| | |
|---|---|
| **Area Of Online Internship** | |
| **Intern Name** | **Udiga Bhanu Prakash** |
| **Name Of Institution** | **INDIAN INSTITUTE OF TECHNOLOGY, INDORE** |
| **Faculty Mentor Name** | **Vimal Bhatia** |
| **Duration** | **01/05/2021 TO 31/05/2021** |
| **Date Of Submission** | **29/05/2021** |

## Table Of Contents

# 1.Abstract

Predicting stock market is one of the most difficult tasks in the field of computation. There are many factors involved in the prediction – physical factors vs. physiological, rational and irrational behavior, investor sentiment, market rumors,etc. All these aspects combine to make stock prices volatile and very difficult to predict with a high degree of accuracy. We investigate data analysis as a game changer in this domain.As per efficient market theory when all information related to a company and stock market events are instantly available to all stakeholders/market investors, then the effects of those events already embed themselves in the stock price. So, it is said that only the historical spot price carries the impact of all other market events and can be employed to predict its future movement. Hence, considering the past stock price as the final manifestation of all impacting factors we employ Machine Learning (ML) techniques on historical stock price data to infer future trend. ML techniques have the potential to unearth patterns and insights we didn't see before, and these can be used to make unerringly accurate predictions. We propose a framework using LSTM (Long ShortTerm Memory) model and companies' net growth calculation algorithm to analyze as well as prediction of future growth of a company.

# 2.Introduction

Data analysis have been used in all business for data-driven decision making. In share market, there are many factors that drive the share price, and the pattern of the change of price is not regular. This is why it is tough to take a robust decision on future price. Artificial Neural Network (ANN) has the capability to learn from the past data and make the decision over future. Deep learning networks. such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) etc. works great with multivariate time series data. We train our model from the past stock data and calculate the future price of that stock. This future price use to calculate the future growth of a company. Moreover, we found a future growth curve from different companies. Thus we can analyze and investigate the similarity of one company's future curve over another. Stock price of a listed company in a stock exchange varies every time an order is placed for sell or buy and a transaction completes. An exchange collects all sell bids with expected price per stock (normally it is more than the price paid while bought by the investor) and all buy bids with or

without a price limit (normally an investor expects the future price of the stock will be more than the current price he is paying now) and a buysell transaction is committed when both bids have a match i.e. selling bid price is same with buying bid price of some buy-bid Fama in 1970 proposed efficient market hypothesis which says that in an efficient market (where all events are known to all stakeholders as an when it happens) the effect all market events are already incorporated in stock prices hence it is not possible to predict using past events or prices. The stock price of a company depends on many intrinsic as well as extrinsic attributes. Macro-economic conditions too play an important role in growth or decline of a sector as a whole. Some of the intrinsic factors could be company's net profit, liabilities, demand stability, competion in market, technically advanced assembly line, surplus cash for adverse situations, stakes in raw material supplier and finished product distributors etc.

# 3.Stock Market

Almost every country has one or more stock exchanges, where the shares of listed companies can be sold or bought. It is a secondary market place. When a company first lists itself in any stock exchange to become a public company, the promoter group sells substantial amount of shares to public as per government norms. During incorporation of a company shares are bought by promoter groups or institutional investors in a primary market. Once promoter offloads major portion of the shares to public relail investors, then those could be traded in secondary market i.e. in stock exchanges. In India the BSE(Bombay Stock Exchange) and the NSE(National Stock Exchange) are the two most active stock exchange. The BSE has around 5000 listed companies where as NSE had around 1600. Both the exchange has similar trading mechanism and market open time, closing time and settlement process. Stock exchanges helps individual investors to take part in the share market and allows to buy even a single share of some listed company with the help of a trading account and demat account. These online markets have revolutionized the Indian investment arena along with government initiative like tax benefit on equity investment, National Pension Scheme (NPS) investing in share market etc. Due to continuous reduction in

bank inrest rates and increasing inflation middle class investors are moving towards equity market from the safe heaven of fixed deposites. All these have helped to grow the capitalization of both the exchanges

## 4.LSTM Architecture

### 4.1 An overview of Recurrent Neural Network (RNN)

In a classical neural network, final outputs seldom act as an output for the next step but if we pay attention to a real-world phenomenon, we observe that in many situations our final output depends not only the external inputs but also on earlier output. For example, when humans read a book, understanding of each sentence depends not only on the current list of words but also on the understanding of the previous sentence or on the context that is created using past sentences. Humans don't start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. This concept of 'context' or 'persistence' is not available with classical neural networks. Inability to use context-based reasoning becomes a major limitation of traditional neural network. Recurrent neural networks (RNN) are conceptualized to alleviate this limitation.RNN are networked with feedback loops within to allow persistence of information. The Figure 1Error! Reference source not found. shows a simple RNN with a feedback loop and its unrolled equivalent version side by side

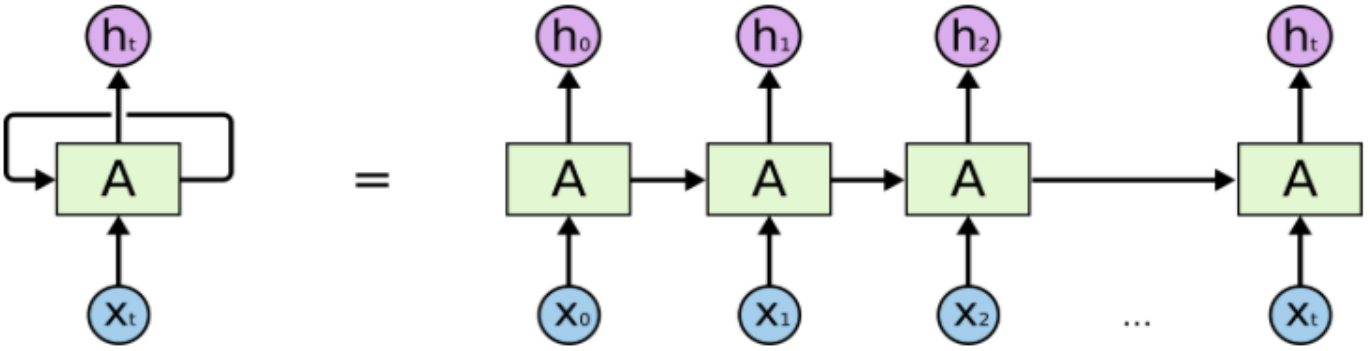



Figure-1:An unrolled recurrent  neural network

Initially (at time step t) for some input Xt the RNN generates an output of ht. In the next time step (t+1) the RNN takes two input Xt+1 and ht to generate the output ht+1. A loop allows information to be passed from one step of the network to the next. RNNs are not free from limitations though. When the 'context' is from near past it works great towards the correct output. But when an RNN has to depend on a distant 'context' (i.e. something learned long past) to produce 6 correct output, it fails miserably This limitation of the RNNs was discussed in great detail by Hochreiter [8] and Bengio, et al. [9]. They also traced back to the fundamental aspects to understand why RNNs may not work in long-term scenarios. The good news is that the LSTMs are designed to overcome the above problem.

## 4.2 LSTM Networks

Hochreiter Schmidhuber [10] introduced a special type of RNN which is capable of learning long term dependencies. Later on many other researchers improved upon this pioneering work in [11] [12] [13] [14]. LSTMs are perfected over the time to mitigate the long-term dependency issue. The evolution and development of LSTM from RNNs areexplained in [15] [16]. Recurrent neural networks are in the form of a chain of repeating modules of the neural network. In standard RNNs, this repeating module has a simple structure like a single tanh layer as shown in Figure 2.

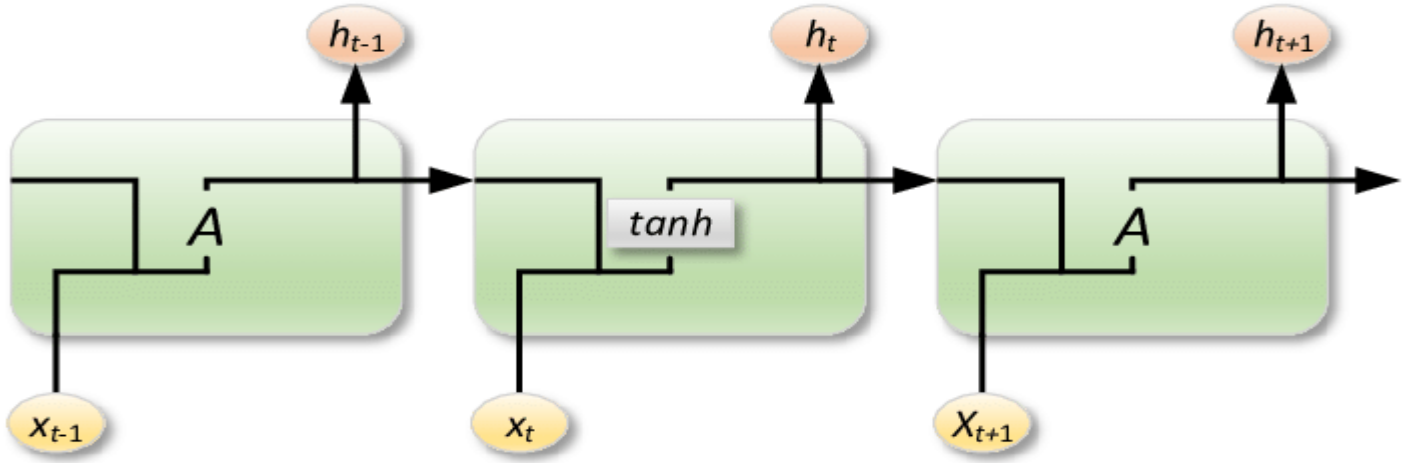
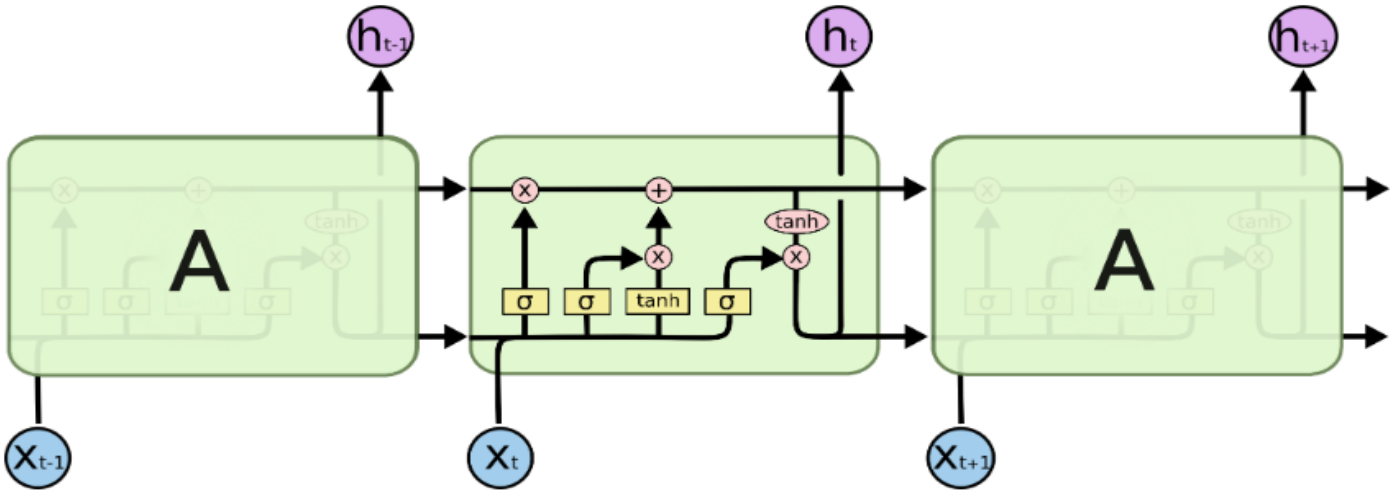



Figure-2:The repeating module in a standard RNN contains a single layer.

LSTMs follow this chain-like structure, however the repeating module has a different structure. Instead of having a single neural network layer, there are four layers, interacting in a very special way as shown in Figure 3. In Figure 3, every line represents an entire feature vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denotes its content being copied and the copies going to different locations.




The repeating module in an LSTM contains four interacting layers.

## 4.3 The Working of LSTM

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.The cell state is like a conveyor belt. This runs straight down the entire chain, having some minor linear interactions. LSTM has the ability to add or remove information to the cell state, controlled by structures called gates. Gates are used for optionally let information through. Gates are composed of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between 0 and 1, describing how much of each component should be let through. A value of 0 means “let nothing through,” while a value of 1 means “let everything through!”An LSTM has three of these gates, to protect and control the cell state The first step of LSTM is to decide what information are to be thrown out from the cell state. It is made by a sigmoid layer called the “forget gate layer.” It looks at ht , ht−1and xtxt, and outputs a number between 00 and 11 for each number in the cell state ct−1ct−1. A 11 represents “completely keep this” while a 00

represents “completely remove this.” In the next step it is decided what new information are going to be stored in the cell state. It has two parts. First, a sigmoid layer called the “input gate layer” decides which values are to be updated. Thereafter, a tanh layer creates a vector of new candidate values, C tC t, that could be added to the state. In the next step, these two are combined to create an update to the state. It is now time to update the old cell state, ct−1ct−1, into the new cell state ctct . We multiply the old state by ftft. Then we add itC titC t. This is the new candidate values, scaled by how much we decide to update each state value. Finally, we need to decide on the output. The output will be a filtered version of the cell state. First, we run a sigmoid layer which decides what parts of the cell state we’re going to output. Then, we put the cell state through tanh-tanh (to push the values to be between 11 and 11) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

## 5.Implementation

## Steps 5.1 Step1: Raw Stock Price Dataset:

Day-wise past stock prices of selected companies are collected from the yahoo finance official website.

## 5.2 Step2: Pre-processing:

This step incorporates the following: a) Data discretization: Part of data reduction but with particular importance, especially for numerical data b) Data transformation: Normalization. c) Data cleaning: Fill in missing values. d) Data integration: Integration of data files. After the dataset is transformed into a clean dataset, the dataset is divided into training and testing sets so as to evaluate. Creating a data structure with 60 timesteps and 1 output.

## 5.3 Step3: Feature Selection:

In this step, data attributes are chosen that are going to be fed to the neural network.In this study Date Close Price are chosen as selected features.

## 5.4 Step 4: Train the RNN model:

The NN model is trained by feeding the training dataset. The model is initiated using random weights and biases. Proposed LSTM model consists of a sequential input layer followed by 3 LSTM layers and then a dense layer with activation. The output layer again consists of a dense layer with a linear activation function.

## 5.5 Step5: Output Generation:

The RNN generated output is compared with the target values and error difference is calculated. The Backpropagation algorithm is used to minimize the error difference by adjusting the biases and weights of the neural network.

## 5.6 Step 6: Test Dataset Update:

Step 2 is repeated for the test data set.

## 5.7 Step 7: Error and companies' net growth calculation:

By calculating deviation we check the percentage of error of our prediction with respect to actual price.

## 5.8 Step 8: Visualization:

Using Keras[21] and their function APIs the prediction is

visualized.

## 5.9 Step 9: Investigate different time interval:

We repeated this process to predict the price at different time intervals. For our case, we took 2-month dataset as training to predict 3-month, 6-month, 1 year 3 years of close price of the share. In this different time span, we calculate the percentage of error in the future prediction. This would be different for different sectors. So, this will help to find a frame for the particular sector to predict future companies' net growth.

**<Problem details and solutions>** ●

# Internship Problem

**Stock Prediction Problem** - Predict the stock market price of next few days using previous stock market data (equity or indices) using machine learning or Deep learning.

1. Use News headlines as Data for prediction.

2. Use previous Equity data of Day open, close, low, high for prediction.

3. Any other stock Relative data.

Note: Try to improve the accuracy of the previously built model or implement it from scratch

## 1. Use News headlines as Data for prediction:

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
df=pd.read_csv("/Stock_Data.csv",encoding="ISO-8859-1")
train = df[df['Date'] < '20150101']
test = df[df['Date'] > '20141231']
data=train.iloc[:,2:27]
data.replace("[^a-zA-Z]"," ",regex=True, inplace=True)
list1= [i for i in range(25)]
new_Index=[str(i) for i in list1]
data.columns= new_Index
for index in new_Index:
    data[index]=data[index].str.lower()
headlines = []
```

```
for row in range(0,len(data.index)):
    headlines.append(' '.join(str(x) for x in data.iloc[row,0:25]))
countvector=CountVectorizer(ngram_range=(2,2))
traindataset=countvector.fit_transform(headlines)
randomclassifier=RandomForestClassifier(n_estimators=200,criterion='entropy')
randomclassifier.fit(traindataset,train['Label'])
test_transform= []
for row in range(0,len(test.index)):
    test_transform.append(' '.join(str(x) for x in test.iloc[row,2:27]))
test_dataset = countvector.transform(test_transform)
predictions = randomclassifier.predict(test_dataset)
matrix=confusion_matrix(test['Label'],predictions)
print(matrix)
score=accuracy_score(test['Label'],predictions)
print(score)
report=classification_report(test['Label'],predictions)
print(report)
df1=pd.DataFrame(predictions,index=df.index[df['Date'] > '20141231'])
test.insert(2,"Predicted label",df1)
df3=test.iloc[:,1:3]
```

Google colab link for this problem is :

https://colab.research.google.com/drive/18C9nPnHsc6KMAwwMTzUwU9a6RLWVH9Q?usp=sharing#scrollTo=fQxdfpuhC0Uv

## 2. Use previous Equity data of Day open, close, low, high for prediction.

## I did second problem in two methods

**Model-1:Taking Close Column as Input itself and getting the Output as Close Column Using LSTM**

I have trained the data in such a way that first 1 to 100 values goes into input and get 101 value as output.In next step 2 to 101 values as input and get 102 value as output.next 3 t0 102 values as input and 103 value as

output and vice versa....

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense,Dropout
import math
from sklearn.metrics import mean_squared_error
path="/content/NSE-TATAGLOBAL11.csv"
df=pd.read_csv(path)
df1=df[::-1]
l=[i for i in range(1235)]
df1.insert(8,"Index",l)
df1.set_index("Index",drop=True,inplace=True)
close_column=df1.loc[:,["Close"]]
scaler=MinMaxScaler(feature_range=(0,1))
close_column_scaled=scaler.fit_transform(close_column)
train_size=int(len(close_column_scaled)*0.6)
test_size = int(len(close_column_scaled)-train_size)
train_data=close_column_scaled[0:train_size,:]
test_data=close_column_scaled[train_size:len(close_column_scaled),:]
def Windowed_dataset(dataset,timestamp=1):
    X,Y=[],[]
    for i in range(len(dataset)-timestamp-1):
        A=dataset[i:i+timestamp,0]
        X.append(A)
        Y.append(dataset[i+timestamp,0])
    return np.array(X),np.array(Y)
timestamp=100
X_train,Y_train=Windowed_dataset(train_data,timestamp)
X_test,Y_test=Windowed_dataset(test_data,timestamp)
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)
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))
model.compile(loss='mean_squared_error',optimizer='adam')
history=model.fit(X_train,Y_train,validation_data=(X_test,Y_test),epochs=100,batch_size=64,ve
rbose=1)
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.legend()
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
train_predict = np.repeat(train_predict, close_column.shape[1], axis=-1)
train_predict=scaler.inverse_transform(train_predict)
test_predict=np.repeat(test_predict,close_column.shape[1],axis=-1)
test_predict=scaler.inverse_transform(test_predict)
print(math.sqrt(mean_squared_error(Y_train,train_predict)))
print(math.sqrt(mean_squared_error(Y_test,test_predict)))
look_back=100
trainPredictPlot=np.empty_like(close_column)
trainPredictPlot[:]=np.nan
trainPredictPlot[look_back:len(train_predict)+look_back]=train_predict
testPredictPlot=np.empty_like(close_column)
testPredictPlot[:] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(close_column)-1] = test_predict
df1["Date"]=pd.to_datetime(df1["Date"])
f1={"family":'sans-serif',"color":"k",'size':20}
plt.figure(figsize=(16,6))
plt.plot(df1["Date"],testPredictPlot,label="Predicted Stock Price")
plt.plot(df1["Date"][-393:],close_column[-393:],label="Original Stock Price")
plt.xlabel("Date",fontdict=f1)
plt.ylabel("Price",fontdict=f1)
plt.legend()
plt.tight_layout()
df1["Date"]=pd.to_datetime(df1["Date"])
f2={"family":'Tahoma',"color":"k",'size':20}
```

```
plt.figure(figsize=(16,6))
plt.plot(df1["Date"],df1["Close"],color="k",label="Original Close Prices")
plt.plot(df1["Date"],trainPredictPlot,color="b",label="Predicted train data")
plt.plot(df1["Date"],testPredictPlot,color="r",label="Predicted Test data")
plt.xlabel("Year",fontdict=f2)
plt.ylabel("Price",fontdict=f2)
plt.legend()
#Lets us Predict 30 days
x_input=test_data[394:].reshape(1,-1)
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
lst_output=[]
n_steps=100
i=0
while(i<30):

    if(len(temp_input)>100):
        #print(temp_input)
        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))
        #print(x_input)
        yhat = model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)
        lst_output.extend(yhat.tolist())
        i=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))
```

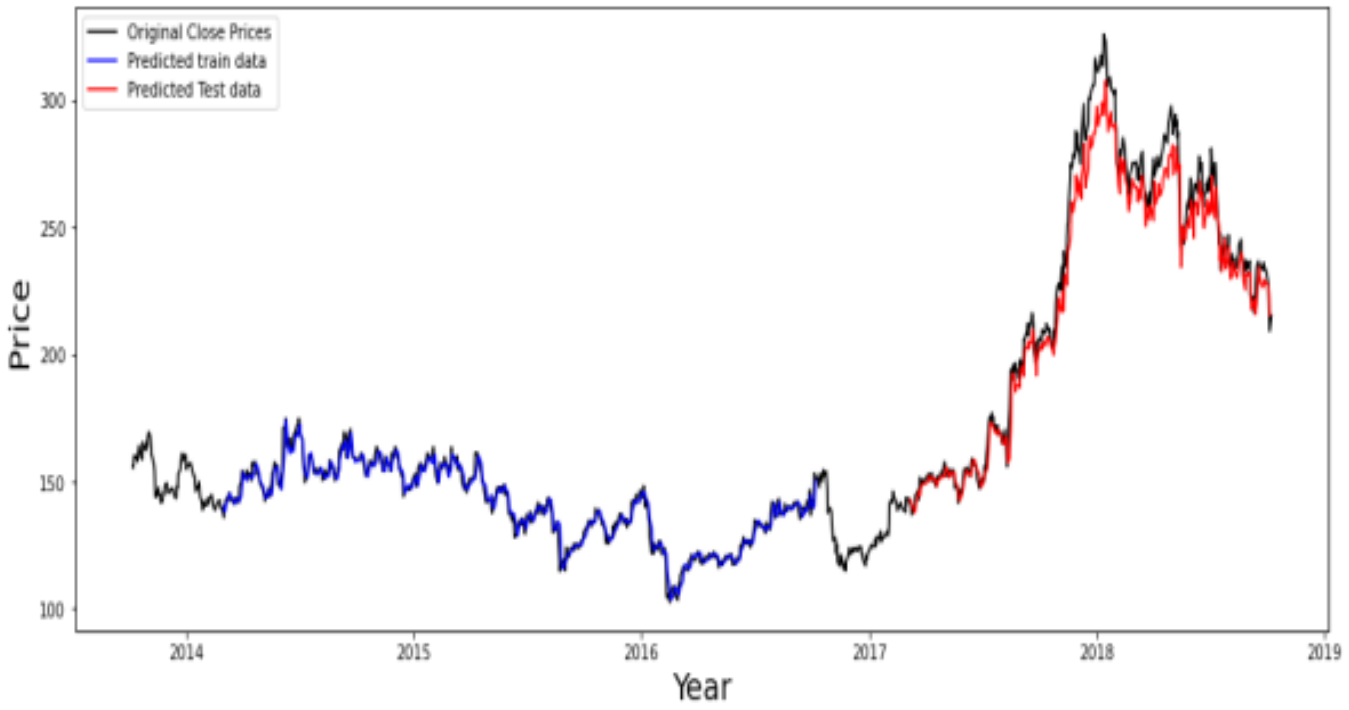
```
        lst_output.extend(yhat.tolist())
        i=i+1


print(lst_output)
day_new=np.arange(1,101)
day_pred=np.arange(101,131)
plt.figure(figsize=(16,6))
plt.plot(day_new,scaler.inverse_transform(close_column_scaled[1135:]),label="Previous 100
days close Prices")
plt.plot(day_pred,scaler.inverse_transform(lst_output),label="Predicted 30 days close
prices")
plt.legend()
train_dates=pd.to_datetime(df1["Date"])
train_dates=list(train_dates)[-1]
predict_period_dates=pd.date_range(train_dates,periods=30,freq="1d").tolist()
forecast_dates = []
for time_i in predict_period_dates:
    forecast_dates.append(time_i.date())


df_forecast = pd.DataFrame({'Date':np.array(forecast_dates),
'Close':scaler.inverse_transform(lst_output)[:,0]})
df_forecast['Date']=pd.to_datetime(df_forecast['Date'])
plt.figure(figsize=(16,6))
plt.plot(df_forecast["Date"],df_forecast["Close"],"r")
plt.xlabel("Date",color="k",fontsize=20)
plt.ylabel("Price",color="k",fontsize=20)
plt.title("Stock price prediction for next 30 days",color="k",fontsize=20)
plt.tight_layout()
Df_forecast
```

**Result:**

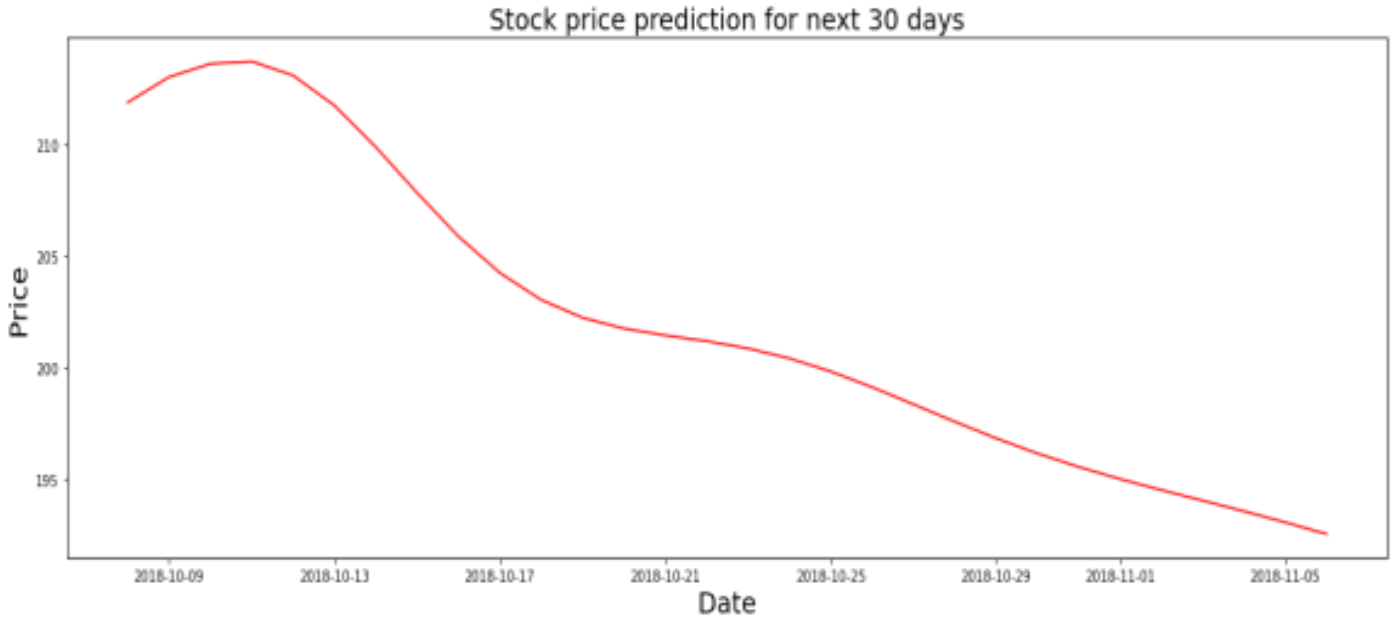
**1.**


Predicted Stock Price
Original Stock Price
Price
325
300
275
250
225
200
175
150
2017-03
2017-05
2017-07
2017-09
2017-11
2018-01
2018-03
2018-05
2018-07
2018-09
2018-11
Date

**2.**




3.




**Link of this code is**

**https://colab.research.google.com/drive/1JUzqlkE97Mik7pVnTse-LBzv6V6HDtxi?usp=sharing**

**Model-2:**

**Taking Open,High,Low,Last columns as input and close column as desired output.**

```
#Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense, Dropout
import math
from sklearn.metrics import mean_squared_error
path="/content/NSE-TATAGLOBAL11.csv"
df=pd.read_csv(path)
```

#Here Date starts from 2018-10-08 and goes upto 2013-10-08.So we have to reverse the DataFrame for better understanding and plotting

```
df1=df[::-1]
l=[i for i in range(1235)]
df1.insert(8,"Index",l)
df1.set_index("Index",drop=True,inplace=True)
```

#Taking input as Open,High,Low,Last columns and Output as close column variable.

```
output=df1[["Close"]]
features = ["Open","High","Low","Last"]
```

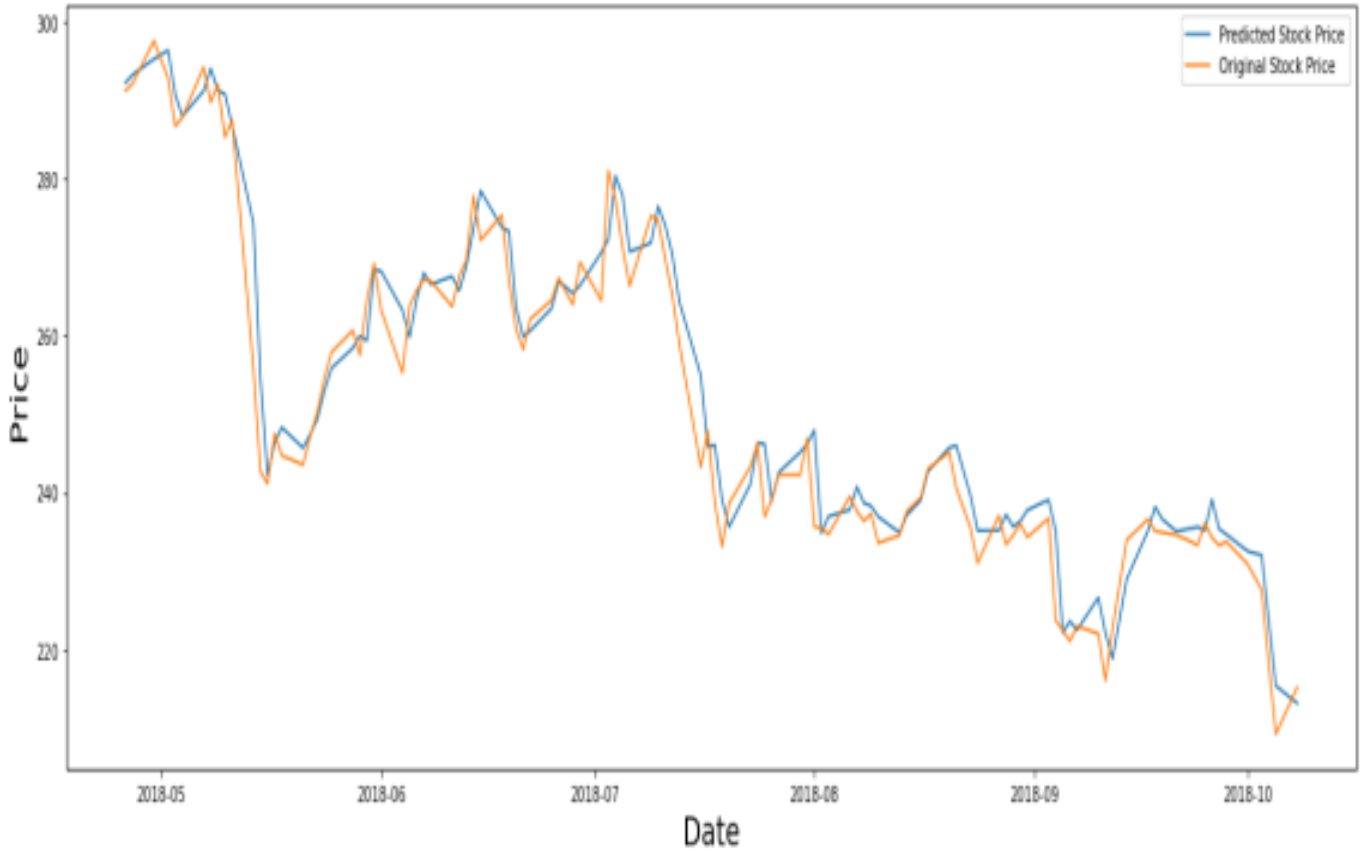
#LSTM uses sigmoid and tanh that are sensitive to magnitude so values need to be normalized Normalize the dataset

```
scaler = MinMaxScaler(feature_range=(0,1))
feature_transform = scaler.fit_transform(df1[features])
output_variable=scaler.fit_transform(output)
feature_transform= pd.DataFrame(columns=features, data=feature_transform, index=df1.index)
from sklearn.model_selection import TimeSeriesSplit
timesplit= TimeSeriesSplit(n_splits=10)
for train_index, test_index in timesplit.split(feature_transform):
```

```
        X_train, X_test = feature_transform[:len(train_index)],
feature_transform[len(train_index): (len(train_index)+len(test_index))]
         y_train, y_test = output_variable[:len(train_index)],
output_variable[len(train_index): (len(train_index)+len(test_index))]
#As required for LSTM networks, we require to reshape an input data into 3D
Tensor trainX =np.array(X_train)
testX =np.array(X_test)
X_train = trainX.reshape(X_train.shape[0],X_train.shape[1],1)
X_test = testX.reshape(X_test.shape[0],X_test.shape[1],1)
model=Sequential()
model.add(LSTM(64,return_sequences=True,input_shape=(trainX.shape[1],1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(32))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
history=model.fit(X_train, y_train,validation_data=(X_test,y_test), epochs=100,
batch_size=64, verbose=1)
#Plot of Training loss vs Validation Loss
# plt.plot(history.history['loss'], label='Training loss')
# plt.plot(history.history['val_loss'], label='Validation loss')
# plt.legend()
train_pred= model.predict(X_train)
test_pred=model.predict(X_test)
#Plot for Test Predictions and Test data
df1["Date"]=pd.to_datetime(df1["Date"])
f1={"family":'sans-serif',"color":"k",'size':20}
plt.figure(figsize=(16,6))
plt.plot(df1["Date"][-112:],scaler.inverse_transform(test_pred),label="Predicted Stock
Price")
plt.plot(df1["Date"][-112:],scaler.inverse_transform(y_test),label="Original Stock Price")
plt.legend()
plt.tight_layout()
l1=train_pred.tolist()
l2=test_pred.tolist()
l1.extend(l2)
l3=np.array(l1)
```

```
predicted=scaler.inverse_transform(l3)
df1["Date"]=pd.to_datetime(df1["Date"])
#Plot of Original and Predicted stock prices
plt.plot(df1["Date"],df1["Close"],label="Original Stock Price")
plt.plot(df1["Date"],predicted,label="Predicted Stock Price")
plt.legend()
df1.insert(8,"Close Predicted",predicted)
df1.drop("Total Trade Quantity",inplace=True,axis=1)
df1.drop("Turnover (Lacs)",inplace=True,axis=1)
df1.to_csv("Newly predicted values.csv")
#New DataFrame with predicted values of close column
Df1
```

**RESULT:**



Link of this code is :

https://colab.research.google.com/drive/1RfZpIg_T6YMRxwJ7k1-AN_Z4J4ZsXv05?usp=sharing#scrollTo=MdAQaYaXq0Jd

**REFERENCES**


1. F. a. o. Eugene, ”Efficient capital markets: a review of theory and empirical work,” Journal of finance, vol. 25, no. 2, pp. 383-417, 1970.

2. Z. A. Farhath, B. Arputhamary and L. Arockiam, ”A Survey on ARIMA Forecasting Using Time Series Model,” Int. J. Comput. Sci. Mobile Comput, vol. 5, pp. 104-109, 2016.

3. S. Wichaidit and S. Kittitornkun, ”Predicting SET50 stock prices using CARIMA (cross correlation ARIMA),” in 2015 International Computer Science and Engineering Conference (ICSEC), IEEE, 2015, pp. 1-4.

4. D. Mondal, G. Maji, T. Goto, N. C. Debnath and S. Sen, ”A Data Warehouse Based Modelling Technique for Stock Market Analysis,” International Journal of Engineering Technology, vol. 3, no. 13, pp. 165-170, 2018.

5. G. Maji, S. Sen and A. Sarkar, ”Share Market Sectoral Indices Movement Forecast with Lagged Correlation and Association Rule Mining,” in International Conference on Computer Information Systems and Industrial Management, Bialystok, Poland, Sprigner, 2017, pp. 327-340.

6. M. Roondiwala, H. Patel and S. Varma, ”Predicting stock prices using LSTM,” International Journal of Science and Research (IJSR), vol. 6, no. 4, pp. 1754-1756, 2017.

7. T. Kim and H. Y. Kim, ”Forecasting stock prices with a feature fusion LSTM-CNN model using different representations of the same data,” PloS one, vol. 14, no. 2, p. e0212320, April 2019.

8. S. Selvin, R. Vinayakumar, E. A. Gopalkrishnan, V. K. Menon and K. P. Soman, ”Stock price prediction using LSTM, RNN and CNN-sliding window model,” in International Conference on Advances in Computing, Communications and Informatics, 2017.

9. S. Hochreiter, ”Untersuchungen zu dynamischen neuronalen Netzen,” Diploma, Technische Universit¨at M¨unchen, vol. 91, no. 1, 1991.

10. Y. Bengio, P. Simard, P. Frasconi and others, ”Learning long-term dependencies with gradient descent is difficult,” IEEE transactions on neural networks, vol. 5, no. 2, pp. 157-166, 1994.