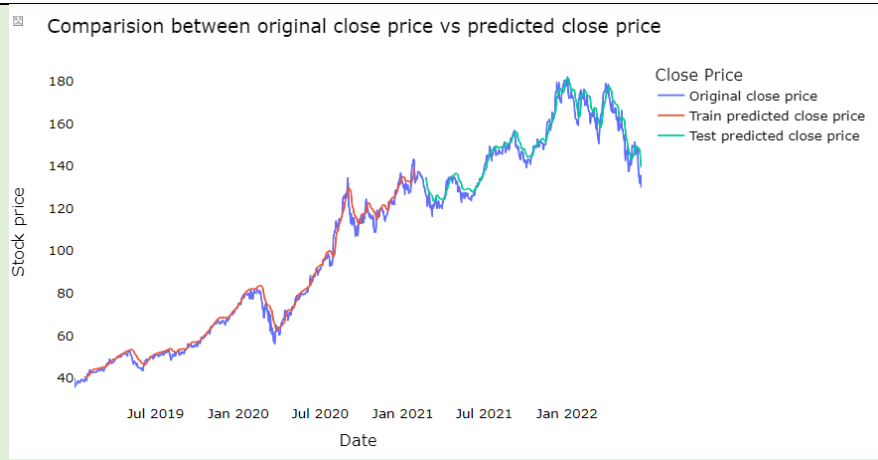
4.3.1.1 Tuning of number of LSTM layers

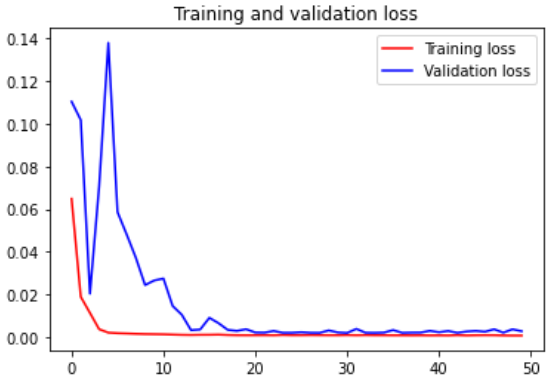
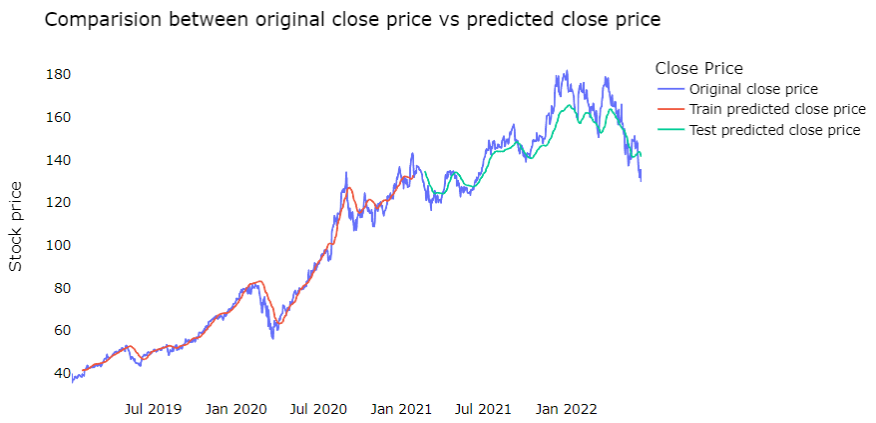
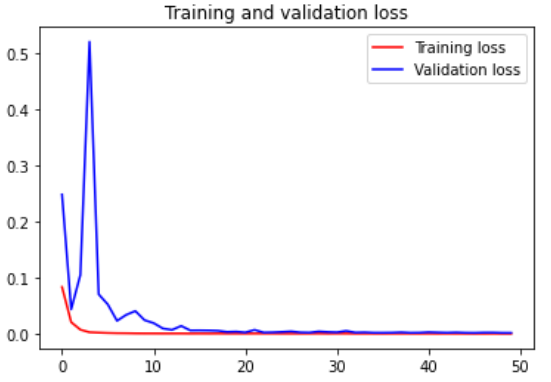
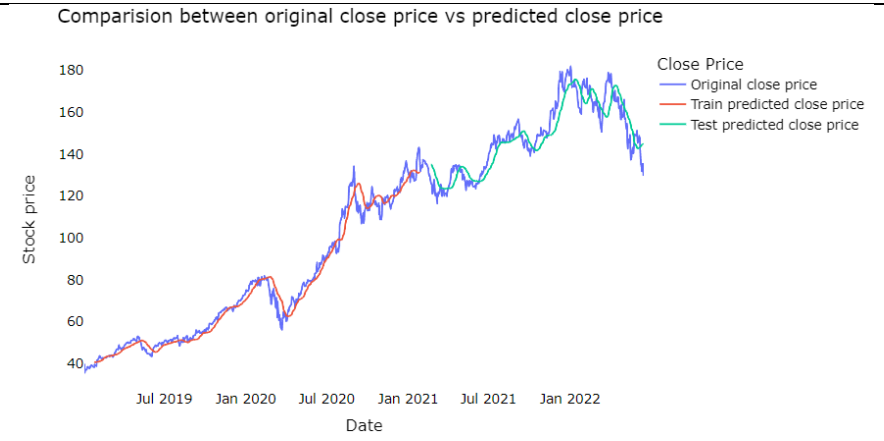
In this section, two, three and four LSTM layers are test. The best result architecture will be used in the next tuning session. The other default parameters included 50 epochs, 32 batch size, 32 units, adam optimizer, loss function using the mean square error, using relu activation function in the hidden layer and linear activation function in the output layer. One of the sample codes of 1 LSTM input layer, 1 LSTM hidden layer, and 1 dense output layer model building is shown below:

```
]: # Initialising the RNN
lstm_model = Sequential()

# Adding the first LSTM layer
lstm_model.add(LSTM(units = 32, return_sequences = True, activation= 'relu', input_shape = (time_step, 1)))


# Adding a second LSTM layer and some Dropout regularisation
# This layer produces a single output
lstm_model.add(LSTM(units = 32,activation= 'relu'))


# Adding the output layer
# This is the final layer produces the required numerical prediction
lstm_model.add(Dense(units = 1,activation= 'linear'))

# Compiling the RNN
lstm_model.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Fitting the RNN to the Training set
history = lstm_model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=50,batch_size=32,verbose=1)
```

Number of LSTM layers	Model Summary
1 LSTM input layers, 1 LSTM hidden layer, 1 Dense output layer	Model: "sequential_28" _____ Layer (type) Output Shape Param # ===== lstm_62 (LSTM) (None, 15, 32) 4352 _____ lstm_63 (LSTM) (None, 32) 8320 _____ dense_28 (Dense) (None, 1) 33 ===== Total params: 12,705 Trainable params: 12,705 Non-trainable params: 0 _____
1 LSTM input layers, 2 LSTM hidden layer, 1 Dense output layer	Model: "sequential_8" _____ Layer (type) Output Shape Param # ===== lstm_19 (LSTM) (None, 15, 32) 4352 _____ lstm_20 (LSTM) (None, 15, 32) 8320 _____ lstm_21 (LSTM) (None, 32) 8320 _____ dense_8 (Dense) (None, 1) 33 ===== Total params: 21,025 Trainable params: 21,025 Non-trainable params: 0 _____

1 LSTM input layers, 3 LSTM hidden layer, 1 Dense output layer	Model: "sequential_30" _____ Layer (type) Output Shape Param # ===== lstm_67 (LSTM) (None, 15, 32) 4352 _____ lstm_68 (LSTM) (None, 15, 32) 8320 _____ lstm_69 (LSTM) (None, 15, 32) 8320 _____ lstm_70 (LSTM) (None, 32) 8320 _____ dense_30 (Dense) (None, 1) 33 ===== Total params: 29,345 Trainable params: 29,345 Non-trainable params: 0 _____

	Training and validation loss	RMSE	R score	Predicted close price versus original close price
1 LSTM input layers, 1 LSTM hidden layer, 1 Dense output layer		Train data RMSE: 3.217725874492939 Test data RMSE: 4.690701156172092	Train data R2 score: 0.9872352375731307 Test data R2 score: 0.9241726605911745	

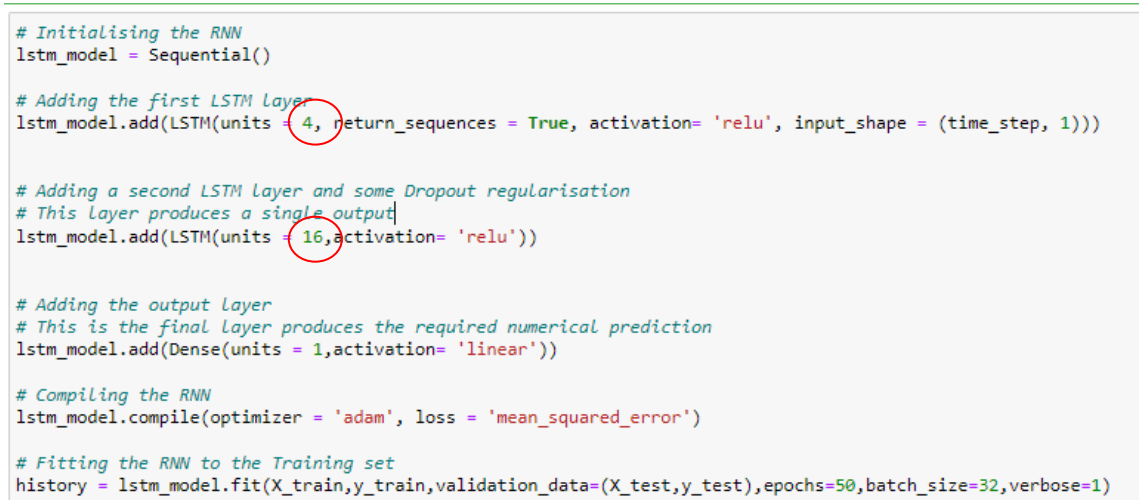
1 LSTM input layers, 2 LSTM hidden layer, 1 Dense output layer		Train data RMSE: 4.1119773401294 08 Test data RMSE: 7.771 70027812242 6	Train data R2 score: 0.9791543 195442978 Test data R2 score: 0.7918471797318274	
1 LSTM input layers, 3 LSTM hidden layer, 1 Dense		Train data RMSE: 4.3792069623203 67 Test data RMSE: 6.986 84482726944 2	Train data R2 score: 0.9763568 361594799 Test data R2 score: 0.8317665276611 67	

output layer				

In this tuning session, 1 LSTM input layers, 1 LSTM hidden layer, 1 dense output layer has the highest performance with 0.9241726605911745 R^2 score and 4.69070156172092 RMSE value when compared with others. Hence, the 1 LSTM input layers, 1 LSTM hidden layer, 1 Dense output layer architecture is used in the next tuning session which is the tuning of the neuron number.

4.3.1.2 Number of neurons

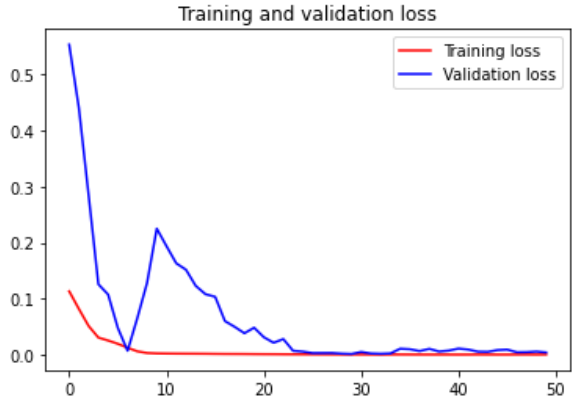
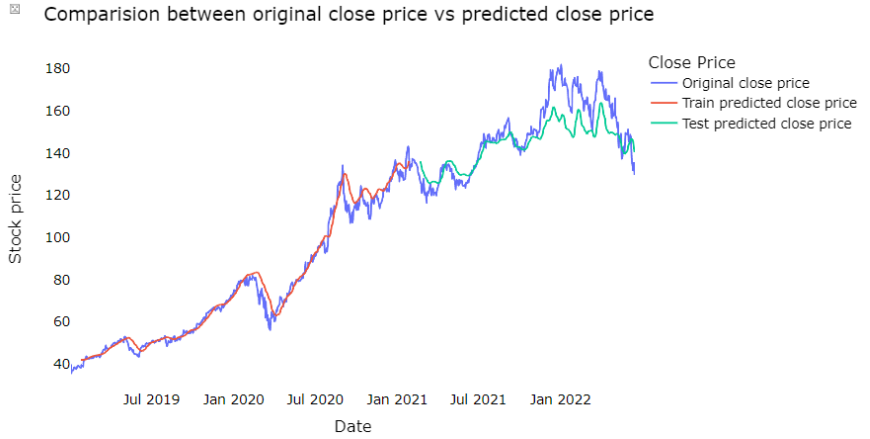
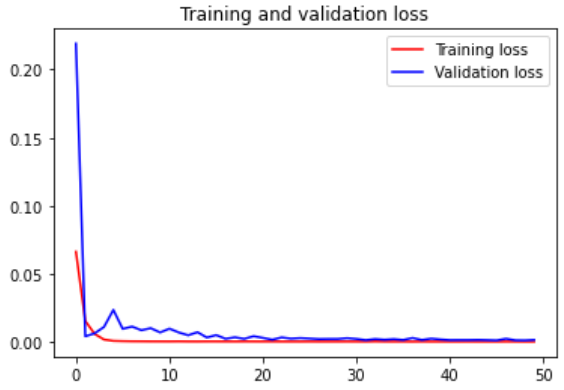
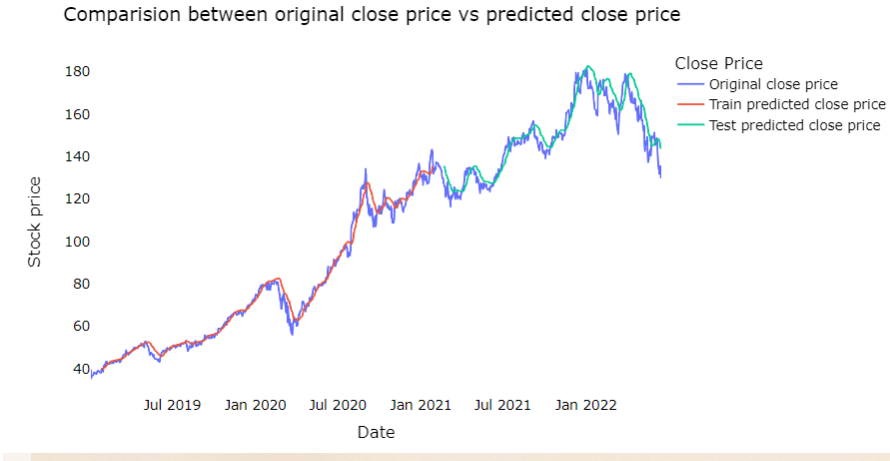
In this session, 1 LSTM input layers, 1 LSTM hidden layer, 1 dense output layer are used to understand the best number of neurons with the default parameters that were mentioned previously. The set of neurons number included 4 with 16 neurons, 16 with 32 neurons, 32 with 64 neurons, 64 with 128 neurons, 128 with 256 neurons, and 256 with 512 neurons. The figure below showed the sample coding of 4 and 16 units. The units that were circled indicate the parameter that needs to be tuned:

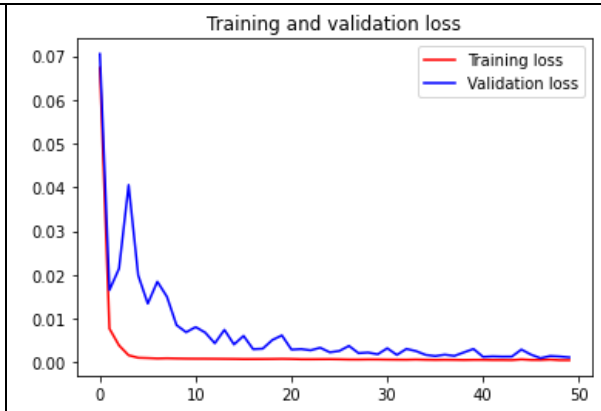
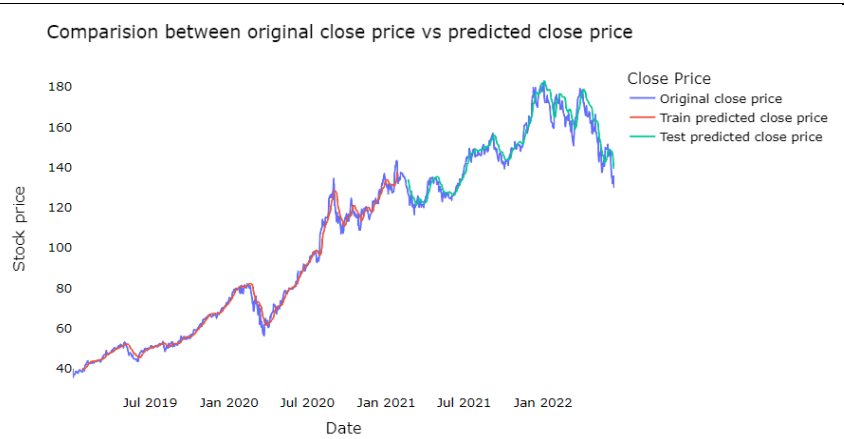
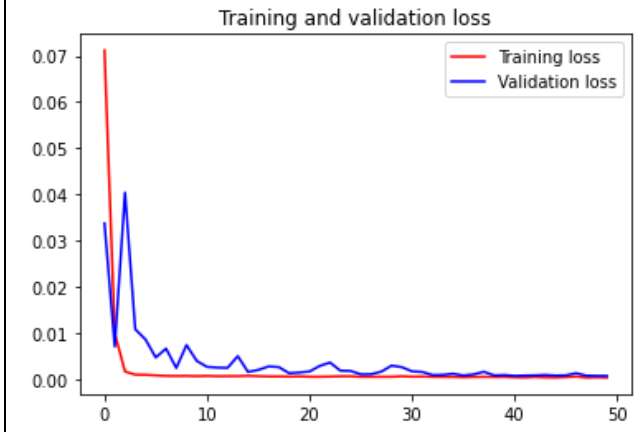
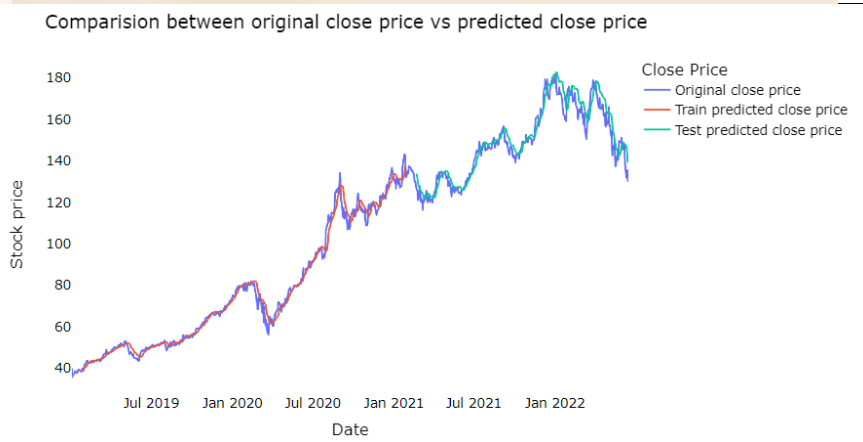


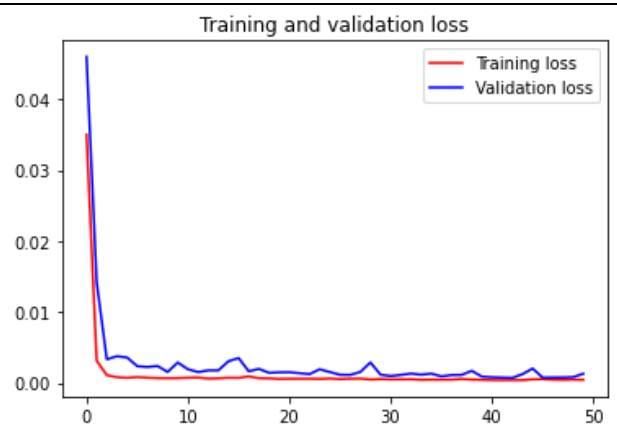
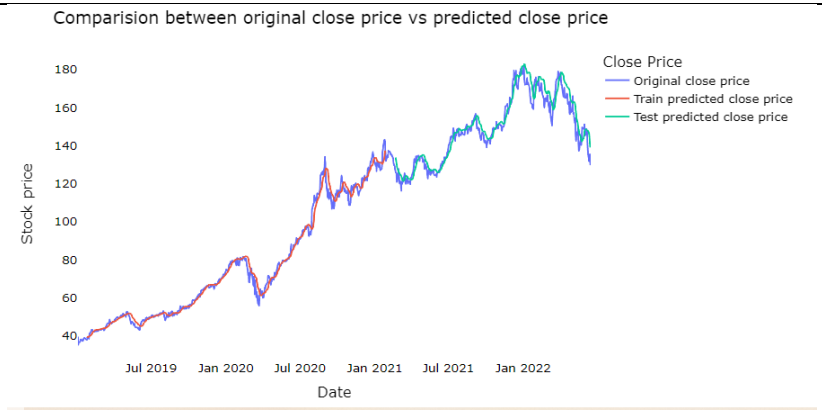
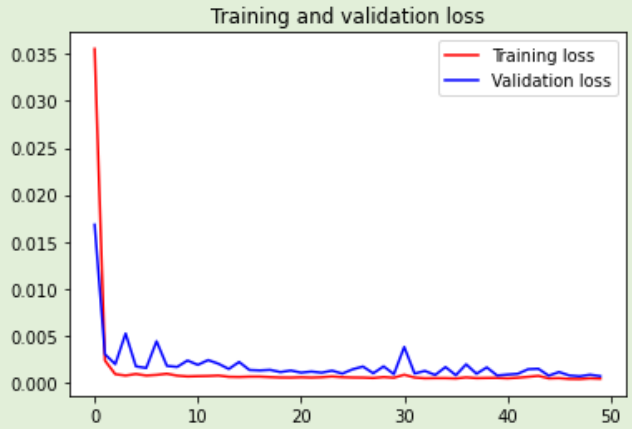
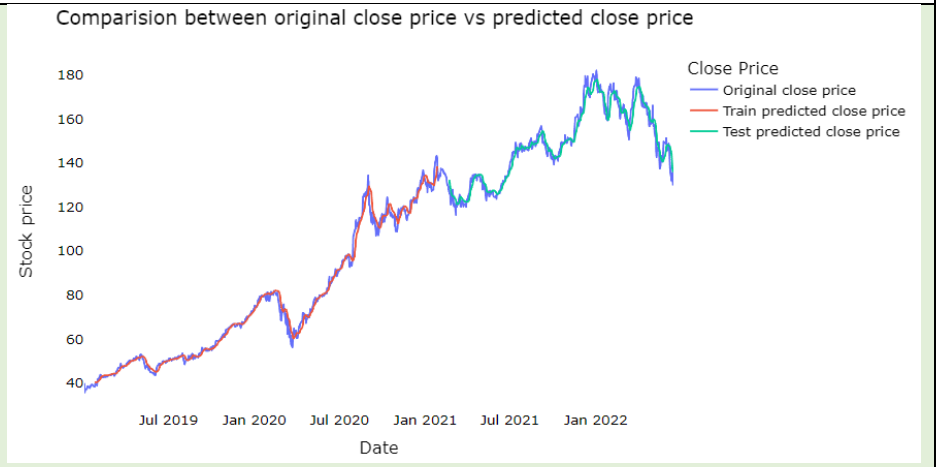
```
# Initialising the RNN
lstm_model = Sequential()

# Adding the first LSTM layer
lstm_model.add(LSTM(units = 4, return_sequences = True, activation= 'relu', input_shape = (time_step, 1)))


# Adding a second LSTM layer and some Dropout regularisation
# This layer produces a single output
lstm_model.add(LSTM(units = 16,activation= 'relu'))


# Adding the output layer
# This is the final layer produces the required numerical prediction
lstm_model.add(Dense(units = 1,activation= 'linear'))

# Compiling the RNN
lstm_model.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Fitting the RNN to the Training set
history = lstm_model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=50,batch_size=32,verbose=1)
```

Number of neurons	Model Summary
4 neurons and 16 neurons	Model: "sequential_10" _____ Layer (type) Output Shape Param # ===== lstm_26 (LSTM) (None, 15, 4) 96 _____ lstm_27 (LSTM) (None, 16) 1344 _____ dense_10 (Dense) (None, 1) 17 ===== Total params: 1,457 Trainable params: 1,457 Non-trainable params: 0
16 neurons and 32 neurons	Model: "sequential_11" _____ Layer (type) Output Shape Param # ===== lstm_28 (LSTM) (None, 15, 16) 1152 _____ lstm_29 (LSTM) (None, 32) 6272 _____ dense_11 (Dense) (None, 1) 33 ===== Total params: 7,457 Trainable params: 7,457 Non-trainable params: 0 _____
32 neurons and 64 neurons	Model: "sequential_12" _____ Layer (type) Output Shape Param # ===== lstm_30 (LSTM) (None, 15, 32) 4352 _____ lstm_31 (LSTM) (None, 64) 24832 _____ dense_12 (Dense) (None, 1) 65 ===== Total params: 29,249 Trainable params: 29,249 Non-trainable params: 0 _____

64 neurons and 128 neurons	Model: "sequential_13" _____ Layer (type) Output Shape Param # ===== lstm_32 (LSTM) (None, 15, 64) 16896 _____ lstm_33 (LSTM) (None, 128) 98816 _____ dense_13 (Dense) (None, 1) 129 ===== Total params: 115,841 Trainable params: 115,841 Non-trainable params: 0 _____
128 neurons and 256 neurons	Model: "sequential_14" _____ Layer (type) Output Shape Param # ===== lstm_34 (LSTM) (None, 15, 128) 66560 _____ lstm_35 (LSTM) (None, 256) 394240 _____ dense_14 (Dense) (None, 1) 257 ===== Total params: 461,057 Trainable params: 461,057 Non-trainable params: 0 _____
256 neurons and 512 neurons	Model: "sequential_15" _____ Layer (type) Output Shape Param # ===== lstm_36 (LSTM) (None, 15, 256) 264192 _____ lstm_37 (LSTM) (None, 512) 1574912 _____ dense_15 (Dense) (None, 1) 513 ===== Total params: 1,839,617 Trainable params: 1,839,617 Non-trainable params: 0

Number of neurons	Training and validation loss	RMSE	R2 score	Predicted close price versus original close price
4 neurons and 16 neurons		Train data RMSE: 4.1146717576 7478 Test data RMSE: 9.7783457273 84457	Train data R2 score: 0.9791269 91879 5159 Test data R2 score: 0.670 48059 90190 768	
16 neurons and 32 neurons		Train data RMSE: 3.4642905070 448595 Test data RMSE: 6.1942696714 70789	Train data R2 score: 0.9852040 36263 565 Test data R2 score: 0.867 76986 83579 428	

32 neurons and 64 neurons		Train data RMSE: 3.01897308 1179927 Test data RMSE: 4.90047333 984007	Train data R2 score: 0.98876344 62914958 Test data R2 score: 0.91723889 61553438	
64 neurons and 128 neurons		Train data RMSE: 2.84392509 58134976 Test data RMSE: 4.11868388 3595195	Train data R2 score: 0.99002871 8937 8706 Test data R2 score: 0.94153887 96988 947	

128 neurons and 256 neurons		Train data RMSE: 3.252618815692968 7 Test data RMSE: 5.313008925491158	Train data R2 score: 0.9869568949944169 Test data R2 score: 0.9027182658286826	
256 neurons and 512 neurons		Train data RMSE: 2.669144209222162 Test data RMSE: 3.801355046383226	Train data R2 score: 0.9912166795480827 Test data R2 score: 0.9502002592349138	

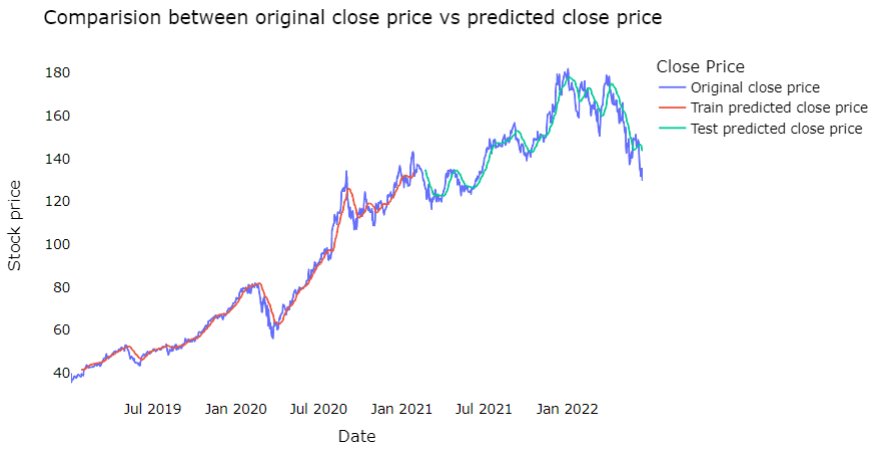
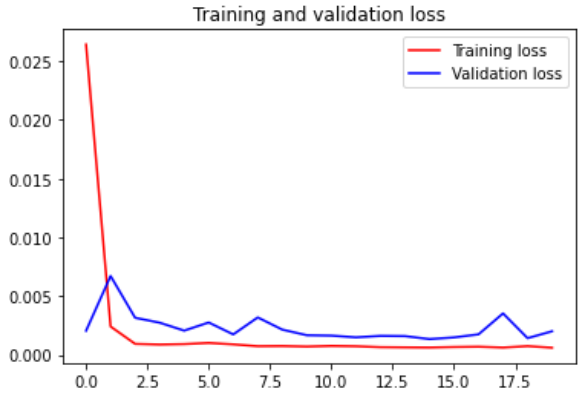
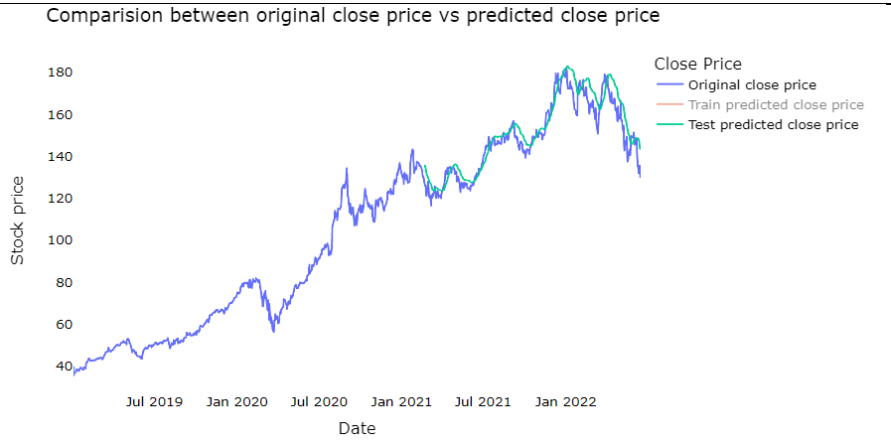
Based on the results, it shown that 256 neurons and 512 neurons under the 1 LSTM input layers, 1 LSTM hidden layer, 1 dense output layer has the best result which is only 3.8 RMSE and 0.95 R^2 score. This architecture is then used for the epochs number tuning in the next section.

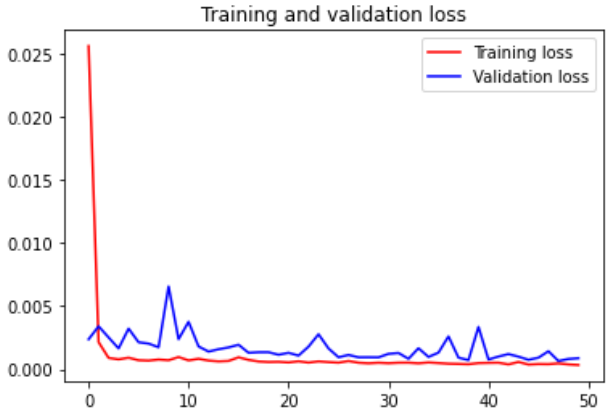
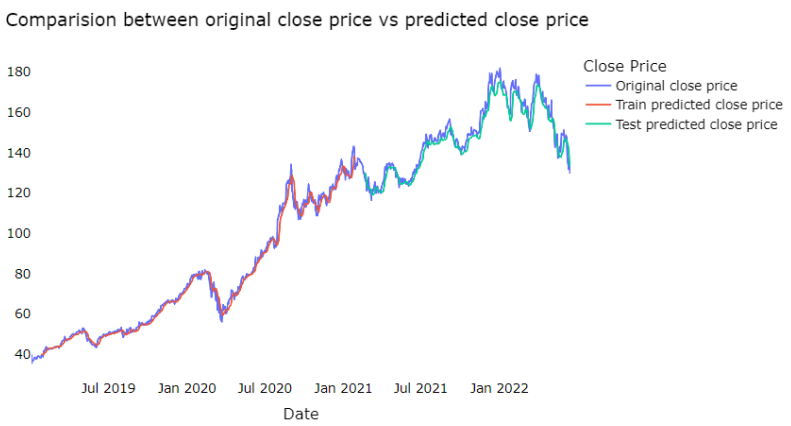
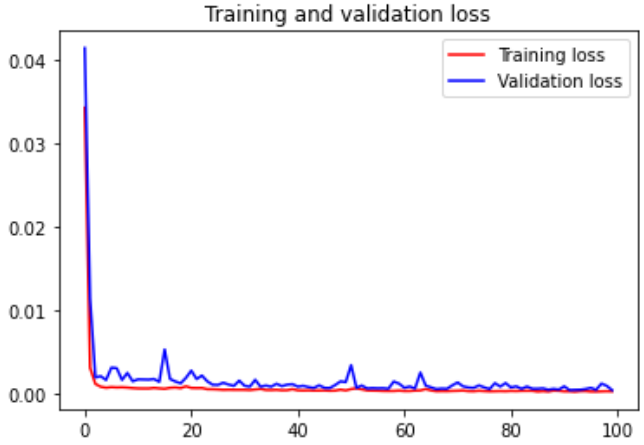
4.3.1.3 Number of epochs

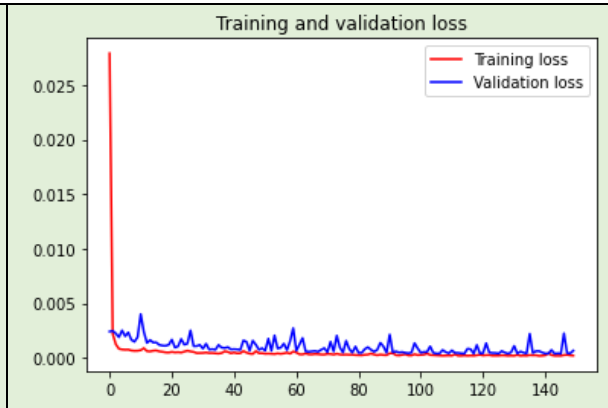
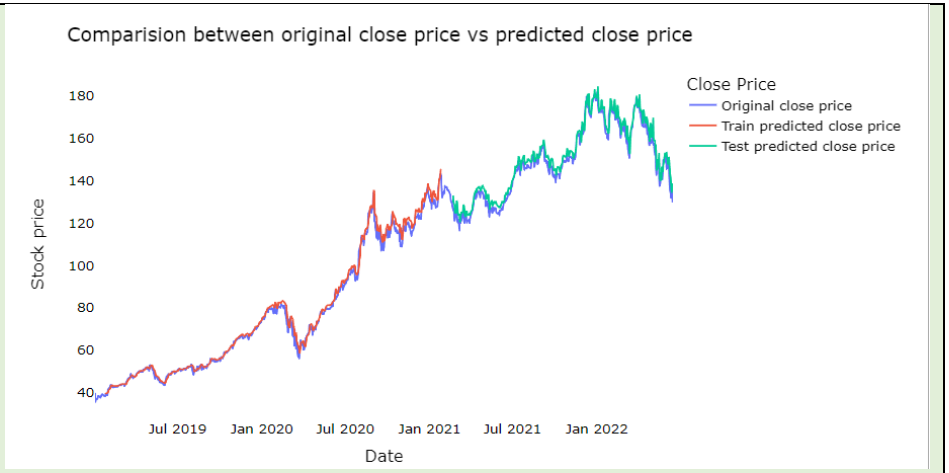
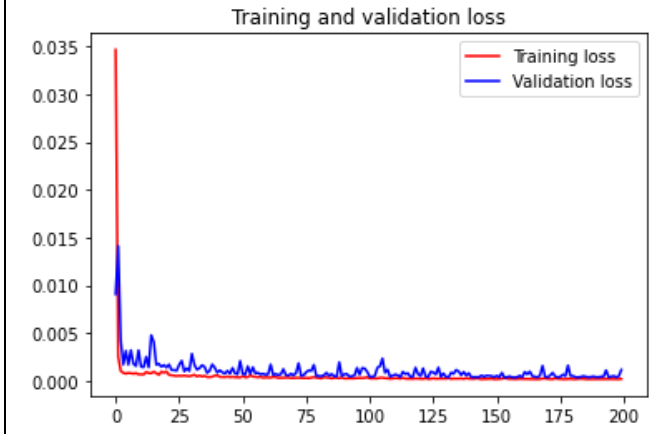
The LSTM model architecture that used in this section is 1 LSTM input layers, 1 LSTM hidden layer, 1 dense output layer with 256 neurons and 512 neurons. The default parameter of this architecture is mentioned previously. The number of epochs that was tuned in this section included 10 epochs, 20 epochs, 50 epochs, 100 epochs, 150 epochs and 200 epochs. The figure below showed the sample coding of 10 epochs. The units that were circled indicate the parameter that needs to be tuned:

```
# Initialising the RNN
lstm_model = Sequential()

# Adding the first LSTM layer
lstm_model.add(LSTM(units = 256, return_sequences = True, activation= 'relu', input_shape = (time_step, 1)))


# This layer produces a single output
lstm_model.add(LSTM(units = 512,activation= 'relu'))


# Adding the output layer
# This is the final layer produces the required numerical prediction
lstm_model.add(Dense(units = 1,activation= 'linear'))

# Compiling the RNN
lstm_model.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Fitting the RNN to the Training set
history = lstm_model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=10,batch_size=32,verbose=1)
```

Epochs	Training and validation loss	RMSE	R2 score	Comparison between original close price and predicted close price

10		Train data RMSE: 3.7420601261227757 Test data RMSE: 5.7425171588 63509	Train data R2 score: 0.9827362 08855 0267 Test data R2 score: 0.886 35382 68945 089	
20		Train data RMSE: 4.1256529395 27955 Test data RMSE: 6.5643046478 48962	Train data R2 score: 0.9790154 31994 1959 Test data R2 score: 0.851 49958 42361 153	

50		Train data RMSE: 2.766439912491594 Test data RMSE: 4.344052190385812 5	Train data R2 score: 0.9905646691322 83 Test data R2 score: 0.9349660277284 042	
100		Train data RMSE: 2.766439912491594 Test data RMSE: 4.344052190385812 5	Train data R2 score: 0.9905646691322 83 Test data R2 score: 0.9349660277284 042	

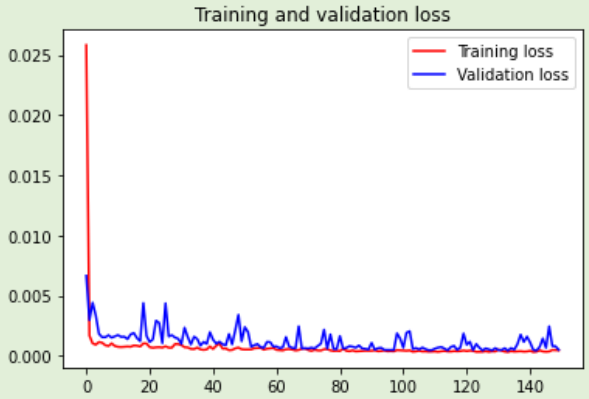
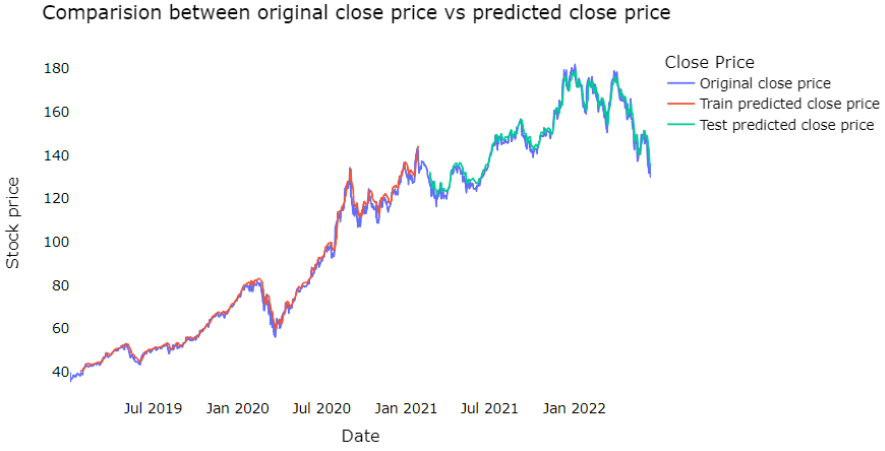
150		Train data RMSE: 2.4150147507548651 Test data RMSE: 3.7861470755806454	Train data R2 score: 0.9928095807959736 Test data R2 score: 0.9505979269266811	
200		Train data RMSE: 2.4519393080747673 Test data RMSE: 5.0017901635795 89	Train data R2 score: 0.9925880226617806 Test data R2 score: 0.9137813637588637	

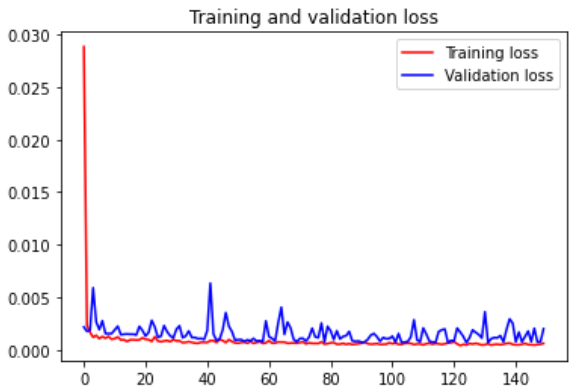
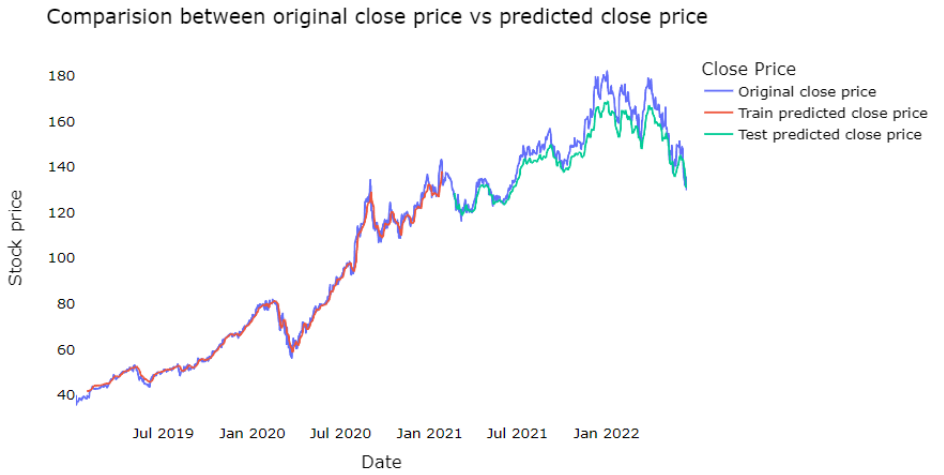
In this section, the best epoch that achieved the highest performance is 150 epochs with 0.95 R-squared score and only 3.79 RMSE. In the next session, the architecture that continues the dropout rate is 1 LSTM input layer, 1 LSTM hidden layer, 1 dense output layer with 256 and 512 units, with 150 epochs and other default parameters that were mentioned previously.

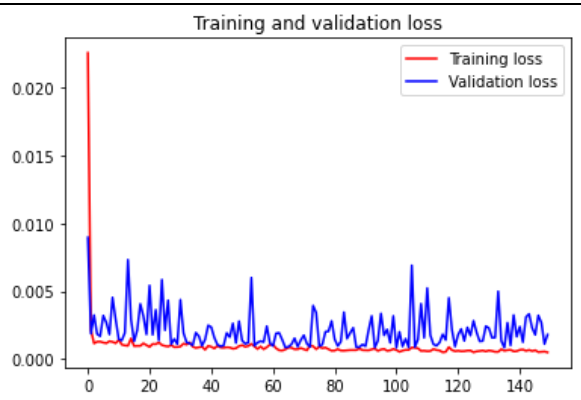
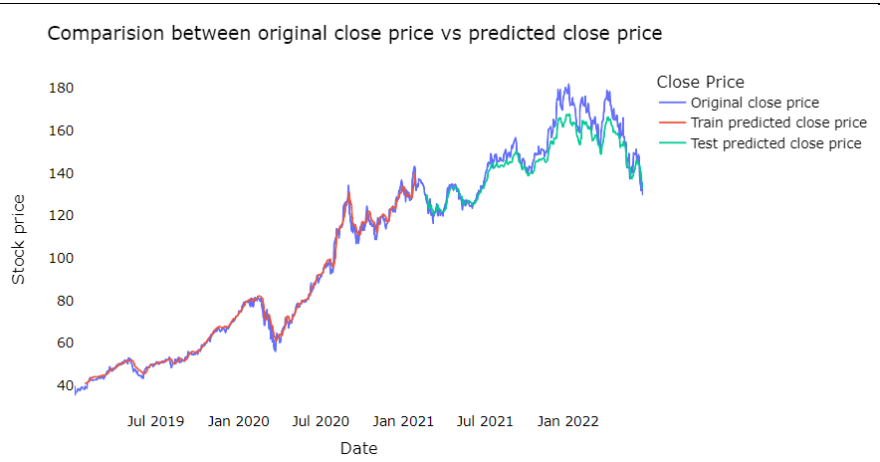
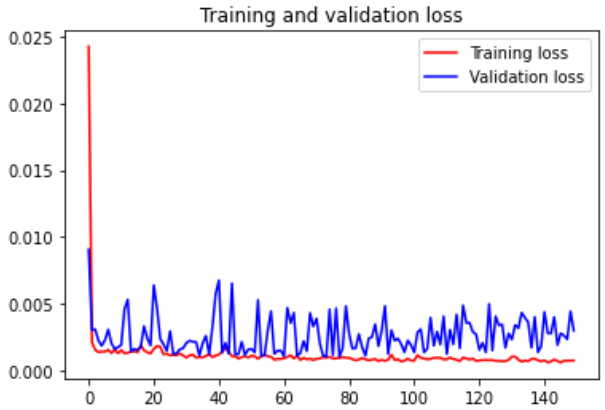
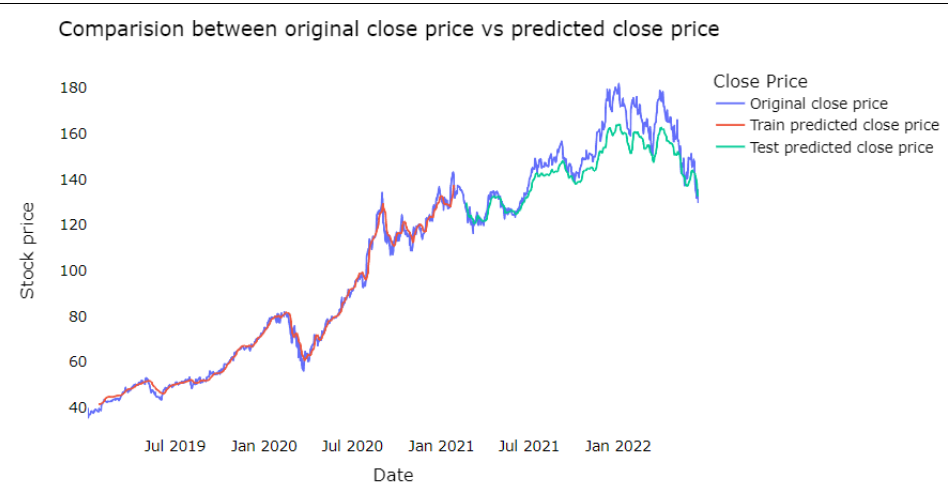
4.3.1.4 Dropout rate

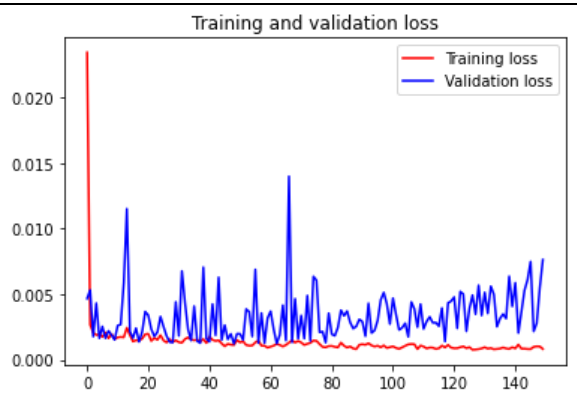
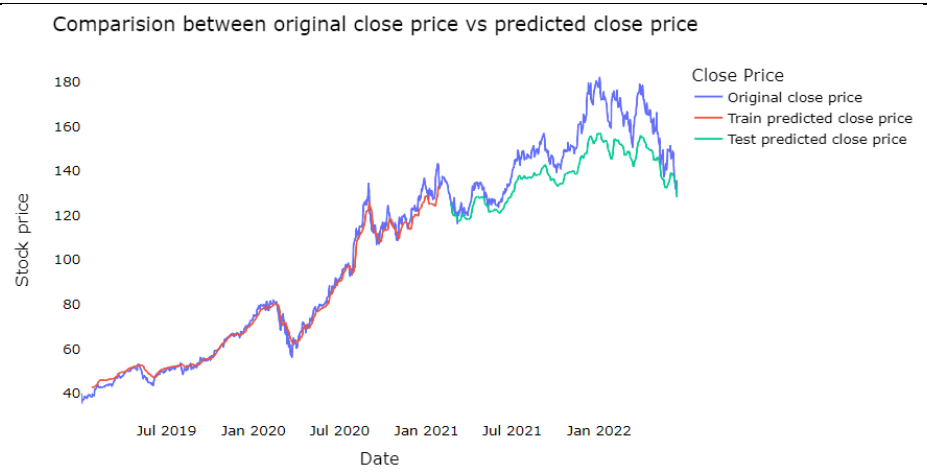
In this section, the architecture that is used to tune the dropout rate is the best model in the previous session which is 1 LSTM input layer, 1 LSTM hidden layer, 1 dense output layer with 256 and 512 units, with 150 epochs and other default parameters that was mentioned previously. Dropout is a regularization function that is used to regularize the model. The dropout rate used to train in this session is 0.1, 0.2, 0.3, 0.4 and 0.5. The figure below showed the sample coding of a 0.1 dropout rate. The units that were circled indicate the parameter that needs to be tuned:

```
# Initialising the RNN
lstm_model = Sequential()

# Adding the first LSTM layer
lstm_model.add(LSTM(units = 256, return_sequences = True, activation= 'relu', input_shape = (time_step, 1)))
lstm_model.add(Dropout(0.1))

# Adding a second LSTM layer and some Dropout regularisation
# This layer produces a single output
lstm_model.add(LSTM(units = 512,activation= 'relu'))
lstm_model.add(Dropout(0.1))

# Adding the output layer
# This is the final layer produces the required numerical prediction
lstm_model.add(Dense(units = 1,activation= 'linear'))

# Compiling the RNN
lstm_model.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Fitting the RNN to the Training set
history = lstm_model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=150,batch_size=32,verbose=1)
```

Dropout rate	Training and validation loss	RMSE	R2 score	Comparison between original close price and predicted close price
0.1		Train data RMSE: 2.47054306613721 67 Test data RMSE: 3.305133	Train data R2 score: 0.9924751212159914 Test data R2 sco	

		36892 0817	re: 0.96 2353 1892 7358 79	
0.2		Train data RMS E: 2.432 36909 77394 61 Test data RMSE: 6.5 26864 37444 6574	Trai n da ta R 2 sc ore: 0.9 9270 5868 2212 381 Test dat a R2 sco re: 0.85 3188 7320 4611 38	

0.3		Train data RMSE: 2.3995388968 99571 Test data RMSE: 6.2047179115 24347	Train data R2 score: 0.9929014 40297 5228 Test data R2 score: 0.867 32341 14160 673	
0.4		Train data RMSE: 2.5514031175 579004 Test data RMSE: 7.9989115835 17126	Train data R2 score: 0.9919744 86780 9556 Test data R2 score: 0.779 49826 78130 435	

0.5		Train data RMSE: 3.3003008539 26265 Test data RMSE: 12.7863350279 50047	Train data R2 score: 0.9865716 790040581 Test data R2 score: 0.43656688 02066146	

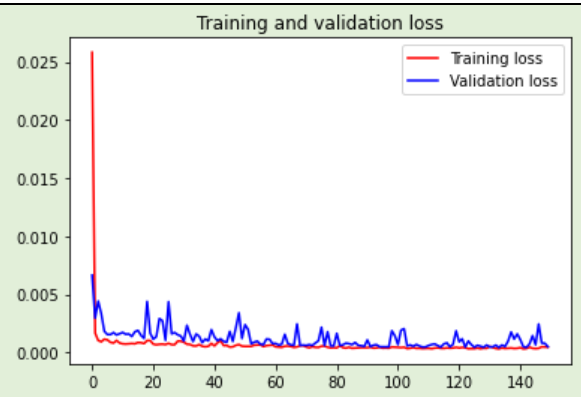
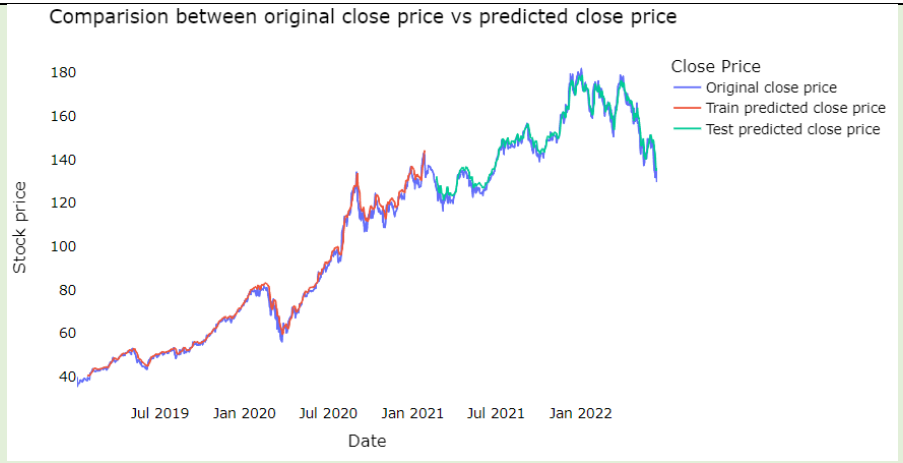
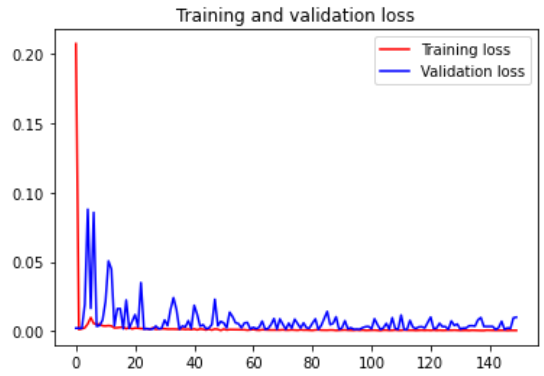
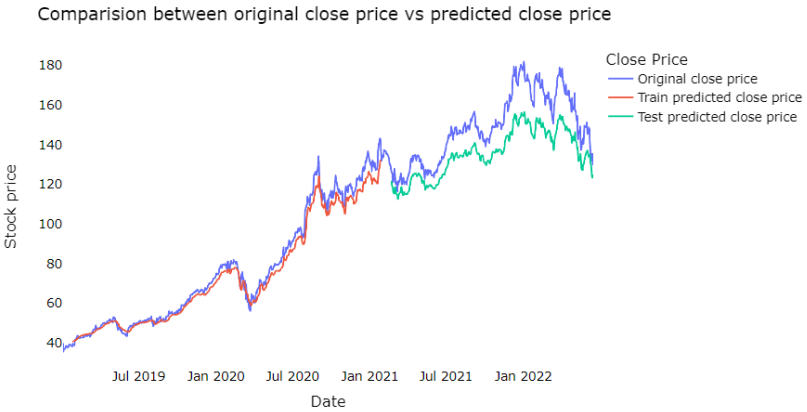
Based on the result above, it showed that the best model is using the 0.1 dropout rate. Hence, this model is then used in this next section to choose the most suitable optimizer.

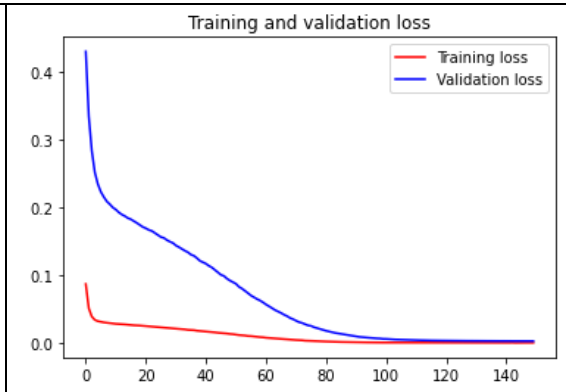
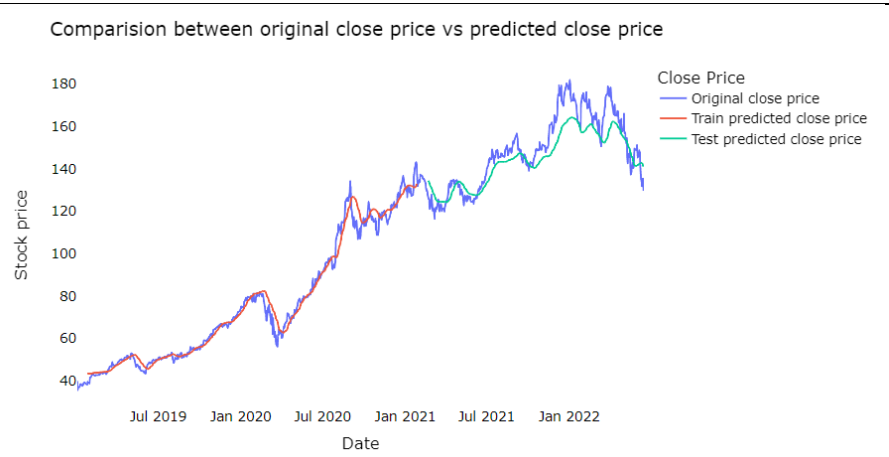
4.3.1.5 Optimizer

In this session, the model architecture that was used is the best model in the previous session. The optimizer that are tuned included adam optimizer, RMSprop and SGD optimizer. The figure below showed the sample coding of Adam optimizer. The units that were circled indicate the parameter that needs to be tuned:

```
# Initialising the RNN
lstm_model = Sequential()

# Adding the first LSTM layer
lstm_model.add(LSTM(units = 256, return_sequences = True, activation= 'relu', input_shape = (time_step, 1)))
lstm_model.add(Dropout(0.1))

# Adding a second LSTM layer and some Dropout regularisation
# This layer produces a single output
lstm_model.add(LSTM(units = 512,activation= 'relu'))
lstm_model.add(Dropout(0.1))

# Adding the output layer
# This is the final layer produces the required numerical prediction
lstm_model.add(Dense(units = 1,activation= 'linear'))

# Compiling the RNN
lstm_model.compile(optimizer = 'adam', loss = 'mean_squared_error')

# Fitting the RNN to the Training set
history = lstm_model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=150,batch_size=32,verbose=1)
```

Optimizer	Training and validation loss	RMSE	R2 score	Comparison between original close price and predicted close price

Adam		Train data RMSE: 2.4705430661372167 Test data RMSE: 3.3051333689208817	Train data R2 score: 0.9924751212159914 Test data R2 score: 0.9623531892735879	
RMSprop		Train data RMSE: 4.212041890443418 Test data RMSE: 14.687730023229546	Train data R2 score: 0.978127419970 5463 Test data R2 score: 0.2565366314708 4416	

SGD		Train data RMSE: 3.870659569115344 Test data RMSE: 8.28557298026113	Train data R2 score: 0.9815292469096 21 Test data R2 score: 0.7634105869538816	

In the end, it was found the adam optimizer is the best model. In final, we can conclude that the best model is 1 LSTM input layers, 1 LSTM hidden layer, 1 dense output layer with adam optimizer, 150 epochs, 256 and 512 units, 0.1 dropout rate, relu activation function in the hidden layer and linear activation function in the output layer

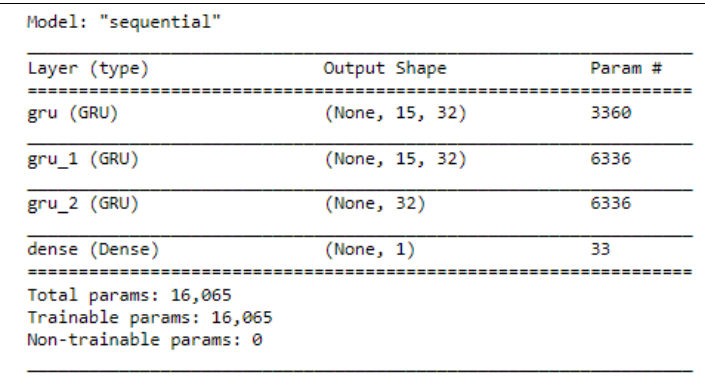
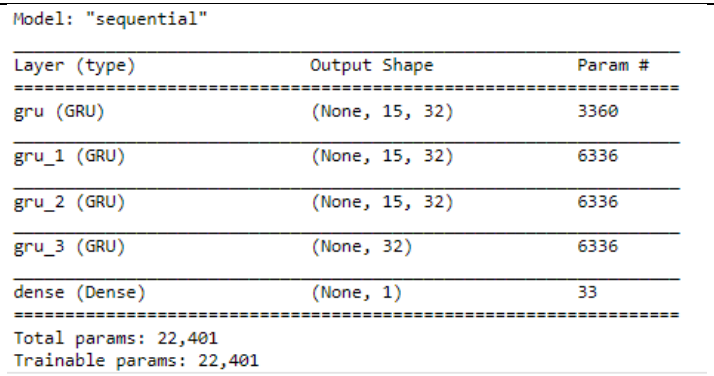
4.3.2 Gated Recurrent Unit (GRU)

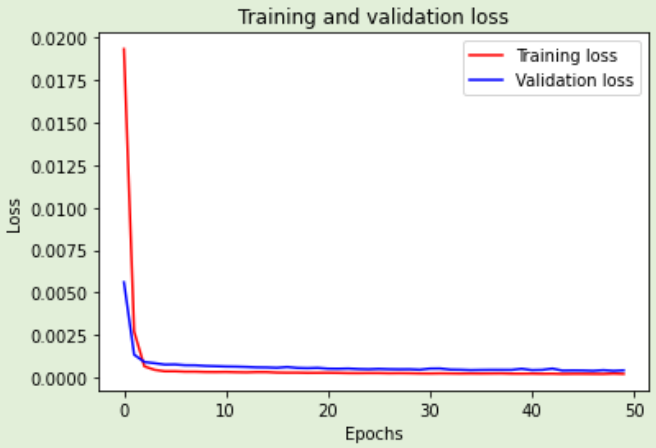
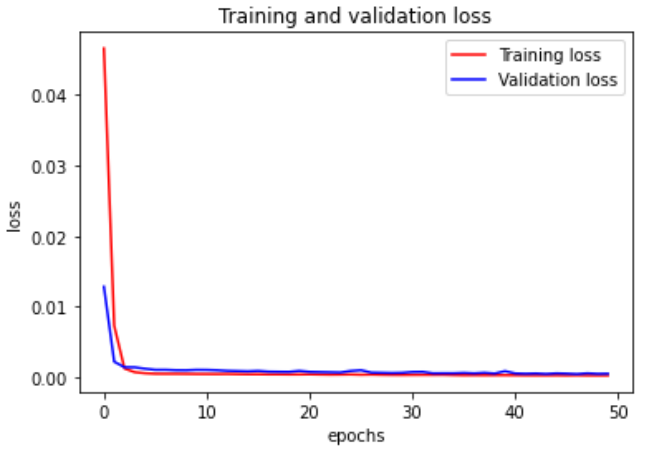
4.3.2.1 Tuning of number of GRU layers

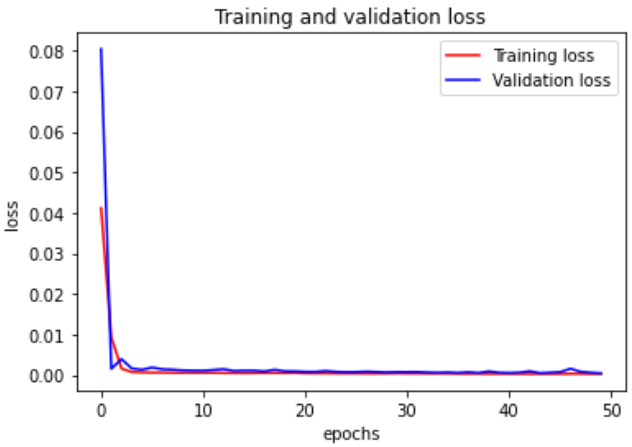
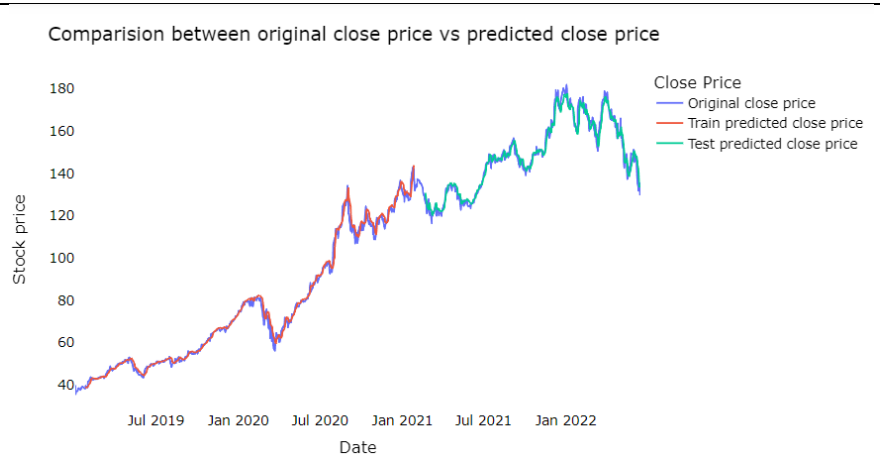
In this section, one, two and three GRU hidden layers are test. The best result architecture will be used in the next tuning session. The other default parameters included 50 epochs, 32 batch size, 32 units, adam optimizer, loss function using the mean square error, using relu activation function in the hidden layer and linear activation function in the output layer. The figure below showed the sample coding of 1 GRU input layer, 1 GRU hidden layer, 1 dense output layer model. The units that were circled indicate the parameter that needs to be tuned:

```
: tf.keras.backend.clear_session()
GRU_model=Sequential()
GRU_model.add(GRU(32,return_sequences=True,input_shape=(time_step,1),activation= 'relu'))
GRU_model.add(GRU(32,activation= 'relu'))
GRU_model.add(Dense(1,activation= 'linear'))
GRU_model.compile(loss='mean_squared_error',optimizer='adam')
history = GRU_model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=50,batch_size=32,verbose=1)
```

Number of GRU layers	Summary
1 GRU input layer, 1 GRU hidden layer, 1 dense output layer	Model: "sequential" _____ Layer (type) Output Shape Param # ===== gru (GRU) (None, 15, 32) 3360 _____ gru_1 (GRU) (None, 32) 6336 _____ dense (Dense) (None, 1) 33 ===== Total params: 9,729 Trainable params: 9,729 Non-trainable params: 0 _____

1 GRU input layer, 2 GRU hidden layer, 1 dense output layer	
1 GRU input layer, 3 GRU hidden layer, 1 dense output layer	

	Training and validation loss	RMSE	R score	Predicted close price versus original close price
1 GRU input layer, 1 GRU hidden layer, 1 dense output layer		Train data RMSE: 2.0928488418157185 Test data RMSE: 3.0138098121426395	Train data R2 score: 0.9946000420877088 Test data R2 score: 0.9686972928078185	
1 GRU input layer, 2 GRU hidden layer, 1		Train data RMSE: 2.2728925323897324 Test data RMSE: 3.288332255617528	Train data R2 score: 0.9936309823233126 Test data R2 score: 0.9627349594685956	

dense output layer				
1 GRU input layer, 3 GRU hidden layer, 1 dense output layer		Train data RMSE: 2.294350299561332 Test data RMSE: 3.055568530414 73	Train data R2 score: 0.993510158303049 Test data R2 score: 0.9678238357696901	

The result showed 1 GRU input layer, 1 GRU hidden layer, 1 dense output layer has the highest performance, which is 3.0138098121426395 test RMSE data and 0.968697292878185 R2 score. Hence, this architecture of model is used in the following tuning session.

4.3.2.2 Number of neurons

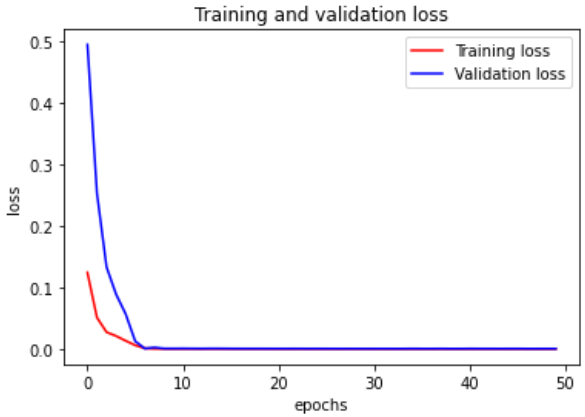
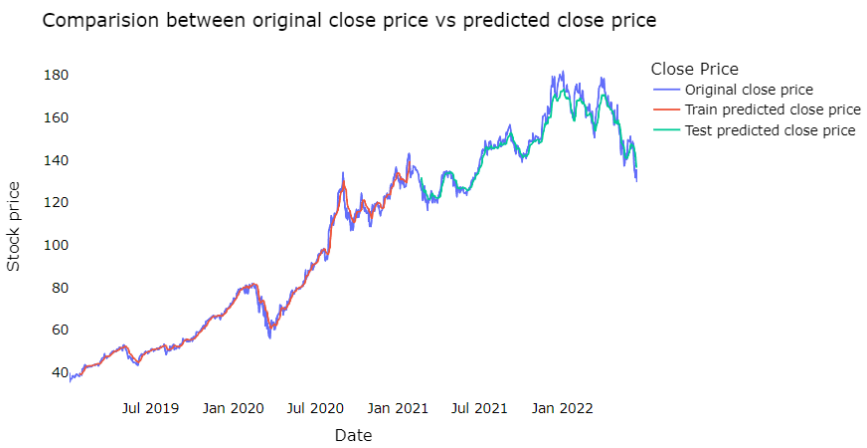
In this session, by using the previous best model that was filtered, the model continues the tuning process with 4 with 16 neurons, 16 with 32 neurons, 32 with 64 neurons, 64 with 128 neurons, 128 with 256 neurons, and 256 neurons with 512 neurons. The figure below showed the sample coding of 1 GRU input layer, 1 GRU hidden layer, 1 dense output layer model with 4 and 16 units. The units that were circled indicate the parameter that needs to be tuned:

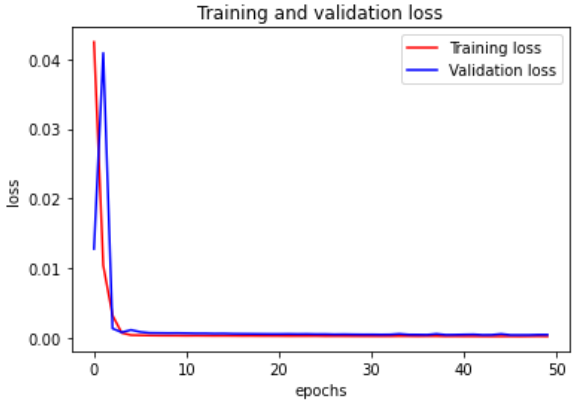
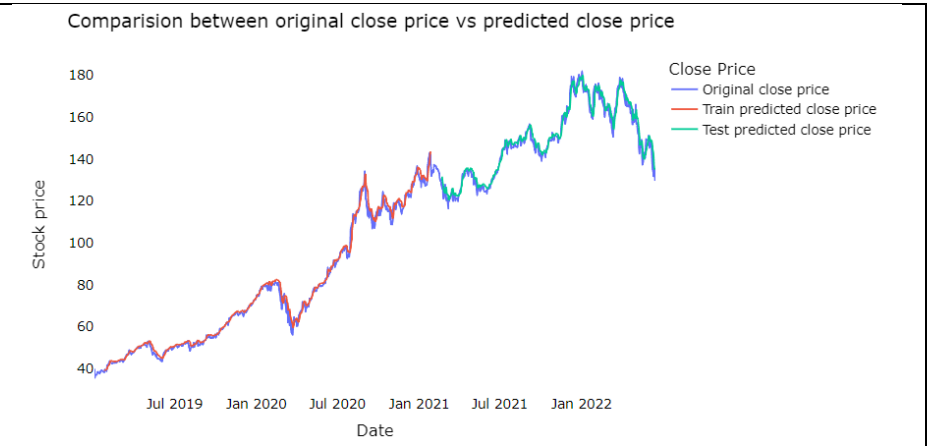
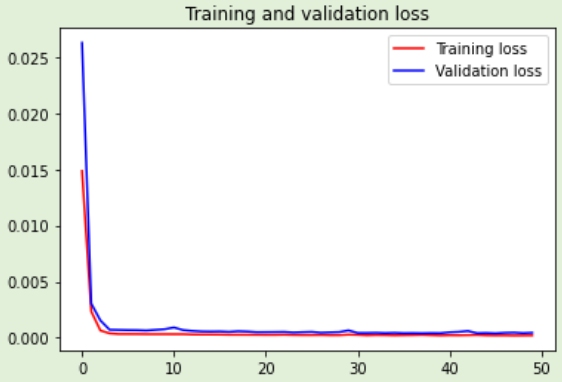
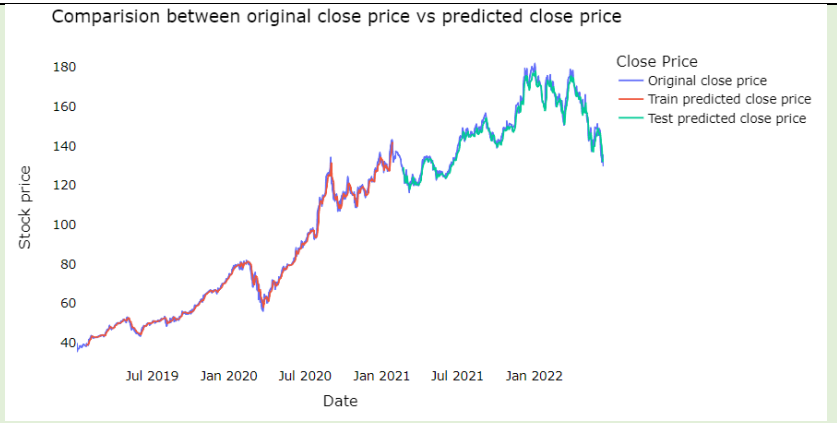
```
tf.keras.backend.clear_session()
GRU_model=Sequential()
GRU_model.add(GRU(4,return_sequences=True,,activation= 'relu',input_shape=(time_step,1)))
GRU_model.add(GRU(16,activation= 'relu'))
GRU_model.add(Dense(1,activation= 'linear'))
GRU_model.compile(loss='mean_squared_error',optimizer='adam')
history = GRU_model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=50,batch_size=32,verbose=1)
```

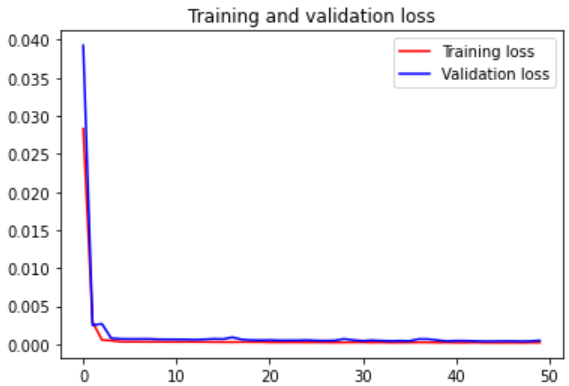
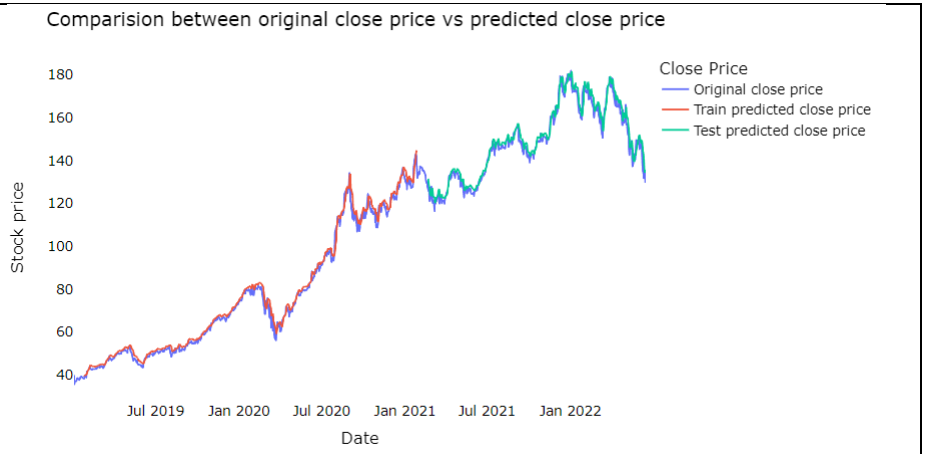
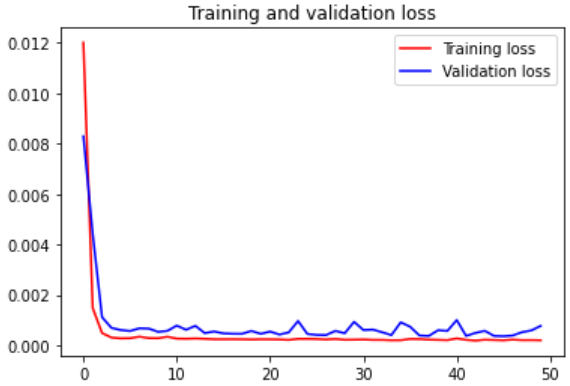
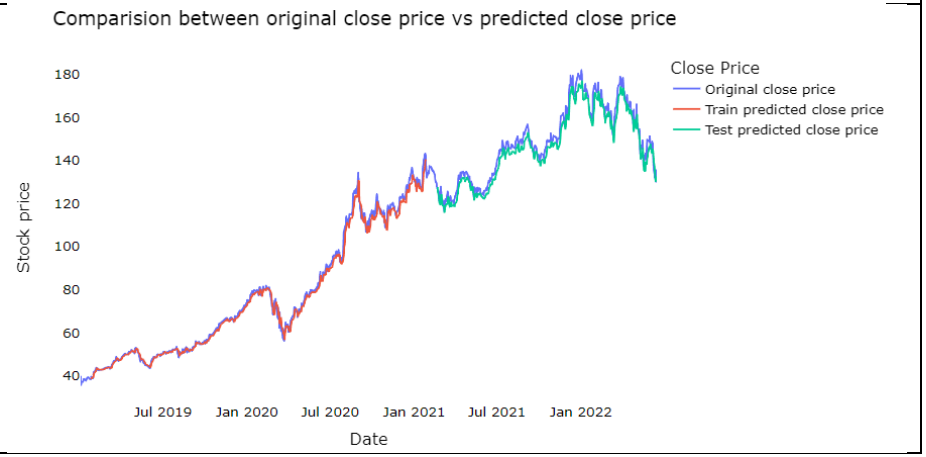
Number of neurons	Model Summary
4 neurons and 16 neurons	Model: "sequential" _____ Layer (type) Output Shape Param # ===== gru (GRU) (None, 15, 4) 84 _____ gru_1 (GRU) (None, 16) 1056 _____ dense (Dense) (None, 1) 17 ===== Total params: 1,157 Trainable params: 1,157 Non-trainable params: 0 _____
16 neurons and 32 neurons	Model: "sequential" _____ Layer (type) Output Shape Param # ===== gru (GRU) (None, 15, 16) 912 _____ gru_1 (GRU) (None, 32) 4800 _____ dense (Dense) (None, 1) 33 ===== Total params: 5,745 Trainable params: 5,745 Non-trainable params: 0

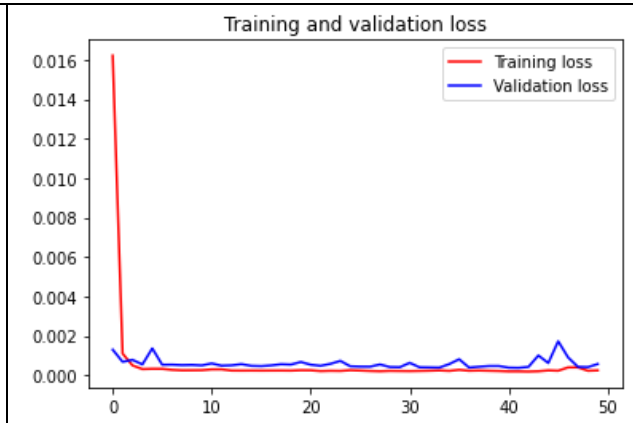
32 neurons and 64 neurons		Model: "sequential" _____ Layer (type) Output Shape Param # ===== gru (GRU) (None, 15, 32) 3360 _____ gru_1 (GRU) (None, 64) 18816 _____ dense (Dense) (None, 1) 65 ===== Total params: 22,241 Trainable params: 22,241 Non-trainable params: 0 _____
64 neurons and 128 neurons		Model: "sequential" _____ Layer (type) Output Shape Param # ===== gru (GRU) (None, 15, 64) 12864 _____ gru_1 (GRU) (None, 128) 74496 _____ dense (Dense) (None, 1) 129 ===== Total params: 87,489 Trainable params: 87,489 Non-trainable params: 0 _____
128 neurons and 256 neurons		Model: "sequential" _____ Layer (type) Output Shape Param # ===== gru (GRU) (None, 15, 128) 50304 _____ gru_1 (GRU) (None, 256) 296448 _____ dense (Dense) (None, 1) 257 ===== Total params: 347,009 Trainable params: 347,009 Non-trainable params: 0 _____

256 neurons and 512 neurons	Model: "sequential" _____ Layer (type) Output Shape Param # ===== gru (GRU) (None, 15, 256) 198912 _____ gru_1 (GRU) (None, 512) 1182720 _____ dense (Dense) (None, 1) 513 ===== Total params: 1,382,145 Trainable params: 1,382,145 Non-trainable params: 0 _____

Number of neurons	Training and validation loss	RMSE	R2 score	Predicted close price versus original close price
4 neurons and 16 neurons		Train data RMSE: 2.4936545233964953 Test data RMSE: 4.1070673763 3711	Train data R2 score: 0.9923336 750955473 Test data R2 score: 0.941 86818 69954 683	

16 neurons and 32 neurons		Train data RMSE: 2.261060106643063 Test data RMSE: 3.132117128924061 7	Train data R2 score: 0.9936971225101983 Test data R2 score: 0.96619147668842 14	
32 neurons and 64 neurons		Train data RMSE: 2.0913795861354014 Test data RMSE: 3.1173958164213197	Train data R2 score: 0.9946076213583456 Test data R2 score: 0.9665085377554311	

64 neurons and 128 neurons		Train data RMSE: 2.390253254502861 4 Test data RMSE: 3.26284606463928	Train data R2 score: 0.9929562734573159 Test data R2 score: 0.9633103657080 62	
128 neurons and 256 neurons		Train data RMSE: 2.336058988874579 5 Test data RMSE: 4.07175455462977	Train data R2 score: 0.9932720576570119 Test data R2 score: 0.9428635314184757	

256 neurons and 512 neurons		Train data RMSE: 2.25203297 6952377 Test data RMSE: 3.50056930 75394557	Train data R2 score: 0.99374734 96703333 Test data R2 score: 0.95776936 86552701	

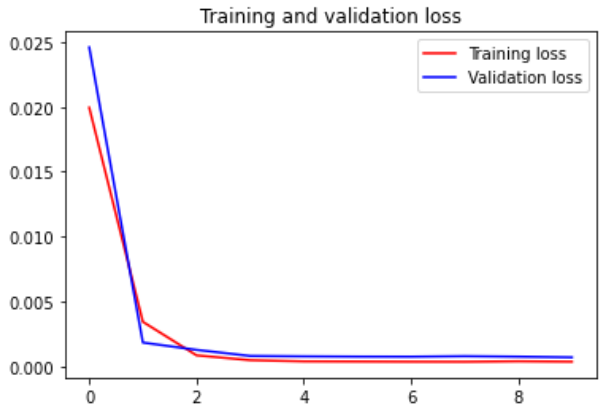
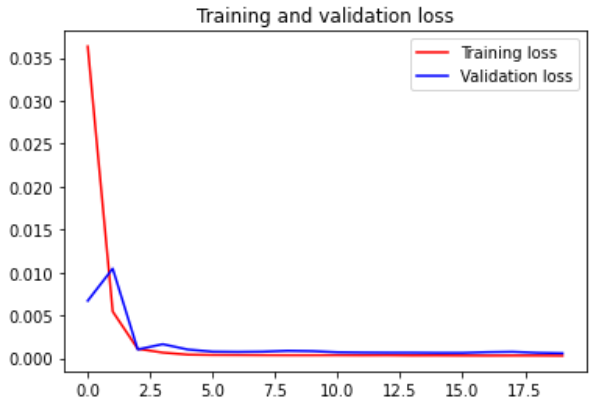
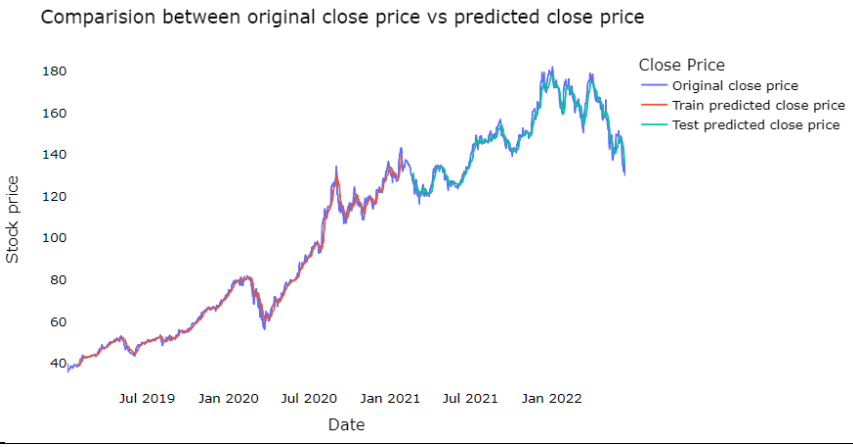
Based on the result above, it was found that 32 and 64 neurons model is the best model with minimum validation loss, only 3.117395816421319 RMSE and 0.9665085377554311 R^2 score. This model is then used in the next tuning session.

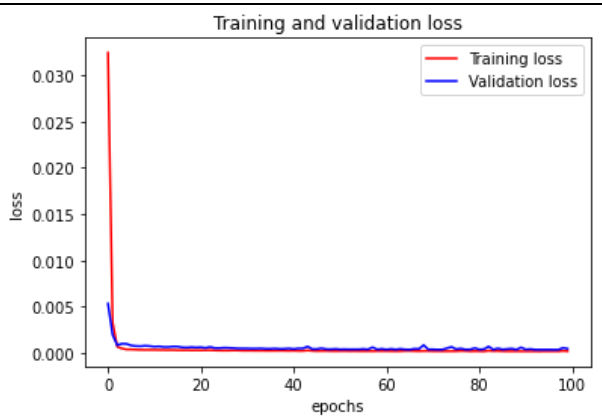
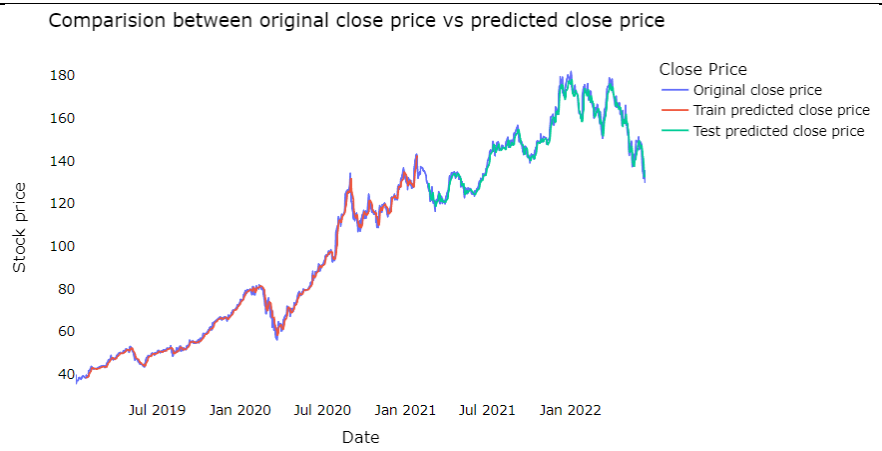
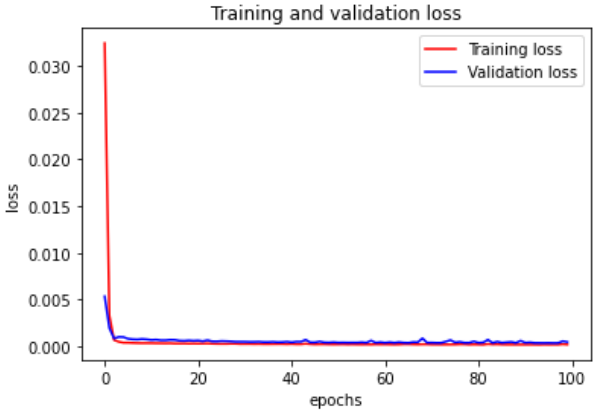
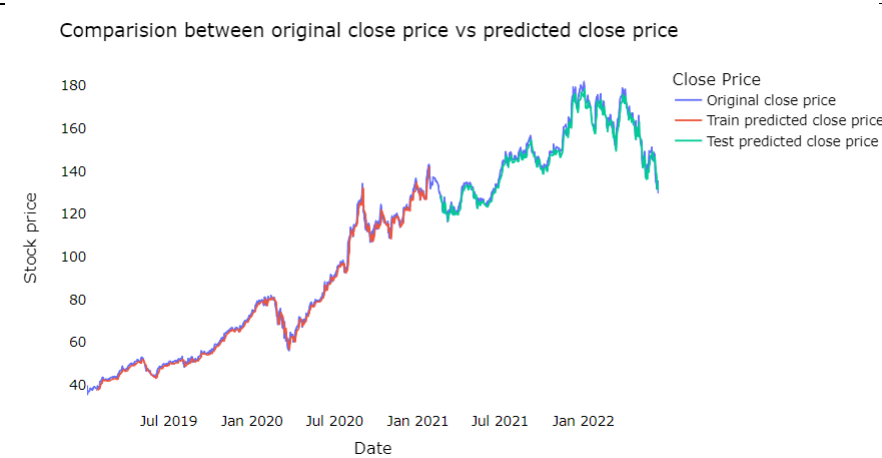
4.3.2.3 Number of epochs

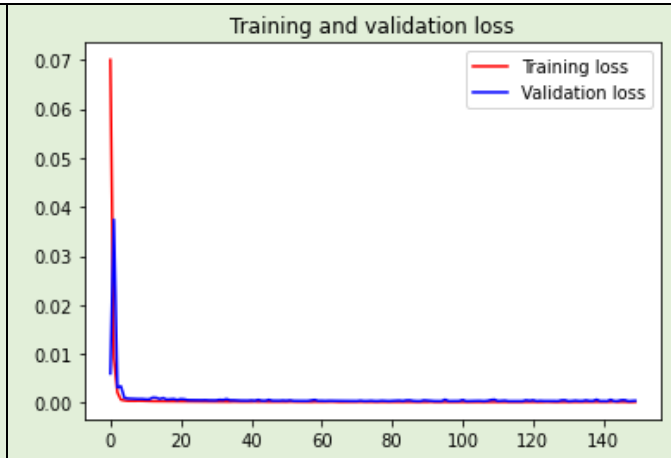
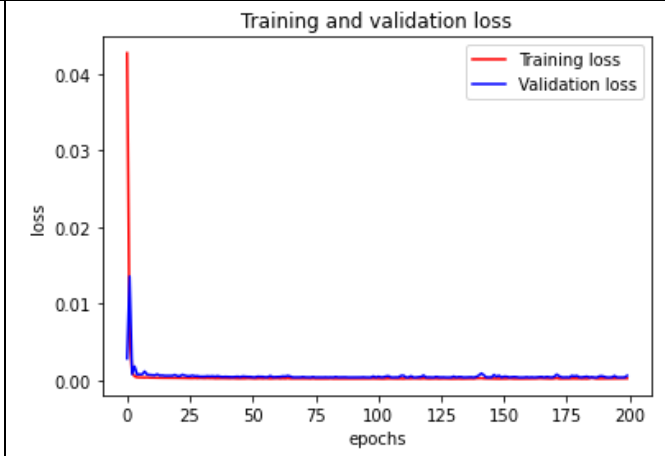
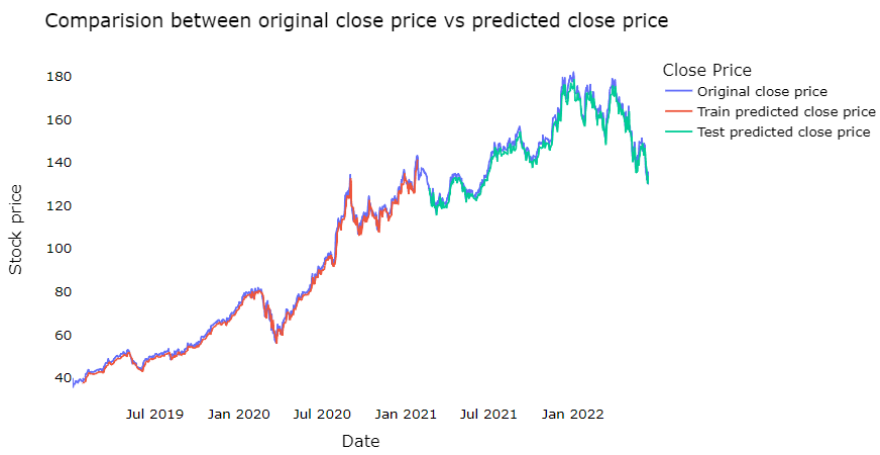
In this session, the best model previous is then used to tune the number of epochs. The number of epochs that are tuned here included 10 epochs, 20 epochs, 50 epochs, 100 epochs and 150 epochs. The figure below showed the sample coding of the GRU model with 10 epochs. The units that were circled indicate the parameter that needs to be tuned:

10 epochs

```
tf.keras.backend.clear_session()
GRU_model=Sequential()
GRU_model.add(GRU(32,return_sequences=True,,activation= 'relu',input_shape=(time_step,1)))
GRU_model.add(GRU(64,activation= 'relu'))
GRU_model.add(Dense(1,activation= 'linear'))
GRU_model.compile(loss='mean_squared_error',optimizer='adam')
history = GRU_model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=10,batch_size=32,verbose=1)
```

Epochs	Training and validation loss	RMSE	R2 score	Comparison between original close price and predicted close price
10		Train data RMSE: 2.629953797992049 3 Test data RMSE: 3.823028899321931	Train data R2 score: 0.991472712830 0233 Test data R2 score: 0.94963 0762690 334	
20		Train data RMSE: 2.496040275909735 Test data RMSE: 3.507621869175997 7	Train data R2 score: 0.992318998881 9785 Test data R2 score: 0.95759 9033989 092	

50		Train data RMSE: 2.12921467167444 5 Test data RMSE: 3.02852944 5707684	Train data R2 score: 0.99441 0766625 8121 Test data R2 score: 0.96839 0777465 4138	
100		Train data RMSE: 2.17205931493 40104 Test data RMSE: 3.17894706124 50508	Train data R2 score: 0.9941835 498217 059 Test data R2 score: 0.9651729 407898 597	

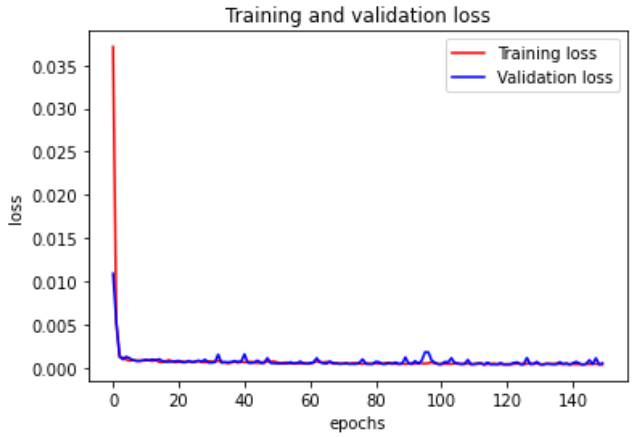
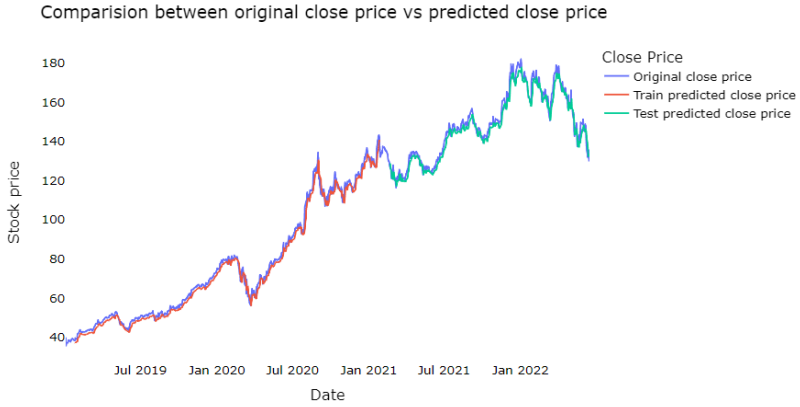
150		Train data RMSE: 2.041998085019022 Test data RMSE: 3.085893553873031 4	Train data R2 score: 0.9948592638793544 Test data R2 score: 0.9671820011578576	
200		Train data RMSE: 2.383787444600767 Test data RMSE: 3.556173800340273	Train data R2 score: 0.9929943295065078 Test data R2 score: 0.9564170955697796	

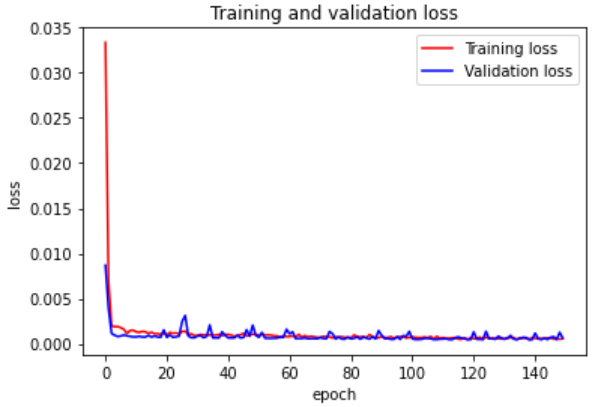
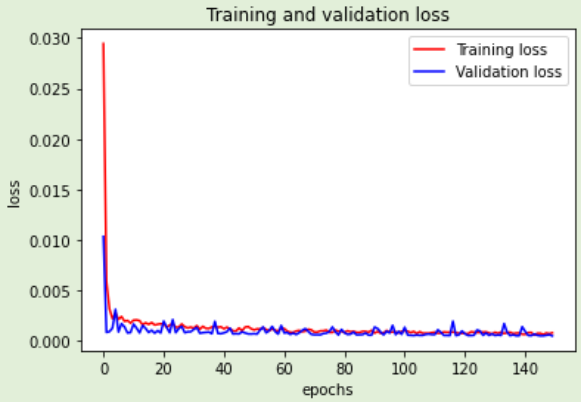
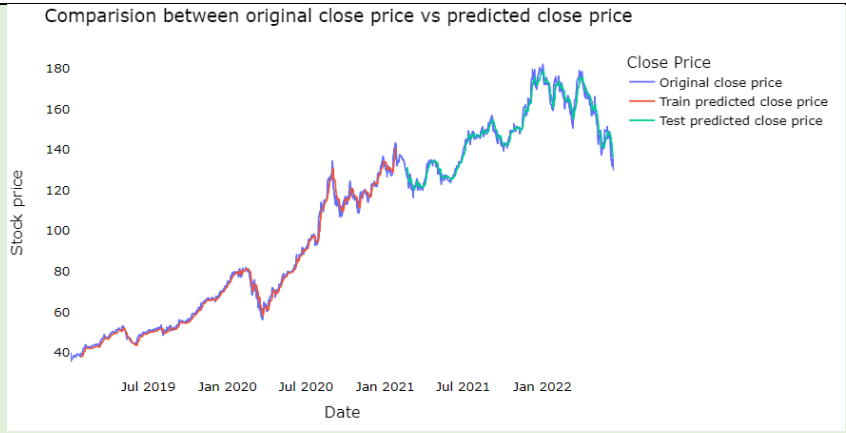
Based on the result, it was found the 150 epochs model achieved the highest performance, which is 0.97 R^2 score and only 3.1 RMSE with minimum validation loss. Hence, this model is then used in the next section.

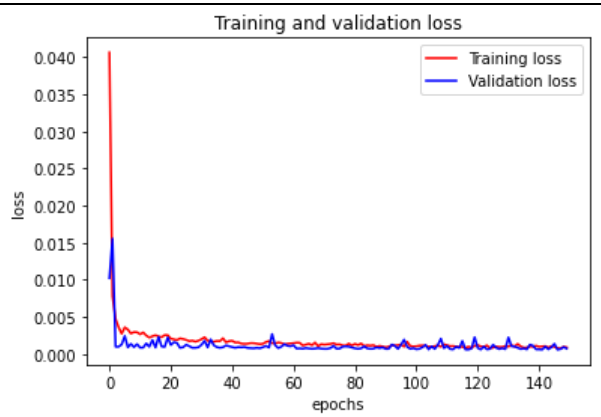
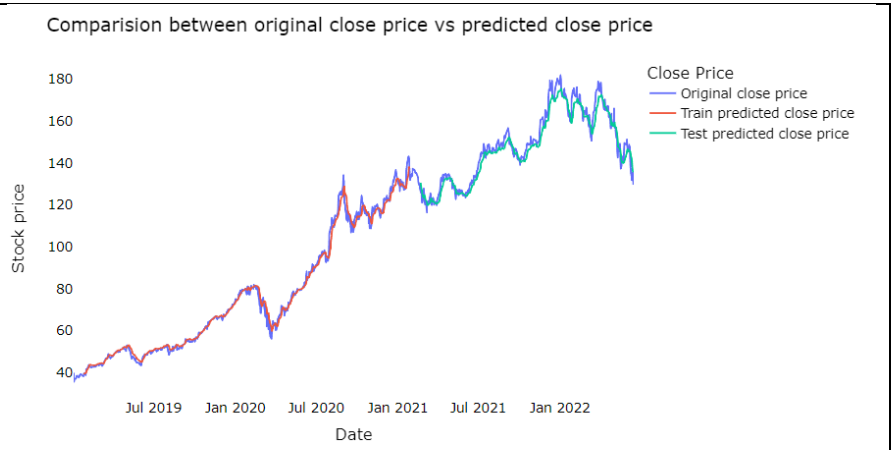
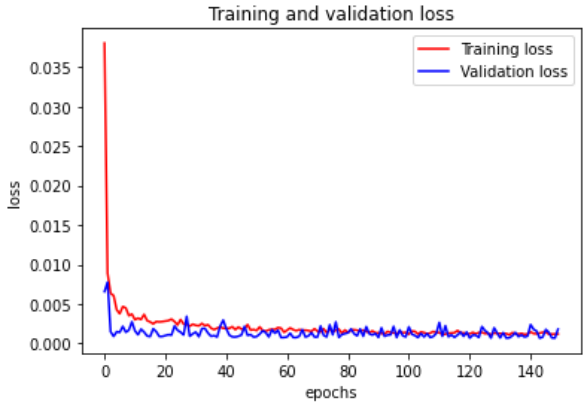
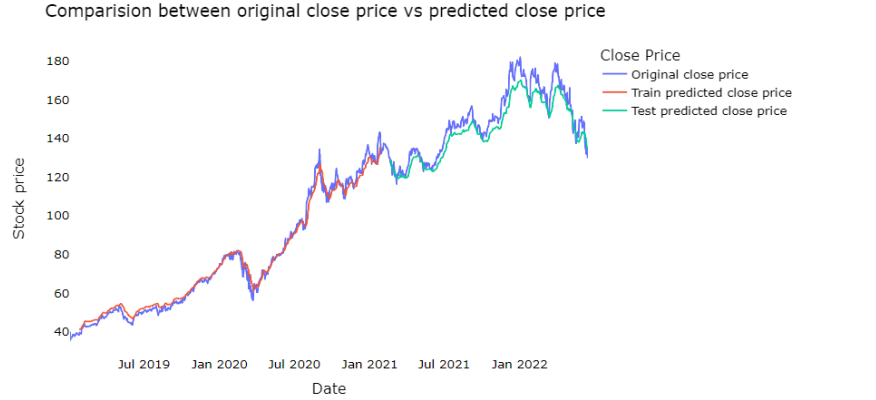
4.3.2.4 Dropout rate

In this section, the last best model in the previous session is used to add regularization and find out the best-fit dropout rate. The dropout rate that was tested included 0.1, 0.2,0.3,0.4 and 0.5. The figure below showed the sample coding of the GRU model with a 0.1 dropout rate. The parameter that was circled indicates the parameter that needs to be tuned:

```
tf.keras.backend.clear_session()
GRU_model=Sequential()
GRU_model.add(GRU(32,return_sequences=True,activation= 'relu',input_shape=(time_step,1)))
GRU_model.add(Dropout(0.1))
GRU_model.add(GRU(64,activation= 'relu'))
GRU_model.add(Dropout(0.1))
GRU_model.add(Dense(1,activation= 'linear'))
GRU_model.compile(loss='mean_squared_error',optimizer='adam')
history = GRU_model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=150,batch_size=32,verbose=1)
```

Dropout rate	Training and validation loss	RMSE	R2 score	Comparison between original close price and predicted close price
0.1		Train data RMSE: 2.5837378254340972 Test data RMSE: 3.395234624937455	Train data R2 score: 0.9917697782411286 Test data R2 score: 0.9602726318186583	

0.2		Train data RMSE: 2.38815723057930115 Test data RMSE: 3.653087365319889 6	Train data R2 score: 0.9929686213924868 Test data R2 score: 0.9540092671228754	
0.3		Train data RMSE: 2.283362651123300 5 Test data RMSE: 3.247065972465407	Train data R2 score: 0.9935721691927373 Test data R2 score: 0.9636643914722226	

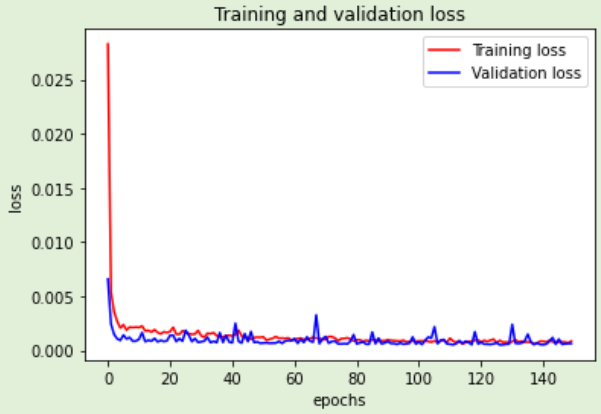
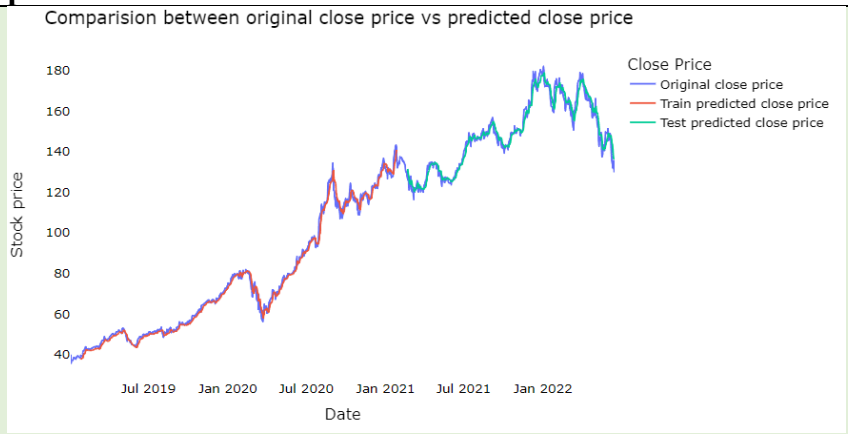
0.4		Train data RMSE: 2.41081788264593003 Test data RMSE: 3.94909701376847 3	Train data R2 score: 0.9928345501756385 Test data R2 score: 0.9462540407632118	
0.5		Train data RMSE: 3.04620592950498 15 Test data RMSE: 6.23327526 6403326	Train data R2 score: 0.9885598117995934 Test data R2 score: 0.8660993068846 05	

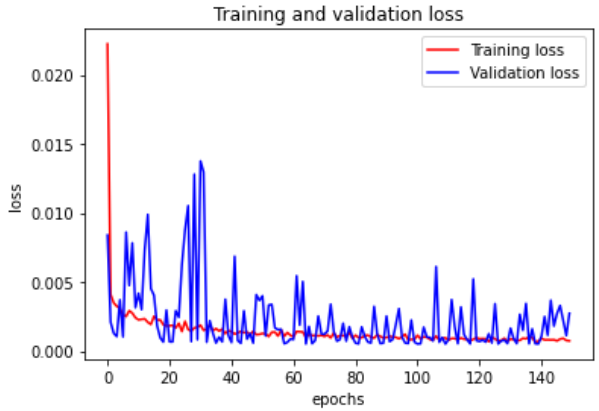
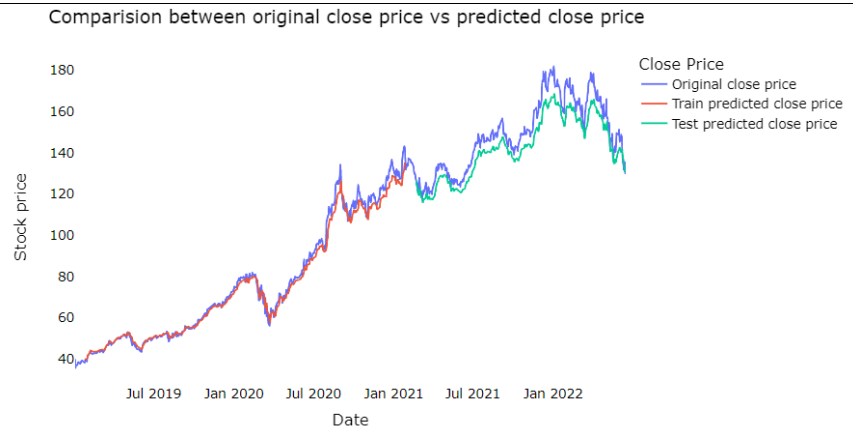
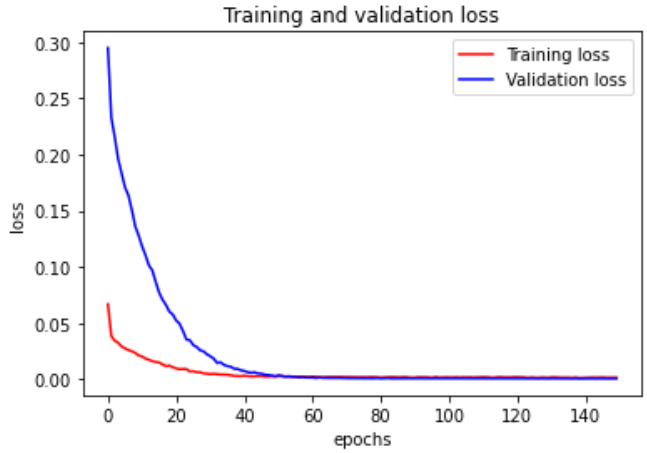
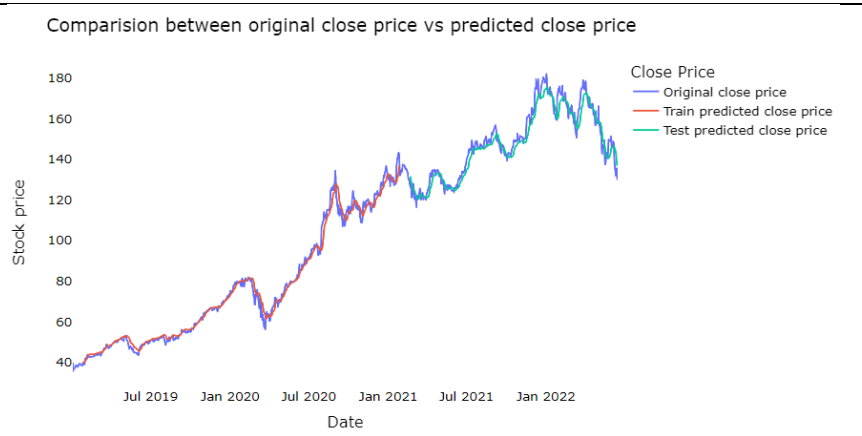
Based on the result, showed that a 0.3 dropout rate achieved the highest performance. Hence, this model is then used in the next session for optimizer selection.

4.3.2.5 Optimizer

By using the 0.3 dropout rate model in the previous session, the model is then used to select the best optimizer. The optimizer selection included adam, RMSprop, and SDG. The figure below showed the sample coding of the GRU model with the Adam optimizer. The parameter that was circled indicates the parameter that needs to be tuned:

```
tf.keras.backend.clear_session()
GRU_model=Sequential()
GRU_model.add(GRU(32,return_sequences=True,activation= 'relu',input_shape=(time_step,1)))
GRU_model.add(Dropout(0.3))
GRU_model.add(GRU(64,activation= 'relu'))
GRU_model.add(Dropout(0.3))
GRU_model.add(Dense(1,activation= 'linear'))
GRU_model.compile(loss='mean_squared_error',optimizer='adam')
history = GRU_model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=150,batch_size=32,verbose=1)
```

Dropout rate	Training and validation loss	RMSE	R2 score	Comparison between original close price and predicted close price
Adam		Train data RMSE: 2.28336265112330005 Test data RMSE: 3.247065972465407	Train data R2 score: 0.9935721691927373 Test data R2 score: 0.96366439147222226	

RMSprop		Train data RMSE: 3.10044418363730 05 Test data RMSE: 7.652207424899164 5	Train data R2 score: 0.9881487956894 69 Test data R2 score: 0.79819882977761 913	
SDG		Train data RMSE: 2.83704957022947 87 Test data RMSE: 4.363103929039976	Train data R2 score: 0.9900768741616861 Test data R2 score: 0.93439339698582	

Based on the result, it was found that adam optimizer achieved the highest performance with 3.67701699541243 RMSE and 0.95 R^2 scores. Hence, we can conclude that in the GRU model, the best model is 1 GRU input layer, 1 GRU hidden layer, 1 dense output layer with 32 and 64 units, with 150 epochs, 0.3 drop-out rate, and using adam optimizer.

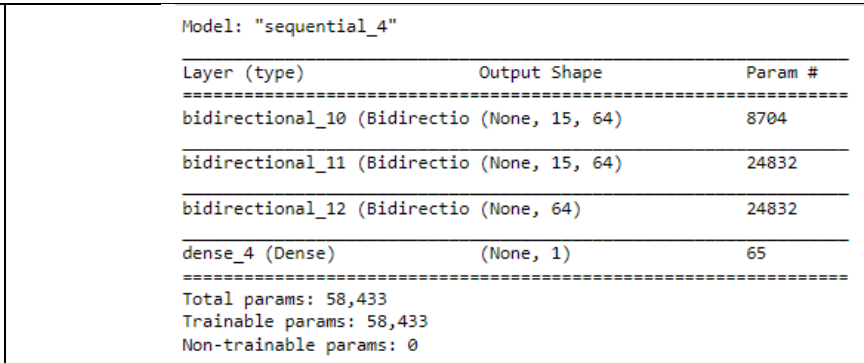
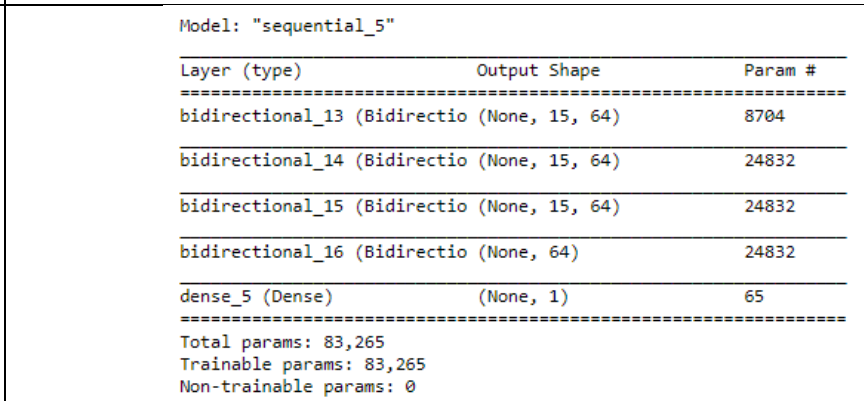
4.3.3 Bidirectional LSTM

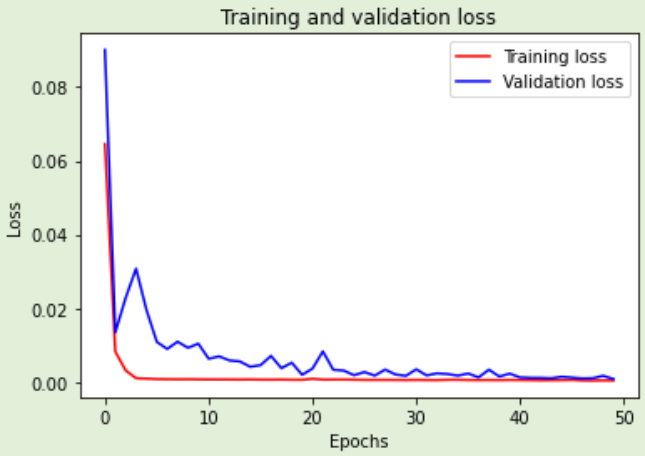
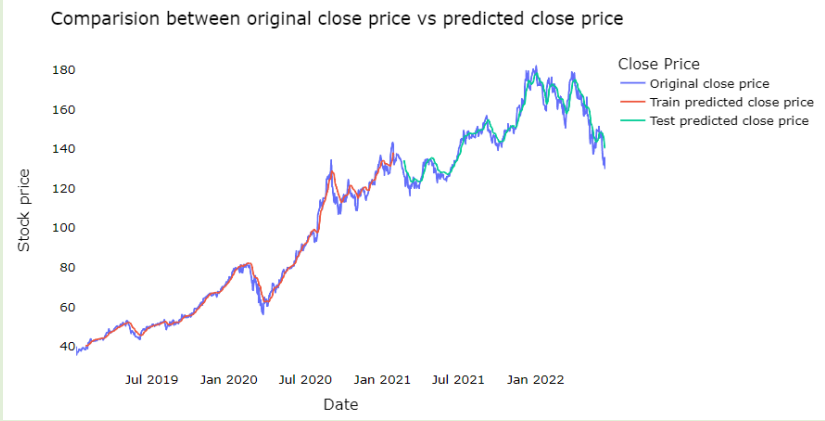
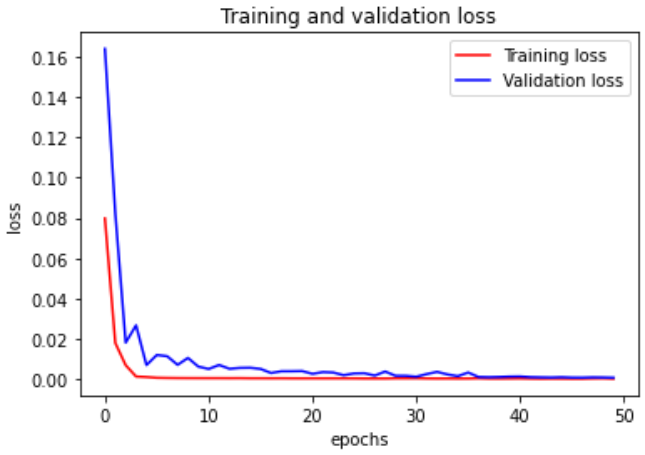
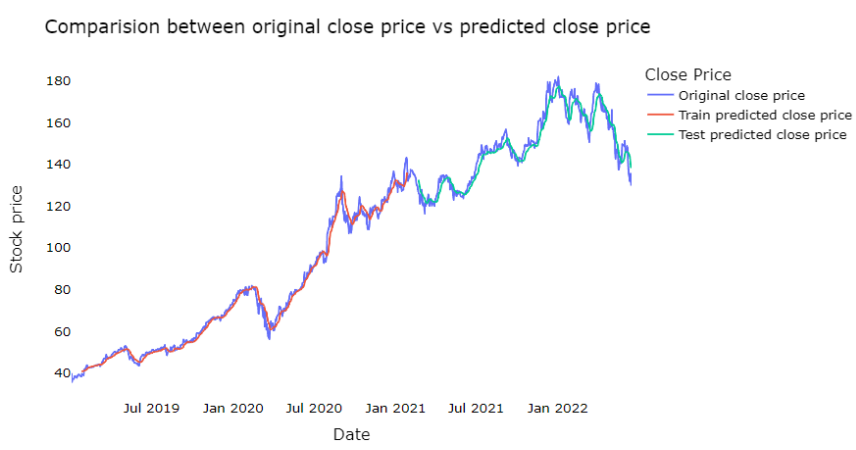
4.3.3.1 Tuning of the number of Bi-LSTM layers

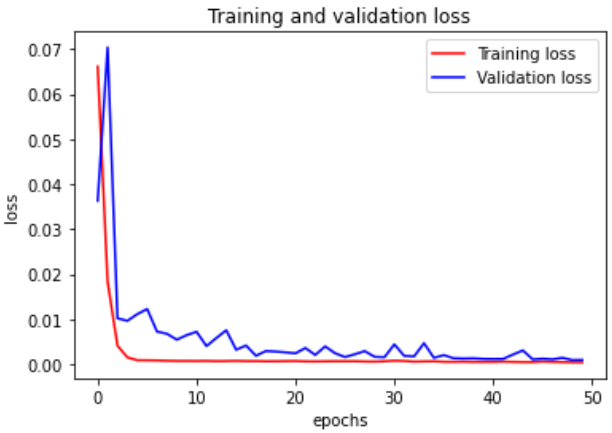
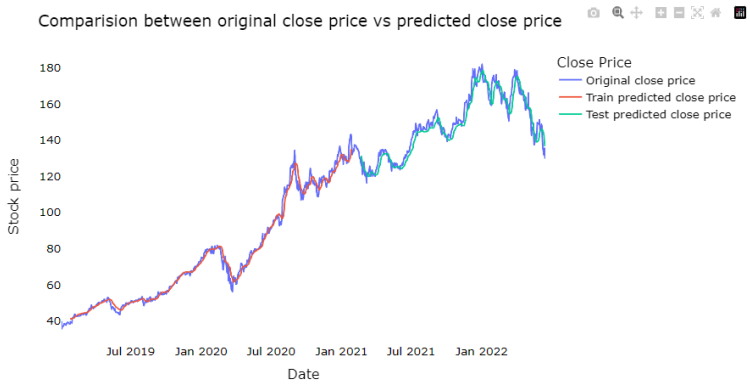
In this section, one, two and three bi-LSTM hidden layer are tested. The best result architecture will be used in the next tuning session. The other default parameters included 50 epochs, 32 batch size, 32 units, adam optimizer, loss function using the mean square error, using relu activation function in the hidden layer and linear activation function in the output layer. One of the sample codes of 1 bi-LSTM hidden layer is shown below:

```
model_biLSTM = Sequential()
model_biLSTM.add(Bidirectional(LSTM(units=32,return_sequences=True,activation= 'relu',input_shape=(time_step,1))))
model_biLSTM.add(Bidirectional(LSTM(units=32,activation= 'relu')))
model_biLSTM.add(Dense(units=1,activation= 'linear'))
model_biLSTM.compile(loss='mean_squared_error',optimizer='adam')
history = model_biLSTM.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=50,batch_size=32,verbose=1)
```

Number of BI-LSTM layers	Summary
1 bi-LSTM input layer, 1 bi-LSTM hidden layer, 1 dense output layer	Model: "sequential_1" _____ Layer (type) Output Shape Param # ===== bidirectional_2 (Bidirection (None, 15, 64) 8704 _____ bidirectional_3 (Bidirection (None, 64) 24832 _____ dense_1 (Dense) (None, 1) 65 ===== Total params: 33,601 Trainable params: 33,601 Non-trainable params: 0

1 bi-LSTM input layer, 2 bi-LSTM hidden layer, 1 dense output layer	
1 bi-LSTM input layer, 3 bi-LSTM hidden layer, 1 dense output layer	

	Training and validation loss	RMSE	R score	Predicted close price versus original close price
1 bi-LSTM input layer, 1 bi-LSTM hidden layer, 1 dense output layer		Train data RMSE: 2.943084091092149 Test data RMSE: 4.340630695292728	Train data R2 score: 0.9893212604847106 Test data R2 score: 0.9350684324778945	
1 bi-LSTM input layer, 2 bi-LSTM hidden layer, 1		Train data RMSE: 2.926248208682935 6 Test data RMSE: 4.492987027927521	Train data R2 score: 0.9894430862737233 Test data R2 score: 0.9304302340514259	

dense output layer				
1 bi-LSTM input layer, 3 bi-LSTM hidden layer, 1 dense output layer		Train data RMSE: 2.954968223648204 Test data RMSE: 4.588439414541919	Train data R2 score: 0.9892348451615551 Test data R2 score: 0.9274428498902333	

Based on the above result, it was found that 1 bi-LSTM input layer, 1 bi-LSTM hidden layer, 1 dense output layer have the best performance with only 4.340630695292728 RMSE and 0.9350684324778945 R2 score. This model is then used in the next neuron tuning section.

4.3.3.2 Number of neurons

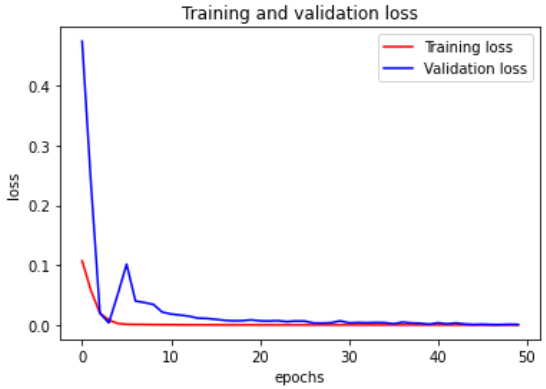
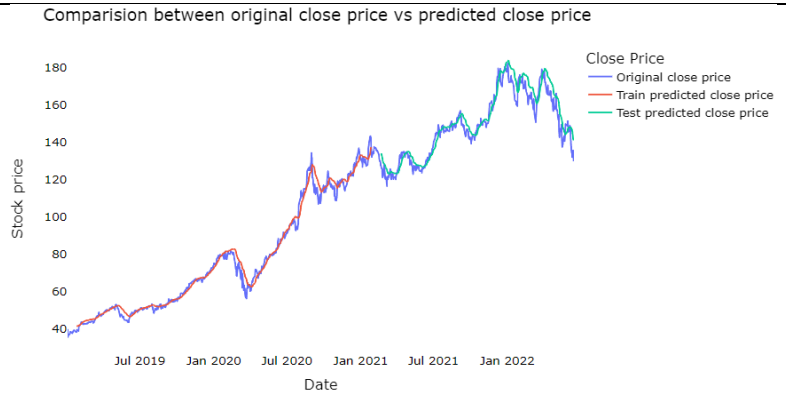
In this session, 1 bi-LSTM input layer, 1 bi-LSTM hidden layer, 1 dense output layer are used to understand the best number of neurons with the default parameters that were mentioned previously. The set of neurons number included 4 with 16 neurons, 16 with 32 neurons, 32 with 64 neurons, 64 with 128

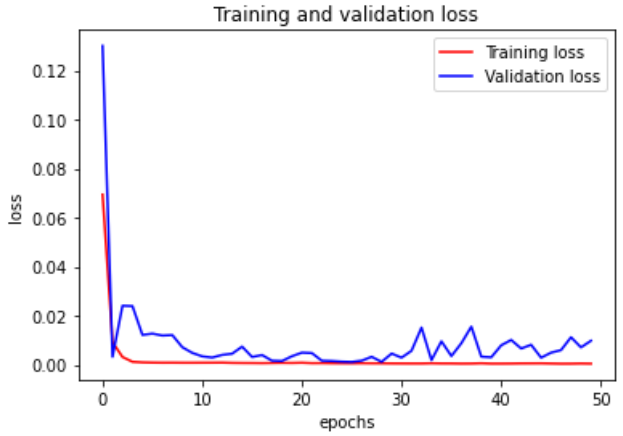
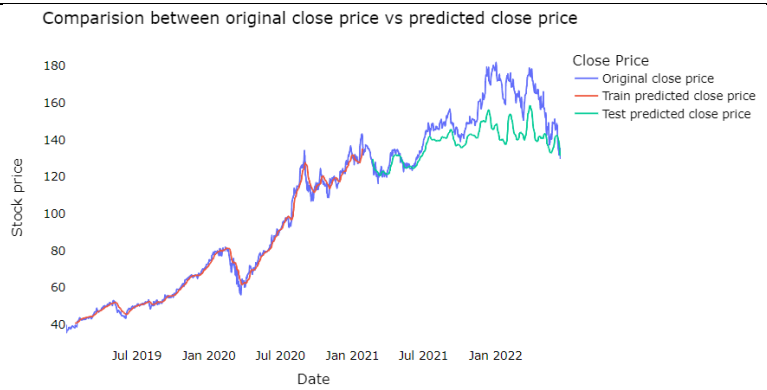
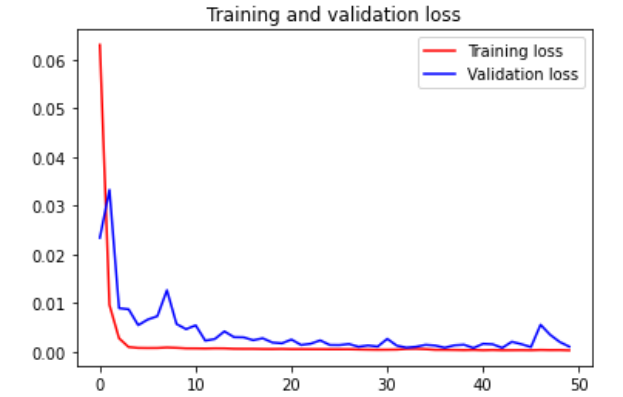
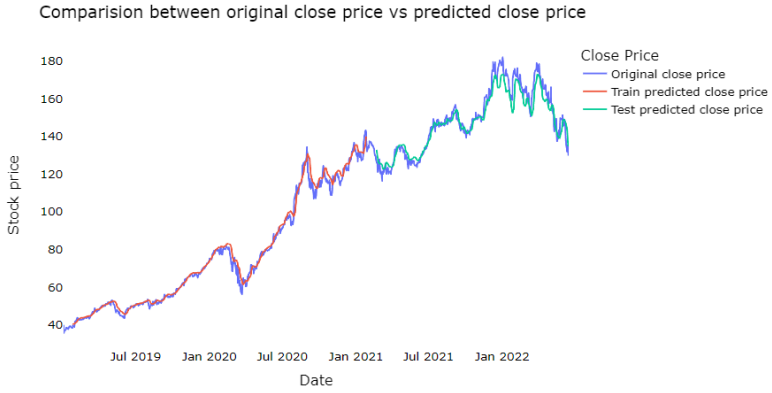
neurons, 128 with 256 neurons, and 256 with 512 neurons. The figure below showed the sample coding of 4 and 16 units. The units that were circled indicate the parameter that needs to be tuned:

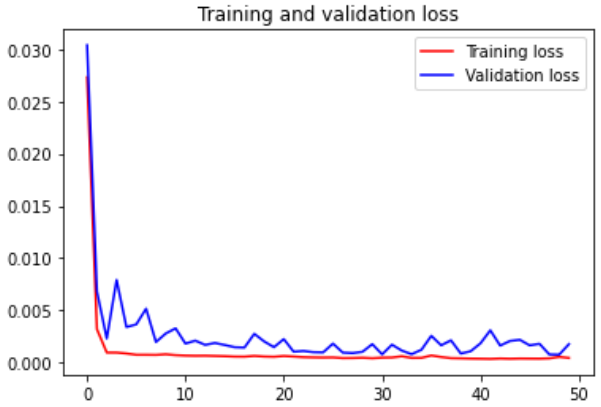
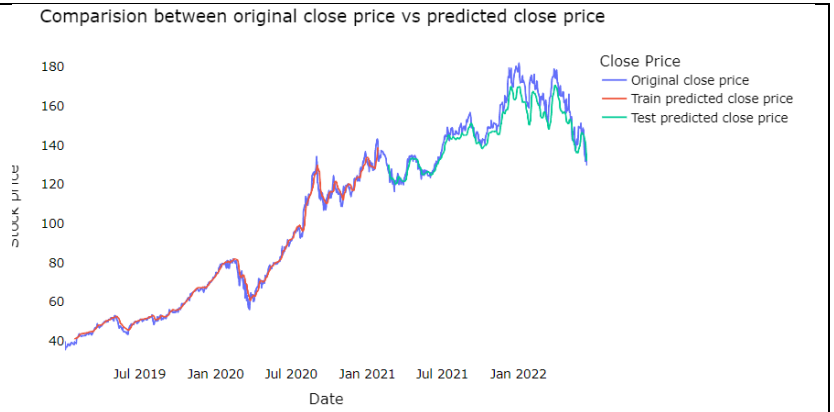
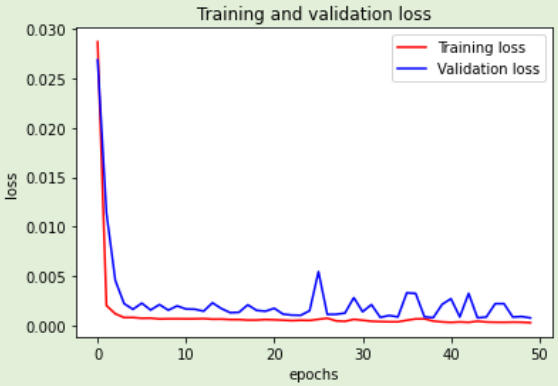
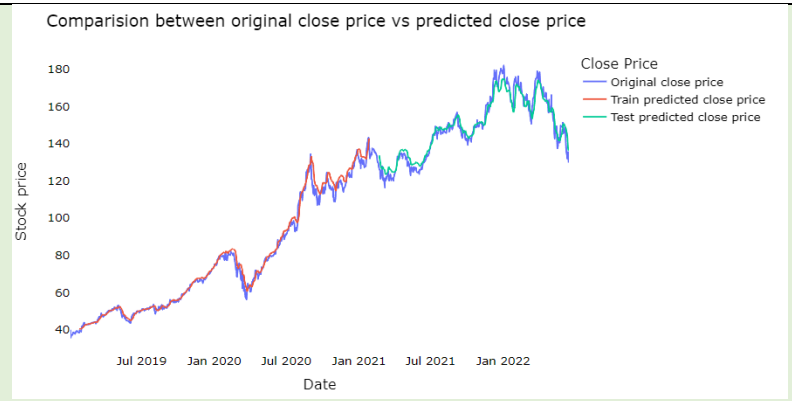
```
model_biLSTM = Sequential()
model_biLSTM.add(Bidirectional(LSTM(units=4,return_sequences=True,activation= 'relu',input_shape=(time_step,1))))
model_biLSTM.add(Bidirectional(LSTM(units=16,activation= 'relu')))
model_biLSTM.add(Dense(units=1,activation= 'linear'))
model_biLSTM.compile(loss='mean_squared_error',optimizer='adam')
history = model_biLSTM.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=50,batch_size=32,verbose=1)
```

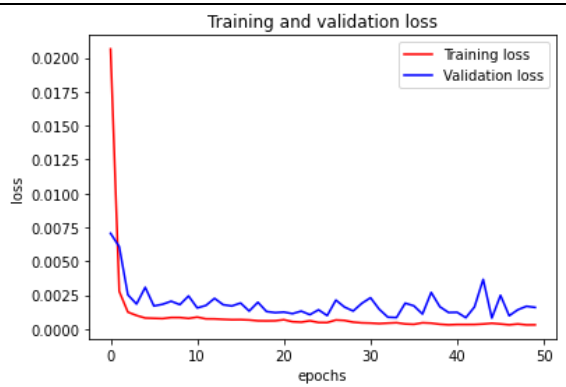
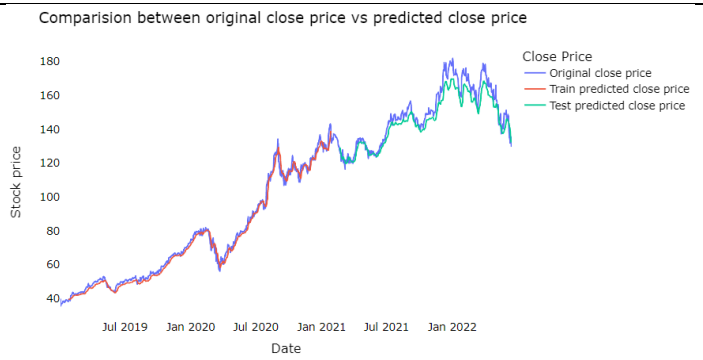
Number of neurons	Model Summary
4 neurons and 16 neurons	Model: "sequential_6" _____ Layer (type) Output Shape Param # ===== bidirectional_17 (Bidirectio (None, 15, 8) 192 _____ bidirectional_18 (Bidirectio (None, 32) 3200 _____ dense_6 (Dense) (None, 1) 33 ===== Total params: 3,425 Trainable params: 3,425 Non-trainable params: 0 _____
16 neurons and 32 neurons	Model: "sequential_7" _____ Layer (type) Output Shape Param # ===== bidirectional_19 (Bidirectio (None, 15, 32) 2304 _____ bidirectional_20 (Bidirectio (None, 64) 16640 _____ dense_7 (Dense) (None, 1) 65 ===== Total params: 19,009 Trainable params: 19,009 Non-trainable params: 0

32 neurons and 64 neurons	Model: "sequential_8" _____ Layer (type) Output Shape Param # ========== bidirectional_21 (Bidirectio (None, 15, 64) 8704 _____ bidirectional_22 (Bidirectio (None, 128) 66048 _____ dense_8 (Dense) (None, 1) 129 ========== Total params: 74,881 Trainable params: 74,881 Non-trainable params: 0
64 neurons and 128 neurons	Model: "sequential_9" _____ Layer (type) Output Shape Param # ========== bidirectional_23 (Bidirectio (None, 15, 128) 33792 _____ bidirectional_24 (Bidirectio (None, 256) 263168 _____ dense_9 (Dense) (None, 1) 257 ========== Total params: 297,217 Trainable params: 297,217 Non-trainable params: 0 _____
128 neurons and 256 neurons	Model: "sequential_10" _____ Layer (type) Output Shape Param # ========== bidirectional_25 (Bidirectio (None, 15, 256) 133120 _____ bidirectional_26 (Bidirectio (None, 512) 1050624 _____ dense_10 (Dense) (None, 1) 513 ========== Total params: 1,184,257 Trainable params: 1,184,257 Non-trainable params: 0
256 neurons and 512 neurons	Model: "sequential_11" _____ Layer (type) Output Shape Param # ========== bidirectional_27 (Bidirectio (None, 15, 512) 528384 _____ bidirectional_28 (Bidirectio (None, 1024) 4198400 _____ dense_11 (Dense) (None, 1) 1025 ========== Total params: 4,727,809 Trainable params: 4,727,809 Non-trainable params: 0

Number of neurons	Training and validation loss	RMSE	R2 score	Predicted close price versus original close price
4 neurons and 16 neurons		Train data RMSE: 3.208222559989269 8 Test data RMSE: 5.187779982825075 5	Train data R2 score: 0.9873105257861152 Test data R2 score: 0.9072501296875212	

16 neurons and 32 neurons		Train data RMSE: 2.85454011 0604968 Test data RMSE: 14.45867356 2396518	Train data R2 score: 0.98995414 39562003 Test data R2 score: 0.27954457 13142 2616	
32 neurons and 64 neurons		Train data RMSE: 2.80957523 7943154 Test data RMSE: 4.86459713 0153314	Train data R2 score: 0.99026813 70584 383 Test data R2 score: 0.91844624 326924	

64 neurons and 128 neurons		Train data RMSE: 2.5324291706457465 Test data RMSE: 6.146753328505390	Train data R2 score: 0.9920934091485564 Test data R2 score: 0.8697907663059135	
128 neurons and 256 neurons		Train data RMSE: 2.8460688652020379 Test data RMSE: 4.088366379203311	Train data R2 score: 0.9900136804378274 Test data R2 score: 0.9423963730380612	

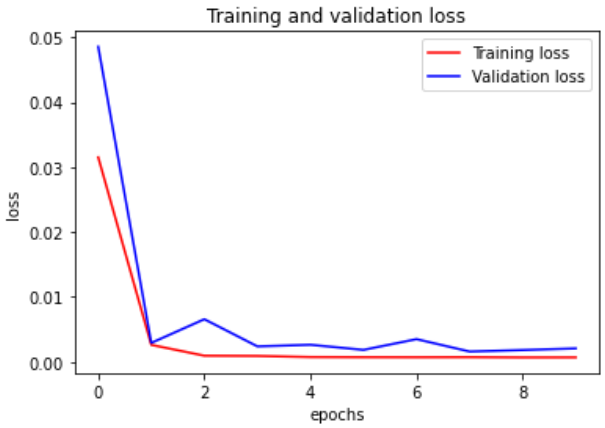
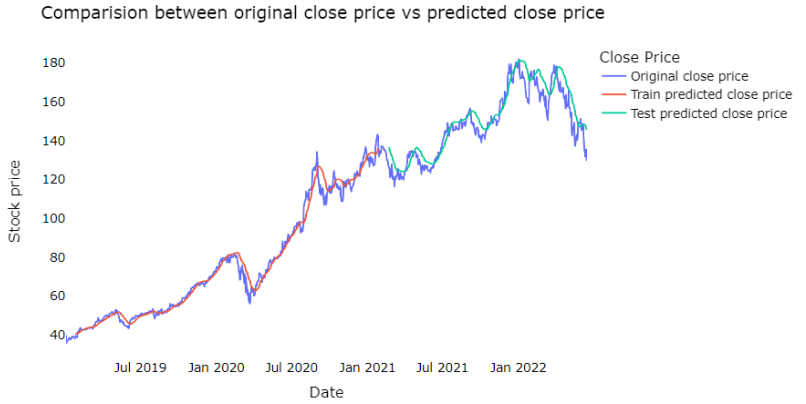
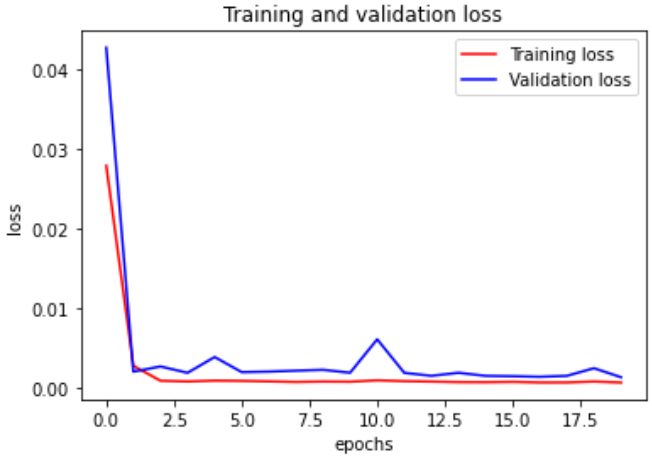
256 neurons and 512 neurons		Train data RMSE: 2.656736820009262 3 Test data RMSE: 5.834726312895937 5	Train data R2 score: 0.9912981474473194 Test data R2 score: 0.8826748300343723	

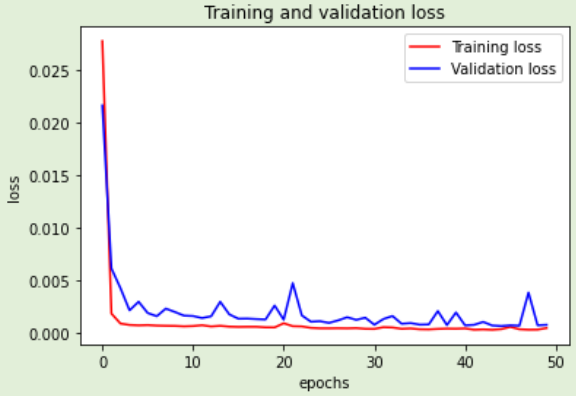
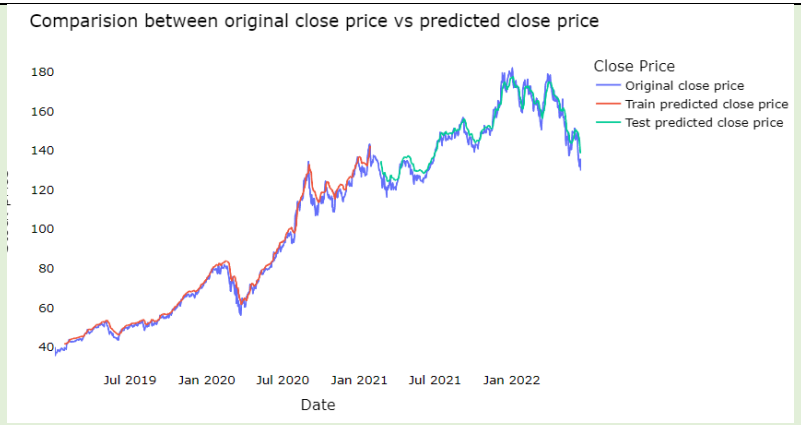
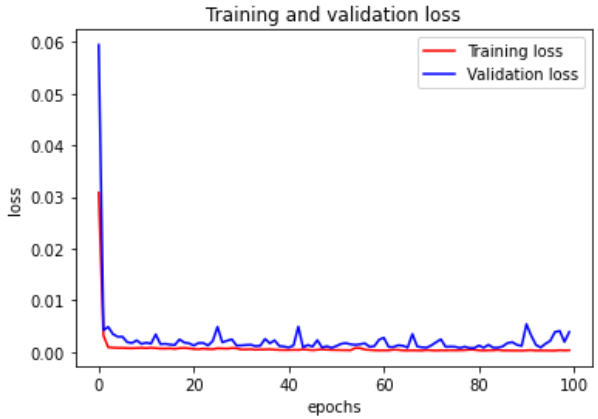
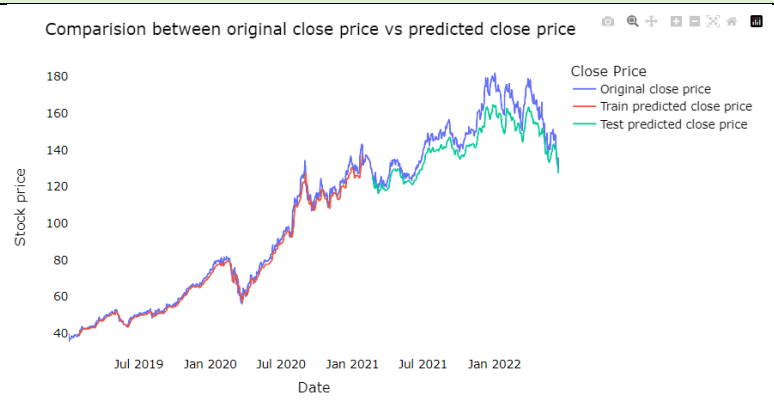
Based on the result above, it was found that the 128 and 256 neurons model is the best model with minimum validation loss, only `4.088366379203311` RMSE and `0.9423963730380612` R^2 scores. This model is then used in the next tuning session.

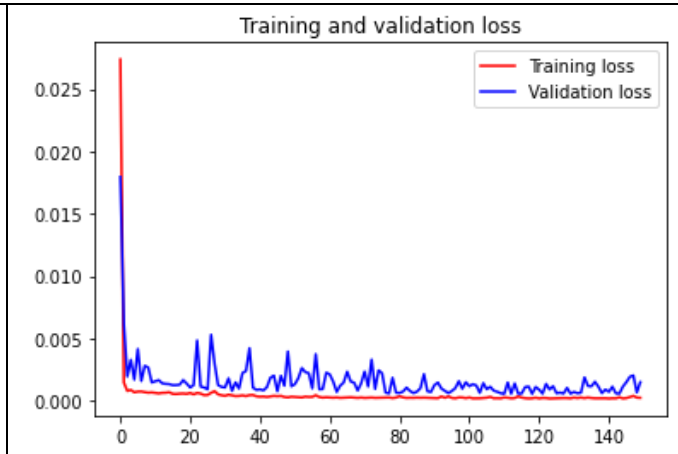
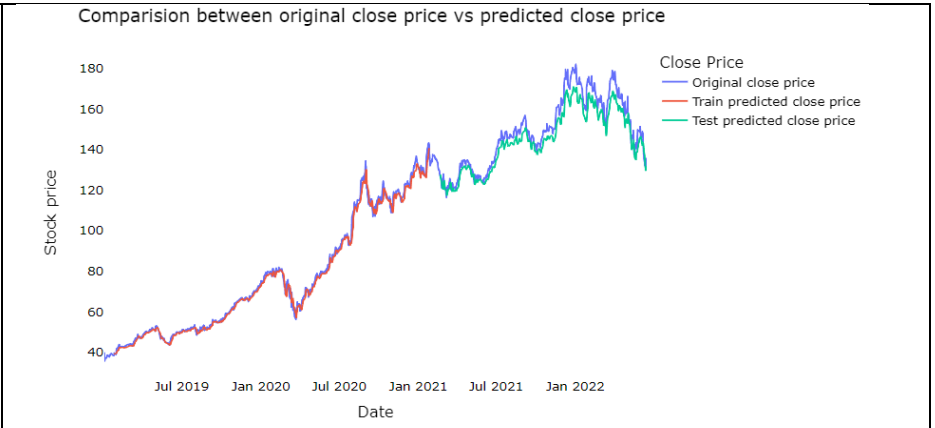
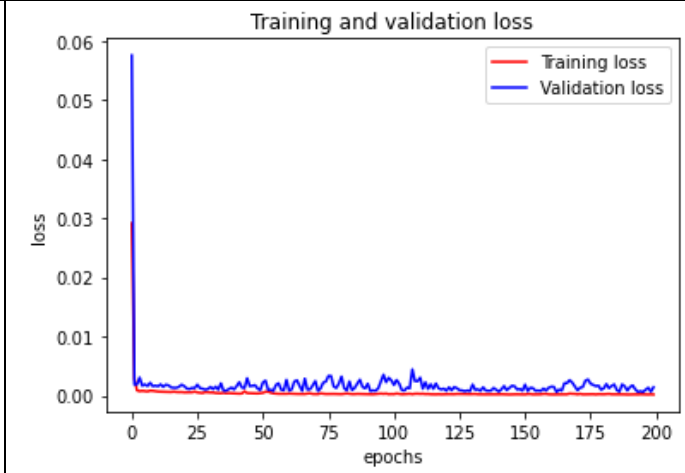
4.3.3.3 Number of epochs

In this session, the best model previous is then used to tune the number of epochs. The number of epochs that are tuned here included 10 epochs, 20 epochs, 50 epochs, 100 epochs and 150 epochs. The figure below showed the sample coding of the bi-LSTM model with 10 epochs. The parameter that was circled indicates the parameter that needs to be tuned:

```
model_biLSTM = Sequential()
model_biLSTM.add(Bidirectional(LSTM(units=128,return_sequences=True,activation= 'relu',input_shape=(time_step,1))))
model_biLSTM.add(Bidirectional(LSTM(units=256,activation= 'relu')))
model_biLSTM.add(Dense(units=1,activation= 'linear'))
model_biLSTM.compile(loss='mean_squared_error',optimizer='adam')
history = model_biLSTM.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=10,batch_size=32,verbose=1)
```

Epochs	Training and validation loss	RMSE	R2 score	Comparison between original close price and predicted close price
10		Train data RMSE: 3.7027738308043934 Test data RMSE: 6.653798515833355	Train data R2 score: 0.9830967964182803 Test data R2 score: 0.8474228478560706	
20		Train data RMSE: 3.4256150473 2442 Test data RMSE: 5.2732168357 63904	Train data R2 score: 0.9855325 57466 2872 Test data R2 score: 0.9041700	

			033831 334	
50		Train data RMSE: 3.0934848708750002 Test data RMSE: 4.106775204892778	Train data R2 score: 0.9882019388320026 Test data R2 score: 0.9418764575447848	
100		Train data RMSE: 2.9574215133019397 Test data RMSE: 9.065172972708154	Train data R2 score: 0.9892169627316063 Test data R2 score: 0.7167940304091089	

150		Train data RMSE: 2.279103023471447 Test data RMSE: 5.690568358491165	Train data R2 score: 0.9935961291422417 Test data R2 score: 0.8884006918315146	
200		Train data RMSE: 2.0062385447616937 Test data RMSE: 5.515465542467588	Train data R2 score: 0.9950377368562092 Test data R2 score: 0.8951630049425263	

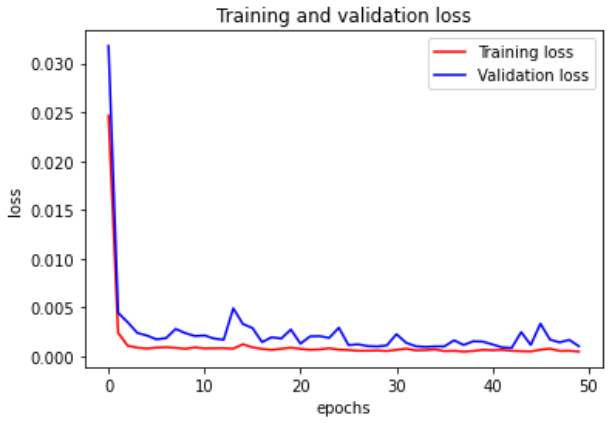
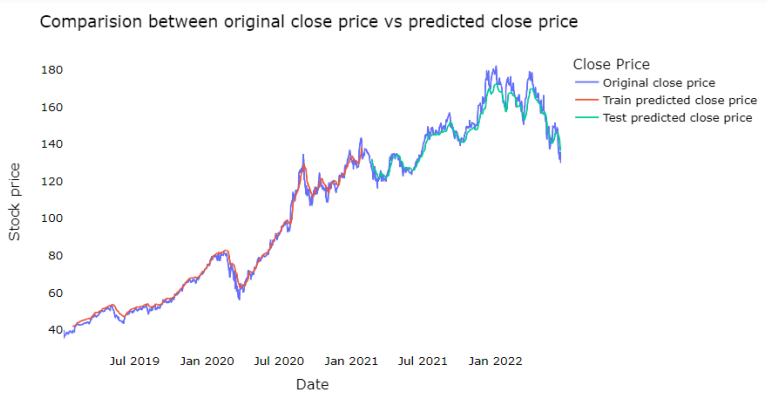
Based on the result, it was found the 50 epochs model achieved the highest performance, which is `0.9418764575447848` R^2 score and only `4.106775204892778` RMSE with minimum validation loss. Hence, this model is then used in the next section.

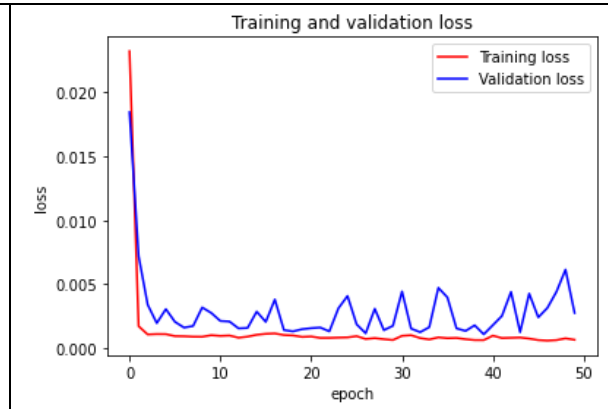
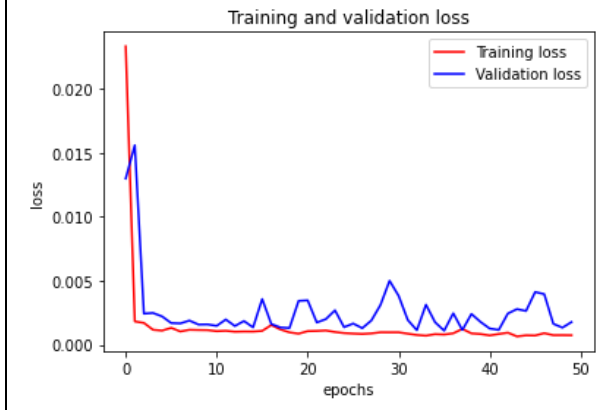
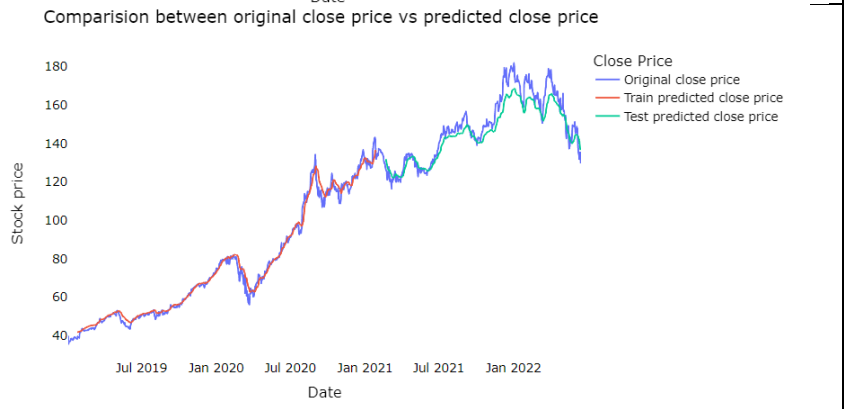
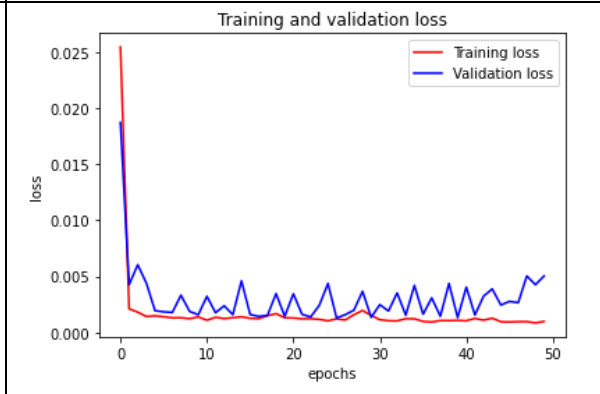
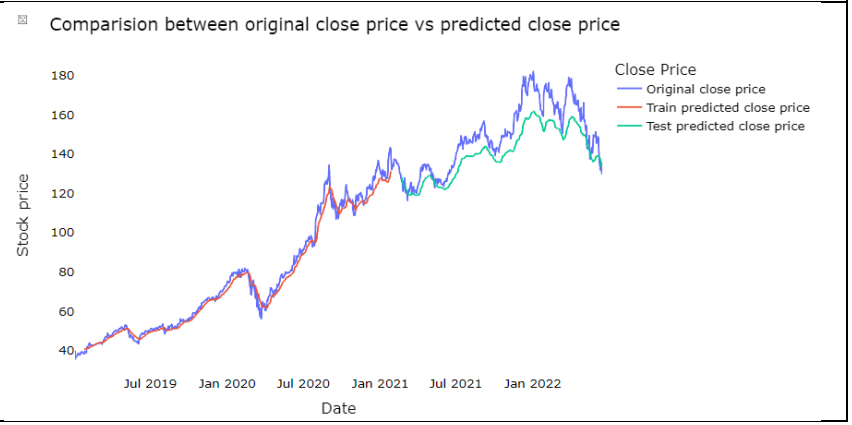
4.3.3.4 Dropout rate

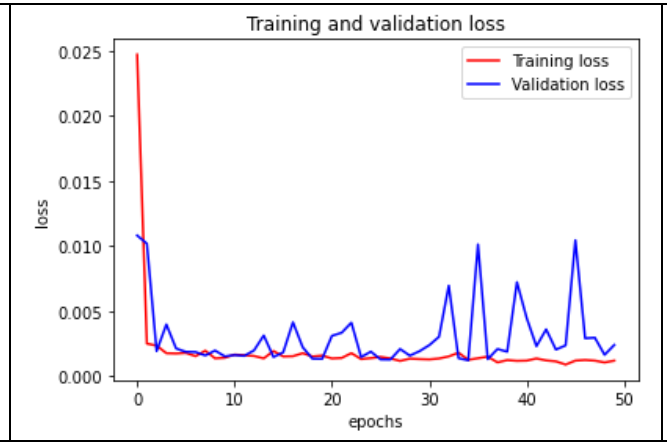
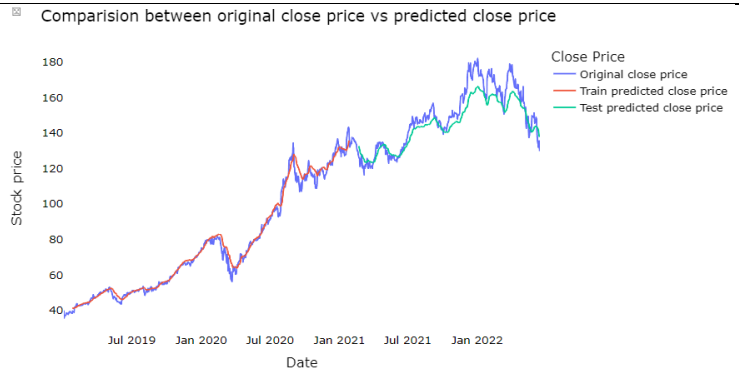
In this section, the last best model in the previous session is used to add regularization and find out the best-fit dropout rate. The dropout rate that was tested included 0.1, 0.2,0.3,0.4 and 0.5. The figure below showed the sample coding of the bi-LSTM model with a 0.1 dropout rate. The parameter that was circled indicates the parameter that needs to be tuned:

```
model_biLSTM = Sequential()
model_biLSTM.add(Bidirectional(LSTM(units=128,return_sequences=True,activation= 'relu',input_shape=(time_step,1))))
model_biLSTM.add(Dropout(0.1))
model_biLSTM.add(Bidirectional(LSTM(units=256,activation= 'relu')))
model_biLSTM.add(Dropout(0.1))
model_biLSTM.add(Dense(units=1,activation= 'linear'))
model_biLSTM.compile(loss='mean_squared_error',optimizer='adam')
history = model_biLSTM.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=50,batch_size=32,verbose=1)
```

.

Dropout rate	Training and validation loss	RMSE	R2 score	Comparison between original close price and predicted close price
0.1		Train data RMSE: 2.834428996757 9503 Test data RMSE: 4.724530701036065	Train data R2 score: 0.9900951976133202 Test data R2 score: 0.9230749895685323	

0.2		Train data RMSE: 2.903687860808 3954 Test data RMSE: 7.702332490389433	Train data R2 score: 0.9896052389926294 Test data R2 score: 0.7955464117324 34	
0.3		Train data RMSE: 2.804805291751 8245 Test data RMSE: 6.177790806979309	Train data R2 score: 0.9903011534780451 Test data R2 score: 0.8684724867754767	
0.4		Train data RMSE: 3.660500007911 0015 Test data RMSE: 10.391513450556 177	Train data R2 score: 0.9834805541393306 Test data R2 score: 0.6278587423112474	

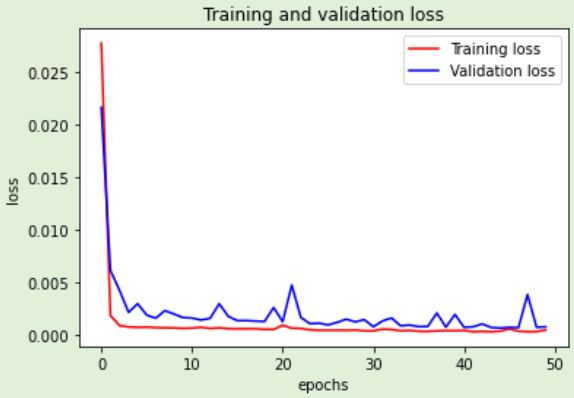
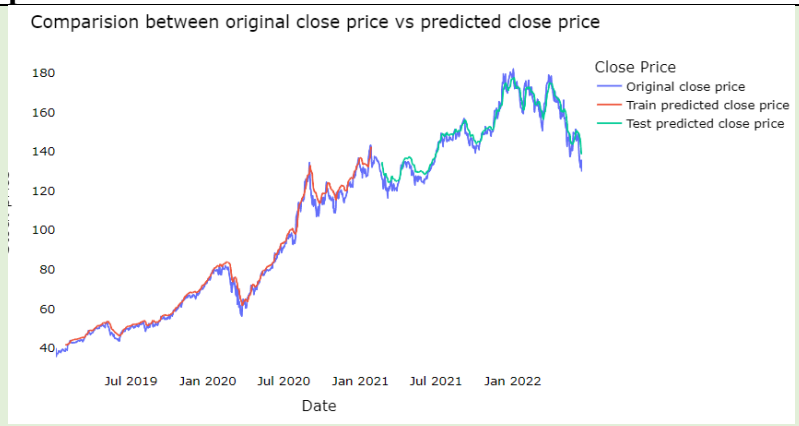
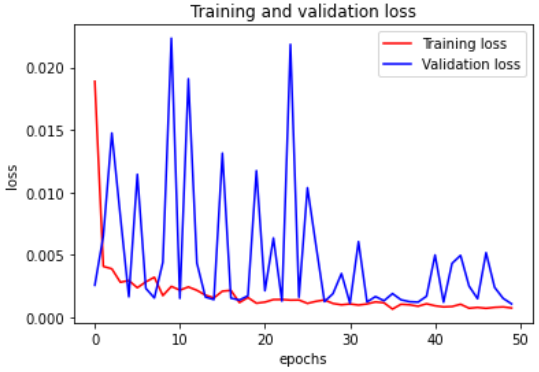
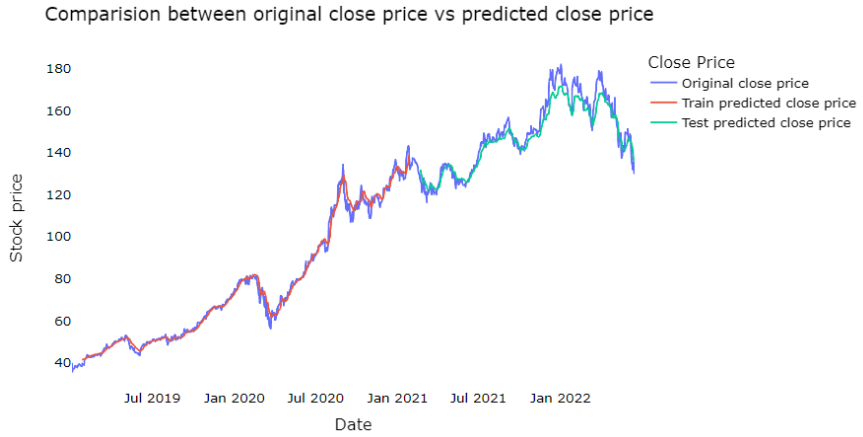
0.5		Train data RMSE: 3.045942960837221 Test data RMSE: 7.1601639842636 53	Train data R2 score: 0.9885617868999422 Test data R2 score: 0.8233164362923143	

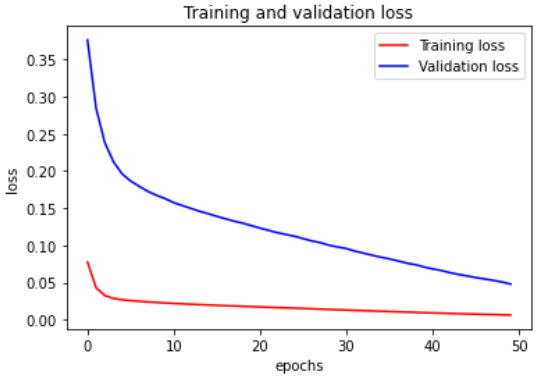
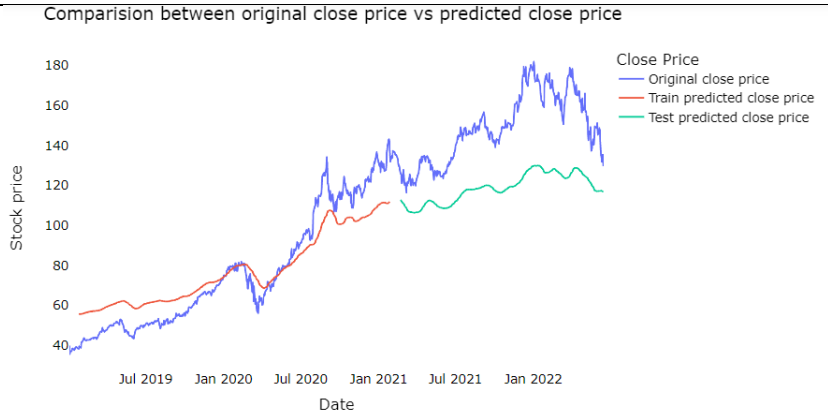
Based on the result, it was found that the model in the previous step has higher performance than adding the regularization. Hence, there is no model selected in this session. We used the previous best model for the optimizer selection in the next session.

4.3.3.5 Optimizer

The optimizer selection included adam, RMSprop, and SDG. The figure below showed the sample coding of the bi-LSTM model with the RMSprop optimizer. The parameter that was circled indicates the parameter that needs to be tuned:

```
model_biLSTM = Sequential()
model_biLSTM.add(Bidirectional(LSTM(units=128,return_sequences=True,activation= 'relu',input_shape=(time_step,1))))
model_biLSTM.add(Bidirectional(LSTM(units=256,activation= 'relu')))
model_biLSTM.add(Dense(units=1,activation= 'linear'))
model_biLSTM.compile(loss='mean_squared_error',optimizer='RMSprop')
history = model_biLSTM.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=50,batch_size=32,verbose=1)
```

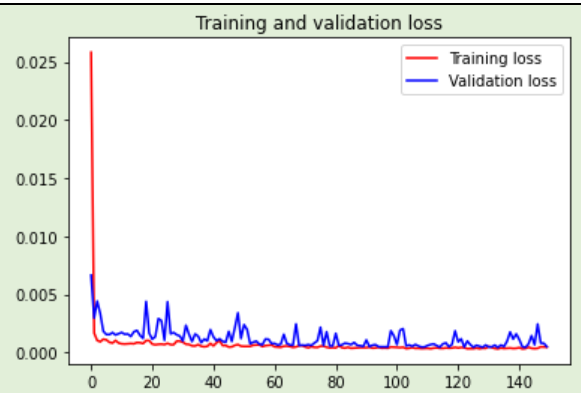
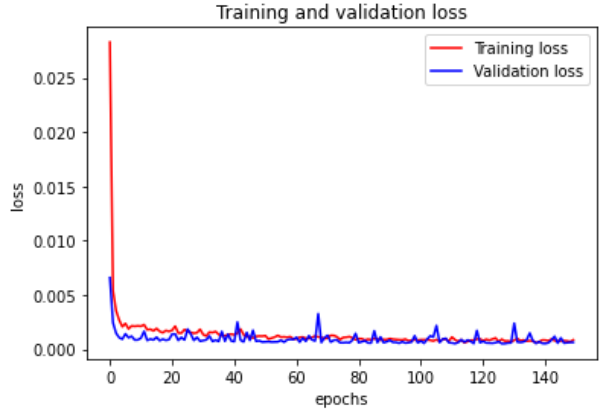
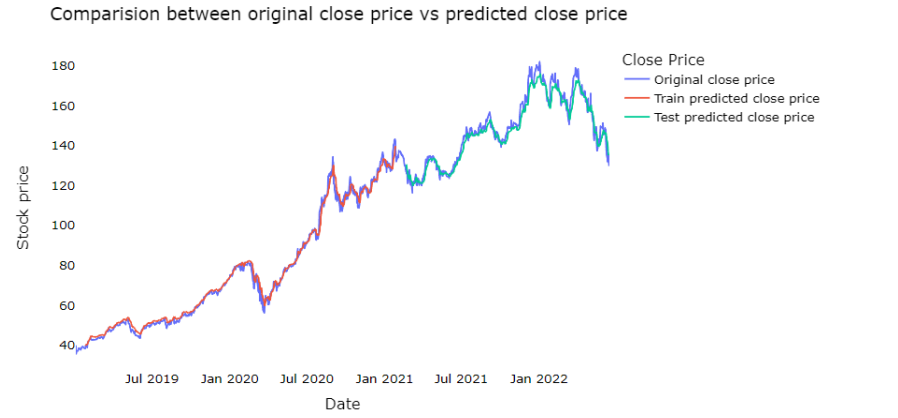
Dropout rate	Training and validation loss	RMSE	R2 score	Comparison between original close price and predicted close price
Adam		Train data RMSE: 3.0934848708750002 Test data RMSE: 4.106775204892778	Train data R2 score: 0.9882019388320026 Test data R2 score: 0.9418764575447848	
RMSprop		Train data RMSE: 2.5246272579435263 Test data RMSE: 4.799219467423498	Train data R2 score: 0.9921420513852799 Test data R2 score: 0.9206235934534208	

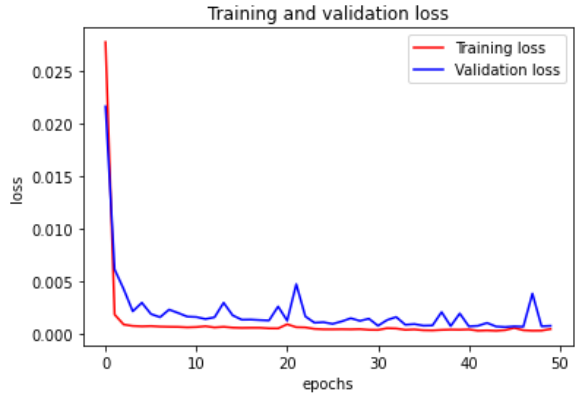
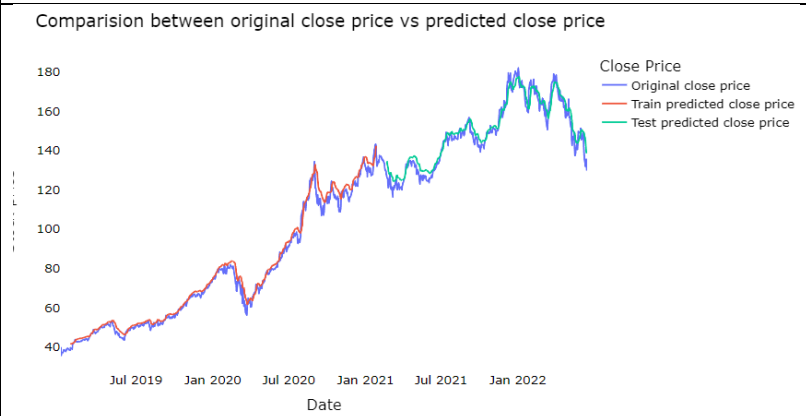
SDG		Train data RMSE: 11.565934016362847 Test data RMSE: 32.070481613920876	Train data R2 score: 0.8350789575168556 Test data R2 score: -2.5445505694223276	

Based on the result above, we can see that the adam optimizer achieved the highest performance with more stable validation loss than the RMSprop and SDG optimizer. Moreover, it achieved the lowest RMSE (`4.106775204892778`) and the highest R^2 score (`2.5445505694223276`).

4.3.4 Final Result

Architecture	Training and validation loss	RMSE	R2 score	Comparison between original close price and predicted close price

1 LSTM input layer, 1 LSTM hidden layers and 1 output layer (256 units and 512 units, 150 epoch, 0.1 dropout rate and adam optimizer)		Train data RMSE: 2.4705430661372167 Test data RMSE: 3.3051333689 20817	Train data R2 score: 0.9924751212159914 Test data R2 score: 0.96235318 92735879	
1 GRU input layer, 1 GRU hidden layers and 1 output layer (32 and 64 units, 150 epochs, 0.1 dropout rate and adam optimizer)		Train data RMSE: 2.3484120759827873 Test data RMSE: 3.6770169954 1243	Train data R2 score: 0.9932007147641 7 Test data R2 score: 0.95340476 70521613	

1 bi-LSTM input layer, 1 bi-LSTM hidden layers and 1 output layer (128 and 256 units, 50 epochs, and adam optimizer)		Train data RMSE: 3.0934848708750002 Test data RMSE: 4.106775204892778	Train data R2 score: 0.9882019388320026 Test data R2 score: 0.9418764575447848	

By comparing all the best results in LSTM, GRU and the bidirectional-LSTM model, it was found that the LSTM model is the best model that can be used to predict Apple stock closing price. Hence, we can conclude that the LSTM model with 1 LSTM input layer, 1 LSTM hidden layer and 1 output layer with 256 units and 512 units, 150 epoch, 0.1 dropout rate and adam optimizer is the best model for Apple stock closing price prediction.

CHAPTER V

DISCUSSION

In this project, there are 3 kinds of models were focused on which included long short-term memory model, the bidirectional long short-term memory model and the gated recurrent unit model. Based on the literature review, most of the studies are using LSTM, bidirectional LSTM and GRU for stock price prediction. However, based on these 3 kinds of models, LSTM still ranked the highest selection for stock price forecasting. In this study, the 3 kinds of models are built and compared with different architecture. The number of layers, number of neurons, epochs, dropout rate and optimizer are adjusted and compared to the results. In the end, it was found the LSTM with 1 LSTM input layer, 1 LSTM layer and 1 output layer with 256 units and 512 units, 150 epoch, 0.1 dropout rate and adam optimizer is the most suitable model for Apple stock price prediction.

In terms of tuning in the number of layers, it was found that in the LSTM model, it has a greater effect when the number of layer increase. When the number of LSTM layer increase from 1 LSTM hidden layer to 2 LSTM hidden layer, the test root square mean error significantly increases from 4.7 to 7.8 while the R squared score significantly drops from 0.92 to 0.79. However, the effect of layers in the GRU and bi-LSTM model are not as significant as in the LSTM model. Based on this finding, we can understand that in the LSTM model, the number of layers plays a more important role in determining the performance of the model than the bidirectional LSTM and GRU models. Moreover, it was also found the model with the least number of layers has a better performance. Based on the result, all the model with only one hidden layer performs the best. Based on the recent study of Adil et al. (2022) which is a study that carried out a detailed investigation to choose the number of layers and neurons for the mixed design model, the researcher claimed that simple neural networks with one or two hidden layers outperformed those with three or more such layers in terms of performance.

In terms of tuning in the number of neurons, the result showed that the best number of neurons in LSTM is 256 and 512 neurons, 32 and 64 neurons in GRU and 128 and 256 neurons in bidirectional LSTM. The result brings information that when the neuron is too less such as 4 and 16 units, it normally does not bring the desired result. Based on an article from Heaton Research, the author mentioned that when there are not enough neurons in the hidden layers to

properly identify the information in a complex data set, underfitting takes place (Heaton, 2017). However, using an excessive number of neurons in the hidden layers might lead to overfitting. Moreover, the study of Adil et al. (2022) also claimed the same point as Heaton, (2017). The study by Yilmaz, (2011) which is the study that focuses on the impact of neuron numbers on neural networks also pointed out that it takes some trial and error to determine the ideal number of neurons. The model may overfit and produce incorrect findings if the right number of neurons is not chosen.

In term of tuning in the number of epochs, it was found that in LSTM and GRU layer, 150 epochs achieved the best result while in the bidirectional LSTM model, 50 epochs achieve the best result. In bidirectional LSTM, the RMSE results increase significantly from 4.1 to 9.1 when the epoch changes from 50 to 100. The same goes with the R squared score decrease from 0.94 to 0.72 in this process. An epoch is the overall number of repetitions required to train the deep learning model using all of the training data at once. When the set of data has made both forward and backward passes, one pass is recorded. It specifies how many times the learning algorithm must process the complete data collection (TechTarget, 2021). Based on the result, this study found that bidirectional LSTM needs a lesser epoch number to obtain the optimum result when compared to LSTM and GRU, this might be due to the forward and backward direction of the model reducing the optimal number of epochs. Based on the study by Afaq, (2020) which is research that focused on the impact of epochs on training neural networks, the authors that there is no perfect number of epochs. In reality, the number of epochs varies from dataset to dataset, but the important aspect that matters is the training and validation error.

The preferred approach to minimize overfitting in neural networks has been to use dropout layers. It is the master of regularisation in the deep learning era of today (Yadav, 2022). In terms of tuning in the dropout rate, it was found that the LSTM used only 0.1 dropout rate, the GRU used only a 0.3 dropout rate while the bidirectional LSTM has a better result when no regularization is carried out. Based on the tuning session, the dropout rate that was tested ranged from 0.1 to 0.5. It is suspected that the optimum dropout rate might be above 0.5 for the bidirectional LSTM model. Hence, it is suggested more range of a dropout range can be tested in the next study to find out the optimum rate.

In terms of tuning in the optimizer, it was found that adam optimizer is among the best optimizer that gives the highest performance in LSTM, biLSTM and GRU models. However, RMSprop and SDG bring fewer effective results with higher RMSE and lower R- squared scores. SGD is a fairly simple technique that is currently rarely used in applications because of how slowly it computes. Additionally, it struggles to effectively manage saddle points. The only difference between RMSProp and the gradient descent technique using momentum is how the gradients are calculated. The Adam optimizer inherited the positive traits of RMSProp as well as other algorithms. The Adam optimizer produces outcomes that are usually superior to those of conventional optimization methods, take less time to compute, and need fewer control variables. Adam is suggested as the default optimizer for the majority of applications as a result of all of that (Gupta, 2021). Empirical findings show that Adam performs admirably in real-world applications and is comparable to other stochastic optimization techniques. Based on the literature review, most of the studies are also using adam optimizer to achieve the best performance. After experimenting in this study, the result showed that the outcome is compatible with the literature review. Hence, it can be concluded that adam optimizer is the best optimizer for stock price prediction in all the time series forecasting models.

In the nutshell, the result of this study concluded that LSTM after hyperparameter tuning is the most suitable model that can be used to predict Apple stock closing price when compared to others. This result is compatible with the literature review as there are major study support LSTM as the best model for stock price prediction. In this study, it can be pointed out that the model's success depends on a variety of parameters that must be set or adjusted to enable proper model training and an excellent outcome. In this project, the main challenge is choosing the right range of values for the parameter tuning. By using the manual method, although it can understand the effect of the parameter on the model, it also brings the risk of missing out on the value that might have a better result than the manually chosen value. In future research, it is suggested that authors can apply a tuner application for hyperparameter tunings such as random search, grid search or Keras tuner. Moreover, different time frames such as long term with more than 10 years and short term with less than 12 months also can be compared to examine the effectiveness of the model in a different time period. Furthermore, the author also can examine stock prices in different sectors to understand the effectiveness more deeply.

CONCLUSION

Different stock price studies are frequently debated and regarded as a special form of time series. Due to the complexity of time series models, some major trends cannot be accurately detected using conventional techniques that depend on linear regression techniques. Furthermore, without more powerful, extraordinarily nonlinear showing tools, it is difficult to predict or calculate the financial transaction undertaken. The study concluded that LSTM model with 1 LSTM input layer, 1 LSTM hidden layer and 1 output layer with 256 units and 512 units, 150 epoch, 0.1 dropout rate and adam optimizer is the best model for Apple stock closing price prediction. As a result, both individuals and businesses can make use of this suggested forecasting model employing LSTM for stock price prediction. This can enable investors to make significant monetary gains while maintaining a stable share price atmosphere. It can be concluded that the aim and objective of this project is successfully achieved. The author of this study intends to examine data from additional stock sectors in additional areas in the future to assess how well this strategy is working.

REFERENCE

Adil, M., Ullah, R., Noor, S., & Gohar, N. (2022). Effect of the number of neurons and layers in an artificial neural network for generalized concrete mix design. *Neural Computing & Applications, 34*(11), 8355–8363. https://doi.org/10.1007/s00521-020-05305-8

Afaq, S., & Rao, S. (n.d.). *Significance of epochs on training A neural network*. Ijstr.org. Retrieved November 27, 2022, from https://www.ijstr.org/final-print/jun2020/Significance-Of-Epochs-On-Training-A-Neural-Network.pdf

Althelaya, K. A., El-Alfy, E.-S. M., & Mohammed, S. (2018). Evaluation of bidirectional LSTM for short-and long-term stock market prediction. *2018 9th International Conference on Information and Communication Systems (ICICS)*.

Baeldung. (n.d.-a). *Training and Validation Loss in Deep Learning*. Baeldung.com. Retrieved November 21, 2022, from https://www.baeldung.com/cs/training-validation-loss-deep-learning#:~:text=4.,the%20performance%20of%20the%20model.

Baeldung. (n.d.-b). *Training and Validation Loss in Deep Learning*. Baeldung.com. Retrieved November 23, 2022, from https://www.baeldung.com/cs/training-validation-loss-deep-learning

Biswas, S. (n.d.). *Advantages of deep learning, plus use cases and examples*. Width.Ai. Retrieved November 11, 2022, from https://www.width.ai/post/advantages-of-deep-learning

Brownlee, J. (2017a, May 23). *A gentle introduction to long Short-Term Memory networks by the experts*. Machinelearningmastery.com; Machine Learning Mastery. https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/

Brownlee, J. (2017b, July 2). *Gentle introduction to the Adam optimization algorithm for deep learning*. Machinelearningmastery.com; Machine Learning Mastery. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Budiharto, W. (2021). Data science approach to stock prices forecasting in Indonesia during Covid-19 using Long Short-Term Memory (LSTM). *Journal of Big Data*, *8*(1), 47. https://doi.org/10.1186/s40537-021-00430-0

Chandola, D., Mehta, A., Singh, S., Tikkiwal, V. A., & Agrawal, H. (2022). Forecasting directional movement of stock prices using deep learning. *Annals of Data Science*. https://doi.org/10.1007/s40745-022-00432-6

Chen, J. (2022, August 3). *What is the stock market, what does it do, and how does it work?* Investopedia. https://www.investopedia.com/terms/s/stockmarket.asp

Dobilas, S. (2022, February 21). *GRU recurrent neural networks — A smart way to predict sequences in Python*. Towards Data Science. https://towardsdatascience.com/gru-recurrent-neural-networks-a-smart-way-to-predict-sequences-in-python-80864e4fe9f6

Dutta, A., Kumar, S., & Basu, M. (2020). A gated recurrent unit approach to Bitcoin price prediction. *Journal of Risk and Financial Management*, *13*(2), 23. https://doi.org/10.3390/jrfm13020023

Fernando, J. (2003, November 25). *R-squared formula, regression, and interpretations*. Investopedia. https://www.investopedia.com/terms/r/r-squared.asp

Ghosh, A., Bose, S., Maji, G., Debnath, N. C., & Sen, S. (Eds.). (2019). *Stock Price Prediction Using LSTM on Indian Share Market* (Vol. 63). EPiC Series in Computing.

Gudikandula, P. (2019, March 27). *Recurrent Neural Networks and LSTM explained - purnasai gudikandula*. Medium. https://purnasaigudikandula.medium.com/recurrent-neural-networks-and-lstm-explained-7f51c7f6bbb9

Gupta, A. (2021, October 7). *A comprehensive guide on deep learning optimizers*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/

Heaton, J. (2017, June 1). *The number of hidden layers*. Heaton Research. https://www.heatonresearch.com/2017/06/01/hidden-layers.html

Hoare, M. T. (n.d.). *Stock price prediction using time series models*. Dbs.Ie. Retrieved November 11, 2022, from https://esource.dbs.ie/bitstream/handle/10788/3830/msc_chouksey_s_2019.pdf?sequence=1&isAllowed=y

Hu, Z., Zhao, Y., & Khushi, M. (2021). A survey of Forex and stock price prediction using deep learning. *Applied System Innovation*, *4*(1), 9. https://doi.org/10.3390/asi4010009

IBM Cloud Education. (2020, September 14). *What are Recurrent Neural Networks?* Ibm.com. https://www.ibm.com/cloud/learn/recurrent-neural-networks

Introduction to recurrent neural network. (2018, October 3). GeeksforGeeks. https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

Istiake Sunny, M. A., Maswood, M. M. S., & Alharbi, A. G. (2020). Deep learning-based stock price prediction using LSTM and bi-directional LSTM model. *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*.

Joo, I.-T., & Choi, S.-H. (2018). Stock prediction model based on bidirectional LSTM recurrent neural network. *The Journal of Korea Institute of Information Electronics and Communication Technology*, *11*(2), 204–208. https://doi.org/10.17661/jkiiect.2018.11.2.204

Lawton, G. (2022, January 31). *Data preprocessing: Definition, key steps and concepts*. SearchDataManagement; TechTarget. https://www.techtarget.com/searchdatamanagement/definition/data-preprocessing

Magnimind. (n.d.). *Deep Learning and Its 5 Advantages*. Becominghuman.Ai. Retrieved November 11, 2022, from https://becominghuman.ai/deep-learning-and-its-5-advantages-eaeee1f31c86?gi=eded689f79da#:~:text=One%20of%20the%20biggest%20advantages,told%20to%20do%20so%20explicitly.

MarketMuse. (n.d.). *Gated Recurrent Unit (GRU)*. MarketMuse Blog. Retrieved November 20, 2022, from https://blog.marketmuse.com/glossary/gated-recurrent-unit-gru-definition/

Murphy, C. B. (2018, May 22). *Why do companies care about their stock prices?* Investopedia. https://www.investopedia.com/investing/why-do-companies-care-about-their-stock-prices/

Navas, J. (n.d.). *What is hyperparameter tuning?* Anyscale. Retrieved November 21, 2022, from https://www.anyscale.com/blog/what-is-hyperparameter-tuning

Padhma, M. (2021, October 28). *Evaluation metric for regression models*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/10/evaluation-metric-for-regression-models/

Patel, V. M. R. (Ed.). (2017). *Predicting Stock Prices Using LSTM*. International Journal of Science and Research (IJSR).

Patil, P. (2018, March 23). *What is Exploratory Data Analysis?* Towards Data Science. https://towardsdatascience.com/exploratory-data-analysis-8fc1cb20fd15

Pawar, K., Jalem, R. S., & Tiwari, V. (2019). Stock market price prediction using LSTM RNN. In *Advances in Intelligent Systems and Computing* (pp. 493–503). Springer Singapore.

Pramod, B., & Shastry, M. (Eds.). (2020). *Stock Price Prediction Using LSTM* (Vol. 83). TEST Engineering and Management.

Price, M. (n.d.). *How are stock prices determined?* The Motley Fool. Retrieved November 10, 2022, from https://www.fool.com/investing/how-to-invest/stocks/how-are-stock-prices-determined/

Roondiwala, M. (Ed.). (2017). *Predicting Stock Prices Using LSTM*. International Journal of Science and Research (IJSR). https://www.researchgate.net/profile/Murtaza-Roondiwala/publication/327967988_Predicting_Stock_Prices_Using_LSTM/links/5bafbe6692851ca9ed30ceb9/Predicting-Stock-Prices-Using-LSTM.pdf

Saxena, S. (2021, March 16). *LSTM*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/

Sen, J., & Mehtab, S. (2022). Long-and-short-term memory (LSTM) NetworksArchitectures and applications in stock price prediction. In *Emerging Computing Paradigms* (pp. 143–160). Wiley. https://doi.org/10.1002/9781119813439.ch8

Sethia, A., & Raut, P. (2019). Application of LSTM, GRU and ICA for stock price prediction. In *Information and Communication Technology for Intelligent Systems* (pp. 479–487). Springer Singapore.

Stock price. (2019, April 28). Corporate Finance Institute. https://corporatefinanceinstitute.com/resources/wealth-management/stock-price/

Subhi Alzazah, F., & Cheng, X. (2021). Recent advances in stock market prediction using text mining: A survey. In R. M. X. Wu & M. Mircea (Eds.), *E-Business - Higher Education and Intelligence Applications*. IntechOpen.

Wei, D. (2019). Prediction of stock price based on LSTM neural network. *2019 International Conference on Artificial Intelligence and Advanced Manufacturing (AIAM)*.

What is epoch? (2021, June 28). SearchDataCenter; TechTarget. https://www.techtarget.com/searchdatacenter/definition/epoch

What is LSTM - introduction to Long Short Term Memory. (2022, November 18). Intellipaat Blog; Intellipaat. https://intellipaat.com/blog/what-is-lstm/

Yadav, H. (2022, July 5). *Dropout in neural networks*. Towards Data Science. https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9

Yilmaz, M., & Arslan, E. (2011). Effect of increasing number of neurons using artificial neural network to estimate geoid heights. *International Journal of the Physical Sciences*, *6*(3), 529–533. https://doi.org/10.5897/IJPS10.626

Zvornicanin, E. (2022, November 6). *Differences Between Bidirectional and Unidirectional LSTM*. Baeldung.com. https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm