A Thesis/Project/Dissertation Report

on

STOCK MARKET PREDICTION USING STACKED LSTM

Submitted in partial fulfillment of the

requirement for the award of the degree of

B.tech (H) specialization in cyber security and computer network

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

Under The Supervision of
Name of Supervisor : Mr.Bharat Bhushan Naib

Submitted By:-

Varundeep singh 18SCSE1140043
Shikha Raj Bharti 18SCSE1140062

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
DECEMBER 2022

Contents

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the thesis/project/dissertation, entitled **"Stock market prediction using stacked LSTM"** in partial fulfillment of the requirements for the award of the **B.tech (H) specialization in cyber security and computer network** submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of **December 2021**, under the supervision of **Mr.Bharat Bhushan Naib** Department of Computer Science and Engineering/Computer Application and Information and Science, of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the thesis/project/dissertation has not been submitted by me/us for the award of any other degree of this or any other places.

Varundeep Singh 18SCSE1140043

Shikha Raj Bharti 18SCSE1140062

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Supervisor Name
Bharat bhushan Naib (Asst professor)

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of Varundeep Singh 18SCSE1140043 and shikha raj Bharti 18SCSE1140062 has been held on _____ and his/her work is recommended for the award of **B.tech (H) specialization in cyber security and computer network**

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date: December , 2021

Acknowledgement

Place: Gr.Noida

Date: December 2021

I would like to acknowledge that my assignment has been completed and I am ensuring that this was done by me and not copied.

In this accomplishment, I would like to express my special gratitude to all my teachers and most importantly our Guide **Mr./Mrs. BHARAT BHUSHAN NAIB** of **GALGOTIAS UNIVERSITY**, without their guidance and feedback it is not possible to complete this assignment.

Finally, I would like to thank my reviewer **Mrs.Kirti shukla** and my Batchmates who helped me a lot in finishing this assignment.

VARUNDEEP SINGH 18SCSE1140043

SHIKHA RAJ BHARTI 18SCSE1140062

Signature.

Abstract

The direction of the financial market is always stochastic and volatile and the return of the security return is deemed to be unpredictable. Analysts now are trying to apply the modeling techniques from Natural Language Processing into the field of Finance as the similarity of having the sequential property in the data. In this research, we have constructed and applied the state-of-art deep learning sequential model, namely Long Short Term Memory Model (LSTM), StackedLSTM and Attention-Based LSTM, along with the traditional ARIMA model, into the prediction of stock prices on the next day. Moreover, using our prediction, we built up two trading strategies and compared them with the benchmark. Our input data not only contains traditional end-day price and trading volumes, but also includes corporate accounting statistics, which are carefully selected and applied into the models. The accounting data are regarded as the news and price shocks to a corporation, hence are expected to increase the predictive power of the model. The result has shown that Attention-LSTM beats all other models in terms of prediction error and shows much higher return in our trading strategy over other models. Furthermore, we discovered that the stacked-LSTM model does not improve the predictive power over LSTM, even though it has a more complex model structure. The prediction of stock value is a complex task which needs a robust algorithm background in order to compute the longer term share prices. Stock prices are correlated within the nature of the market; hence it will be difficult to predict the costs. The proposed algorithm uses the market data to predict the share price using machine learning techniques like recurrent neural network named as Long Short Term Memory, in that process weights are corrected for each data point using stochastic gradient descent. This system will provide accurate outcomes in comparison to currently available stock price predictor algorithms. The network is trained and evaluated with various sizes of input data to urge the graphical outcomes.

List of Table

List of Figures

Acronyms

B.Tech.	Bachelor of Technology
M.Tech.	Master of Technology
BCA	Bachelor of Computer Applications
MCA	Master of Computer Applications
B.Sc. (CS)	Bachelor of Science in Computer Science
M.Sc. (CS)	Master of Science in Computer Science
SCSE	School of Computing Science and Engineering

CHAPTER-1

1.1 Introduction

The share market is a place where the shares of a public company are traded. As discussed in the volatile nature of the stock market makes it an area which needs an abundance of analysis with the old data predicated. The previous stock trend prediction algorithms use the historic time series stock data. The typical scientific stock price forecasting procedures are focused on the statistical analysis of stock data. The paper will develop a stock data predictor program that uses previous stock prices and data will be treated as training sets for the program to predict the stock prices of a particular share. This program develops a procedure.

This work consists of three parts: data extraction and pre-processing of the Chinese stock market dataset, carrying out feature engineering, and stock price trend prediction model based on the long short-term memory (LSTM). We collected, cleaned-up, and structured 2 years of Chinese stock market data. We reviewed different techniques often used by real-world investors, developed a new algorithm component, and named it as feature extension, which proved to be effective.

The system is customized by assembling the feature engineering procedure with an LSTM prediction model, achieving high prediction accuracy that outperforms the leading models in most related works.Our proposed solution is a unique customization as compared to the previous works because rather than just proposing yet another state-of-the-art LSTM model, we proposed a fine-tuned and customized deep learning prediction system along with utilization of comprehensive feature engineering and combined it with LSTM to perform prediction. we have seen how to predict a stock price, this is a simple algorithm. You can make use of auto-ml so that the adding of new data will be easy. Refer a lot of Deep Learning Algorithms, Machine Learning … etc

We will use the Long Short-Term Memory(LSTM) method to create a Machine Learning model to forecast Microsoft Corporation stock values.

They are used to make minor changes to the information by multiplying and adding. Long-term memory (LSTM) is a deep learning artificial recurrent neural network (RNN) architecture.

1.2 FORMULATION OF PROBLEM :-

With an understanding of what stock and the stock market is, let's understand why people are fascinated by a Machine Learning model that can predict the stock price. A Machine Learning model predicts the value of an observation based on several inputs that are predictors. The stock market is working similarly, i.e., based on several inputs, the stock price fluctuates. However, the bone of contention is that, unlike other problems that generally are predicted, the predictions of stock prices are rather complicated as it is often prey to the butterfly effect. Anything, any event in the outside world, can influence the price of the stock. These include political events, economic news, competitors or related stock movements, and other challenges to capture concepts such as rumor, anxiety, and other psychological factors. Capturing all these inputs makes stock market prediction challenging and Machine Learning a potential candidate for it. It can take a large number of inputs and assess the patterns and their relation to a dependent variable (the stock price) We analyzed the best possible approach for predicting short-term price trends from different aspects: feature engineering, financial domain knowledge, and prediction algorithm. Then we addressed three research questions in each aspect, respectively: How can feature engineering benefit model prediction accuracy? How do findings from the financial domain benefit prediction model design? And what is the best algorithm for predicting short-term price trends?

The first research question is about feature engineering. We would like to know how the feature selection method benefits the performance of prediction models. From the abundance of the previous works, we can conclude that stock price data embedded with a high level of noise, and there are also correlations between features, which makes the price prediction

notoriously difficult. That is also the primary reason for most of the previous works introducing the feature engineering part as an optimization module.

The second research question is evaluating the effectiveness of findings we extracted from the financial domain. Different from the previous works, besides the common evaluation of data models such as the training costs and scores, our evaluation will emphasize the effectiveness of newly added features that we extracted from the financial domain.

The third research question is which algorithms are we going to model our data? From the previous works, researchers have been putting efforts into the exact price prediction. We decompose the problem into predicting the trend and then the exact number

In the previous works, researchers have been using a variety of models for predicting stock price trends. While most of the best-performed models are based on machine learning techniques, in this work, we will compare our approach with the outperformed machine learning models in the evaluation part and find the solution for this research question.

The Adjusted Close Value reflects the stock's value after dividends have been declared (too technical!). Furthermore, the total volume of the stocks in the market is provided. With this information, it is up to the job of a Machine Learning/Data Scientist to look at the data and develop different algorithms that may extract patterns from the historical data of the Microsoft Corporation stock.

1.3 TOOLS AND TECHNOLOGY USED :-

Applying feature extension:-

The three feature extension methods are max–min scaling, polarizing, and calculating fluctuation percentage. Not all the technical indices are applicable for all three of the feature extension methods; this procedure only applies the meaningful extension methods on technical indices.

Applying recursive feature elimination

After the feature extension above, we explore the most effective i features by using the Recursive Feature Elimination (RFE) algorithm [6]. We estimate all the features by two attributes, coefficient, and feature importance. We also limit the features that are removed from the pool by one, which means we will remove one feature at each step and retain all the relevant features. Then the output of the RFE block will be the input of the next step, which refers to PCA.

Applying principal component analysis (PCA)

The very first step before leveraging PCA is feature pre-processing. Because some of the features after RFE are percentage data, while others are very large numbers, i.e., the output from RFE are in different units. It will affect the principal component extraction result.

Fitting long short-term memory (LSTM) model

PCA reduces the dimensions of the input data, while the data pre-processing is mandatory before feeding the data into the LSTM layer. The reason for adding the data pre-processing step before the LSTM model is that the input matrix formed by principal components has no time steps. While one of the most important parameters of training an LSTM is the number of time steps. Hence, we have to model the matrix into corresponding time steps for both training and testing dataset.

CHAPTER 2

LITERATURE/SURVEY PROJECT

While doing the literature survey, the data about Stock market prediction systems that are as of now being utilized are considered. Over the most recent two decades determining stock returns has become a significant field of research. In the majority of the cases the scientists had endeavored to build up a straight connection between the information macroeconomic factors what's more, the stock returns, be that as it may, with the revelation of non linear slants in the financial exchange record returns, there has been an incredible move in the focal point of the scientists towards the nonlinear expectation of the stock returns. Despite the fact that, thereafter numerous writings have come up in nonlinear measurable displaying of the stock returns, the majority of them required that the nonlinear model be indicated before the estimation is done. in any case, for the explanation that the financial exchange returns are boisterous, unsure, confused and nonlinear in nature. There are various functions used to forecast the parameters. Mainly include, binary threshold, linear threshold, hyperbolic sigmoid, and brown. The Investigation of Stock Market Prediction Using Machine Learning Approach has been mentioned. The stock exchange forecast has become a sharp area of interest. Particular assessment is one of them, yet it does not reliably deliver specific results, so it is essential to develop strategies for progressively accurate gauge.

RautSushrut et al.[1] suggested that supervised learning classifiers be used to forecast stock price movement based on financial index data, and determine their ability. In the financial market computational analytical approaches have been portfolio modeling. A discussion about the statistical AI methodology has been addressed; the usage of SVM methodology has been shown in the paper and also shown that tactical methodologies can be applied to predict the stock prices.

Manoj S Hegde et al.[2] investigated that The Long Short Term Memory (LSTM) networks are a type of recurrent neural network (RNN) capable of

solving in volute linear problems, and also there is a discussion about the usage of RNN (Recurrent Neural Networks) to predict the share prices.

M. Roondiwala et al.[3] proposed that the Long ShortTerm Memory is the most popular RNN architecture. In the secret network layer, LSTM introduces a memory cell; a processing device that replaces conventional artificial neurons, using these memory cells, networks can effectively link memory and remote input in time, making it suitable to dynamically capture data structure over time with a high predictive limit. It is also shown in the paper that the stock prediction can be done on the NIFTY50 shares. The data collection is one of the major steps and later the training of our model and there is a need to test the algorithm by applying a different data set to the algorithm. Our procedure will be discussed in coming sections

As Kim and H. Y. Kim et al.[4] identified that .another significant issue with basic ANNs for stock forecast is the marvel of detonating fleeting inclination, where the loads of a gigantically huge system either become excessively massively enormous or too minuscule (respectively),drastically easing back their union to the ideal worth. This is regularly brought about by two components: loads are instated self-assertively and the loads progressively proximate to the end of the system moreover slope to transmute significantly more than those at the beginning. It is also mentioned in the paper that the usage of LSTM networks can be applied in procedure of predicting share prices.

As discussed by S. Selvin et al.[5], Customary types for dealing with the financial exchange investigation and the stock-value forecast include a major review of the past stock-exhibition gander and the general credibility of the organization itself, and a measurable investigation that is solely concerned with the calculation and recognition of stock-value designs, it also mentioned in the paper that different types of analysis that can be performed in order to predict the stock value.

CHAPTER 3

FUNCTIONALITY / WORKING OF PROJECT

In [ ]:
```
import pandas_datareader as pdr
import tensorflow as tf
import pandas as pd
```

Loading the Dataset

In [ ]:
```
df = pd.read_csv('/content/drive/MyDrive/DataSets/NSE-TATAGLOBAL.csv')
df.head()
```

Out[ ]:

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	2018-09-28	234.05	235.95	230.20	233.50	233.75	3069914	7162.35
1	2018-09-27	234.55	236.80	231.10	233.80	233.25	5082859	11859.95
2	2018-09-26	240.00	240.00	232.50	235.00	234.25	2240909	5248.60
3	2018-09-25	233.30	236.75	232.00	236.25	236.10	2349368	5503.90
4	2018-09-24	233.55	239.20	230.75	234.00	233.30	3423509	7999.55

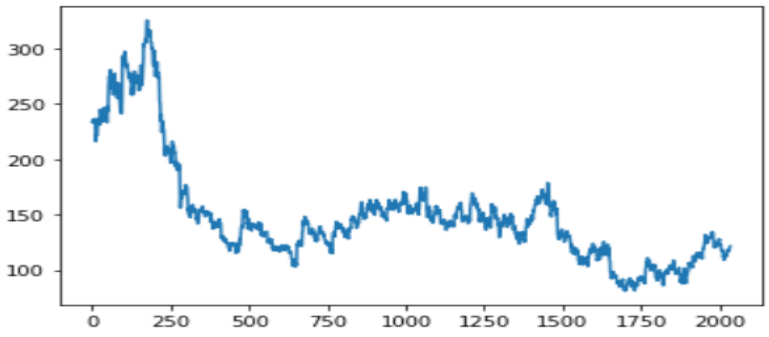
In [ ]:
```
# Get the 'Close' values
df_close = df.reset_index()['Close']
```

In [ ]:
```
df_close.shape
```

Plot the 'Close' values

In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(df_close)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x7f9b9b1fe310>]
```



Use Scaler

In [ ]:
```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
df_close = scaler.fit_transform(np.array(df_close).reshape(-1, 1))
df_close.shape
```

Out[ ]:
```
(2035, 1)
```

Training and Testing Data

In [26]:
```
# split train & test
train_size = int(len(df_close)*0.65)
test_size = len(df_close) - train_size
train_data, test_data = df_close[0:train_size,:], df_close[train_size:len(df1),:1]
```

In [32]:
```
import numpy
# convert values to 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]
    dataX.append(a)
    dataY.append(dataset[i + time_step, 0])
  return numpy.array(dataX), numpy.array(dataY)
```

In [33]:
```
time_step=100
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)
```

In [34]:
```
print(X_train)
```

```
[[0.62418301 0.62214052 0.62622549 ... 0.83455882 0.86213235 0.85273693]
 [0.62214052 0.62622549 0.63378268 ... 0.86213235 0.85273693 0.87111928]
 [0.62622549 0.63378268 0.62234477 ... 0.85273693 0.87111928 0.84497549]
 ...
 [0.34517974 0.31781046 0.33047386 ... 0.2816585  0.27001634 0.26531863]
 [0.31781046 0.33047386 0.32128268 ... 0.27001634 0.26531863 0.27389706]
 [0.33047386 0.32128268 0.34007353 ... 0.26531863 0.27389706 0.25347222]]
```

In [35]:
```
print(X_train.shape), print(y_train.shape)
```

Out[35]:
```
(1221, 100)
(1221,)
(None, None)
```

In [36]:
```
print(X_test.shape), print(y_test.shape)
```

Out[36]:
```
(612, 100)
(612,)
(None, None)
```

Reshape input into 3D format as required for LSTM

In [37]:
```
# reshape input to 3D [samples, time steps, features]
# 3D input 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)
```

Create Stacked LSTM Model

In [38]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

In [44]:
```
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(100,1)))
model.add(LSTM(50, return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
# compile model
model.compile(loss='mean_squared_error', optimizer='adam')
```

In [46]:
```
# view model summary
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 100, 50)           10400
_________________________________________________________________
lstm_4 (LSTM)                (None, 100, 50)           20200
_________________________________________________________________
lstm_5 (LSTM)                (None, 50)                20200
_________________________________________________________________
```

dense_1 (Dense) (None, 1) 51

===

Total params: 50,851

Trainable params: 50,851

Non-trainable params: 0

_____

In [48]:
```
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=100, batch_size=64, verbose=1)
```

Epoch 1/10020/20 [==============================] - 12s 286ms/step - loss: 0.0319 - val_loss: 0.0054

Epoch 2/100 20/20 [==============================] - 4s 220ms/step - loss: 0.0031 - val_loss: 9.8749e-04

Epoch 3/100 20/20 [==============================] - 4s 220ms/step - loss: 0.0023 - val_loss: 0.0012

Epoch 4/100 20/20 [==============================] - 4s 217ms/step - loss: 0.0016 - val_loss: 0.0012

Epoch 5/100 20/20 [==============================] - 4s 215ms/step - loss: 0.0014 - val_loss: 0.0012

Epoch 6/100 20/20 [==============================] - 4s 214ms/step - loss: 0.0015 - val_loss: 0.0014

Epoch 7/100 20/20 [==============================] - 4s 211ms/step - loss: 0.0013 - val_loss: 0.0011

Epoch 8/100 20/20 [==============================] - 4s 215ms/step - loss: 0.0012 - val_loss: 0.0010

Epoch 9/100 20/20 [==============================] - 4s 218ms/step - loss: 0.0016 - val_loss: 8.5848e-04

Epoch 10/100 20/20 [==============================] - 4s 214ms/step - loss: 0.0013 - val_loss: 0.0012

Epoch 11/100 20/20 [==============================] - 4s 214ms/step - loss: 0.0012 - val_loss: 0.0011

Epoch 12/100 20/20 [==============================] - 4s 211ms/step - loss: 0.0010 - val_loss: 9.6296e-04

Epoch 13/100 20/20 [==============================] - 4s 208ms/step - loss: 0.0010 - val_loss: 0.0011

Epoch 14/100 20/20 [==============================] - 4s 206ms/step - loss: 9.5370e-04 - val_loss: 9.0927e-04

Epoch 15/100 20/20 [==============================] - 4s 206ms/step - loss: 9.0375e-04 - val_loss: 6.8417e-04

Epoch 16/100 20/20 [==============================] - 4s 207ms/step - loss: 9.3395e-04 - val_loss: 7.8754e-04

Epoch 17/100 20/20 [==============================] - 4s 207ms/step - loss: 9.7269e-04 - val_loss: 0.0011

Epoch 18/100 20/20 [==============================] - 4s 208ms/step - loss: 8.6621e-04 - val_loss: 6.9723e-04

Epoch 19/100 20/20 [==============================] - 4s 204ms/step - loss: 8.0821e-04 - val_loss: 8.3797e-04

Epoch 20/100 20/20 [==============================] - 4s 208ms/step - loss: 8.4106e-04 - val_loss: 0.0010

Epoch 21/100 20/20 [==============================] - 4s 208ms/step - loss: 7.4750e-04 - val_loss: 8.8635e-04

Epoch 22/100 20/20 [==============================] - 4s 208ms/step - loss: 7.7887e-04 - val_loss: 8.9053e-04

Epoch 23/100 20/20 [==============================] - 4s 207ms/step - loss: 6.9998e-04 - val_loss: 6.5799e-04

Epoch 24/100 20/20 [==============================] - 4s 209ms/step - loss: 7.5385e-04 - val_loss: 6.3814e-04

Epoch 25/100 20/20 [==============================] - 4s 210ms/step - loss: 8.0346e-04 - val_loss: 8.6116e-04

Epoch 26/100 20/20 [==============================] - 4s 212ms/step - loss: 6.8039e-04 - val_loss: 7.0478e-04

Epoch 27/100 20/20 [==============================] - 4s 210ms/step - loss: 7.3889e-04 - val_loss: 7.8271e-04

Epoch 28/100 20/20 [==============================] - 4s 207ms/step - loss: 6.7102e-04 - val_loss: 8.3201e-04

Epoch 29/100 20/20 [==============================] - 4s 206ms/step - loss: 6.4831e-04 - val_loss: 7.7534e-04

Epoch 30/100 20/20 [==============================] - 4s 206ms/step - loss: 6.1869e-04 - val_loss: 6.4983e-04

Epoch 31/100 20/20 [==============================] - 4s 210ms/step - loss: 7.8578e-04 - val_loss: 6.2451e-04

Epoch 32/100 20/20 [==============================] - 4s 205ms/step - loss: 6.2462e-04 - val_loss: 8.3214e-04

Epoch 33/100 20/20 [==============================] - 4s 204ms/step - loss: 6.2834e-04 - val_loss: 6.4353e-04

Epoch 34/100 20/20 [==============================] - 4s 207ms/step - loss: 6.0474e-04 - val_loss: 7.1306e-04

Epoch 35/100 20/20 [==============================] - 4s 207ms/step - loss: 6.9425e-04 - val_loss: 5.4889e-04

Epoch 36/100 20/20 [==============================] - 4s 206ms/step - loss: 6.3973e-04 - val_loss: 6.3465e-04

Epoch 37/100 20/20 [==============================] - 4s 207ms/step - loss: 6.9146e-04 - val_loss: 4.7941e-04

Epoch 38/100 20/20 [==============================] - 4s 207ms/step - loss: 5.9533e-04 - val_loss: 6.7071e-04

Epoch 39/100 20/20 [==============================] - 4s 206ms/step - loss: 5.4858e-04 - val_loss: 6.9497e-04

Epoch 40/100 20/20 [==============================] - 4s 208ms/step - loss: 5.6759e-04 - val_loss: 8.7421e-04

Epoch 41/100 20/20 [==============================] - 4s 208ms/step - loss: 7.5182e-04 - val_loss: 5.8749e-04

Epoch 42/100 20/20 [==============================] - 4s 210ms/step - loss: 5.4717e-04 - val_loss: 6.6544e-04

Epoch 43/100 20/20 [==============================] - 4s 206ms/step - loss: 5.4734e-04 - val_loss: 4.7607e-04

Epoch 44/100 20/20 [==============================] - 4s 207ms/step - loss: 5.5067e-04 - val_loss: 4.9458e-04

Epoch 45/100 20/20 [==============================] - 4s 206ms/step - loss: 5.1409e-04 - val_loss: 4.9200e-04

Epoch 46/100 20/20 [==============================] - 4s 199ms/step - loss: 4.9899e-04 - val_loss: 4.9104e-04

Epoch 47/100 20/20 [==============================] - 4s 201ms/step - loss: 5.0069e-04 - val_loss: 6.8338e-04

Epoch 48/100 20/20 [==============================] - 4s 209ms/step - loss: 4.9988e-04 - val_loss: 4.8467e-04

Epoch 49/100 20/20 [==============================] - 4s 208ms/step - loss: 4.7753e-04 - val_loss: 5.0506e-04

Epoch 50/100 20/20 [==============================] - 4s 205ms/step - loss: 4.9396e-04 - val_loss: 4.8932e-04

Epoch 51/100 20/20 [==============================] - 4s 205ms/step - loss: 5.3644e-04 - val_loss: 4.0617e-04

Epoch 52/100 20/20 [==============================] - 4s 203ms/step - loss: 5.4318e-04 - val_loss: 6.0807e-04

Epoch 53/100 20/20 [==============================] - 4s 202ms/step - loss: 5.5832e-04 - val_loss: 4.3495e-04

Epoch 54/100 20/20 [==============================] - 4s 199ms/step - loss: 6.3838e-04 - val_loss: 5.2618e-04

Epoch 55/100 20/20 [==============================] - 4s 193ms/step - loss: 4.6889e-04 - val_loss: 4.3246e-04

Epoch 56/100 20/20 [==============================] - 4s 197ms/step - loss: 4.8364e-04 - val_loss: 6.5795e-04

Epoch 57/100 20/20 [==============================] - 4s 195ms/step - loss: 4.6583e-04 - val_loss: 6.1314e-04

Epoch 58/100 20/20 [==============================] - 4s 203ms/step - loss: 5.0248e-04 - val_loss: 5.3957e-04

Epoch 59/100 20/20 [==============================] - 4s 199ms/step - loss: 4.4935e-04 - val_loss: 4.2343e-04

Epoch 60/100 20/20 [==============================] - 4s 201ms/step - loss: 4.8008e-04 - val_loss: 4.7334e-04

Epoch 61/100 20/20 [==============================] - 4s 203ms/step - loss: 4.1341e-04 - val_loss: 4.7851e-04

Epoch 62/100 20/20 [==============================] - 4s 204ms/step - loss: 4.4707e-04 - val_loss: 3.8688e-04

Epoch 63/100 20/20 [==============================] - 4s 201ms/step - loss: 4.1293e-04 - val_loss: 5.4444e-04

Epoch 64/100 20/20 [==============================] - 4s 199ms/step - loss: 4.2526e-04 - val_loss: 3.8633e-04

Epoch 65/100 20/20 [==============================] - 4s 194ms/step - loss: 3.8733e-04 - val_loss: 3.1159e-04

Epoch 66/100 20/20 [==============================] - 4s 198ms/step - loss: 3.8148e-04 - val_loss: 3.7236e-04

Epoch 67/100 20/20 [==============================] - 4s 199ms/step - loss: 4.1487e-04 - val_loss: 3.2144e-04

Epoch 68/100 20/20 [==============================] - 4s 197ms/step - loss: 3.9209e-04 - val_loss: 3.9558e-04

Epoch 69/100 20/20 [==============================] - 4s 197ms/step - loss: 3.8213e-04 - val_loss: 4.4212e-04

Epoch 70/100 20/20 [==============================] - 4s 199ms/step - loss: 3.3993e-04 - val_loss: 3.1317e-04

Epoch 71/100 20/20 [==============================] - 4s 201ms/step - loss: 3.5453e-04 - val_loss: 3.1094e-04

Epoch 72/100 20/20 [==============================] - 4s 199ms/step - loss: 3.2541e-04 - val_loss: 3.6810e-04

Epoch 73/100 20/20 [==============================] - 4s 193ms/step - loss: 3.2715e-04 - val_loss: 4.8602e-04

Epoch 74/100 20/20 [==============================] - 4s 194ms/step - loss: 3.8025e-04 - val_loss: 2.5199e-04

Epoch 75/100 20/20 [==============================] - 4s 196ms/step - loss: 3.1497e-04 - val_loss: 2.3976e-04

Epoch 76/100 20/20 [==============================] - 4s 195ms/step - loss: 3.0870e-04 - val_loss: 2.5330e-04

Epoch 77/100 20/20 [==============================] - 4s 201ms/step - loss: 2.9720e-04 - val_loss: 3.2768e-04

Epoch 78/100 20/20 [==============================] - 4s 203ms/step - loss: 2.9525e-04 - val_loss: 2.7359e-04

Epoch 79/100 20/20 [==============================] - 4s 202ms/step - loss: 3.6068e-04 - val_loss: 2.5968e-04

Epoch 80/100 20/20 [==============================] - 4s 205ms/step - loss: 3.0645e-04 - val_loss: 2.2749e-04

Epoch 81/100 20/20 [==============================] - 4s 205ms/step - loss: 2.7999e-04 - val_loss: 2.5462e-04

Epoch 82/100 20/20 [==============================] - 4s 199ms/step - loss: 3.4639e-04 - val_loss: 2.3313e-04

Epoch 83/100 20/20 [==============================] - 4s 198ms/step - loss: 2.7821e-04 - val_loss: 2.8185e-04

Epoch 84/100 20/20 [==============================] - 4s 195ms/step - loss: 2.5725e-04 - val_loss: 2.6936e-04

Epoch 85/100 20/20 [==============================] - 4s 196ms/step - loss: 2.6199e-04 - val_loss: 3.0618e-04

Epoch 86/100 20/20 [==============================] - 4s 201ms/step - loss: 2.8042e-04 - val_loss: 2.5239e-04

Epoch 87/100 20/20 [==============================] - 4s 196ms/step - loss: 2.4904e-04 - val_loss: 2.2943e-04

Epoch 88/100 20/20 [==============================] - 4s 195ms/step - loss: 2.5913e-04 - val_loss: 2.5312e-04

Epoch 89/100 20/20 [==============================] - 4s 192ms/step - loss: 2.6569e-04 - val_loss: 2.1963e-04

Epoch 90/100 20/20 [==============================] - 4s 192ms/step - loss: 2.5214e-04 - val_loss: 2.6289e-04

Epoch 91/100 20/20 [==============================] - 4s 197ms/step - loss: 2.5203e-04 - val_loss: 1.8534e-04

Epoch 92/100 20/20 [==============================] - 4s 195ms/step - loss: 2.4460e-04 - val_loss: 3.0759e-04

Epoch 93/100 20/20 [==============================] - 4s 197ms/step - loss: 7.4677e-04 - val_loss: 2.6719e-04

```
Epoch 94/100 20/20 [==============================] - 4s 194ms/step - loss: 3.3179e-04 - val_loss: 2.5489e-04

Epoch 95/100 20/20 [==============================] - 4s 196ms/step - loss: 2.4554e-04 - val_loss: 2.2221e-04

Epoch 96/100 20/20 [==============================] - 4s 189ms/step - loss: 2.4522e-04 - val_loss: 2.0084e-04

Epoch 97/100 20/20 [==============================] - 4s 189ms/step - loss: 2.5320e-04 - val_loss: 2.2505e-04

Epoch 98/100 20/20 [==============================] - 4s 188ms/step - loss: 2.2103e-04 - val_loss: 2.6176e-04

Epoch 99/100 20/20 [==============================] - 4s 191ms/step - loss: 2.4204e-04 - val_loss: 2.7066e-04

Epoch 100/100 20/20 [==============================] - 4s 192ms/step - loss: 2.7308e-04 - val_loss: 2.4121e-04
```

Prediction & Performance Metrics

In [49]:
```
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
```

In [50]:
```
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
```

In [53]:
```
import math
from sklearn.metrics import mean_squared_error

# Calculate RMSE performance metrics
math.sqrt(mean_squared_error(y_train, train_predict))
```

Out[53]:
```
166.02246091533394
```

In [55]:
```
# test data RMSE
math.sqrt(mean_squared_error(y_test, test_predict))
```

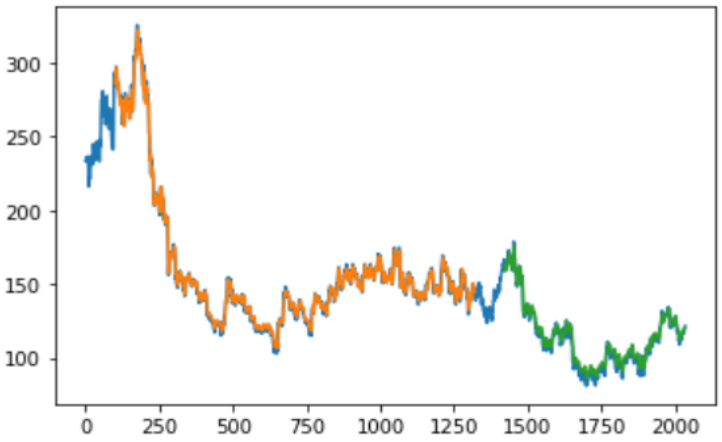
Out[55]:
```
116.52687747323411
```

Plotting Results

In [56]:
```
# shift train predictions for plotting
look_back = 100
trainPredictPlot = np.empty_like(df_close)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict

# shift test predictions for plotting
testPredictPlot = np.empty_like(df_close)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df_close)-1, :] = test_predict

# plot baseline and predictions
plt.plot(scaler.inverse_transform(df_close))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



In [58]:
```
len(test_data)
```

Out[58]:
```
713
```

In [59]:
```
x_input = test_data[613:].reshape(1, -1)
x_input.shape
```

Out[59]:
```
(1, 100)
```

In [60]:
```
temp_input = list(x_input)
temp_input = temp_input[0].tolist()
```

Prediction for next 30 days

In [63]:

```
# demonstrate prediction for next 30 days
from numpy import array

lst_op = []
n_steps = 100
i = 0
while i < 30:
  if(len(temp_input)>100):
    x_input = np.array(temp_input[1:])
    print("{} day input {}".format(i, x_input))
    x_input = x_input.reshape(1, -1)
    x_input = x_input.reshape((1, n_steps, 1))
    yhat = model.predict(x_input, verbose = 0)
    print("{} day output {}".format(i, yhat))
    temp_input.extend(yhat[0].tolist())
    temp_input = temp_input[1:]
    lst_op.extend(yhat.tolist())
    i += 1
  else:
    x_input = x_input.reshape((1, n_steps, 1))
    yhat = model.predict(x_input, verbose=0)
    print(yhat[0])
    temp_input.extend(yhat[0].tolist())
    print(len(temp_input))
    lst_op.extend(yhat.tolist())
    i += 1
print(lst_op)
```

```
[0.16743809]
101
```

[0.16743809]

101

1 day input [0.13848039 0.14011438 0.13888889 0.13541667 0.14011438 0.1380719

0.13071895 0.13071895 0.12867647 0.11846405 0.14644608 0.14808007

0.15910948 0.15992647 0.15788399 0.16441993 0.17892157 0.17933007

0.19260621 0.20812908 0.18974673 0.18055556 0.18239379 0.17708333

0.17810458 0.18055556 0.17810458 0.17851307 0.19607843 0.18913399

0.18954248 0.19403595 0.19444444 0.20200163 0.19771242 0.19934641

0.19873366 0.1997549 0.2128268 0.21568627 0.20445261 0.21772876

0.21098856 0.21425654 0.19750817 0.18811275 0.17851307 0.17381536

0.16033497 0.16564542 0.17116013 0.17422386 0.18035131 0.17401961

0.16278595 0.16973039 0.17810458 0.17034314 0.16830065 0.17279412

0.17544935 0.18382353 0.19138072 0.18913399 0.19097222 0.17238562

0.16830065 0.1693219 0.17177288 0.16156046 0.14971405 0.1503268

0.15196078 0.14726307 0.14501634 0.14603758 0.12479575 0.13112745

0.11397059 0.1190768 0.12377451 0.13562092 0.12908497 0.13459967

0.12806373 0.13031046 0.12724673 0.13521242 0.14522059 0.15257353

0.14848856 0.14338235 0.14562908 0.15236928 0.15400327 0.14971405

0.1621732 0.16319444 0.16584967 0.16743809]

1 day output [[0.17011692]]

2 day input [0.14011438 0.13888889 0.13541667 0.14011438 0.1380719 0.13071895

0.13071895 0.12867647 0.11846405 0.14644608 0.14808007 0.15910948

0.15992647 0.15788399 0.16441993 0.17892157 0.17933007 0.19260621

0.20812908 0.18974673 0.18055556 0.18239379 0.17708333 0.17810458

0.18055556 0.17810458 0.17851307 0.19607843 0.18913399 0.18954248

0.19403595 0.19444444 0.20200163 0.19771242 0.19934641 0.19873366

0.1997549 0.2128268 0.21568627 0.20445261 0.21772876 0.21098856

0.21425654 0.19750817 0.18811275 0.17851307 0.17381536 0.16033497

0.16564542 0.17116013 0.17422386 0.18035131 0.17401961 0.16278595

0.16973039 0.17810458 0.17034314 0.16830065 0.17279412 0.17544935

0.18382353 0.19138072 0.18913399 0.19097222 0.17238562 0.16830065

0.1693219 0.17177288 0.16156046 0.14971405 0.1503268 0.15196078

0.14726307 0.14501634 0.14603758 0.12479575 0.13112745 0.11397059

0.1190768 0.12377451 0.13562092 0.12908497 0.13459967 0.12806373

0.13031046 0.12724673 0.13521242 0.14522059 0.15257353 0.14848856

0.14338235 0.14562908 0.15236928 0.15400327 0.14971405 0.1621732

0.16319444 0.16584967 0.16743809 0.17011692]

2 day output [[0.1722583]]
3 day input [0.13888889 0.13541667 0.14011438 0.1380719 0.13071895 0.13071895

0.12867647 0.11846405 0.14644608 0.14808007 0.15910948 0.15992647

0.15788399 0.16441993 0.17892157 0.17933007 0.19260621 0.20812908

0.18974673 0.18055556 0.18239379 0.17708333 0.17810458 0.18055556

0.17810458 0.17851307 0.19607843 0.18913399 0.18954248 0.19403595

0.19444444 0.20200163 0.19771242 0.19934641 0.19873366 0.1997549

0.2128268 0.21568627 0.20445261 0.21772876 0.21098856 0.21425654

0.19750817 0.18811275 0.17851307 0.17381536 0.16033497 0.16564542

0.17116013 0.17422386 0.18035131 0.17401961 0.16278595 0.16973039

0.17810458 0.17034314 0.16830065 0.17279412 0.17544935 0.18382353

0.19138072 0.18913399 0.19097222 0.17238562 0.16830065 0.1693219

0.17177288 0.16156046 0.14971405 0.1503268 0.15196078 0.14726307

0.14501634 0.14603758 0.12479575 0.13112745 0.11397059 0.1190768

0.12377451 0.13562092 0.12908497 0.13459967 0.12806373 0.13031046

0.12724673 0.13521242 0.14522059 0.15257353 0.14848856 0.14338235

0.14562908 0.15236928 0.15400327 0.14971405 0.1621732 0.16319444

0.16584967 0.16743809 0.17011692 0.1722583]

3 day output [[0.17413737]]
4 day input [0.13541667 0.14011438 0.1380719 0.13071895 0.13071895 0.12867647

0.11846405 0.14644608 0.14808007 0.15910948 0.15992647 0.15788399

0.16441993 0.17892157 0.17933007 0.19260621 0.20812908 0.18974673

0.18055556 0.18239379 0.17708333 0.17810458 0.18055556 0.17810458

0.17851307 0.19607843 0.18913399 0.18954248 0.19403595 0.19444444

0.20200163 0.19771242 0.19934641 0.19873366 0.1997549 0.2128268

0.21568627 0.20445261 0.21772876 0.21098856 0.21425654 0.19750817

0.18811275 0.17851307 0.17381536 0.16033497 0.16564542 0.17116013

0.17422386 0.18035131 0.17401961 0.16278595 0.16973039 0.17810458

0.17034314 0.16830065 0.17279412 0.17544935 0.18382353 0.19138072

0.18913399 0.19097222 0.17238562 0.16830065 0.1693219 0.17177288

0.16156046 0.14971405 0.1503268 0.15196078 0.14726307 0.14501634

0.14603758 0.12479575 0.13112745 0.11397059 0.1190768 0.12377451

0.13562092 0.12908497 0.13459967 0.12806373 0.13031046 0.12724673

0.13521242 0.14522059 0.15257353 0.14848856 0.14338235 0.14562908

0.15236928 0.15400327 0.14971405 0.1621732 0.16319444 0.16584967

0.16743809 0.17011692 0.1722583 0.17413737]

4 day output [[0.17584969]]
5 day input [0.14011438 0.1380719 0.13071895 0.13071895 0.12867647 0.11846405

0.14644608 0.14808007 0.15910948 0.15992647 0.15788399 0.16441993

0.17892157 0.17933007 0.19260621 0.20812908 0.18974673 0.18055556

0.18239379 0.17708333 0.17810458 0.18055556 0.17810458 0.17851307

0.19607843 0.18913399 0.18954248 0.19403595 0.19444444 0.20200163

0.19771242 0.19934641 0.19873366 0.1997549 0.2128268 0.21568627

0.20445261 0.21772876 0.21098856 0.21425654 0.19750817 0.18811275

0.17851307 0.17381536 0.16033497 0.16564542 0.17116013 0.17422386

0.18035131 0.17401961 0.16278595 0.16973039 0.17810458 0.17034314

0.16830065 0.17279412 0.17544935 0.18382353 0.19138072 0.18913399

0.19097222 0.17238562 0.16830065 0.1693219 0.17177288 0.16156046

0.14971405 0.1503268 0.15196078 0.14726307 0.14501634 0.14603758

0.12479575 0.13112745 0.11397059 0.1190768 0.12377451 0.13562092

0.12908497 0.13459967 0.12806373 0.13031046 0.12724673 0.13521242

0.14522059 0.15257353 0.14848856 0.14338235 0.14562908 0.15236928

0.15400327 0.14971405 0.1621732 0.16319444 0.16584967 0.16743809

0.17011692 0.1722583 0.17413737 0.17584969]

5 day output [[0.17743438]]
5 day output [[0.17743438]]

6 day input [0.1380719 0.13071895 0.13071895 0.12867647 0.11846405 0.14644608

0.14808007 0.15910948 0.15992647 0.15788399 0.16441993 0.17892157

0.17933007 0.19260621 0.20812908 0.18974673 0.18055556 0.18239379

0.17708333 0.17810458 0.18055556 0.17810458 0.17851307 0.19607843

0.21772876 0.21098856 0.21425654 0.19750817 0.18811275 0.17851307

0.17381536 0.16033497 0.16564542 0.17116013 0.17422386 0.18035131

0.17401961 0.16278595 0.16973039 0.17810458 0.17034314 0.16830065

0.17279412 0.17544935 0.18382353 0.19138072 0.18913399 0.19097222

0.17238562 0.16830065 0.1693219 0.17177288 0.16156046 0.14971405

0.1503268 0.15196078 0.14726307 0.14501634 0.14603758 0.12479575

0.13112745 0.11397059 0.1190768 0.12377451 0.13562092 0.12908497

0.13459967 0.12806373 0.13031046 0.12724673 0.13521242 0.14522059

0.15257353 0.14848856 0.14338235 0.14562908 0.15236928 0.15400327

0.14971405 0.1621732 0.16319444 0.16584967 0.16743809 0.17011692

0.1722583 0.17413737 0.17584969 0.17743438]

6 day output [[0.1789193]]
7 day input [0.13071895 0.13071895 0.12867647 0.11846405 0.14644608 0.14808007

0.15910948 0.15992647 0.15788399 0.16441993 0.17892157 0.17933007

0.19260621 0.20812908 0.18974673 0.18055556 0.18239379 0.17708333

0.17810458 0.18055556 0.17810458 0.17851307 0.19607843 0.18913399

0.18954248 0.19403595 0.19444444 0.20200163 0.19771242 0.19934641

0.19873366 0.1997549 0.2128268 0.21568627 0.20445261 0.21772876

0.21098856 0.21425654 0.19750817 0.18811275 0.17851307 0.17381536

0.16033497 0.16564542 0.17116013 0.17422386 0.18035131 0.17401961

0.16278595 0.16973039 0.17810458 0.17034314 0.16830065 0.17279412

0.17544935 0.18382353 0.19138072 0.18913399 0.19097222 0.17238562

0.16830065 0.1693219 0.17177288 0.16156046 0.14971405 0.1503268

0.15196078 0.14726307 0.14501634 0.14603758 0.12479575 0.13112745

0.11397059 0.1190768 0.12377451 0.13562092 0.12908497 0.13459967

0.12806373 0.13031046 0.12724673 0.13521242 0.14522059 0.15257353

0.14848856 0.14338235 0.14562908 0.15236928 0.15400327 0.14971405

0.1621732 0.16319444 0.16584967 0.16743809 0.17011692 0.1722583

0.17413737 0.17584969 0.17743438 0.1789193]

7 day output [[0.18032923]]

8 day input [0.13071895 0.12867647 0.11846405 0.14644608 0.14808007 0.15910948

0.15992647 0.15788399 0.16441993 0.17892157 0.17933007 0.19260621

0.20812908 0.18974673 0.18055556 0.18239379 0.17708333 0.17810458

0.18055556 0.17810458 0.17851307 0.19607843 0.18913399 0.18954248

0.19403595 0.19444444 0.20200163 0.19771242 0.19934641 0.19873366

0.1997549 0.2128268 0.21568627 0.20445261 0.21772876 0.21098856

0.21425654 0.19750817 0.18811275 0.17851307 0.17381536 0.16033497

0.16564542 0.17116013 0.17422386 0.18035131 0.17401961 0.16278595

0.16973039 0.17810458 0.17034314 0.16830065 0.17279412 0.17544935

0.18382353 0.19138072 0.18913399 0.19097222 0.17238562 0.16830065

0.1693219 0.17177288 0.16156046 0.14971405 0.1503268 0.15196078

0.14726307 0.14501634 0.14603758 0.12479575 0.13112745 0.11397059

0.1190768 0.12377451 0.13562092 0.12908497 0.13459967 0.12806373

0.13031046 0.12724673 0.13521242 0.14522059 0.15257353 0.14848856

0.14338235 0.14562908 0.15236928 0.15400327 0.14971405 0.1621732

0.16319444 0.16584967 0.16743809 0.17011692 0.1722583 0.17413737

0.17584969 0.17743438 0.1789193 0.18032923]

8 day output [[0.18168488]]

9 day input [0.12867647 0.11846405 0.14644608 0.14808007 0.15910948 0.15992647
0.15788399 0.16441993 0.17892157 0.17933007 0.19260621 0.20812908

0.18974673 0.18055556 0.18239379 0.17708333 0.17810458 0.18055556

0.17810458 0.17851307 0.19607843 0.18913399 0.18954248 0.19403595

0.19444444 0.20200163 0.19771242 0.19934641 0.19873366 0.1997549

0.2128268 0.21568627 0.20445261 0.21772876 0.21098856 0.21425654

0.19750817 0.18811275 0.17851307 0.17381536 0.16033497 0.16564542

0.17116013 0.17422386 0.18035131 0.17401961 0.16278595 0.16973039

0.17810458 0.17034314 0.16830065 0.17279412 0.17544935 0.18382353

0.19138072 0.18913399 0.19097222 0.17238562 0.16830065 0.1693219

0.17177288 0.16156046 0.14971405 0.1503268 0.15196078 0.14726307

0.14501634 0.14603758 0.12479575 0.13112745 0.11397059 0.1190768

0.12377451 0.13562092 0.12908497 0.13459967 0.12806373 0.13031046

9 day output [[0.18300194]]

10 day input [0.11846405 0.14644608 0.14808007 0.15910948 0.15992647 0.15788399

0.16441993 0.17892157 0.17933007 0.19260621 0.20812908 0.18974673

0.18055556 0.18239379 0.17708333 0.17810458 0.18055556 0.17810458

0.17851307 0.19607843 0.18913399 0.18954248 0.19403595 0.19444444

0.20200163 0.19771242 0.19934641 0.19873366 0.1997549 0.2128268

0.21568627 0.20445261 0.21772876 0.21098856 0.21425654 0.19750817

0.18811275 0.17851307 0.17381536 0.16033497 0.16564542 0.17116013

0.17422386 0.18035131 0.17401961 0.16278595 0.16973039 0.17810458

0.17034314 0.16830065 0.17279412 0.17544935 0.18382353 0.19138072

10 day output [[0.18429129]]

.....

29 day input [0.19607843 0.18913399 0.18954248 0.19403595 0.19444444 0.20200163

0.19771242 0.19934641 0.19873366 0.1997549 0.2128268 0.21568627

0.20445261 0.21772876 0.21098856 0.21425654 0.19750817 0.18811275

0.17851307 0.17381536 0.16033497 0.16564542 0.17116013 0.17422386

0.18035131 0.17401961 0.16278595 0.16973039 0.17810458 0.17034314

0.16830065 0.17279412 0.17544935 0.18382353 0.19138072 0.18913399

0.19097222 0.17238562 0.16830065 0.1693219 0.17177288 0.16156046

0.14971405 0.1503268 0.15196078 0.14726307 0.14501634 0.14603758

0.12479575 0.13112745 0.11397059 0.1190768 0.12377451 0.13562092

0.12908497 0.13459967 0.12806373 0.13031046 0.12724673 0.13521242

0.14522059 0.15257353 0.14848856 0.14338235 0.14562908 0.15236928

0.15400327 0.14971405 0.1621732 0.16319444 0.16584967 0.16743809

0.17011692 0.1722583 0.17413737 0.17584969 0.17743438 0.1789193

0.18032923 0.18168488 0.18300194 0.18429129 0.1855596 0.18681036

0.18804459 0.18926202 0.19046153 0.19164188 0.19280186 0.19394074

0.19505823 0.19615443 0.19722986 0.19828527 0.19932155 0.2003396

0.20134029 0.20232445 0.20329262 0.20424539]

29 day output [[0.20518292]]

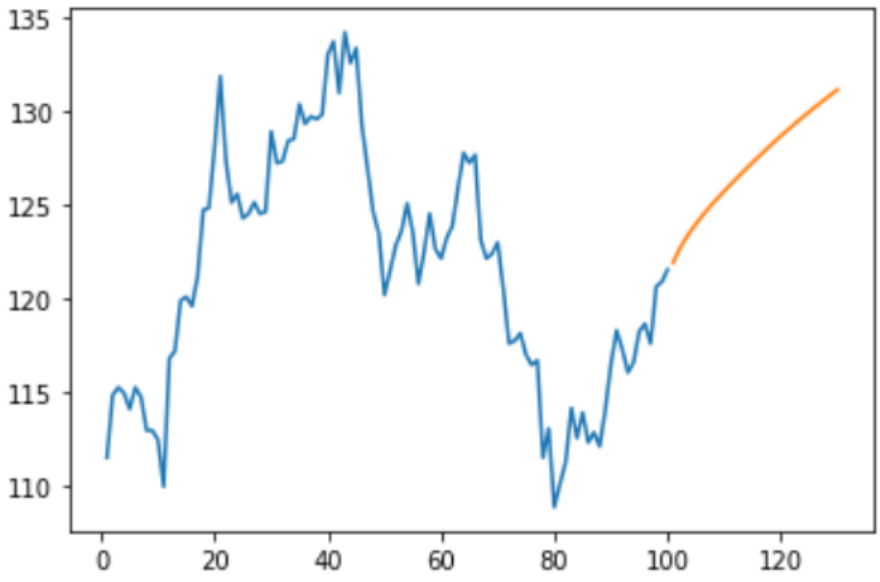
[[0.16743808984756470], [0.17011691629886627], [0.17225830256938934], [0.17413736879825592], [0.17584969103336334], [0.17743438482284546], [0.17891930043697357], [0.18032923340797424], [0.18168488144874573], [0.18300193548202515], [0.18429128825664520], [0.18555960059165955], [0.18681035935878754], [0.18804459273815155], [0.18926201760768890], [0.19046153128147125], [0.19164188206195830], [0.19280186295509338], [0.19394074380397797], [0.19505822658538818], [0.19615443050861360], [0.19722986221313477], [0.19828526675701140], [0.19932155311076350], [0.20033960044384003], [0.20134028792381287], [0.20232445001602173], [0.20329262316226960], [0.20424538850784302], [0.20518292486667633]]

In [65]:
```
day_new = np.arange(1, 101)
day_pred = np.arange(101, 131)
```

Plotting Predictions

In [86]:
```
plt.plot(day_new, scaler.inverse_transform(df_close[1935:]))
plt.plot(day_pred, scaler.inverse_transform(lst_op))
```

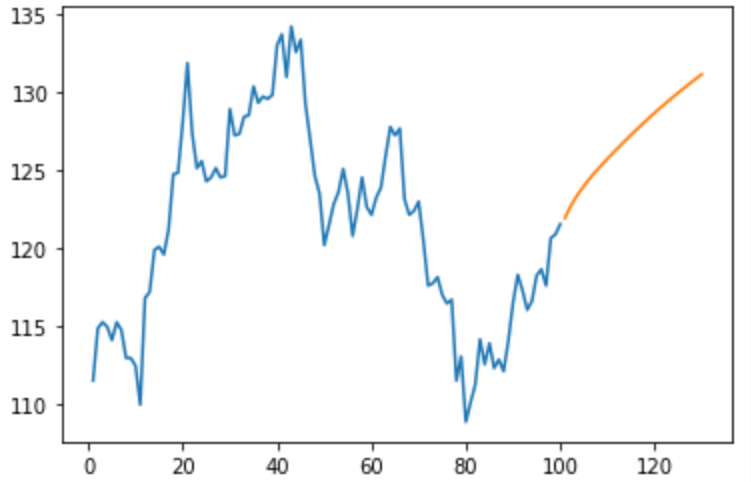
Out[86]:
```
[<matplotlib.lines.Line2D at 0x7f9b9128e090>]
```



CHAPTER - 4

RESULT AND DISCUSSION

The implementation of the proposed LSTM model using python which predicts the future price of share based on its historical data. The below visualization figure shows the visualization of SHARE prediction. In our paper the implementation of an algorithm which predicts the stock price of a share for a given period of time, the below graph from our algorithm will show the predicted price of shares. The result shown in the below graph is the plotted form of our algorithm outcome by applying 96 LSTM units for achieving the accuracy. Fig 2 is drawn from the original dataset and also shows the result by comparing its correctness with the trained model from the algorithm that is defined in the previous section. the “x” axis is share price. The “y” axis is days. The data for a slot is shown in Fig 1



CHAPTER - 5

CONCLUSION AND FUTURE SCOPE

Predicting stock market returns is a challenging task due to consistently changing stock values which are dependent on multiple parameters which form complex patterns.

Future direction could be:

1. analyzing the correlation between different cryptocurrencies and how that would affect the performance of our model.
2. adding features using technical analysis to check the model performance.
3. adding features from fundamental analysis to check how those affect the mode.
4. adding sentiment analysis from social networking e.g. twitter and new report to check model performance.

5. GRU network can also be tried with different activation e.g. ‘softsign” to check the performance.

Multivariate analysis of this nature requires a great amount of effort and time on feature engineering, data analysis, model training etc.

REFERENCES:-

- Hiba Sadia,- “Stock Market Prediction Using Machine Learning Algorithms”, IJEAT, 2019
- *Vijh, M., Chandola, D., Tikkiwal, V. A., & Kumar, A. (2020). Stock Closing Price Prediction using Machine Learning Techniques. Procedia Computer Science, 167, 599–606.*
- *Vanelin Valkov (2015). Hackers guide to machine learning*