Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering

Master's Thesis

NSE stock market prediction using Deep Recurrent Neural Network and Comparison with ARIMA

Adithyan Chettiyattil Pankajakshan

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

ADITHYAN CHETTIYATTIL PANKAJAKSHAN

Systems Engineering and Informatics

Thesis title

NSE STOCK MARKET PREDICTION USING DEEP RECURRENT NEURAL NETWORK AND COMPARISON WITH ARIMA

Objectives of thesis

Understand the deep recurrent neural network model in Time Series index forecasting.

Creating Deep RNN Predictive algorithm and testing the accuracy of of Deep RNN in the field of stock market index prediction. Creating ARIMA statistical model of index forecasting and comparison of accuracy with Deep RNN model.

Reviewing feasibility of transition from classical statistical forecasting methods to Deep RNN forecasting in the field of Stock Market Index.

Methodology

NSE Stock market index data of a selected company to be collected from online brokerage firm ZerodhaKite application with API call. Deep Recurrent neural network model to be made in Google collab notebook. Keras module of tensorflow to be used to program the deep RNN model. NSE stock market indices will be collected and input parameters will be decided and output parameter will be stock market Index. The dataset will be split into two training set and test set. The model will be tested to estimate the accuracy of the Deep RNN model. Machine learning model of ARIMA will be created and run the same dataset to find out the accuracy of the ARIMA model. Comparison of both accuracy to be done to reach the feasibility of using Deep RNN in stock market forecasting.

The proposed extent of the thesis

60 pages

Keywords

Deep learning, RNN, prediction, ARIMA

Recommended information sources

GÉRON, A. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems.* Beijing; Boston; Farnham; Sevastopol; Tokyo: O'Reilly, 2019. ISBN 978-1-492-03264-9.

Expected date of thesis defence

2022/23 WS – FEM

The Diploma Thesis Supervisor

doc. Ing. Arnošt Veselý, CSc.

Supervising department

Department of Information Engineering

Electronic approval: 4. 11. 2022	Electronic approval: 28. 11. 2022
Ing. Martin Pelikán, Ph.D.	Ing. Martin Pelikán, Ph.D.
Head of department	Dean

Declaration

I declare that I have worked on my master's thesis titled "NSE Stock market prediction using Deep RNN and comparison with ARIMA" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the master's thesis, I declare that the thesis does not break any copyrights.

In Prague on 30/03/2023

Acknowledgement

I would like to thank for the super vision of the thesis by doc. Ing. Arnošt Veselý. I also express gratitude to my brother Arjun, parents, Navya and friends for their support to complete this.

NSE stock market prediction using Deep Recurrent Neural Network and Comparison with ARIMA

Abstract

The National Stock Exchange (NSE) stock index is a crucial indicator of the performance of the Indian stock market and is used by investors and financial institutions to make informed decisions. Accurately predicting the future values of the NSE stock index is a challenging task and requires a combination of sound technical and financial analysis. In this study, we compare the performance of two popular methods for predicting the NSE stock index: deep Recurrent Neural Networks (RNNs) and Autoregressive Integrated Moving Average (ARIMA) models.

The results of our study show that the deep RNN model outperforms the ARIMA model in terms of prediction accuracy, with RNN having lesser RMSE value than ARIMA model.

This study provides insights into the relative strengths and weaknesses of deep RNNs and ARIMA models for predicting the NSE stock index and highlights the potential for deep learning techniques to improve the accuracy of stock market predictions. The results of this study could be useful for investors, financial institutions, and researchers interested in stock market predictions and financial time-series analysis.

Předpovídání indického akciového trhu NSE pomocí hlubokých rekurentních neuronových sítí a srovnání s ARIMA.

Abstrakt

Národní burzovní index (NSE) je klíčovým ukazatelem vývoje indického akciového trhu a slouží investorům a finančním institucím k informovanému rozhodování. Přesné předpovídání budoucích hodnot NSE indexu je náročný úkol a vyžaduje kombinaci zvukové technické a finanční analýzy. V této studii porovnáváme výkon dvou populárních metod pro předpovídání NSE indexu: hluboké rekurentní neuronové sítě (RNN) a autoregresivní integrovaný pohyblivý průměr (ARIMA) modely.

Výsledky naší studie ukazují, že hluboký RNN model překonává ARIMA model v přesnosti předpovědi, přičemž RNN má menší hodnotu RMSE než ARIMA model.

Tato studie poskytuje informace o relativních silných a slabých stránkách hlubokých RNN a ARIMA modelů pro předpovídání NSE indexu a poukazuje na potenciál technik hlubokého učení pro zlepšení přesnosti předpovědí na akciových trzích. Výsledky této studie by mohly být užitečné pro investory, finanční instituce a výzkumníky zabývající se předpověďmi na akciových trzích a finanční analýzou časových řad.

Table of Contents

1. Introduction

Time series forecasting, especially stock price prediction has always been a great concern. It is a challenging and crucial problem to tackle not only by economists but also researchers [30]. Stock price forecasting is challenging because the stock data is affected by many factors which make it unstable and volatile. It is also difficult to forecast due to sudden shock, uncertainty, historical data that may not capture all the pattern, or even the noise and irrelevant information that lies in the data itself. Although it is challenging, the need for accurate prediction of stock prices plays an increasingly crucial role in the stock market [30]. Investor and companies rely on accurate stock price forecasting to make the critical business decisions for their success.

Stock data is part of time series data. Many researchers have used classical time series models such as ARIMA, GARCH, VAR, and so on. However, this method is sometimes limited to some assumptions such as stationarity and linearity [27]. At the same time, the development of artificial intelligence makes more researchers choose to use machine learning and deep learning models for prediction instead. Latest methods such as Neural Network, RNN, LSTM have been widely used [12]. Even so, there are many research gaps about it. Some research might find ARIMA still works best, but another research stated that the complexity of Neural Networks outperform the classical model [15].

Therefore, this study aims to explore the various methods and techniques used in stock price forecasting, including classical methods of ARIMA and more recent approaches such as RNN. Additionally, this study will compare the models based on appropriate evaluation metrics.

2. Objective and methodology

2.1 Objective

The purpose of this study is to develop a granular and robust stock prediction or forecasting framework using classical model and deep learning model to compare the performance of each models.
The proposed models are ARIMA and RNN. With that being said, three main goals are taken into consideration. First, to obtain the optimal model for each framework. Second, to conduct side by side comparison of the selected model for each framework and determine which one is best for forecasting the stock price. Third, to perform deeper analysis and make recommendations about what is the best model for prediction.

2.2 Methodology

The rest of the study is structured as follows. In section 2, a clear statement of the scope and objective of study is presented. In section 3, literature review and prior research are provided. In section 4, detailed discussion and analysis results are explained. And, in section 5, the conclusion and recommendation of study is drawn.
As mentioned above, the methodology of study involves brief explanation of the statistical models and deep learning models for time series. This approach will result in a better understanding about the current state of time series forecasting. It also involves reviewing all the research gaps to see the finding patterns in the finance domain. It will also give a big picture about the tendency for each short term and long-term prediction.
Next, the main task is to predict the stock price using ARIMA and RNN models using available historical dataset. The historical data is made into a dataset. This dataset is cleansed and transformed into an excel file. The data is further analyzed and modified for optimal time series features.
This data is then split into train, validation and test data. Train : Test split is in the ratio of 80:20. Validation data is 36% of the train data. Model is trained and optimal architecture is found for both the Models for a given look back period of time. Then the model is trained, validated and tested and metrics are evaluated.

Another crucial step is to find the best evaluation metric for the model performance comparison. RSME will be the metric taken into account to evaluate the model prediction accuracy. All models will be developed in Python version 3.10 with the help of open source software such as Keras, Tensorflow, Pandas, Seaborn, Numpy, StatsModel, and so on. The dataset is collected from nseindia.com, the website of stock market index of National stock exchange.
The result of each model will later be compared and evaluated. Finally, deeper analysis and recommendation will be drawn from the analysis findings.

3. Literature review

3.1 Terminology

3.1.1 Time Series Model

Time series is a set of observations which the variable takes at different times or listed in the order of time. In other words, the type of data that is collected at a regular time interval, such as daily, weekly, monthly, quarterly, or even annually[13]. Time series models can be divided into descriptive modeling which is sometimes called time series analysis and predictive modeling which is known as time series forecasting. Kotu, [20] also added that time series forecasting can be differentiated into several approaches, time series forecasting based on decomposition, smoothing technique, regression based, and machine learning based. Decomposition approach is a method that deconstructs time series models into components of trend, seasonality, and noise. Next, the smoothing approach is a method that smooths past observation and then projects it to the future. The regression approach is similar to the concept of regression but differs in the independent variable which is now time [20]. Furthermore, a more sophisticated approach than regression based is a model that fosters the concept of autocorrelation, which is the phenomenon that data are correlated in time series. The most popular method among this approach is ARIMA [26]. In addition, machine learning approaches can also be used based on supervised learning concepts with the target or label and its features. In this approach, the features are derived from window technique, transforming time series dataset by using lagged data [20].

3.1.2 Stationarity in Time Series

In the time series concept, the collection of random variables ordered in time is well known as the stochastic process [13]. Part of this concept that received big attention is stationary stochastic. The stationarity in the stochastic process is achieved when mean and variance are constant over time and the value of covariate between two periods depends only on its lag. In short, stationarity in time series refers to the concept that the value of time series is not dependent on time [20]. Without the stationarity, the interpretation of the time series forecasting's result would be problematic (Manuca, 1996). When the stationary concept is violated, the data can be called a nonstationary time series. One example of it is the random walk model. Nonstationary time series data will have time varied mean, time varied variance, or even both [13]. Even so, there is a way to tackle this violation. Non-stationary time series can be changed into stationary ones with the differencing method, which is the change between consecutive data in time series [20]. The formula is calculated as follow:

$$y'_t = y_t - y_{t-1}$$

where,

y'_t = difference value of y_t

y_t = current value at time t

y_{t-1} = past value

In several cases, there might be a situation where data is still non stationary even if the differencing is conducted. In such a case, second order differencing after first order differenced time series will be needed [13]).

3.1.3 Detecting Stationarity

There are several approaches to detect whether the data is stationary or not, which are graphical test, correlogram, and unit root test. The most well known unit root test for detecting stationarity is Augmented Dickey Fuller (ADF) test (Gujati) This statistic test works by augmenting the equations by adding lagged values of dependent variable of ΔY_t. ADF formula is written below.

$$\Delta Y_t = \beta_1 + \beta_2 t + \delta Y_{t-1} + \sum_{i=1}^{m} \alpha_i \Delta Y_{t-i} + \varepsilon_t$$

where,

ε_t = white noise term

$\Delta Y_{t-1} = Y_{t-1} - Y_{t-2}$

The null hypothesis of ADF is to check whether the $\delta = 0$. The ADF test follows the same asymptotic distribution as the Dickey Fuller test. Therefore, the same critical values can be used in hypothesis testing [13]).

3.1.4 ARIMA

According to Gujarati [13], the autoregressive integrated moving average (ARIMA), or popularly known as the Box–Jenkins model is a time series forecasting technique that was proposed in 1976 by George Box and Gwilym Jenkins. The basic difference between ARIMA and regression models is, unlike the regression models, ARIMA model allows the dependent or target variable (Yt) to be explained by past or lagged values, Y itself and stochastic error terms. It is best suited for short term time series forecasting for 12 months or less [9]. The ARIMA model is basically derived by modification of the autoregressive moving average (ARMA) model and written as ARIMA (p,d,q). With that being said, the model has three components, which are p that denote the autoregressive (AR), q that depicts moving average (MA), and d that denotes the integrated (I) part [26]. The integrated component depicts the level of integration of variables which can be stationary with differentiation and all of p,d,q are nonnegative integers [17].

3.1.4.1 Autoregressive Component (AR)

AR component attempts to learn the pattern between current period and previous period [9]. AR will forecast future value by using linear combination of past data value and a white noise term which are random variables with mean of zero and constant variance [22]. According to Box and Jenkins (1976), in general, AR can be formulated described as follow:

$$(Y_t - \delta) \ = \ \alpha_1(Y_{t-1} - \delta) + \alpha_2(Y_{t-2} - \delta) + \ldots . + \alpha_p(Y_{t-p} - \delta) + u_t$$

where,

Y_t = the value or variable that wants to be predicted,

δ = the mean of Y

u_t = uncorrelated random error term with zero mean and constant variance of $\boldsymbol{\sigma}^2$

The component of u_t is also called as white noise. This formula depicts that the forecast value of Y at time t is simply a sum of proportion (denote by α) of previous value plus the random shock or disturbance at time t. The value of Y_t is expressed as deviation from the mean value and depends on its previous p time periods value and the white noise. In other words, Y_t is a pth-order autoregressive or AR(p) model.

3.1.4.2 Moving Average Component (MA)

MA component attempts to measure the adaptation of the new forecast to previous forecast error [9]. The general form of MA is:

$$Y_t = \mu + \beta_0(u_t) + \beta_1(u_{t-1}) + \beta_2(u_{t-2}) + \ldots + \beta_q(u_{t-q})$$

where,

μ = constant

u_t = white noise stochastic error term

Basically, a moving average component is a linear combination of white noise error terms. In other words, Y_t is a qth-order moving average or MA(q)model.

3.1.4.3 Autoregressive Integrated Moving Average (ARIMA)

In time series, it is quite likely that Y_t has both components of AR and MA. Thus, they will form a new model ARMA [13]. ARMA(p,q) model contains p autoregressive and q moving average. It can be formulated as follow:

$$Y_t = \theta + \alpha_1 Y_{t-1} + \ldots + \alpha_p Y_{t-p} - \delta) + \beta_0(u_t) + \ldots + \beta_q(u_{t-q})$$

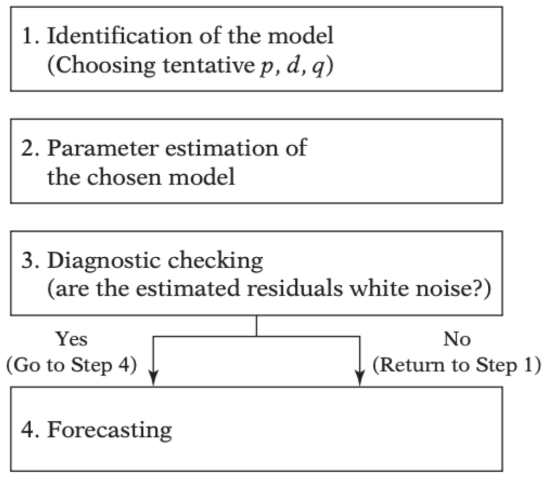
where,

θ = constant term

The ARMA model has the underlying assumption of stationarity, which requires the mean to be constant and covariance to be time invariant [24]. However, in the real world, many economic time series are non stationary or integrated. Therefore, one needs to difference the time series of d times to make it to be stationare and apply ARMA (p,q) after that. This is how ARIMA(p,d,q) was formed, with d denoting the number of times the series needs to be differenced [13].

3.1.4.4 Box Jenkins Methodology

The Box-Jenkins methodology comes in handy when building the ARIMA model [13]. It contains four consecutive steps, which are identification, estimation, diagnostic checking and forecasting.


1. Identification of the model
(Choosing tentative p, d, q)
2. Parameter estimation of the chosen model
3. Diagnostic checking
(are the estimated residuals white noise?)
Yes
(Go to Step 4)
No
(Return to Step 1)
4. Forecasting

The Box-Jenkins Methodology

Source: Gujarati (2008)

First, the identification process. The purpose of this step is to determine the optimal value of p, d, and q in the ARIMA model. The most common tools for p,d,q detection are the autocorrelation function (ACF) and the partial autocorrelation function (PACF) with their respective correlograms and AIC criteria. These are simply the plots of ACF and PACF against the lag length [13]. Even so, sometimes model identification for ARIMA is done by an auto procedure, which iteratively fits all possible model structures and then uses the goodness of fit statistic to select the best ARIMA model [10].

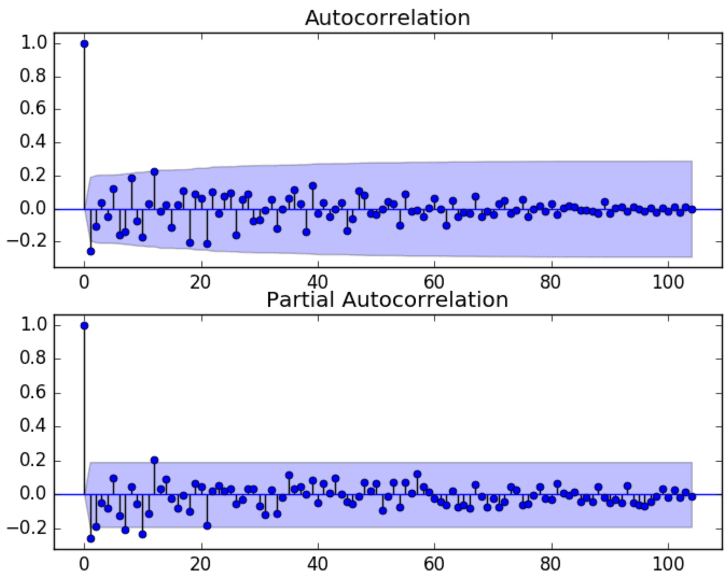


Figure 1.ACF and PACF Plots Example of Monthly Boston Robberies

Source: Machine Learning Mastery

Second, the model estimation. After identifying the optimal value of p,d, and q, the coefficients of ARIMA models are estimated. The calculation and the sum of residual squared is minimized by using least squares method. But, sometimes one will have to foster the non linear (in parameter) estimation method [13].
Third, the diagnostic checking. The purpose of this step is to check whether the estimated model fits the data reasonably well. The important element of the step is to make sure that the residuals estimated from the models are white noise, random, and normally distributed [10]. Another element is to check whether the parameters used are statistically significant. The fitting process usually follows the parsimony principle [13].
The fourth step is forecasting or predicting the desired period of time series. In many cases, the forecast obtained by ARIMA is more reliable than traditional econometric modeling, let alone for short term forecasts [13].

3.1.4.5 Limitation of ARIMA

Aside from its popularity and its success in catching linear time series patterns, the ARIMA model also has its own limitations [32]. ARIMA model is not heavily used in complex and dynamic domains because of its incapability of capturing non-linear patterns in time series data. However, this phenomenon should be investigated further [1].

3.1.5 Machine Learning

Machine learning can be defined as a subclass of artificial intelligence (AI) that uses statistical learning algorithms to assimilate data and provides the learned solution or response (sharma). In short, ML is the science of computers that can learn from data [12]. ML algorithms can be classified depending on the amount and type of supervision that need to be obtained during the training process.

1. Supervised Learning
   ML model that has target or output labels. Can be divided into regression and classification tasks.
2. Unsupervised Learning
   ML model that has no labels or targets. Usually used in feature reduction, clustering, and so on.

3. Reinforcement Learning
   ML that contains agents that can observe the environment, select actions and get reward in return.

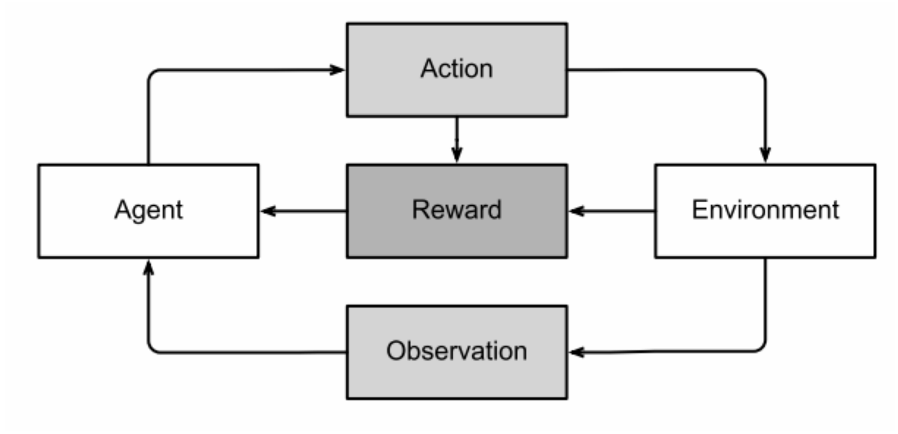


Figure 2. Reinforcement Learning

Source: Machine Learning and Deep Learning Applications-A Vision

3.1.6 Deep Learning

Deep learning is a specific subcategory of machine learning. The deep in deep learning tells the idea of successive layers representation. It is basically a network model with neurons and features. DL provides an automatic learning process and makes it more robust than the ML model. That is the reason why DL is suitable for dealing with large datasets and complex algorithms.

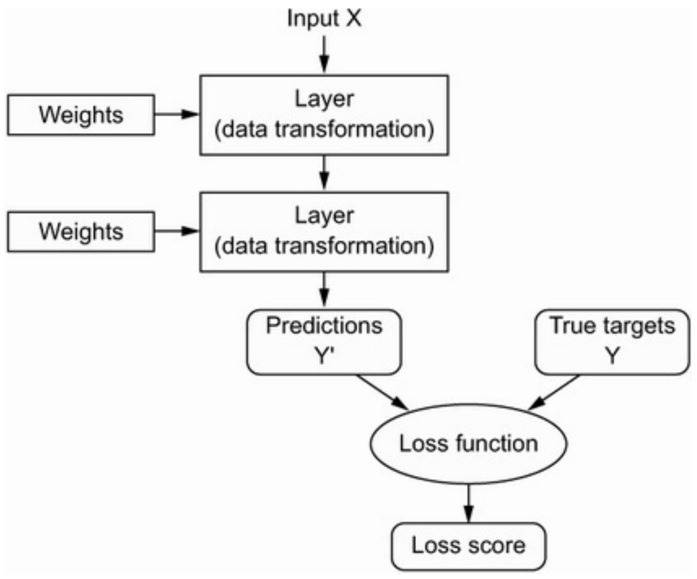


Figure 3. Deep learning model

Source: Deep Learning with Python, 2nd Edition

3.1.7 Artificial Neural Network

Neural Network (NN), which is also known as artificial neural network defined as a form of artificial intelligence (AI) or deep learning that tries to mimic how the human brain works [4]. ANN consists of units of neurons. [5]. Each ANN will contain input, hidden, and output layers. ANN might consist of one hidden layer, many hidden layers, or even no hidden layer [5]. A layer refers to a collection of one or more nodes which each node connects to other nodes in the next layer [8]. When each ANN contains multiple hidden layers, it can be called a deep neural network (DNN). Graph below depicts the example structure of ANN with three input features, one hidden layer with four nodes, and one output.

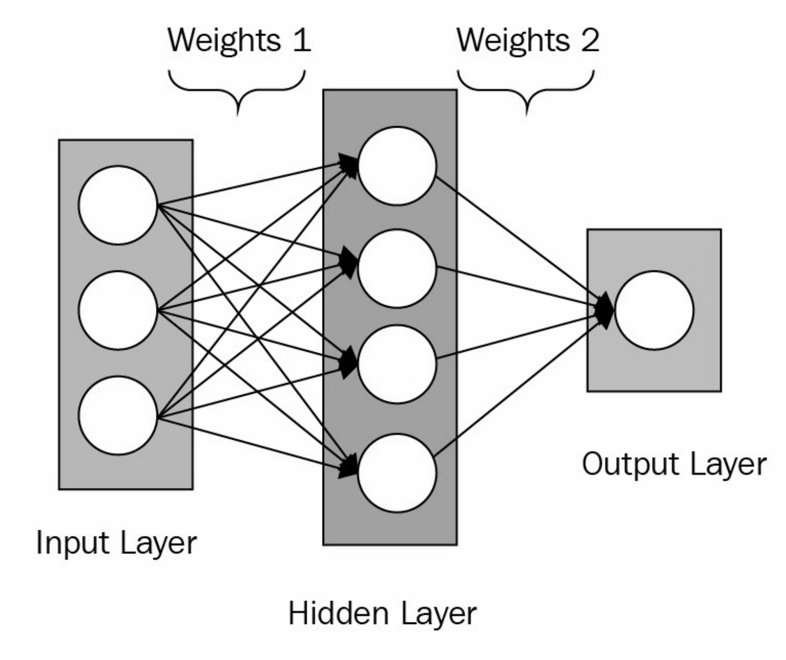


Figure 4. ANN Architecture
Source: Neural Network with Keras Cookbook

The number of nodes in output depends on what output that is trying to be predicted [8]. The neuron has the input of a set of features, which is noted as (x). These input features will affect the output. Next, these inputs are multiplied by weight given to each input feature (w_1, w_2,. .., w_n). This weight can be modified by a correction scalar or sometimes denoted as bias (b). The usual bias used in ANN is 1. After that, this input vector will pass through an activation function (f) that will produce the output. There are many options when it comes to activation functions, one can use ReLu, softmax, tanH or so forth. Finally, this output can either be the input to the neuron in subsequent or hidden layers or just be the final output [4].

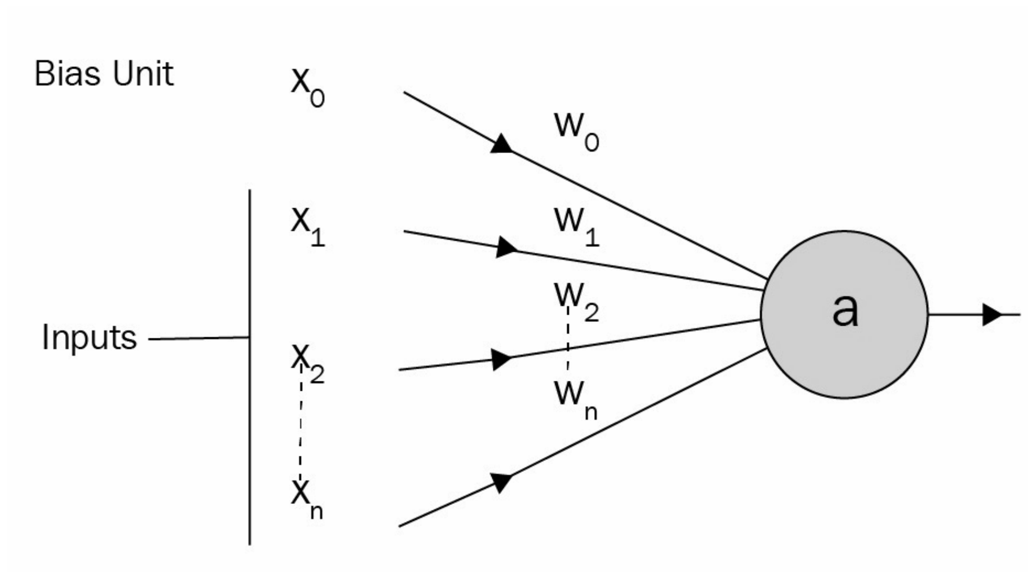


Figure 5. Network Structure

Source: Neural Network with Keras Cookbook

Suppose that a is one of the units in hidden layers. This a will be formulated as:

$$a = f\left(\sum_{i=0}^{N} w_i x_i + b\right)$$

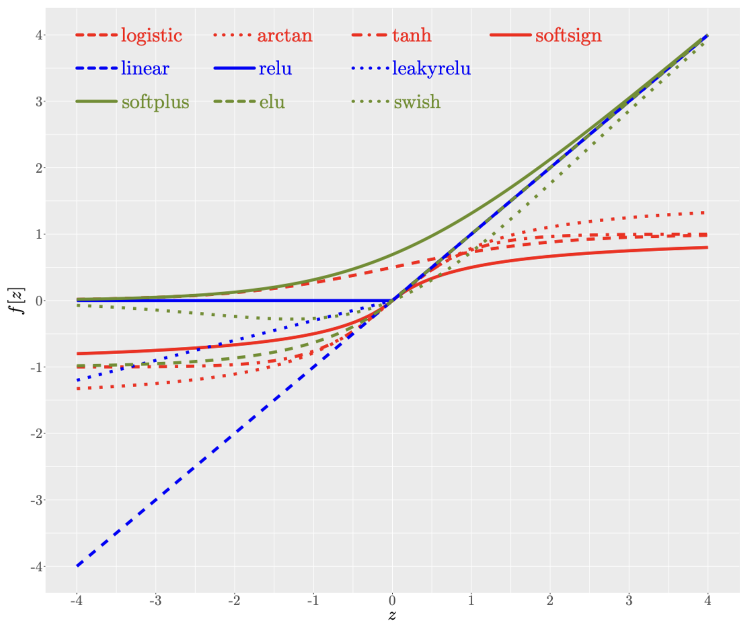


Figure 6. Activation Function in ANN

Source: Activation in ANN Systematic Overview

The number of input, hidden, output layers, and the activation function will define the topology or network architecture of the ANN [4]. According to Casas [4], there are three type of network architecture of ANN:

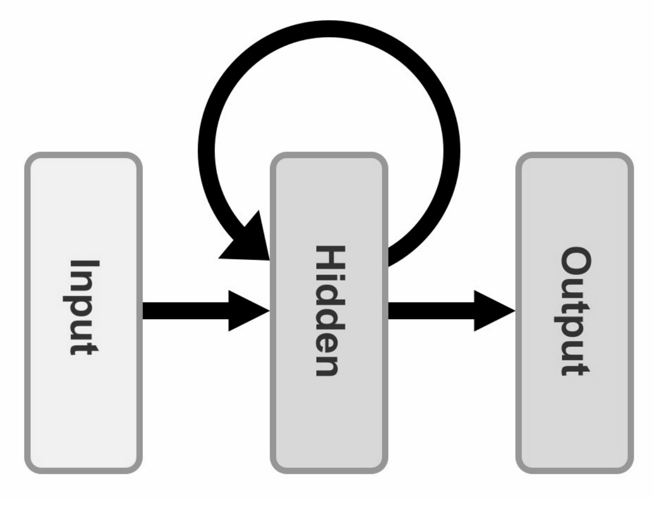
1. Single layer network
2. Multilayer network
3. Recurrent network

In the high level, ANN process follow these basic steps (neural n):

1. Get the dataset and separate them into training and testing set
2. Encode and transform the data.
3. Identify and build network architecture.
4. Train the network until it converges on the training set.
5. Test and evaluate the network on the testing set.

3.1.8 Recurrent Neural Network

Recurrent neural network is a type of ANN architecture that can address sequence-base problems in deep learning. It was first proposed with the invention of Hopfield Network in the 1980s. It utilizes sequential data, which is an ANN that unfolded through time [22]. Differ from standard feedforward networks, RNN uses an additional memory state, known as cyclical hidden states [19]. Therefore, it will make each step dependent on the previous one.



After one input is received at time step, hidden state will incorporate the data from input of previous sequence by combining both current input (x_t) and the previous hidden state (h_{t-1}) [22]. The process can be drawn as

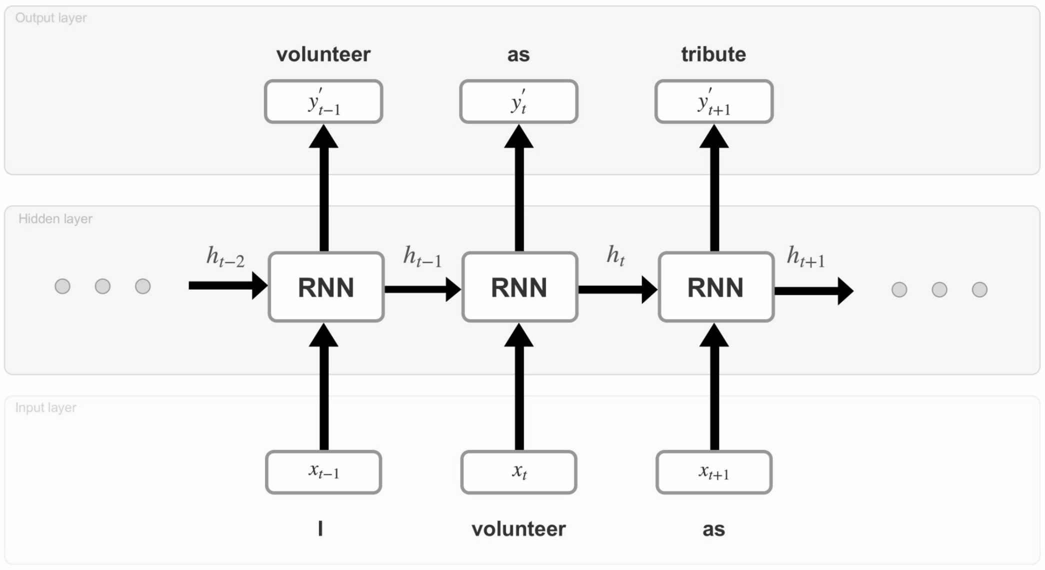


Figure 7. Unfolded RNN with One Hidden Layer

Source: Recurrent Neural Network with Python Quick Start Guide

Graph above depicts how RNN works and the combination if x_{t-1} + RNN + y'_{t-1} shows the process at each t-1 time step. At this step:

1. RNN conducts forward-propagation and computes the prediction error to obtain the loss on training and testing dataset.

$$h_t = Wg(h_{t-1}) + Ux_t$$
$$a_t = g(h_t)$$
$$z_t = V.a_t \, y_t = g(z_t)$$

2. Next, RNN will calculate the gradient descent at each layer and conduct the error back-propagation, across t-1 time step. After that, RNN will update the weights. Furthermore, it will loop around to another forward-propagation. [22]

$$\frac{\vartheta L}{\vartheta W} = \sum_{t=1}^{T} \frac{\vartheta L_t}{\vartheta W} = \sum_{t=1}^{T} \sum_{i=1}^{t} \frac{\vartheta L_t}{\vartheta y_t} \frac{\vartheta y_t}{\vartheta h_t} \left(\prod_{j=i+1}^{t} W^t diag(g'(h_{j-1})) \right) \frac{\vartheta h_i}{\vartheta W}$$

$$\frac{\vartheta h_i}{\vartheta h_i} = \sum_{t=1}^{T} \quad \frac{\vartheta L_t}{\vartheta W} = \prod_{t=1}^{t} \quad \frac{\vartheta h_t}{\vartheta h_{j-1}} W^t diag(g'(h_{j-1}))$$

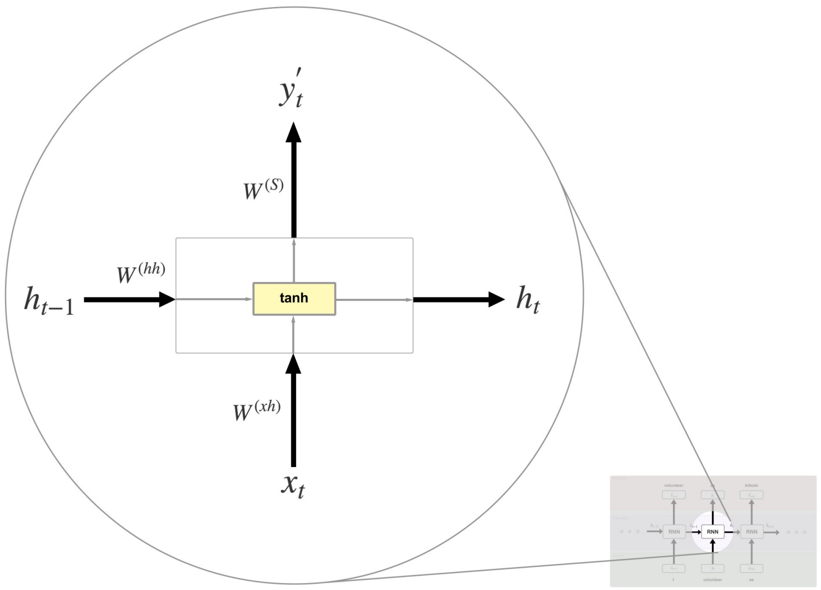


Figure 8. Training RNN

Source: Recurrent Neural Network with Python Quick Start Guide

3.1.9 Limitation of RNN

RNN performs significantly better and less expensive when working on complex tasks with large amounts of data, especially sequence-based problems such as time series [19]. However, it also has drawbacks such as the difficulty to build the right architecture on specific problems. Also, RNN does not yield the best result when data is small [19].

3.1.10 Long Short-Term Memory

According to Bengio [3], even though RNN exhibits good performance when dealing with sequential data, it will suffer from gradient descent and the error criterion may be inadequate to train the model in the long run. One solution that addresses the vanishing error problem is the gradient-based method, known as long short-term memory (LSTM) [3].

In contrast to RNN, the LSTM cell adds long-term memory. It makes small modifications to relevant information through gates that control information flow for each time step. The model then can selectively ignore the irrelevant information that is learnt through back propagation [22]. Figure below describes how LSTM works when there are two weights coming in from the past cell, (c and a at time t-1), and once transformed another two weights are going (c and a at the time t).

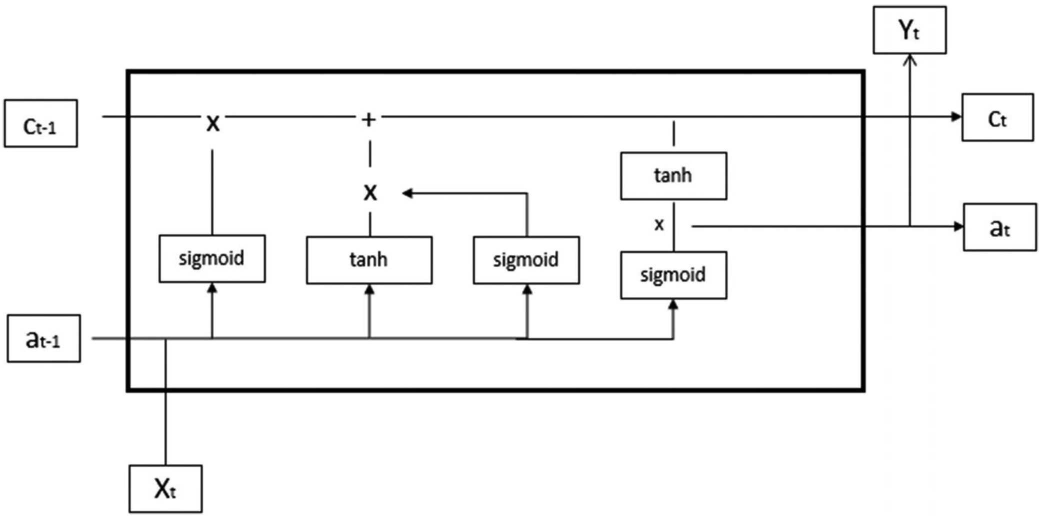


Figure 9. LSTM Cell

Source: Advanced Forecasting with Python

3.2 Current State Overview

This section contains prior research about stock market forecasting using ARIMA and RNN.

3.2.1 Stock Price Prediction Using ARIMA, Neural Network and LSTM Models

The first research is the stock price prediction using ARIMA, NN and LSTM models [14]. The dataset that is used in this paper is historical trading data from 2 January 2020 to 19 January 2021 from Bursa Malaysia closing price. The reason behind this period is, the number of COVID-19 cases increased dramatically in Malaysia, which also triggered a shock and anomaly movement of stock prices in 2020. To compare and evaluate the forecasting models, data is divided into two parts, 70 percent training set (data from 2 January 2020 to 28 September 2020), and the rest of 30 percent (29 September 2020 to 19 January 2021) held as a testing set. The forecasting models that applied to the research are ARIMA, NN, and LSTM. To choose the best model, MAPE and RMSE are chosen as evaluation metrics.

MAPE is the mean absolute percentage error. It is the mean absolute percentage deviation of each value with actual value.

$$\mathrm{MAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$$

A_t is the actual value and F_t is the predicted values. n is the number of observations.

Model	RMSE		MAPE	
	Training	Testing	Training	Testing
LSTM with 2 hidden layers	16.7985	17.5289	0.8322	0.8373
LSTM with 3 hidden layers	16.7170	16.8410	0.8192	0.8184

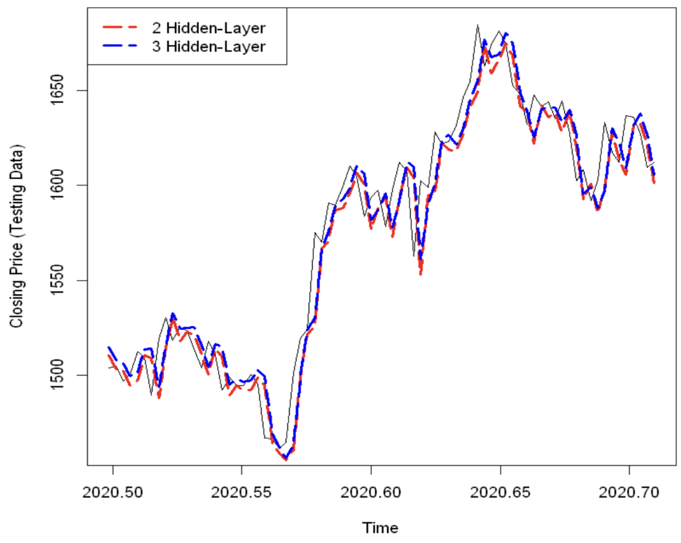


Figure 10. Stock prediction with LSTM

Based on research findings, compared to ARIMA and NN, the LSTM model has the best performance in Bursa Malaysia stock price prediction because it has the smallest MAPE and RMSE values. It is not only able to generate more than 90% of accuracy but also able to explain unpredictable movement of stock prices during this pandemic period.

3.2.2 ARIMA vs LSTM on NASDAQ Stock Exchange Data

The next research is the ARIMA vs LSTM on NASDAQ stock exchange data [17]. Nine of the most popular NASDAQ sectors are chosen, which are IT, automotive, financial, logistics and transport, clothing, food, energy, healthcare, and entertainment along with media. Both ARIMA and LSTM models are used to predict daily or monthly average prices from 2008 to 2021 of the chosen companies. MSE and MAPE were selected as evaluation metrics.

Once you have the absolute percent error for each data entry, you can calculate the MAPE. Add all the absolute percent errors together and divide the sum by the number of errors. For example, if your dataset included 12 entries, you would divide the sum by 12. The final result is the MAPE.

time window	ARIMA (MAPE)	LSTM (MAPE)
1 day	1.64	**1.46**
30 days	**1.64**	5.52
1 month	**4.28**	6.90
3 months	**5.93**	10.53
9 months	**7.55**	16.05

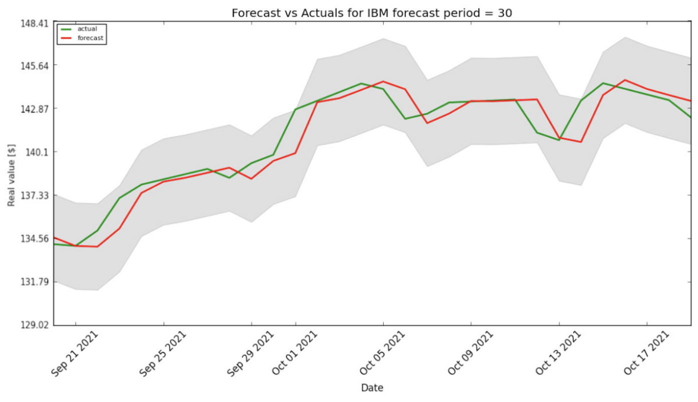


Fig. 3. Prediction for IBM company using ARIMA, 30-day period (zoomed)

Figure 11. ARIMA Predictions chart

The analysis result showed that ARIMA model performs better than LSTM when it is univariate models, only one feature used. The p and q on the ARIMA model were ranged from 0 to 2. For the LSTM, adam optimizer and tanh activation function were used. When forecasting the stock price for 30 days, ARIMA is about 3.4 times better than LSTM. When predicting an average of 3 months, ARIMA was about 1.8 times better than LSTM. Also, when forecasting the average stock price for next 9 months, ARIMA was about 2.1 times better than LSTM.

3.2.3 A Comparison Between Econometric Modeling and Long- Short Term Memory

Another prior research is a comparison between econometric modeling and LSTM [8]. The paper proposed the study to predict the price of stock return within the main index of Romanian stock market. The goal was to explore the possibility of adapting a ML model for the Romanian market. The data is collected from daily returns of BRD stocks that were listed on the Romanian stock market in 2001. The time period was from 16 January 2001 to 4 November 2016. The sample size was 3,809 data points. This sample covered a 15 year stock return that allowed capturing the effect of the financial crisis that began in 2007.

The result of this research concluded that the best activation-optimization to use was the Softsign ADAM combination. However, although the NN model outperformed the classical model of TARCH, the author believed that was due to the low number of features given as input parameters. Based on this consideration, they proposed to build and analyze how a neural network can be trained to learn the dynamics of an entire market, not just the evolution of one stock price.

3.2.4 Stock Price Prediction with ARIMA and Deep Learning Models

The next prior paper that takes into consideration is stock price prediction with ARIMA and deep learning models [11]. Data used in the study is close price movement for 30 listed stocks from the Dow Jones Industrial Average (DIJA). Data was obtained from yahoo finance. For the ARIMA model, the input was a dataset from January 2016 to December 2017. For deep learning models, the training set was taken from January 2006 to December 2015. And, the rest of January 2016 to December 2017 was used as a testing set. The Seq2Seq model algorithm is expressed as below graph.

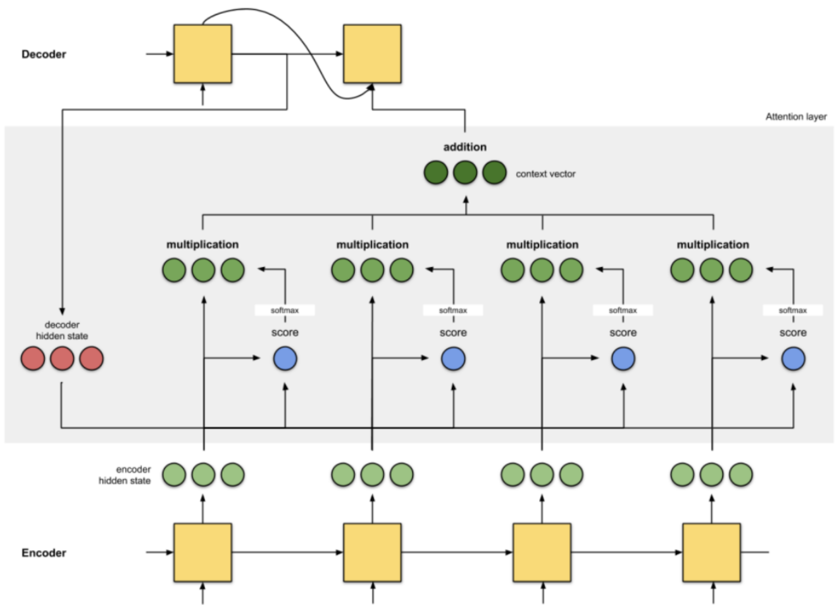


Figure 12. LSTM Architecure in stock prediction using ARIMA and deep learning models

The study concluded that LSTM was proven to be more reliable than vanilla ANN and RNN. Built upon the LSTM model, Seq2Seq models proposed in the study was an excellent predictor for stock prices of several days in the future with even lower mean squared error. In addition, the study also stated that the increase in the complexity of NN will generate better performance.

3.2.5 RMSE

The Root Mean Squared Error (RMSE) is one of the main performance indicators for a neural network regression model. It measures the average difference between values predicted by a model and the actual values. It provides an estimation of how well the model is able to predict the dependent variable.

The lower the rmse value the better the model is; infact a perfect model will have RMSE value equal to 0.

$$RMSE = \sqrt{\frac{SSE_W}{W}} = \sqrt{\frac{1}{W}\sum_{i=1}^{N} w_i u_i^2}$$

wi: weight of the observation, ui : error of observation ,N: number of observation,W: Total weight

3.2.6 Comparison of ARIMA, ANN and LSTM for Stock Price Prediction

Another interesting research that will be taken into account is a study of comparison of ARIMA, ANN, and LSTM for stock prediction [22]. This research tried to forecast DELL's stock price around 2010. The model choices for the study were ARIMA (1,0,0) and ANN (10,17,1).

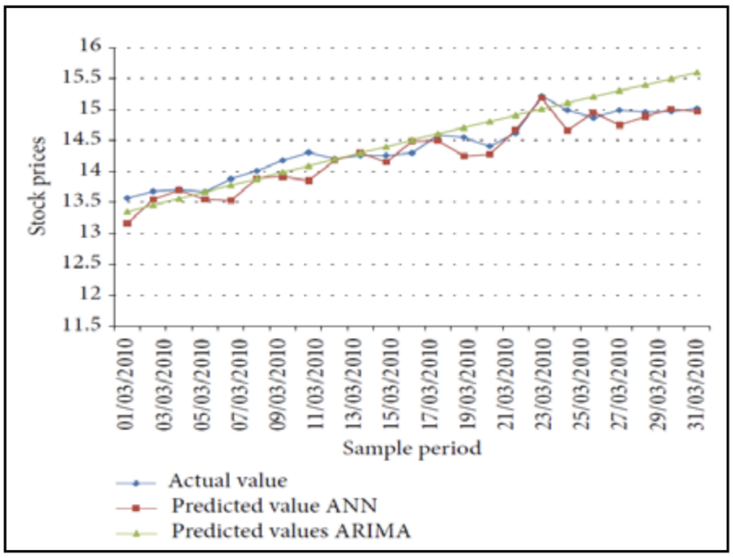


Figure 13. ARIMA ,ANN Predicted plot from comparative study

The conclusion that can be drawn from the study is that the ANN model generated better performance than that of the ARIMA model. Next, the performance of the LSTM model may be more due to the ANN. Last, the ARIMA-GARCH model actually can further improve the accuracy of the ARIMA model by improving the white noise sequence.

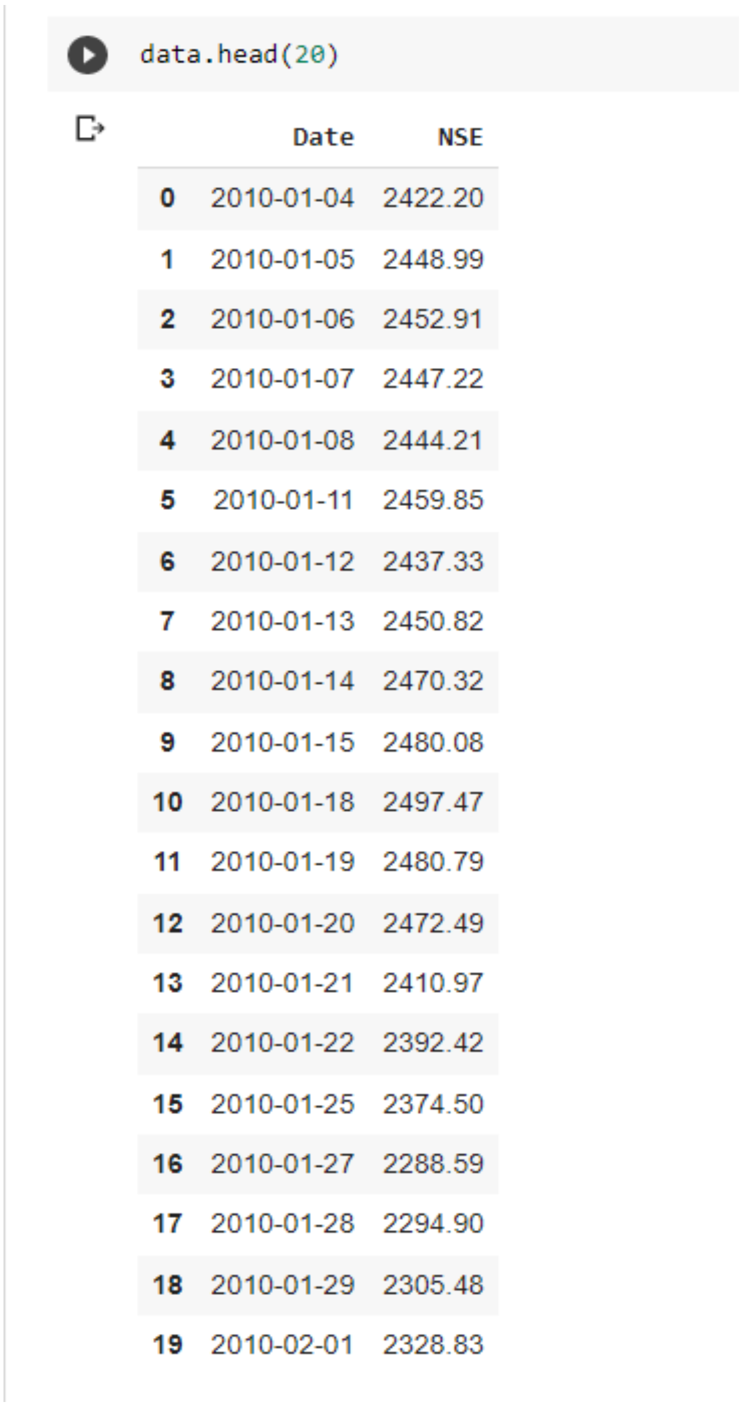
4. Model Implementation

4.1 DEEP RNN Implementation

Previous research sheds the light for the research to go forward using the historical data points. The new research will take the historical data points in batches and predict the index based on it. The research will try to have similar inputs to both the models in order to compare their performance. Since both are predictive models and predicted value are real numbers loss function can be the best measure of model performance.

4.1.1 Dataset

The dataset for the research was taken from nseindia.com, the national stock exchange of India. The daily index of the stock is called NSE index and the NSE index daily view data is downloaded for the purpose of the research. The start date of the NSE index is 04/01/2010 and the end date is 01/01/2021.



```
data.head(20)
```

```
          Date      NSE
0   2010-01-04  2422.20
1   2010-01-05  2448.99
2   2010-01-06  2452.91
3   2010-01-07  2447.22
4   2010-01-08  2444.21
5   2010-01-11  2459.85
6   2010-01-12  2437.33
7   2010-01-13  2450.82
8   2010-01-14  2470.32
9   2010-01-15  2480.08
10  2010-01-18  2497.47
11  2010-01-19  2480.79
12  2010-01-20  2472.49
13  2010-01-21  2410.97
14  2010-01-22  2392.42
15  2010-01-25  2374.50
16  2010-01-27  2288.59
17  2010-01-28  2294.90
18  2010-01-29  2305.48
19  2010-02-01  2328.83
```

Figure 14. NSE Data first 20 values

4.1.2 Attributes

The two attributes of the dataset are date and NSE index.

Date
Format: Datetime

NSE Index
Format: decimal

4.1.3 Data Pre-Processing

Data pre-processing refers to a series of operations performed on raw data to prepare it for analysis, modeling, and decision-making. It is an important step in the data science process as the quality and structure of the data can significantly impact the accuracy and usefulness of the results.
Common steps involved in data pre-processing are data cleaning, data transformation, data integration, data reduction.
Data pre-processing is a crucial step in the data science process, as it can significantly affect the results of the analysis and the performance of machine learning models. By pre-processing the data, you can ensure that the data is of high quality and that the results of your analysis are reliable and accurate.
The libraries used for data pre-processing are described below.
Pandas: Pandas is a powerful and flexible open-source library for data analysis and manipulation in Python. It provides easy-to-use data structures and data analysis tools for handling and manipulating numerical tables and time-series data. The two primary data structures in Pandas are the Series and DataFrame.
Numpy: NumPy is a powerful and popular library for scientific computing in Python. It provides support for arrays and matrices, which are essential for numerical and scientific computing. The primary data structure in NumPy is the ndarray, which stands for N-dimensional array.

4.1.4 Code

Google colab was selected as the IDE environment. It has majority of the python libraries pre installed and is one of the best environment to run neural network models. It works with cloud GPU and computer processor. Google colab is also integrated with google drive and it has an auto save feature that lets us save the work real time in the google drive. Google colab also lets us download the files as ipynb notebooks or .py files.
The required libraries need to be imported initially to the colab environment we are working.

```
# Loading all necessary libraries

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN
from keras.layers import LSTM
from keras.layers import Dropout
from keras.optimizers import Adam
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

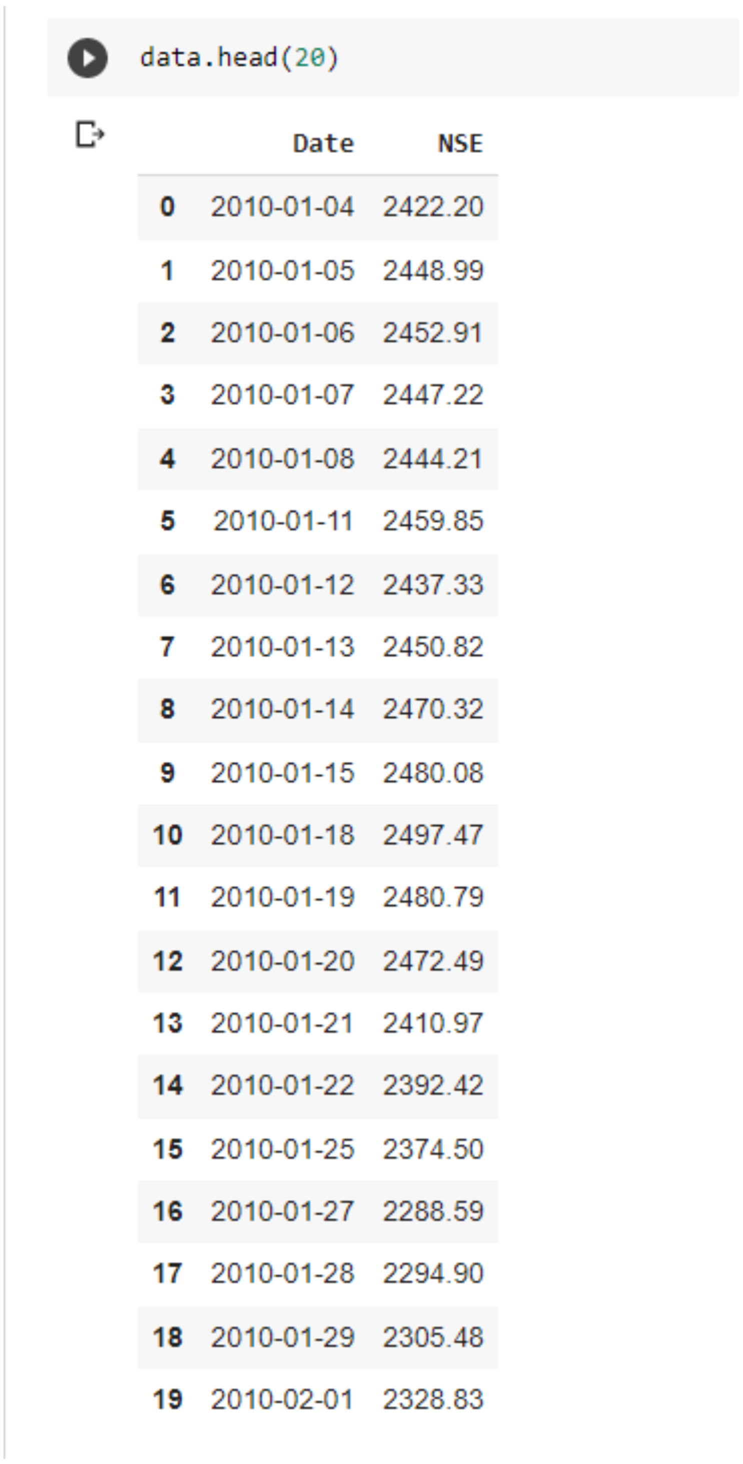
Pandas library was imported as pd.
Numpy was imported as np. The graphical library matplotlib was imported as plt.
Keras is an open-source software library that provides a Python interface for ANNs (Artificial Neural Networks). It acts as an interface for the TensorFlow library. It was developed to enable fast experimentation with deep learning models.
Keras has a user-friendly API that makes it easy to create and train neural networks. It supports convolutional networks (for image classification), recurrent networks (for sequence processing), and combinations of the two.
Keras provides several high-level libraries for building recurrent neural networks (RNNs), including simpleRNN, LSTM, and GRU.

The Dataset excel file named NSE.xlsx is first imported to the google colab. Then the excel file is loaded as a dataframe using Pandas function shown below.

```
# Loading data set
data = pd.read_csv('NSE.csv')
data['Date'] = pd.to_datetime(data['Date'])
```

The date column is set to datetime format from object format.

The head of the data with 20 values is inspected with the code.


```
data.head(20)
```

```
          Date      NSE
0   2010-01-04  2422.20
1   2010-01-05  2448.99
2   2010-01-06  2452.91
3   2010-01-07  2447.22
4   2010-01-08  2444.21
5   2010-01-11  2459.85
6   2010-01-12  2437.33
7   2010-01-13  2450.82
8   2010-01-14  2470.32
9   2010-01-15  2480.08
10  2010-01-18  2497.47
11  2010-01-19  2480.79
12  2010-01-20  2472.49
13  2010-01-21  2410.97
14  2010-01-22  2392.42
15  2010-01-25  2374.50
16  2010-01-27  2288.59
17  2010-01-28  2294.90
18  2010-01-29  2305.48
19  2010-02-01  2328.83
```

As we could see the dataset contains date in one column named “Date” and NSE index in the other column named “NSE”.Total number rows are counted using the len(data) and printed using the print() function.

```
print('Total number of rows in data are: ', len(data))
```

Then a function drop_outlier_IQR is defined and is used to detect outliers that are below 1.5 times interquartile range and above 1.5 times the interquartile range.

```
def drop_outliers_IQR(df):
   q1=df.quantile(0.25)
   q3=df.quantile(0.75)
   IQR=q3-q1
   not_outliers = df[~((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]
   outliers_dropped = not_outliers.dropna().reset_index()
   return outliers_dropped['NSE']

data['NSE'] = drop_outliers_IQR(data['NSE'])
```

Total number of rows removed are zero and total number of rows after outlier was found to be 2977.

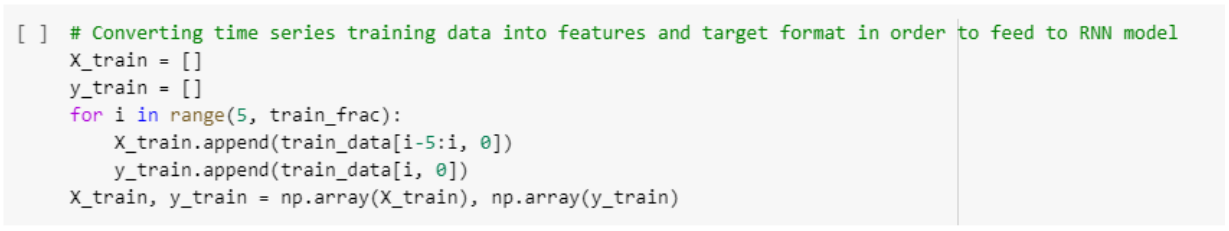
The training set values need to be transformed. Sklearn.preprocessing library is used to transform. This training set is then scaled using minmax scaler. The value is transformed between 0 and 1.

```
# Reshaping the data in order to perform scaling

training_set = data['NSE'].values.reshape(-1,1)
sc = MinMaxScaler()
training_set_scaled = sc.fit_transform(training_set)
```

The next step is to change the dataset into train , validatin and test data. Train set is split to train the model. Training set is made and is 64% of the total set. Validation set is taken at 16% of the total data and Test data is about 20% of the whole data.

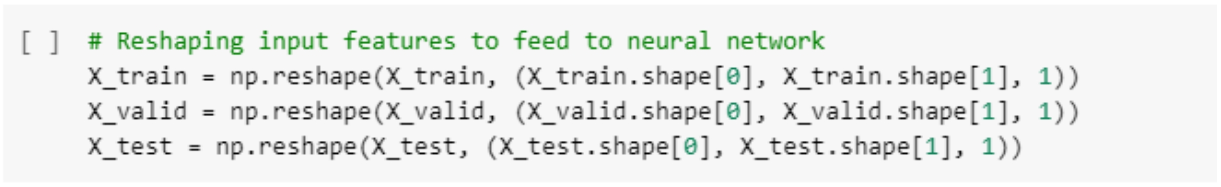
```
# Generating fraction for train-validation-test split

train_frac = int(len(training_set_scaled)*64/100)
train_test_threshold = int(len(training_set_scaled)*80/100)
train_data = training_set_scaled[0:train_frac]
validation_data = training_set_scaled[train_frac:train_test_threshold]
test_data = training_set_scaled[train_test_threshold:]
```

The model is trained using 5 past values and the next value is predicted. In order to get this two np array are created and the X_train has 5 past values and y_train has the next value.



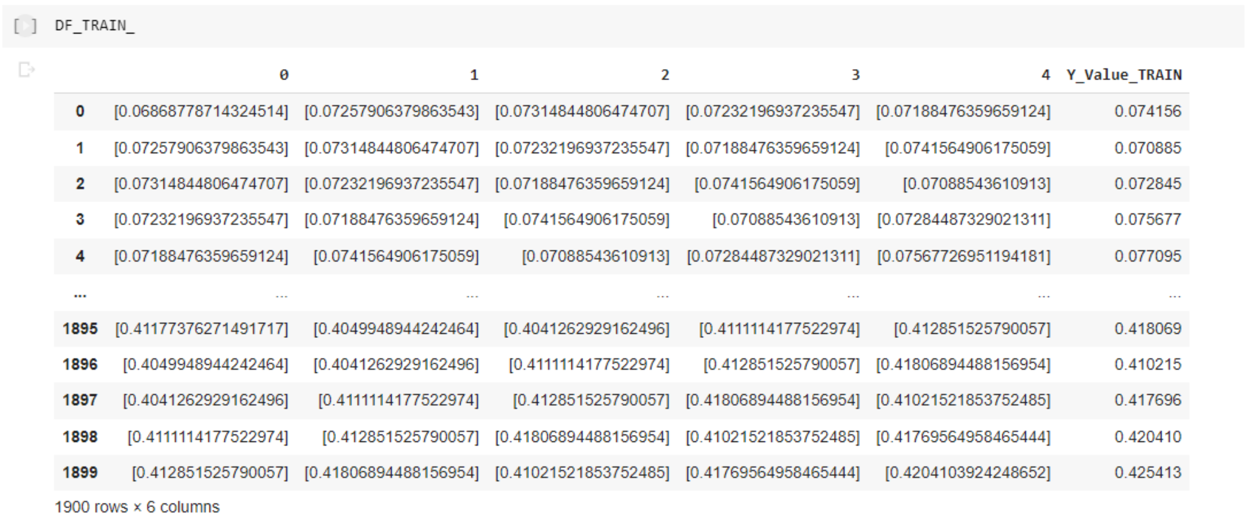
```python
# Converting time series training data into features and target format in order to feed to RNN model
X_train = []
y_train = []
for i in range(5, train_frac):
    X_train.append(train_data[i-5:i, 0])
    y_train.append(train_data[i, 0])
X_train, y_train = np.array(X_train), np.array(y_train)
```

This array is then reshaped to feed into the RNN layer.



```python
# Reshaping input features to feed to neural network
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_valid = np.reshape(X_valid, (X_valid.shape[0], X_valid.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

The DF Dataframe is plotted below. There are 5 X values and Y value.



```
DF_TRAIN_
```

```
                         0                      1                      2                      3                      4  Y_Value_TRAIN
   0  [0.06868778714324514]  [0.07257906379863543]  [0.07314844806474707]  [0.07232196937235547]  [0.07188476359659124]       0.074156
   1  [0.07257906379863543]  [0.07314844806474707]  [0.07232196937235547]  [0.07188476359659124]   [0.0741564906175059]       0.070885
   2  [0.07314844806474707]  [0.07232196937235547]  [0.07188476359659124]   [0.0741564906175059]    [0.07088543610913]       0.072845
   3  [0.07232196937235547]  [0.07188476359659124]   [0.0741564906175059]    [0.07088543610913]  [0.07284487329021311]       0.075677
   4  [0.07188476359659124]   [0.0741564906175059]    [0.07088543610913]  [0.07284487329021311]  [0.07567726951194181]       0.077095
 ...                    ...                    ...                    ...                    ...                    ...            ...
1895  [0.41177376271491717]   [0.4049948944242464]   [0.4041262929162496]   [0.4111114177522974]    [0.412851525790057]       0.418069
1896   [0.4049948944242464]   [0.4041262929162496]   [0.4111114177522974]    [0.412851525790057]  [0.41806894488156954]       0.410215
1897   [0.4041262929162496]   [0.4111114177522974]    [0.412851525790057]  [0.41806894488156954]  [0.41021521853752485]       0.417696
1898   [0.4111114177522974]    [0.412851525790057]  [0.41806894488156954]  [0.41021521853752485]  [0.41769564958465444]       0.420410
1899    [0.412851525790057]  [0.41806894488156954]  [0.41021521853752485]  [0.41769564958465444]   [0.4204103924248652]       0.425413
1900 rows × 6 columns
```

4.1.5 Model Building

The Model architecture of the Deep RNN model is shows below. The First layer A wih 7 RNN layers, second layer B with 7 RNN layers and the final dense layer with Relu as activation function to provide the output.

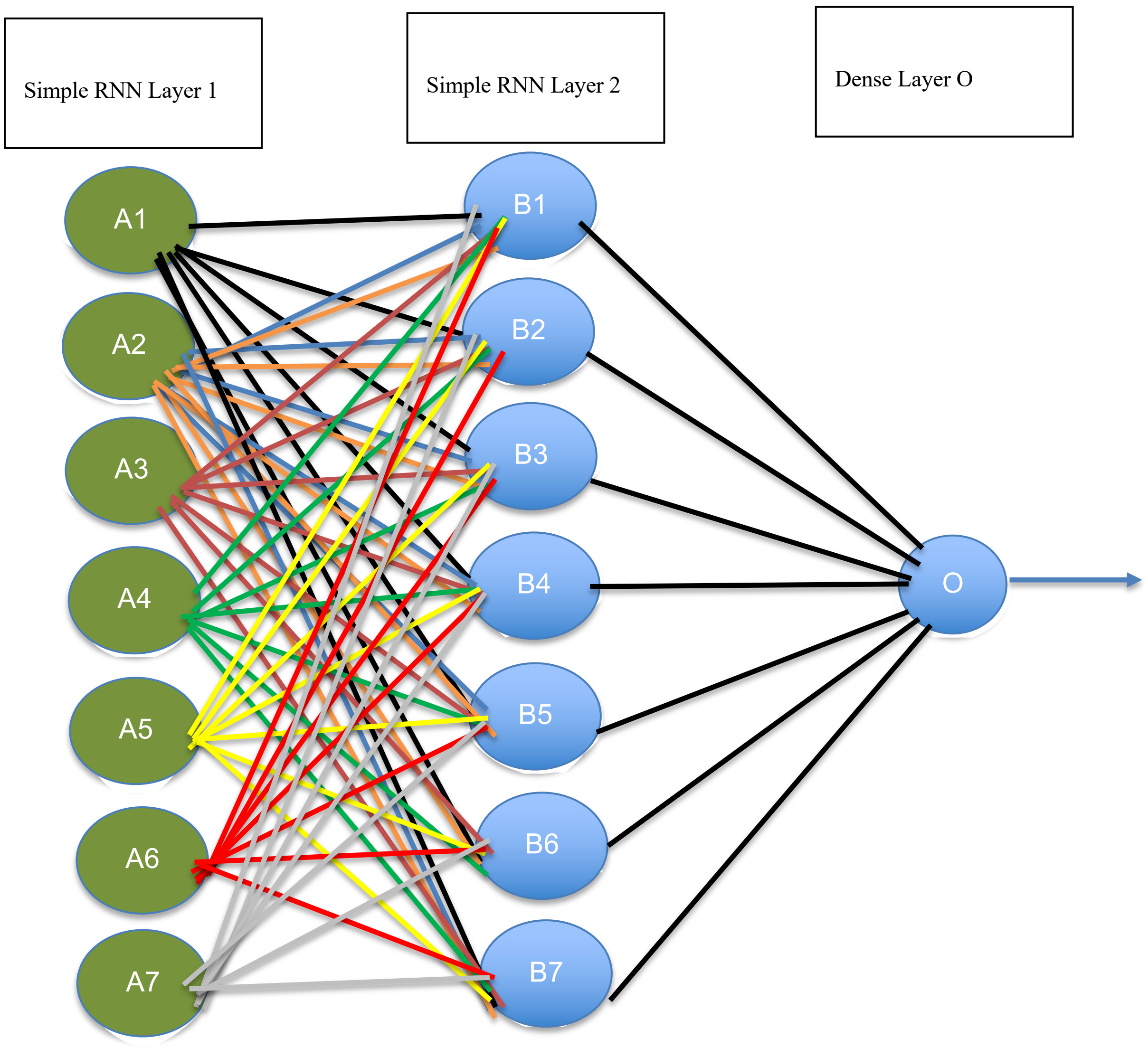
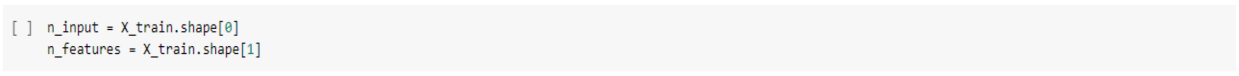
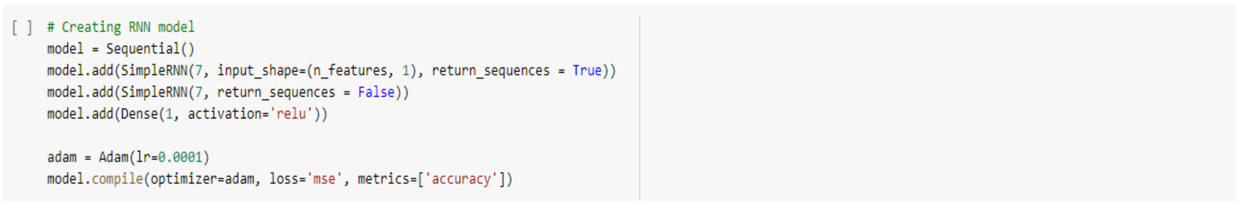


Figure 15. Model RNN Architecture



```
[ ] n_input = X_train.shape[0]
    n_features = X_train.shape[1]
```



```
[ ] # Creating RNN model
    model = Sequential()
    model.add(SimpleRNN(7, input_shape=(n_features, 1), return_sequences = True))
    model.add(SimpleRNN(7, return_sequences = False))
    model.add(Dense(1, activation='relu'))

    adam = Adam(lr=0.0001)
    model.compile(optimizer=adam, loss='mse', metrics=['accuracy'])
```

The Recurrent neural network model is built in the above code. Model is a sequential model with 3 layers. First layer comprises of Simple RNN layer with 7 neurons. Input shape (5,1) is passed into the SimpleRNN and the sequence is returned.

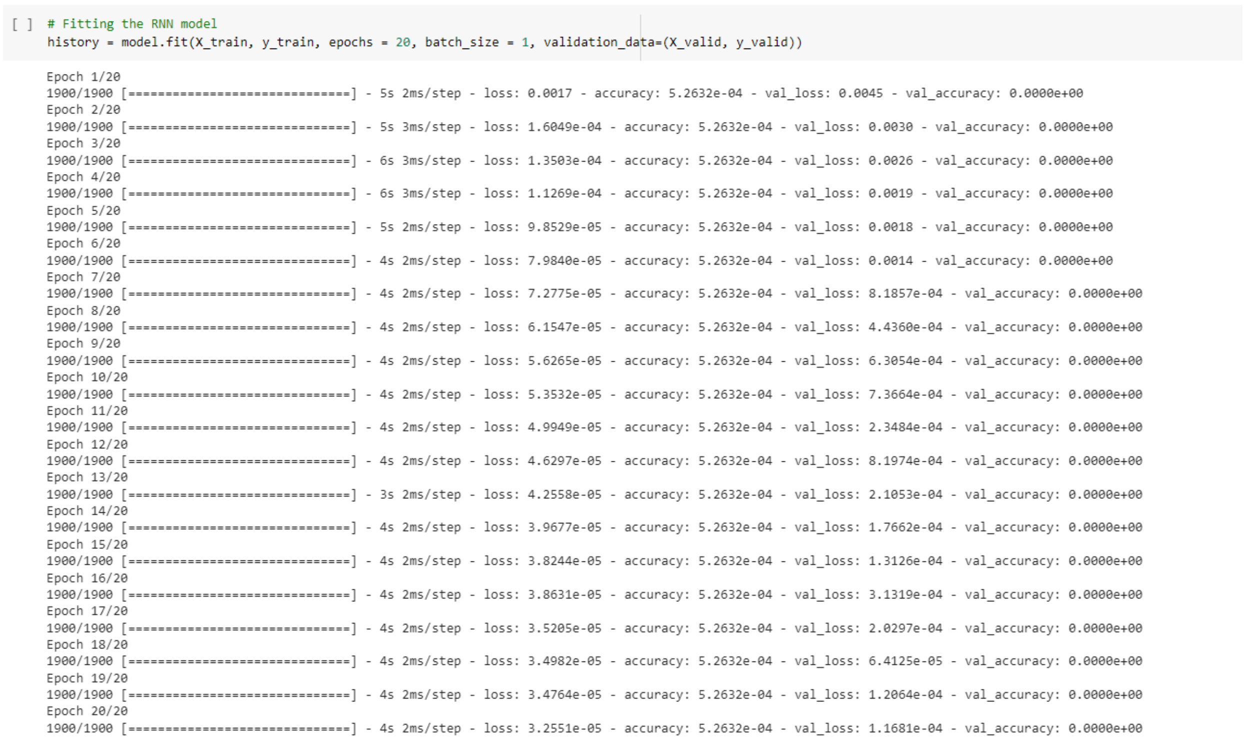
The next layer is again similar layer of 7 SimpleRNN neurons and the last year is a dense layer with activation function relu.

The model optimizer is set as adam and the learning rate is at 0.0001. loss is measured as Mean squared error (MSE).

Now the model is complete and ready for inputs.

The model is fitted into the train data first. For that mode.ft() function is used as assigned to a string called history.

X_train and y_train are inputted into the model. Epoch size is first selected as 20 with batch size of 1. Epoch size is the number of time data is passed through the model. Validation data is also selected for the model with validation_data(X_valid,y_valid) function.



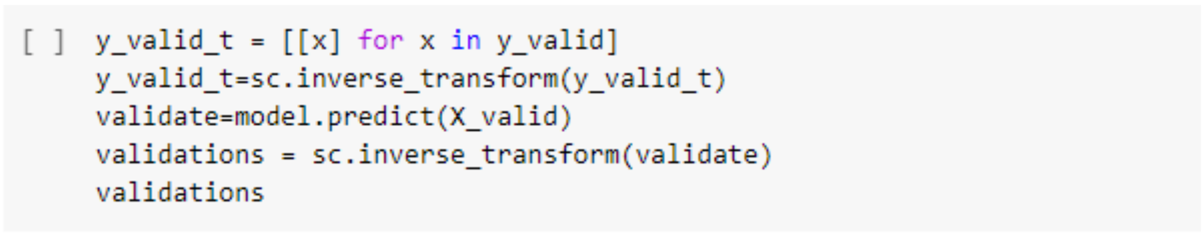
```
[ ] # Fitting the RNN model
    history = model.fit(X_train, y_train, epochs = 20, batch_size = 1, validation_data=(X_valid, y_valid))

    Epoch 1/20
    1900/1900 [==============================] - 5s 2ms/step - loss: 0.0017 - accuracy: 5.2632e-04 - val_loss: 0.0045 - val_accuracy: 0.0000e+00
    Epoch 2/20
    1900/1900 [==============================] - 5s 3ms/step - loss: 1.6049e-04 - accuracy: 5.2632e-04 - val_loss: 0.0030 - val_accuracy: 0.0000e+00
    Epoch 3/20
    1900/1900 [==============================] - 6s 3ms/step - loss: 1.3503e-04 - accuracy: 5.2632e-04 - val_loss: 0.0026 - val_accuracy: 0.0000e+00
    Epoch 4/20
    1900/1900 [==============================] - 6s 3ms/step - loss: 1.1269e-04 - accuracy: 5.2632e-04 - val_loss: 0.0019 - val_accuracy: 0.0000e+00
    Epoch 5/20
    1900/1900 [==============================] - 5s 2ms/step - loss: 9.8529e-05 - accuracy: 5.2632e-04 - val_loss: 0.0018 - val_accuracy: 0.0000e+00
    Epoch 6/20
    1900/1900 [==============================] - 4s 2ms/step - loss: 7.9840e-05 - accuracy: 5.2632e-04 - val_loss: 0.0014 - val_accuracy: 0.0000e+00
    Epoch 7/20
    1900/1900 [==============================] - 4s 2ms/step - loss: 7.2775e-05 - accuracy: 5.2632e-04 - val_loss: 8.1857e-04 - val_accuracy: 0.0000e+00
    Epoch 8/20
    1900/1900 [==============================] - 4s 2ms/step - loss: 6.1547e-05 - accuracy: 5.2632e-04 - val_loss: 4.4360e-04 - val_accuracy: 0.0000e+00
    Epoch 9/20
    1900/1900 [==============================] - 4s 2ms/step - loss: 5.6265e-05 - accuracy: 5.2632e-04 - val_loss: 6.3054e-04 - val_accuracy: 0.0000e+00
    Epoch 10/20
    1900/1900 [==============================] - 4s 2ms/step - loss: 5.3532e-05 - accuracy: 5.2632e-04 - val_loss: 7.3664e-04 - val_accuracy: 0.0000e+00
    Epoch 11/20
    1900/1900 [==============================] - 4s 2ms/step - loss: 4.9949e-05 - accuracy: 5.2632e-04 - val_loss: 2.3484e-04 - val_accuracy: 0.0000e+00
    Epoch 12/20
    1900/1900 [==============================] - 4s 2ms/step - loss: 4.6297e-05 - accuracy: 5.2632e-04 - val_loss: 8.1974e-04 - val_accuracy: 0.0000e+00
    Epoch 13/20
    1900/1900 [==============================] - 3s 2ms/step - loss: 4.2558e-05 - accuracy: 5.2632e-04 - val_loss: 2.1053e-04 - val_accuracy: 0.0000e+00
    Epoch 14/20
    1900/1900 [==============================] - 4s 2ms/step - loss: 3.9677e-05 - accuracy: 5.2632e-04 - val_loss: 1.7662e-04 - val_accuracy: 0.0000e+00
    Epoch 15/20
    1900/1900 [==============================] - 4s 2ms/step - loss: 3.8244e-05 - accuracy: 5.2632e-04 - val_loss: 1.3126e-04 - val_accuracy: 0.0000e+00
    Epoch 16/20
    1900/1900 [==============================] - 4s 2ms/step - loss: 3.8631e-05 - accuracy: 5.2632e-04 - val_loss: 3.1319e-04 - val_accuracy: 0.0000e+00
    Epoch 17/20
    1900/1900 [==============================] - 4s 2ms/step - loss: 3.5205e-05 - accuracy: 5.2632e-04 - val_loss: 2.0297e-04 - val_accuracy: 0.0000e+00
    Epoch 18/20
    1900/1900 [==============================] - 4s 2ms/step - loss: 3.4982e-05 - accuracy: 5.2632e-04 - val_loss: 6.4125e-05 - val_accuracy: 0.0000e+00
    Epoch 19/20
    1900/1900 [==============================] - 4s 2ms/step - loss: 3.4764e-05 - accuracy: 5.2632e-04 - val_loss: 1.2064e-04 - val_accuracy: 0.0000e+00
    Epoch 20/20
    1900/1900 [==============================] - 4s 2ms/step - loss: 3.2551e-05 - accuracy: 5.2632e-04 - val_loss: 1.1681e-04 - val_accuracy: 0.0000e+00
```

Figure 16. Fitting of RNN Model

The model has now completed the 20 epoch runs of the model with the training data.

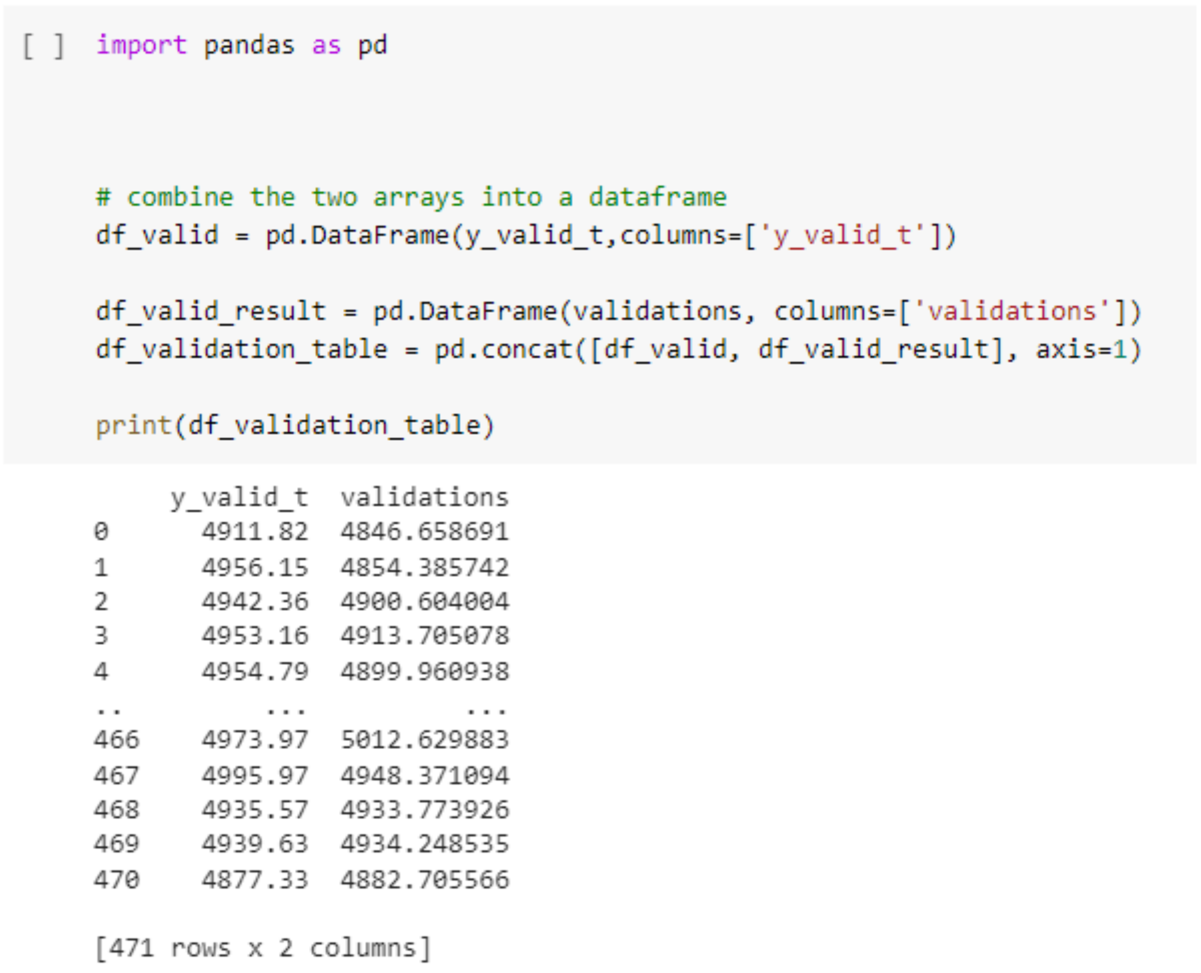
We need the predicted values for validation data. The validation data predicted are in scaled format and need to do inverse transform to get the values in original scale.

sc.inverse_transform() function is used to get the values in original scale.



```
[ ] y_valid_t = [[x] for x in y_valid]
    y_valid_t=sc.inverse_transform(y_valid_t)
    validate=model.predict(X_valid)
    validations = sc.inverse_transform(validate)
    validations
```

Now the actual validation data and predicted validations data are concatenated to form a validation table. The validation table is given below.



```
[ ]  import pandas as pd



     # combine the two arrays into a dataframe
     df_valid = pd.DataFrame(y_valid_t,columns=['y_valid_t'])

     df_valid_result = pd.DataFrame(validations, columns=['validations'])
     df_validation_table = pd.concat([df_valid, df_valid_result], axis=1)

     print(df_validation_table)
```

```
          y_valid_t  validations
     0      4911.82  4846.658691
     1      4956.15  4854.385742
     2      4942.36  4900.604004
     3      4953.16  4913.705078
     4      4954.79  4899.960938
     ..         ...          ...
     466    4973.97  5012.629883
     467    4995.97  4948.371094
     468    4935.57  4933.773926
     469    4939.63  4934.248535
     470    4877.33  4882.705566

     [471 rows x 2 columns]
```

Figure 17. Validation predicted vs actual

The root mean squared error (RSME) is calculated and found to be 74.40. The loss vs epoch graph is plotted. As we could see from the graph the two lines converges first at 17 epoch size. This is the optimal epoch size which we identified to get improved prediction of the model measured in rsme.

```
[ ]  from sklearn.metrics import mean_squared_error

     # calculate the RMSE
     rmse = np.sqrt(mean_squared_error(df_validation_table['y_valid_t'], df_validation_table['validations']))
     print(rmse)

     74.40736640176567
```

```
[ ]  # Plotting Loss v/s Accuracy

     plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.title('model loss')
     plt.ylabel('loss')
     plt.xlabel('epoch')
     plt.legend(['train', 'validation'], loc='upper left')
     plt.show()
```

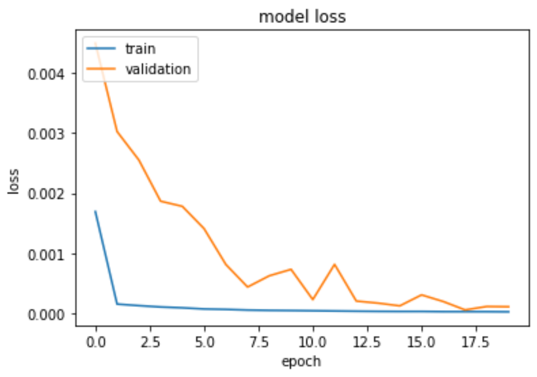
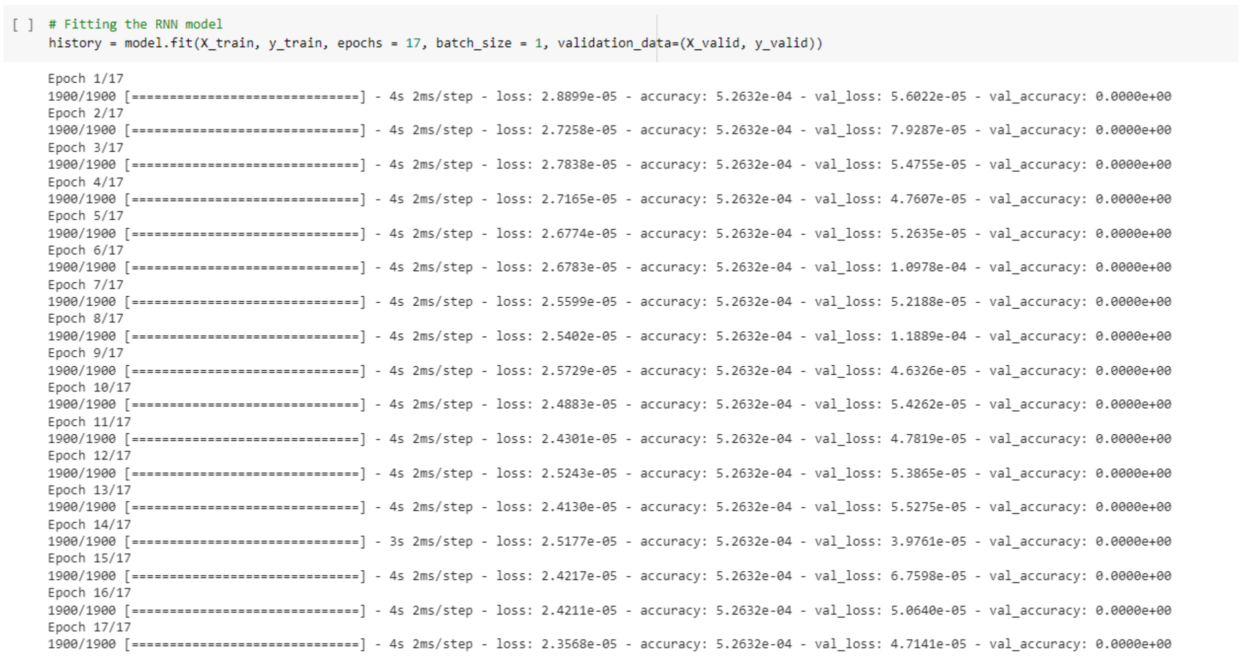


Figure 18. Epoch vs Loss diagram

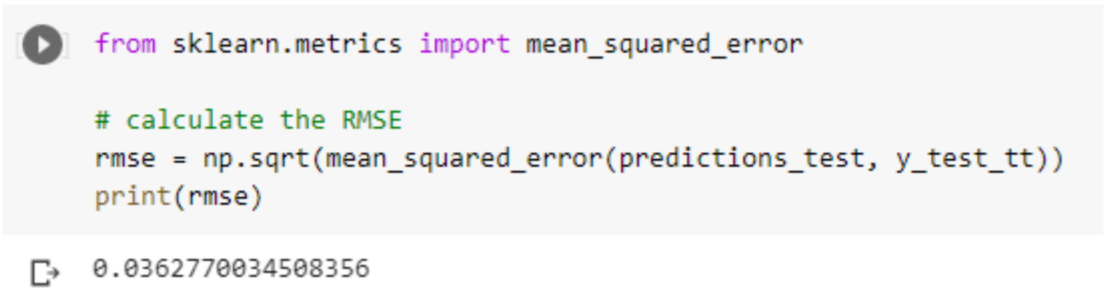
Now the model is fitted onto the optimal epoch value we found from the epoch vs loss plot which is 17. The model is again run on both Train set and it is tested with Test data with new epoch size of 17, keeping all other parameters constant.



```
[ ] # Fitting the RNN model
    history = model.fit(X_train, y_train, epochs = 17, batch_size = 1, validation_data=(X_valid, y_valid))

    Epoch 1/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.8899e-05 - accuracy: 5.2632e-04 - val_loss: 5.6022e-05 - val_accuracy: 0.0000e+00
    Epoch 2/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.7258e-05 - accuracy: 5.2632e-04 - val_loss: 7.9287e-05 - val_accuracy: 0.0000e+00
    Epoch 3/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.7838e-05 - accuracy: 5.2632e-04 - val_loss: 5.4755e-05 - val_accuracy: 0.0000e+00
    Epoch 4/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.7165e-05 - accuracy: 5.2632e-04 - val_loss: 4.7607e-05 - val_accuracy: 0.0000e+00
    Epoch 5/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.6774e-05 - accuracy: 5.2632e-04 - val_loss: 5.2635e-05 - val_accuracy: 0.0000e+00
    Epoch 6/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.6783e-05 - accuracy: 5.2632e-04 - val_loss: 1.0978e-04 - val_accuracy: 0.0000e+00
    Epoch 7/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.5599e-05 - accuracy: 5.2632e-04 - val_loss: 5.2188e-05 - val_accuracy: 0.0000e+00
    Epoch 8/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.5402e-05 - accuracy: 5.2632e-04 - val_loss: 1.1889e-04 - val_accuracy: 0.0000e+00
    Epoch 9/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.5729e-05 - accuracy: 5.2632e-04 - val_loss: 4.6326e-05 - val_accuracy: 0.0000e+00
    Epoch 10/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.4883e-05 - accuracy: 5.2632e-04 - val_loss: 5.4262e-05 - val_accuracy: 0.0000e+00
    Epoch 11/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.4301e-05 - accuracy: 5.2632e-04 - val_loss: 4.7819e-05 - val_accuracy: 0.0000e+00
    Epoch 12/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.5243e-05 - accuracy: 5.2632e-04 - val_loss: 5.3865e-05 - val_accuracy: 0.0000e+00
    Epoch 13/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.4130e-05 - accuracy: 5.2632e-04 - val_loss: 5.5275e-05 - val_accuracy: 0.0000e+00
    Epoch 14/17
    1900/1900 [==============================] - 3s 2ms/step - loss: 2.5177e-05 - accuracy: 5.2632e-04 - val_loss: 3.9761e-05 - val_accuracy: 0.0000e+00
    Epoch 15/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.4217e-05 - accuracy: 5.2632e-04 - val_loss: 6.7598e-05 - val_accuracy: 0.0000e+00
    Epoch 16/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.4211e-05 - accuracy: 5.2632e-04 - val_loss: 5.0640e-05 - val_accuracy: 0.0000e+00
    Epoch 17/17
    1900/1900 [==============================] - 4s 2ms/step - loss: 2.3568e-05 - accuracy: 5.2632e-04 - val_loss: 4.7141e-05 - val_accuracy: 0.0000e+00
```

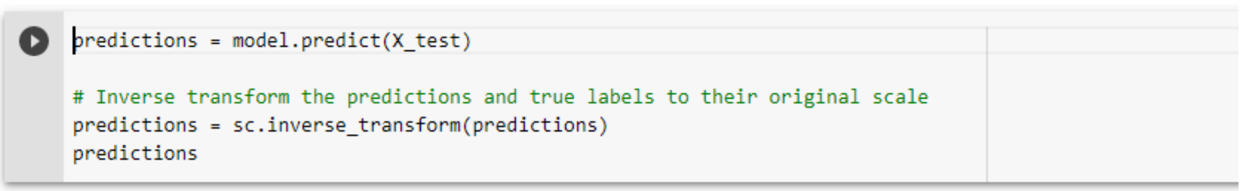
In the next code the X_test values as passed as inputs into the code and the predictions are made on the model.

The predicted scaled values and the actual scaled values are used to calculate RSME value and it was found to be 0.036

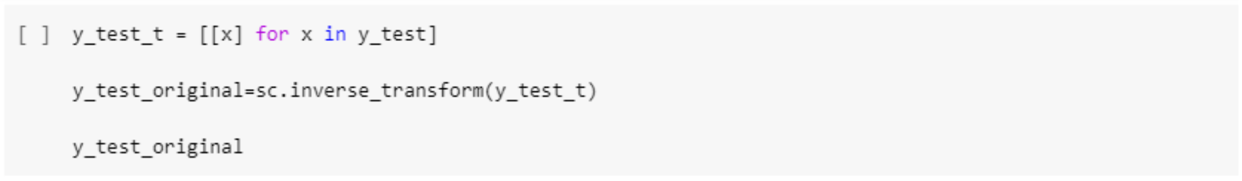


```
from sklearn.metrics import mean_squared_error

# calculate the RMSE
rmse = np.sqrt(mean_squared_error(predictions_test, y_test_tt))
print(rmse)

0.03627700345083 56
```

These predictions are scaled and are to be transformed back to original scale.

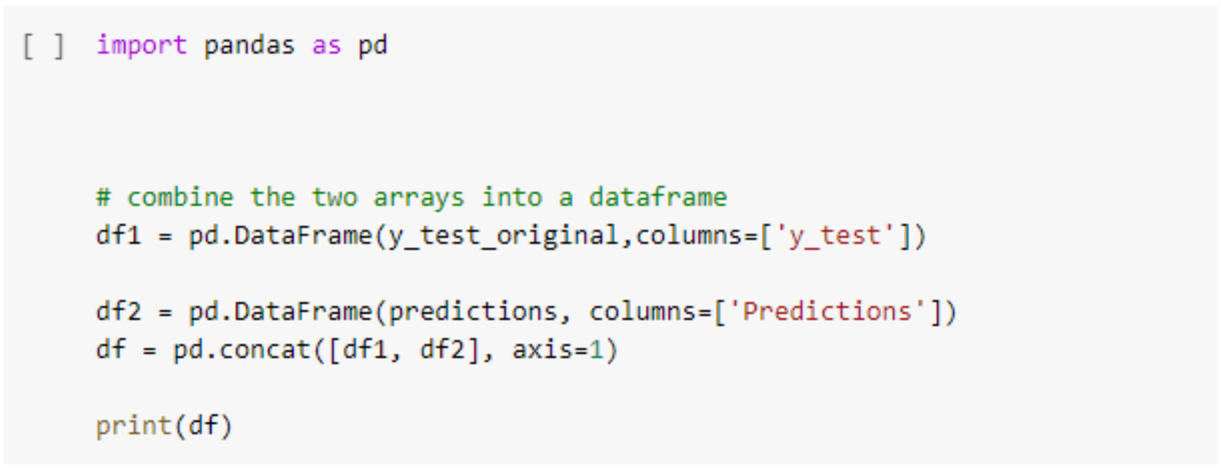


```
predictions = model.predict(X_test)

# Inverse transform the predictions and true labels to their original scale
predictions = sc.inverse_transform(predictions)
predictions
```

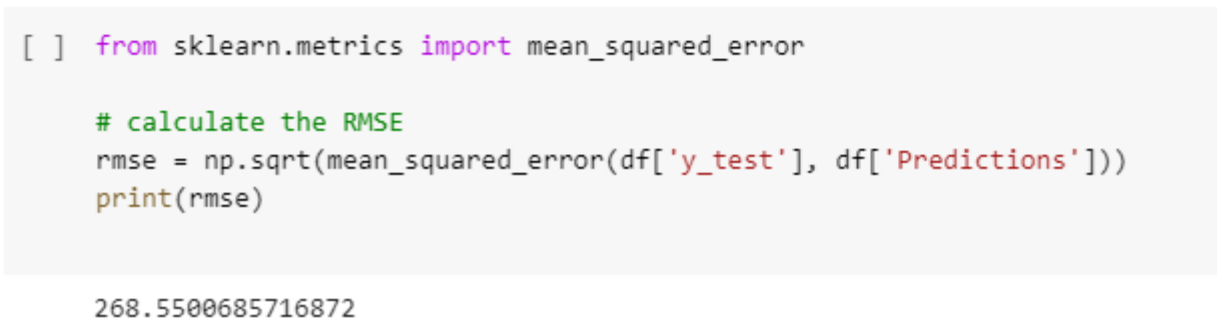


```
y_test_t = [[x] for x in y_test]

y_test_original=sc.inverse_transform(y_test_t)

y_test_original
```

Now the predicted values and original values are made into a dataframe.



```
import pandas as pd



# combine the two arrays into a dataframe
df1 = pd.DataFrame(y_test_original,columns=['y_test'])

df2 = pd.DataFrame(predictions, columns=['Predictions'])
df = pd.concat([df1, df2], axis=1)

print(df)
```



```
from sklearn.metrics import mean_squared_error

# calculate the RMSE
rmse = np.sqrt(mean_squared_error(df['y_test'], df['Predictions']))
print(rmse)
```

```
268.5500685716872
```

The actual and predicted values are compared using RSME. The RSME value was found to be 269. The actual and predicted values are plotted on to graph using matplotlib in the next plot.

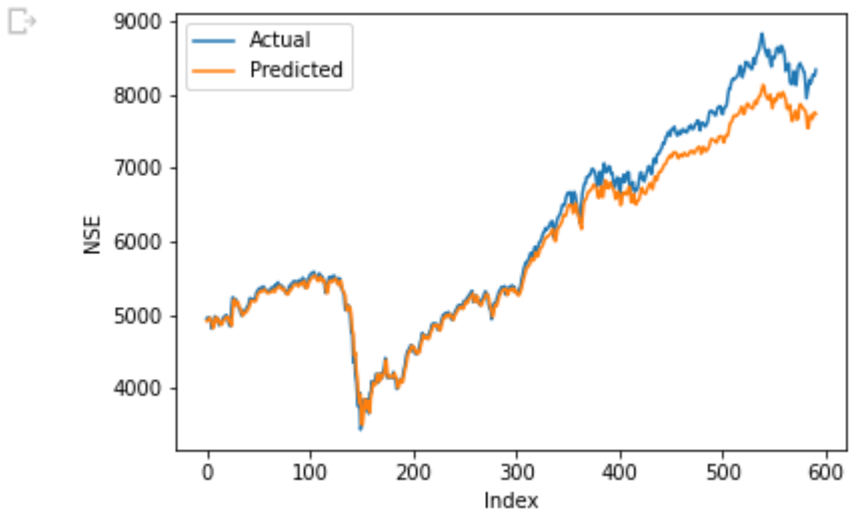
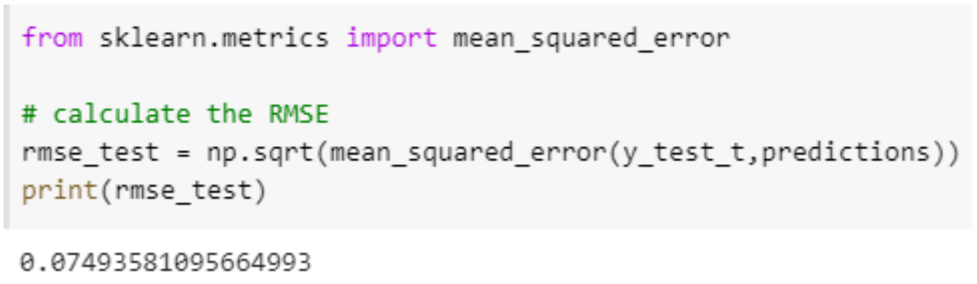


Figure 19. Actual vs Predicted RNN

4.1.6 Method 2: Taking 10 past values as Inputs

As a second method the look back period for the data was changed to 10. Now the input array contains 10 days past values, x and predicts the next days value y.

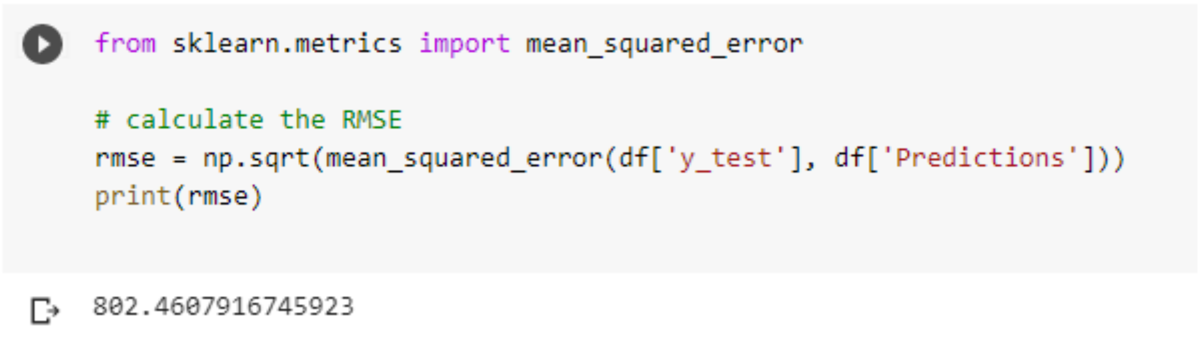
RNN model results are given below for scaled values. The RMSE was found to be 0.0749



```
from sklearn.metrics import mean_squared_error

# calculate the RMSE
rmse_test = np.sqrt(mean_squared_error(y_test_t,predictions))
print(rmse_test)
```

```
0.07493581095664993
```

RMSE for non-scaled values was found to be 802.46

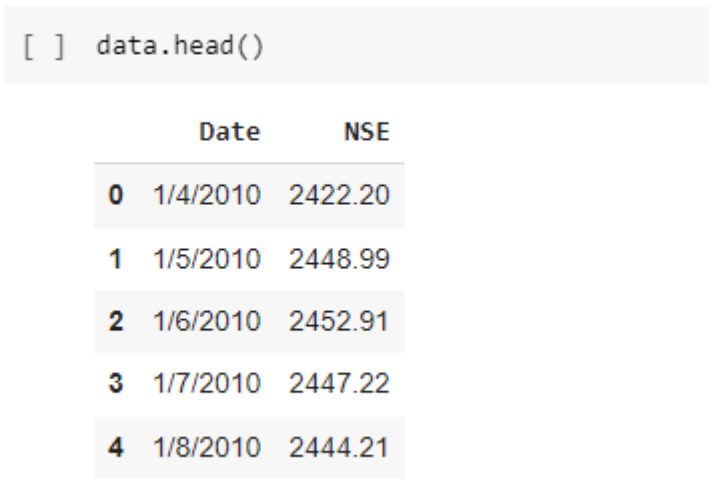

```
from sklearn.metrics import mean_squared_error

# calculate the RMSE
rmse = np.sqrt(mean_squared_error(df['y_test'], df['Predictions']))
print(rmse)
```

```
802.4607916745923
```

4.2 ARIMA Implementation

4.2.1 Data

The data is viewed after the preprocessing stage.


```
[ ] data.head()
```

	Date	NSE
0	1/4/2010	2422.20
1	1/5/2010	2448.99
2	1/6/2010	2452.91
3	1/7/2010	2447.22
4	1/8/2010	2444.21

The Data is scaled between 0 and 1 using Minmax scaler.

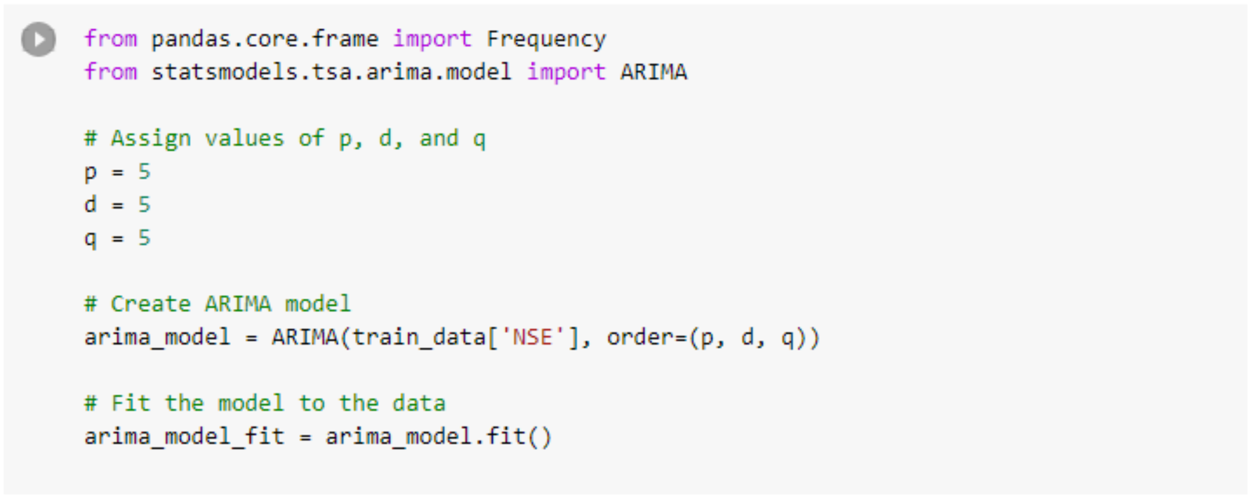
```
training_set = data['NSE'].values.reshape(-1,1)
sc = MinMaxScaler()
training_set_scaled = sc.fit_transform(training_set)
```

The data is then split into train and test data. The train data is 80% of the total data. Test data is 20% of the total data.

```
# Split data into train and test sets
train_data = training_set_scaled[:int(0.8*len(data))]
test_data = training_set_scaled[int(0.8*len(data)):]
```

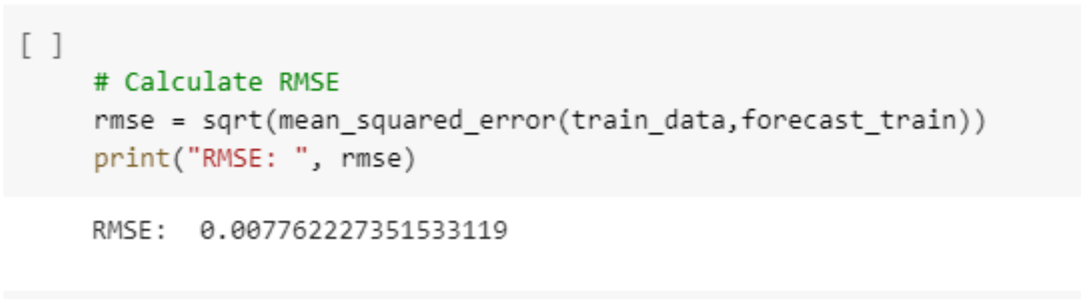
From statsmodel library ARIMA module is imported. This is the machine learning library by which ARIMA model will be built on the data. P is the number autoregressive lags , d is the degree of differencing and q is the number of moving average lags.

In this research p,q and d were taken as 5. The model is fit on the data.



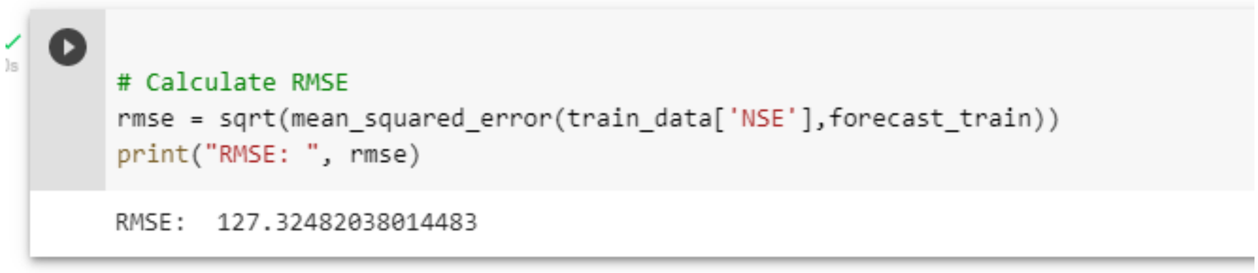
```
from pandas.core.frame import Frequency
from statsmodels.tsa.arima.model import ARIMA

# Assign values of p, d, and q
p = 5
d = 5
q = 5

# Create ARIMA model
arima_model = ARIMA(train_data['NSE'], order=(p, d, q))

# Fit the model to the data
arima_model_fit = arima_model.fit()
```

The Train RMSE for scaled forecast was calculated and it was found to be 0.0077



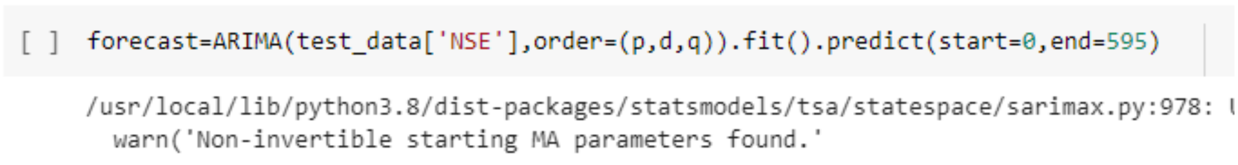
```
[ ]
 # Calculate RMSE
 rmse = sqrt(mean_squared_error(train_data,forecast_train))
 print("RMSE: ", rmse)

 RMSE:  0.007762227351533119
```

The Forecasted values are inverse scaled to get values in original scale and then RMSE is found. Train RMSE is calculated and it was found to be 127.3



```
# Calculate RMSE
rmse = sqrt(mean_squared_error(train_data['NSE'],forecast_train))
print("RMSE: ", rmse)

RMSE:  127.32482038014483
```

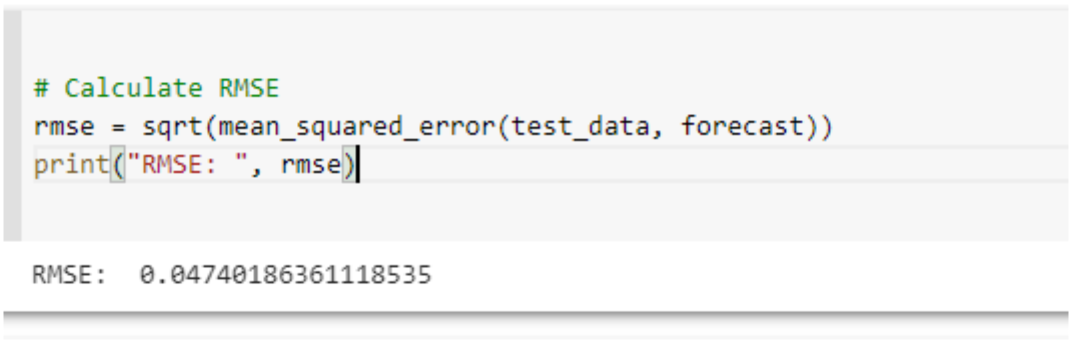
Now the new unseen test data is put into the same model. The forecast is done for the test data and the values are stored as forecast.



```
[ ]  forecast=ARIMA(test_data['NSE'],order=(p,d,q)).fit().predict(start=0,end=595)

     /usr/local/lib/python3.8/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: U
       warn('Non-invertible starting MA parameters found.'
```

The forecasted test values and actual scaled test variables are used to calculate scaled RMSE.

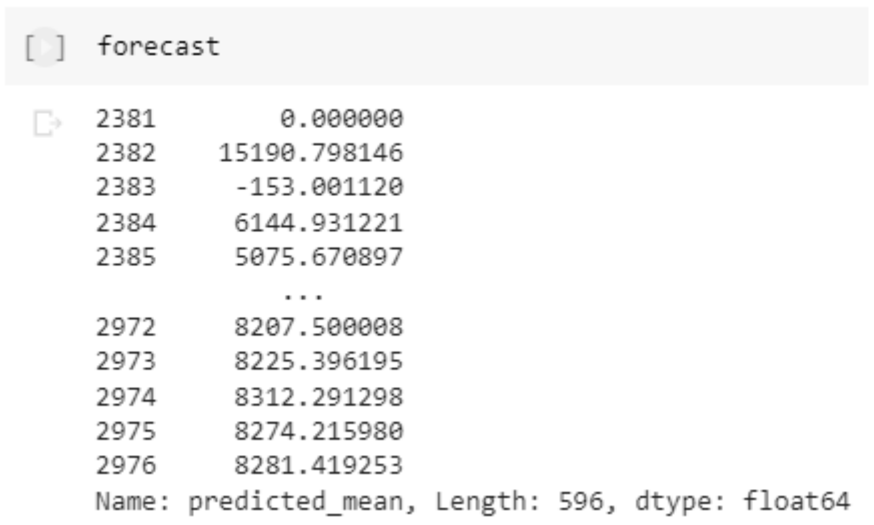
Scaled Test RMSE is found to be 0.047



```
# Calculate RMSE
rmse = sqrt(mean_squared_error(test_data, forecast))
print("RMSE: ", rmse)

RMSE:  0.04740186361118535
```

The scaled values are inverse transformed to get values in original scale.

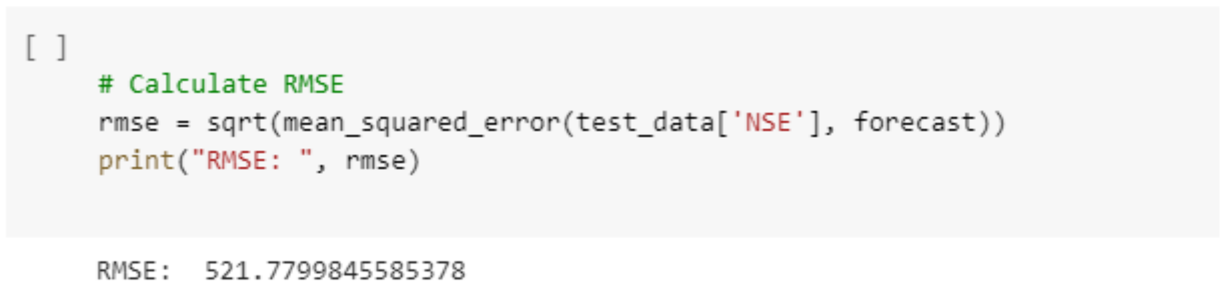
Forecast values are plotted after inverse scaling.



```
[ ]  forecast

     2381         0.000000
     2382     15190.798146
     2383      -153.001120
     2384      6144.931221
     2385      5075.670897
                  ...
     2972      8207.500008
     2973      8225.396195
     2974      8312.291298
     2975      8274.215980
     2976      8281.419253
     Name: predicted_mean, Length: 596, dtype: float64
```

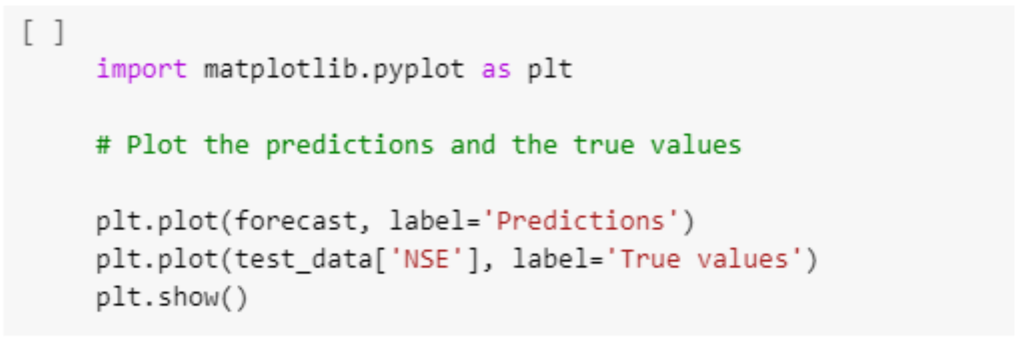
Figure 20. Forecasted values by ARIMA

The test RMSE is calculated. It is found at 521.7



```
[ ]
     # Calculate RMSE
     rmse = sqrt(mean_squared_error(test_data['NSE'], forecast))
     print("RMSE: ", rmse)

     RMSE:  521.7799845585378
```

The result is plotted on the graph below.



```
[ ]
    import matplotlib.pyplot as plt

    # Plot the predictions and the true values

    plt.plot(forecast, label='Predictions')
    plt.plot(test_data['NSE'], label='True values')
    plt.show()
```

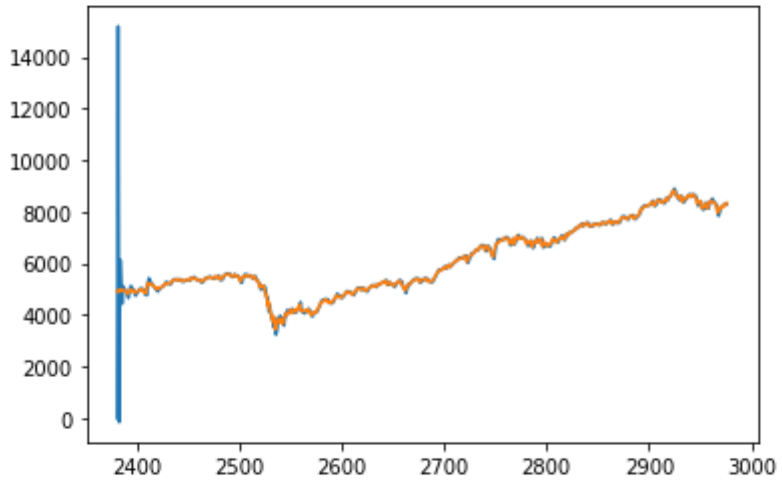
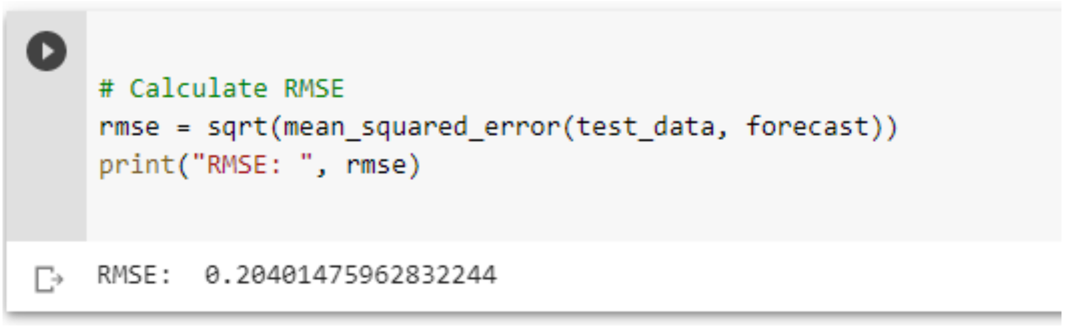


Figure 21. Actual vs Predicted values by ARIMA

4.2.2 Method 2: Taking 10 past values as inputs

The model was now changed to take 10 past values at once. That is p,q, and d values in ARIMA was changed to 10. Now the model will consider the 10 input values at once to predict the next value and so on.

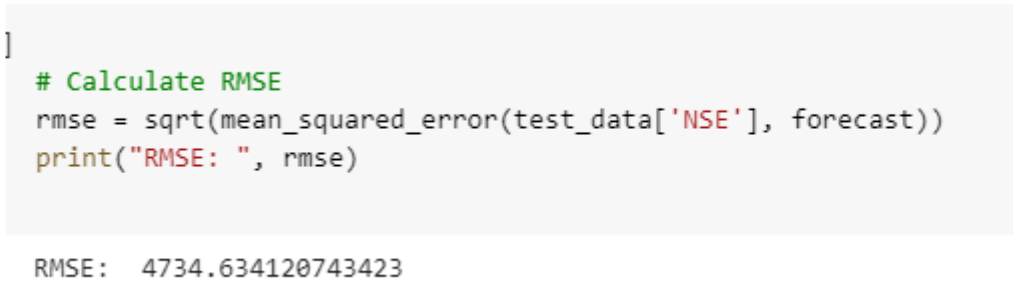
The RMSE for scaled values was calculated for test data and it was found to be 0.20



```
# Calculate RMSE
rmse = sqrt(mean_squared_error(test_data, forecast))
print("RMSE: ", rmse)
```

```
RMSE:  0.20401475962832244
```

The RMSE for non-scaled values for test data was calculated and It was found to be 4734.6



```
# Calculate RMSE
rmse = sqrt(mean_squared_error(test_data['NSE'], forecast))
print("RMSE: ", rmse)
```

```
RMSE:  4734.634120743423
```

5. Conclusion

Prediction of NSE stock exchange value was conducted with both the models and both models performed relatively well. The Deep RNN model had better predictive performance than ARIMA model for the similar look back period of values considering the root mean square values.

Deep RNN model had a scaled test RMSE value of 0.036 on test data while ARIMA gave 0.047 for scaled test RMSE.
Deep RNN model takes much larger processing time compared to ARIMA which is a lot quicker since it is a machine learning model. Both the models performed well considering how volatile the market index usually is from day to day.

The ARIMA model was showing high RMSE error when the input time frame was taken as 10 days. While Deep RNN showed a scaled test RMSE value of .07 ARIMA gave a scaled test RMSE value of 0.204. The ARIMA model significantly underperformed when input values were increased.

Further research can be conducted on the performance of both models by adding exogenous variables, which could make the model complex but more accurate.

6. References

1. ALABDULRAZZAQ, Haneen, ALENEZI, Mohammed N., RAWAJFIH, Yasmeen, ALGHANNAM, Bareeq A., AL-HASSAN, Abeer A. and AL-ANZI, Fawaz S., 2021, On the accuracy of Arima based prediction of COVID-19 spread. *Results in Physics*. 2021. Vol. 27, p. 104509. DOI 10.1016/j.rinp.2021.104509.

2. BAKKER, Indra den, 2017, *Python deep learning cookbook: Over 75 practical recipes on neural network modeling, reinforcement learning, and transfer learning using Python*. Birmingham : Packt Publishing.

3. BENGIO, Y., SIMARD, P. and FRASCONI, P., 1994, Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*. 1994. Vol. 5, no. 2p. 157–166. DOI 10.1109/72.279181.

4. CASAS, I., 2009, Neural networks. *International Encyclopedia of Human Geography*. 2009. P. 419–422. DOI 10.1016/b978-008044910-4.00482-x.

5. CHERUKURI, Harish, PEREZ-BERNABEU, E., SELLES, M.A. and SCHMITZ, Tony L., 2019, A neural network approach for chatter prediction in turning. *Procedia Manufacturing*. 2019. Vol. 34, p. 885–892. DOI 10.1016/j.promfg.2019.06.159.

6. CHOLLET François, 2021, *Deep learning with python*. Shelter Island : Manning.

7. CONNOR, J.T., MARTIN, R.D. and ATLAS, L.E., 1994, Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*. 1994. Vol. 5, no. 2p. 240–254. DOI 10.1109/72.279188.

8. DEZSI, Eva, NISTOR, Ioan Alin, 2016, Can Deep Machine Learning Outsmart The Market? A Comparison Between Econometric Modelling And Long- Short Term Memory," Romanian Economic Business Review, Romanian-American University, vol. 11(4.1), pages 54-73, december.

9. ERIC, Stellwagen, LEN, Tashman, Len. 2013, ARIMA: The Models of Box and Jenkins. Foresight: Int. J. Appl. Forecast.. 28-33.

10. FU, Wangdong, "ARIMA model for forecasting Poisson data: Application to long-term earthquake predictions" (2010). *UNLV Theses, Dissertations, Professional Papers, and Capstones*. 897.

11. GAO, Zihao, 2021, Stock price prediction with Arima and Deep Learning Models. *2021 IEEE 6th International Conference on Big Data Analytics (ICBDA)*. 2021. DOI 10.1109/icbda51983.2021.9403037.

12. GÉRON Aurélien, 2023, *Hands-on machine learning with scikit-learn, keras and tensorflow: Concepts, tools, and techniques to build Intelligent Systems*. Sebastapol, CA : O'Reilly.

13. GUJARATI Damodar N and Dawn C Porter. *Basic Econometrics*. 5th ed. McGraw-Hill 2009.

14. HO, M K, DARMAN, Hazlina and MUSA, Sarah, 2021, Stock price prediction using Arima, Neural Network and LSTM models. *Journal of Physics: Conference Series*. 2021. Vol. 1988, no. 1p. 012041. DOI 10.1088/1742-6596/1988/1/012041.

15. HU, Junguo, 2018, Effects of BP Algorithm-based Activation Functions on Neural Network Convergence

16. KIANG, Melody Y., 2003, Neural networks. *Encyclopedia of Information Systems*. 2003. P. 303–315. DOI 10.1016/b0-12-227240-4/00121-0.

17. KOBIELA, Dariusz, KREFTA, Dawid, KRÓL, Weronika and WEICHBROTH, Paweł, 2022, Arima vs LSTM on NASDAQ Stock Exchange Data. *Procedia Computer Science*. 2022. Vol. 207, p. 3836–3845. DOI 10.1016/j.procs.2022.09.445.

18. KORSTANJE, Joos, 2021, *Advanced forecasting with python: With state-of-the-art-models including LSTMs, Facebook's prophet, and Amazon's DeepAR*. New York, NY : Apress.

19. KOSTADINOV, Simeon, 2018, *Recurrent neural networks with Python Quick Start Guide*. Packt Publishing.

20. KOTU, Vijay and DESHPANDE, Bala, 2019, Time Series forecasting. *Data Science*. 2019. P. 395–445. DOI 10.1016/b978-0-12-814761-0.00012-5.

21. LEDERER, Johannes. "Activation Functions in Artificial Neural Networks: A Systematic Overview." *ArXiv* abs/2101.09957 (2021): n. Pag.

22. MA, Qihang, 2020, Comparison of Arima, ann and LSTM for stock price prediction. *E3S Web of Conferences*. 2020. Vol. 218, p. 01026. DOI 10.1051/e3sconf/202021801026.

23. MANUCA, Radu and SAVIT, Robert, 1996, Stationarity and nonstationarity in time series analysis. *Physica D: Nonlinear Phenomena*. 1996. Vol. 99, no. 2-3p. 134–161. DOI 10.1016/s0167-2789(96)00139-x.

24. MILLS, Terence C., 2019, Arima models for nonstationary time series. *Applied Time Series Analysis*. 2019. P. 57–69. DOI 10.1016/b978-0-12-813117-6.00004-1.

25. MILLS, Terence C., 2019, Unit roots, difference and trend stationarity, and fractional differencing. *Applied Time Series Analysis*. 2019. P. 71–101. DOI 10.1016/b978-0-12-813117-6.00005-3.

26. MONDAL, Prapanna, SHIT, Labani and GOSWAMI, Saptarsi, 2014, Study of effectiveness of time series modeling (ARIMA) in forecasting stock prices. *International Journal of Computer Science, Engineering and Applications*. 2014. Vol. 4, no. 2p. 13–29. DOI 10.5121/ijcsea.2014.4202.

27. NABIPOUR, Mojtaba, NAYYERI, Pooyan, JABANI, Hamed, SHAMSHIRBAND, Shahab and MOSAVI, Amir, 2020, Deep learning for stock market prediction. . 2020. DOI 10.20944/preprints202003.0256.v1.

28. PAI, Ping-Feng and LIN, Chih-Sheng, 2005, A hybrid Arima and support vector machines model in stock price forecasting. *Omega*. 2005. Vol. 33, no. 6p. 497–505. DOI 10.1016/j.omega.2004.07.024.

29. SHARMA, Neha, SHARMA, Reecha and JINDAL, Neeru, 2021, Machine learning and deep learning applications-A Vision. *Global Transitions Proceedings*. 2021. Vol. 2, no. 1p. 24–28. DOI 10.1016/j.gltp.2021.01.004.

30. SHI, Zhiwei, WU, Zhifeng, SHI, Shuaiwei, MAO, Chengzhi, WANG, Yingqiao and ZHAO, Laiqi, 2022, High-frequency forecasting of stock volatility based on model fusion and a feature Reconstruction Neural Network. *Electronics*. 2022. Vol. 11, no. 23p. 4057. DOI 10.3390/electronics11234057.

31. SHMUELI, Galit, 2018, *Practical time series forecasting: A hands-on guide*. Axelrod Schnall Publishers.

32. SWARAJ, Aman, VERMA, Karan, KAUR, Arshpreet, SINGH, Ghanshyam, KUMAR, Ashok and MELO DE SALES, Leandro, 2021, Implementation of stacking based Arima model for prediction of covid-19 cases in India. *Journal of Biomedical Informatics*. 2021. Vol. 121, p. 103887. DOI 10.1016/j.jbi.2021.103887.

7. List of Figures and Appendix

7.1 List of Figures

7.2 Appendix