

# **Application of Deep learning for prediction of Stock Market**

## **Group Members :**

- 1) Utsav Agarwal (20MA20061)
- 2) Sindhi Vishal Maheshbhai (20AG30033)
- 3) Parth Barhate (20BT30016)
- 4) Chittireddy Akhil Reddy (20MA20019)

## **Abstract :**

Stock price forecasting has been a key area of research for a very long time. There are formal arguments proving that precise modelling and the creation of suitable variables can result in models that can be used to predict stock prices and stock price movement patterns quite accurately. In order to find patterns in the price movements of stocks utilising cutting-edge technology for data processing and mining researchers have also studied technical analysis of stocks.

Although proponents of the efficient market hypothesis contend that precise stock price forecasting is impossible, Since financial markets are notoriously unstable, it is well acknowledged that predicting stock prices is a difficult undertaking. However, advancement in computational power and development of cutting edge technologies and methods have made it possible for many market players or researchers to attempt to anticipate stock prices using various statistical, economic, or even neural network models in order to gain profits or comprehend the essence of the equity market.

It is a challenging task to choose which stock rate prediction technique is more trustworthy and accurate in order to generate a buy or sell signal for particular stocks because it is well accepted that wealth brings luxury and comfort. The prediction of stock indexes using traditional time series analysis has shown to be difficult. An artificial neural network might be the best choice for the task. A neural network may extract pertinent information from a large data source. Using the idea of artificial neural networks with NLP and time series, we provide a real-time application of artificial neural networks for stock market projections in this project.

In this paper, we present a hybrid modelling strategy for predicting stock prices using various deep learning and machine-learning based models..

## **I. Introduction :**

Theoretical and practical interest in price forecasting in equities markets is high. On the one hand, a somewhat accurate prognosis maximises returns for investors. Before making an investment choice, many market participants, particularly institutional ones, invest a significant amount of time and money in gathering and analysing pertinent data. On the other side, academics frequently use the ability of price forecasting to assess market effectiveness. Additionally, they create, use, or modify various models to increase the prediction capacity. Finding a "good" strategy to predict stock prices more accurately will always be a hot topic in academia and the financial sector.

The stock market is primarily dynamic, nonlinear, intricate, nonparametric, and chaotic in nature, making equity price prediction a difficult assignment in the financial time series prediction process . Additionally, a variety of macroeconomic factors, including political developments, corporate practices, prevailing economic

conditions, commodity price indices, interest rates, investor expectations, institutional. Other significant elements are the decisions made by investors and their psychological makeup.

Technical analysis of stock prices has been modelled by researchers with the aim of identifying trends in stock movement that benefit investors. Numerous economic and stock price-related indicators have been suggested in the literature for this aim. Bollinger Band, Moving Average Convergence Divergence (MACD), Relative Strength Index (RSI), Moving Average (MA), Momentum Stochastics (MS), and Meta Sine Wave are a few of these indicators (MSW). Other well-known patterns in stock price movements, such as the head and shoulders, triangle, flag, Fibonacci fan, Andrew's pitchfork, etc., are also thought to be significant indicators for stock market investment. Potential investors can use these methods' effective visualisations to help them choose the best investments.

In the current study, a variety of machine learning and deep learning-based prediction models are proposed for precisely forecasting the movement of the APPLE, TESLA, GOOGLE, MICROSOFT and AMAZON stock prices. The training dataset consists of the historical index values of these companies from January 02, 2015, to September 30, 2020. Predictive models are created using the training dataset, and using the models, the open values of the index of these companies are projected for the test period, which ran from October 01, 2020, to February 26, 2021. The powerful deep learning-based long- and short-term memory (LSTM) network is added to the predictive framework, which significantly increases the predictive potential of the models.

The rest of the paper is divided into the following sections. We clearly identify the issue at hand in Section 2. A quick summary of the relevant research on stock

price movement prediction is provided in Section 3. We outline our study technique in Section 4. Extensive Results on how well the predictive models performed are shown in Section 5. The specifics of every predictive model created for this study are described in this section, along with the outcomes they generated. The paper is concluded in Section 6.

## **II. Problem Statement :**

Our research aims to gather APPLE, TESLA, GOOGLE, MICROSOFT and AMAZON stock prices from the Indian NSE over a time span of about five and a half years in order to build a solid framework for predicting these technology companies' index values. We postulate that a deep learning or machine learning model may be able to learn from the characteristics of the daily index values' historical movement patterns of these companies, and that these features may then be successfully used to predict these technology companies' index values in the future.

In the current proposition, we selected a forecast horizon of one year for the machine learning models and one week for the deep learning models, and we showed that the future values of these companies' index can be forecasted using these models with a respectable level of accuracy. In our prior work, we employed RNN-based deep learning models to create extremely accurate predictive frameworks for predicting future index values of these companies, which helped to support our hypothesis. In the current work, we create long and short-term memory (LSTM) network-based models using four different ways to increase the prediction accuracy of our forecasting models.

It should be emphasised that we are not discussing concerns with short-term forecasting that are relevant to intraday traders in this study. The arguments made in this paper, however, are pertinent for medium-term investors who would be interested in a weekly projection of the index values of APPLE, TESLA, GOOGLE, MICROSOFT and AMAZON.

### **III. Related Work :**

Based on the usage of variables and the method used to represent the problem, the current research on time series forecasting and stock price prediction may be roughly divided into three clusters. The models that use bivariate or multivariate regression on cross-sectional data make up the majority of the first type of work.

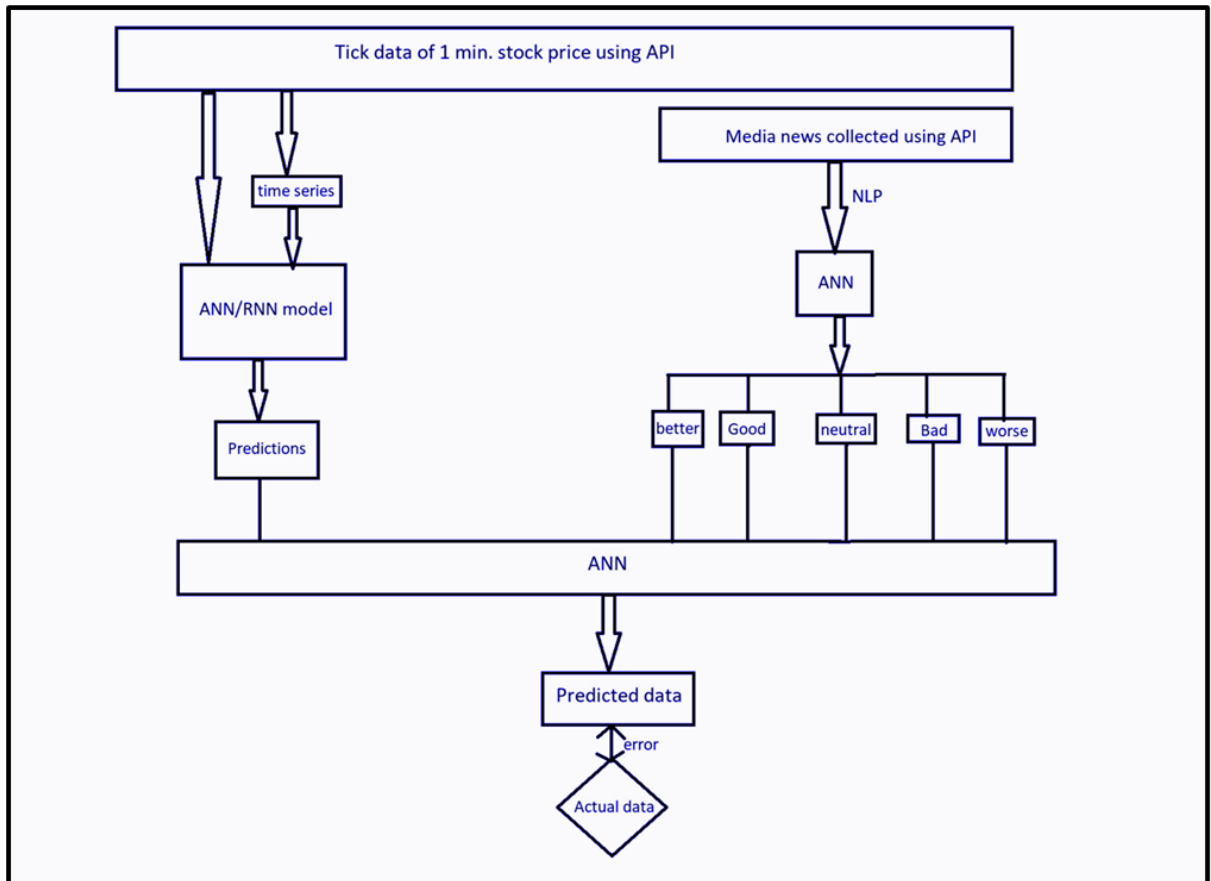
These models frequently fail to yield extremely accurate findings due to their intrinsic simplicity and the falsity of the linearity assumptions they make. The hypotheses in the second category forecast stock prices using time series theory as well as other econometric tools including the Granger Causality Test, autoregressive distributed lag (ARDL), vector autoregression (VAR), and quantile regression. Using machine learning, deep learning, and natural language processing, the third type of work consists of learning-based approach propositions.

One of the major flaws of the current theories proposed in the literature for stock price prediction, with the exception of the category of work that makes use of learning-based approaches, is their inability to correctly predict extremely dynamic and swiftly changing patterns in stock price movement. By utilising the strength of machine learning and deep learning-based models to create a very solid, dependable, and accurate framework for stock index prediction, we aim to solve the issue in this

study. We have specifically deployed an LSTM network-based deep learning model and evaluated how well it performs in forecasting future stock index values.

### III. Methodology :

Flow chart of our project :



And Some of the Models we have used are

#### 1) LSTM :

A Recurrent Neural Network (RNN) based architecture called LSTM (Long Short-Term Memory) is frequently used in time series forecasting and natural language processing. The LSTM fixes the serious short-memory problem that recurrent neural networks experience. The LSTM controls

whether to keep, forget, or ignore data points via a sequence of "gates," each with its own RNN.

Exploding and vanishing gradient issues are also resolved using LSTMs. Simply said, as a neural network trains, these issues arise as a result of recurrent weight modifications. Repeated epochs cause gradients to grow or shrink, and with each change, it becomes simpler for the gradients of the network to compound in either way. The gradients are either made way too big or way too little by this compounding. Exploding and disappearing gradients are significant drawbacks of utilising conventional RNNs, although LSTM architecture significantly reduces these problems. Following a prediction, the model is updated to forecast the following value in the series. The model experiences some inaccuracy with each prediction. Values are "squashed" using (usually) sigmoid & tanh activation functions before gate entrance & output to prevent exploding gradients.

## **2) NLP:**

A computer program's capacity to comprehend natural language, or human language as it is spoken and written, is known as natural language processing (NLP). A part of artificial intelligence, it is.

The study of natural language processing (NLP) is advancing quickly, and there are many new applications for it. Text prediction is one of NLP's tasks. A sentence or group of words are provided to a prediction model in a text prediction task. The model's job is to then show the user the word or words that have the best likelihood of continuing the initial sequence. A text prediction model is trained for this project using a selection of texts that have been assessed.

In this project we built an LSTM-based Recurrent Neural Network (RNN) to predict APPLE, TESLA, GOOGLE, MICROSOFT and AMAZON stock prices step by step. It is split into 7 parts as below

- 1) Problem Statement
- 2) Data processing
- 3) Model building
- 4) Model fitting
- 5) Model prediction
- 6) Result visualisation

#### **A. Problem Statement :**

We have the stock prices of APPLE, TESLA, GOOGLE, MICROSOFT and AMAZON shares from January 02, 2015, to September 30, 2020. The assignment is to forecast the stock price trend from October 01, 2020, to February 26, 2021 and compare with it the actual values. The future changes in stock price are independent of the past, according to Brownian Motion. The stock price cannot be predicted precisely, but the increasing and falling tendencies can be anticipated.

#### **Sentiment News analysis for predicting stock prices**

Earlier news sentiment analysis methods used the aggregate data of the social media platforms like daily twitter post data to predict the overall stock market movement. For example, the stock movement predictions were done using the news articles using Support Vector Machines(SVM) or using some deep learning techniques by applying models like LSTM, RNN and GRUs .

Support vector machines (SVMs) are promising methods for financial time-series prediction because they employ a risk function composed of the empirical error plus a



regularised term derived from the structural risk reduction principle. SVM is used in this work to forecast the stock price index. In addition, by contrasting SVM with back-propagation neural networks and case-based reasoning, this study investigates the viability of using SVM in financial forecasting. The outcomes of the experiment demonstrate that SVM offers a promising substitute for stock market forecasting.

Gated recurrent units (GRUs) is a gating technique for recurrent neural networks. The GRU has fewer parameters than an LSTM because it doesn't have an output gate, but it is similar to an LSTM in that it has a forget gate. It was discovered that the performance of GRU and LSTM were comparable for a few tasks involving polyphonic music modelling, speech signal modelling, and natural language processing. It has been demonstrated that GRUs perform better on some smaller and less frequent datasets.

In this project, our aim is to predict the individual stock movement rather than an aggregate prediction. The difficulties faced in achieving this is, it is difficult to scrape data/news articles relevant to our stock only. Moreover, considering the fact that some of the news articles are purposely written on an underlying bias to the sentiment.

From some experimental results, it is accepted that, for solving single-stock problems, the title of the news article is actually more important than the content present in it. Similarly, it is also found that training the model on the entire article will give the similar results as by training it on the summary of that article only.

First, 5 parameters are collected as an input to train the model on it. These are : 1) opening price, 2) closing price, 3)High, 4)Low, 5)volume and the output is classified into two categories. If the stock is moved up then it is labelled as 1 otherwise 0. NLTK library was used for sentiment predictions on 600 days of Amazon returns and a simple LSTM forecasting with only 8 hidden units and is trained for 5 epochs only. The preferred Python API for NLP (Natural Language Processing) is NLTK (Natural Language Toolkit). To prepare text data for subsequent analysis, such as with ML models, it is a very potent tool. It assists in turning words into numbers, which the model can use to its advantage.

Using neural networks, sentence's event can be extracted by learning 100 -length word embedding vectors using skip-gram algorithms for the words in the news titles. For a more complex approach, StanfordNLP OpenIE extractor is used which extracts a sentence's underlying event for every news article by converting event tuples into word embedding tuples and is fed into a neural tensor network.

Open information extraction (open IE) is the process of extracting relation tuples from plain text, often binary relations, such as (Mark Zuckerberg; founded; Facebook). The primary distinction between this type of information extraction and others is that there is no need to predetermine the schema for these relations; often, the relation name is simply the text that connects two inputs. Barack Obama was born in Hawaii, for instance, would result in the triple (Barack Obama; was born in; Hawaii), which corresponds to the relation was-born-in in the open domain (Barack-Obama, Hawaii). Each sentence is first divided into a group of entailed clauses by the algorithm. The result is a set of entailed reduced sentence fragments, with each clause then being

maximally condensed. The system outputs these pieces after segmenting them into OpenIE triples.

After having sentence embeddings for news headlines and labelled with future up or down movement of the company's stock, the LSTM binary classification was trained and achieved the 52% out-of-sample accuracy for 8000 sentence embeddings provided and the accuracy increases as the model is trained more and more. This concludes that we simply lack enough data to create a robust model as it leads to drastic over-fitting.

Our project's data is in this [link](#) of the codes and predictions of the stock prices of 5 different companies namely APPLE, TESLA, GOOGLE, MICROSOFT and AMAZON. Some of them are shown below :

### **OUR PROJECT's MODEL :**

```
In [1]: import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import tensorflow as tf
from sklearn.metrics import mean_squared_error, mean_absolute_error

import warnings
warnings.filterwarnings('ignore')

In [2]: AAPL = pd.read_csv("/content/drive/MyDrive/Database/Pre_Processed_AAPL.csv")
        TSLA = pd.read_csv("/content/drive/MyDrive/Database/Pre_Processed_TSLA.csv")
        GOOG = pd.read_csv("/content/drive/MyDrive/Database/Pre_Processed_GOOG.csv")
        MSFT = pd.read_csv("/content/drive/MyDrive/Database/Pre_Processed_MSFT.csv")
        AMZN = pd.read_csv("/content/drive/MyDrive/Database/Pre_Processed_AMZN.csv")
```

### **B. Data processing :**

## Data Preprocessing

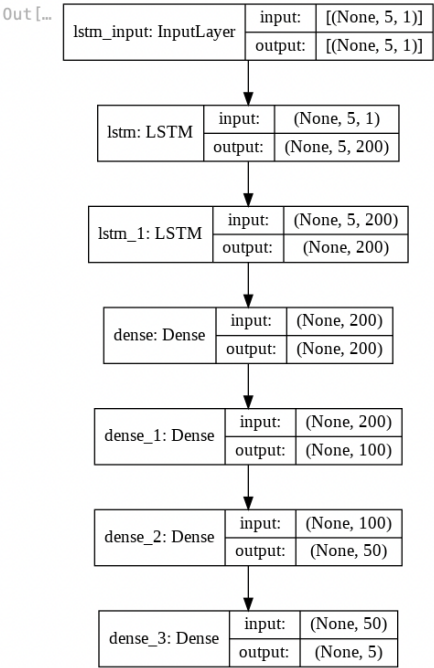
```
In [...]  
def Dataset(Data, Date):  
  
    Train_Data = Data['Adj. Close'][Data['Date'] < Date].to_numpy()  
    Data_Train = []  
    Data_Train_X = []  
    Data_Train_Y = []  
    for i in range(0, len(Train_Data), 5):  
        try:  
            Data_Train.append(Train_Data[i : i + 5])  
        except:  
            pass  
  
    if len(Data_Train[-1]) < 5:  
        Data_Train.pop(-1)  
  
    Data_Train_X = Data_Train[0 : -1]  
    Data_Train_X = np.array(Data_Train_X)  
    Data_Train_X = Data_Train_X.reshape((-1, 5, 1))  
    Data_Train_Y = Data_Train[1 : len(Data_Train)]  
    Data_Train_Y = np.array(Data_Train_Y)  
    Data_Train_Y = Data_Train_Y.reshape((-1, 5, 1))  
  
    Test_Data = Data['Adj. Close'][Data['Date'] >= Date].to_numpy()  
    Data_Test = []  
    Data_Test_X = []  
    Data_Test_Y = []  
    for i in range(0, len(Test_Data), 5):  
        try:  
            Data_Test.append(Test_Data[i : i + 5])  
        except:  
            pass  
  
    if len(Data_Test[-1]) < 5:  
        Data_Test.pop(-1)  
  
    Data_Test_X = Data_Test[0 : -1]  
    Data_Test_X = np.array(Data_Test_X)  
    Data_Test_X = Data_Test_X.reshape((-1, 5, 1))  
    Data_Test_Y = Data_Test[1 : len(Data_Test)]  
    Data_Test_Y = np.array(Data_Test_Y)  
    Data_Test_Y = Data_Test_Y.reshape((-1, 5, 1))  
  
    return Data_Train_X, Data_Train_Y, Data_Test_X, Data_Test_Y
```

## C. Model Building :

### Model

```
In [...]  
def Model():  
    model = tf.keras.models.Sequential([  
        tf.keras.layers.LSTM(200, input_shape = (5, 1), activation =  
            tf.keras.layers.LSTM(200, activation = tf.nn.leaky_relu),  
            tf.keras.layers.Dense(200, activation = tf.nn.leaky_relu),  
            tf.keras.layers.Dense(100, activation = tf.nn.leaky_relu),  
            tf.keras.layers.Dense(50, activation = tf.nn.leaky_relu),  
            tf.keras.layers.Dense(5, activation = tf.nn.leaky_relu)  
        ])  
  
    return model  
  
In [...]  
model = Model()
```

```
In [...]  
tf.keras.utils.plot_model(model, show_shapes=True)
```



```
In [...]  
model.summary()
```

Model: "sequential"

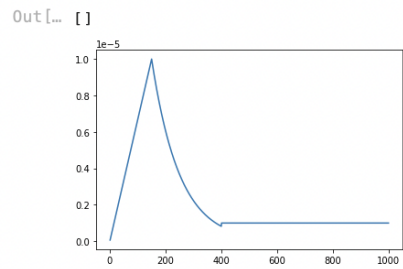
Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 5, 200)	161600
lstm_1 (LSTM)	(None, 200)	320800
dense (Dense)	(None, 200)	40200
dense_1 (Dense)	(None, 100)	20100
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 5)	255

=====  
Total params: 548,005  
Trainable params: 548,005  
Non-trainable params: 0  
=====

### Custom Learning Rate

```
In [...]  
def scheduler(epoch):  
    if epoch <= 150:  
        lrate = (10 ** -5) * (epoch / 150)  
    elif epoch <= 400:  
        initial_lrate = (10 ** -5)  
        k = 0.01  
        lrate = initial_lrate * math.exp(-k * (epoch - 150))  
    else:  
        lrate = (10 ** -6)  
  
    return lrate
```

```
In [...]  
epochs = [i for i in range(1, 1001, 1)]  
lrate = [scheduler(i) for i in range(1, 1001, 1)]  
plt.plot(epochs, lrate)
```



```
In [...]  
callback = tf.keras.callbacks.LearningRateScheduler(scheduler)
```

## D. Model fitting:

### 1) APPLE :

## Apple

```
In [... AAPL.head()
```

```
Out[...      Date  Open  High  Low  Close  Adj. Close  Volume
0  2015-01-02  27.85  27.86  26.84  27.33    24.86  212818400.0
1  2015-01-05  27.07  27.16  26.35  26.56    24.16  257142000.0
2  2015-01-06  26.64  26.86  26.16  26.57    24.16  263188400.0
3  2015-01-07  26.80  27.05  26.67  26.94    24.50  160423600.0
4  2015-01-08  27.31  28.04  27.17  27.97    25.44  237458000.0
```

```
In [... AAPL.info()
```

```
RangeIndex: 1549 entries, 0 to 1548
Data columns (total 7 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Date        1549 non-null   object
1   Open        1549 non-null   float64
2   High        1549 non-null   float64
3   Low         1549 non-null   float64
4   Close       1549 non-null   float64
5   Adj. Close  1549 non-null   float64
6   Volume      1549 non-null   float64
dtypes: float64(6), object(1)
memory usage: 84.8+ KB
```

```
In [... # Change Dtype of Date column
AAPL["Date"] = pd.to_datetime(AAPL["Date"])
```

### Split the Data into Training and Test set

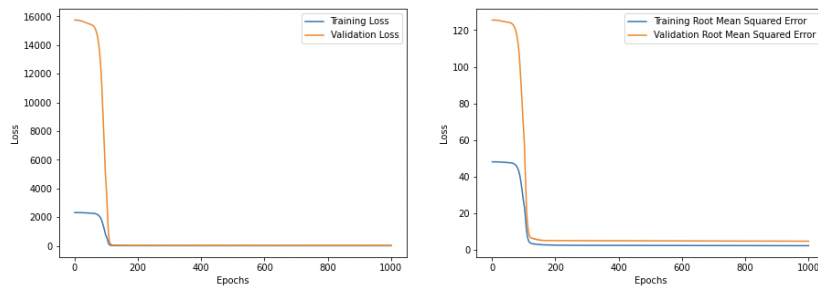
Training Period: 2015-01-02 - 2020-09-30

Testing Period: 2020-10-01 - 2021-02-26

```
In [... AAPL_Date = '2020-10-01'
AAPL_Train_X, AAPL_Train_Y, AAPL_Test_X, AAPL_Test_Y = Dataset(AAPL, AAPL_Date)
```

## Model Fitting

```
In [...]\nAAPL_Model = Model()\n\nIn [...]\nAAPL_Model.compile(optimizer = tf.keras.optimizers.Adam(), loss = 'mse', metrics = tf.keras.metrics\n\nIn [...]\nAAPL_hist = AAPL_Model.fit(AAPL_Train_X, AAPL_Train_Y, epochs = 1000, validation_data = (AAPL_Test_\n\nIn [...]\nhistory_dict = AAPL_hist.history\n\nloss = history_dict["loss"]\nroot_mean_squared_error = history_dict["root_mean_squared_error"]\nval_loss = history_dict["val_loss"]\nval_root_mean_squared_error = history_dict["val_root_mean_squared_error"]\n\nepochs = range(1, len(loss) + 1)\n\nIn [...]\nfig, (ax1, ax2) = plt.subplots(1, 2)\n\nfig.set_figheight(5)\nfig.set_figwidth(15)\n\nax1.plot(epochs, loss, label = 'Training Loss')\nax1.plot(epochs, val_loss, label = 'Validation Loss')\nax1.set(xlabel = "Epochs", ylabel = "Loss")\nax1.legend()\n\nax2.plot(epochs, root_mean_squared_error, label = "Training Root Mean Squared Error")\nax2.plot(epochs, val_root_mean_squared_error, label = "Validation Root Mean Squared Error")\nax2.set(xlabel = "Epochs", ylabel = "Loss")\nax2.legend()\n\nplt.show()
```



## 2) MICROSOFT :



```
In [47]: MSFT.head()
```

```
Out[47]:
```

	Date	Open	High	Low	Close	Adj. Close	Volume
0	2015-01-02	46.66	47.42	46.54	46.76	41.44	27913900.0
1	2015-01-05	46.37	46.73	46.25	46.33	41.06	39673900.0
2	2015-01-06	46.38	46.75	45.54	45.65	40.46	36447900.0
3	2015-01-07	45.98	46.46	45.49	46.23	40.97	29114100.0
4	2015-01-08	46.75	47.75	46.72	47.59	42.18	29645200.0

```
In [48]: MSFT.info()
```

```
RangeIndex: 1549 entries, 0 to 1548  
Data columns (total 7 columns):  
#   Column      Non-Null Count  Dtype  
---  ---  
0   Date         1549 non-null   object  
1   Open         1549 non-null   float64  
2   High         1549 non-null   float64  
3   Low          1549 non-null   float64  
4   Close        1549 non-null   float64  
5   Adj. Close   1549 non-null   float64  
6   Volume       1549 non-null   float64  
dtypes: float64(6), object(1)  
memory usage: 84.8+ KB
```

```
In [49]: # Change Dtype of Date column  
MSFT["Date"] = pd.to_datetime(MSFT["Date"])
```

## Split the Data into Training and Test set

Training Period: 2015-01-02 - 2020-09-30

Testing Period: 2020-10-01 - 2021-02-26

```
In [50]: MSFT_Date = '2020-10-01'  
MSFT_Train_X, MSFT_Train_Y, MSFT_Test_X, MSFT_Test_Y = Dataset(MSFT, MSFT_Date)
```

## Model Fitting

```
MSFT_Model = Model()
```

```
MSFT_Model.compile(optimizer = tf.keras.optimizers.Adam(), loss = 'mse', metrics = tf.keras.metrics.RootMeanSquaredError())
```

```
MSFT_hist = MSFT_Model.fit(MSFT_Train_X, MSFT_Train_Y, epochs = 1000, validation_data = (MSFT_Test_X, MSFT_Test_Y), callbacks=[callback])
```

```
history_dict = MSFT_hist.history
```

```
loss = history_dict["loss"]  
root_mean_squared_error = history_dict["root_mean_squared_error"]  
val_loss = history_dict["val_loss"]  
val_root_mean_squared_error = history_dict["val_root_mean_squared_error"]  
  
epochs = range(1, len(loss) + 1)
```

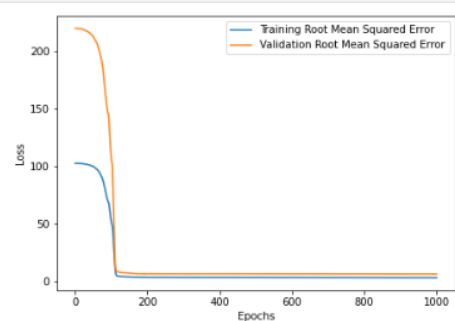
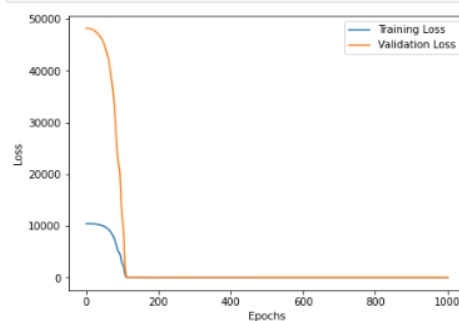
```
fig, (ax1, ax2) = plt.subplots(1, 2)
```

```
fig.set_figheight(5)  
fig.set_figwidth(15)
```

```
ax1.plot(epochs, loss, label = "Training Loss")  
ax1.plot(epochs, val_loss, label = "Validation Loss")  
ax1.set(xlabel = "Epochs", ylabel = "Loss")  
ax1.legend()
```

```
ax2.plot(epochs, root_mean_squared_error, label = "Training Root Mean Squared Error")  
ax2.plot(epochs, val_root_mean_squared_error, label = "Validation Root Mean Squared Error")  
ax2.set(xlabel = "Epochs", ylabel = "Loss")  
ax2.legend()
```

```
plt.show()
```



### 3) AMAZON :

#### Amazon

In [59]: `AMZN.head()`

Out[59]:

	Date	Open	High	Low	Close	Adj. Close	Volume
0	2015-01-02	312.58	314.75	306.96	308.52	308.52	2783200.0
1	2015-01-05	307.01	308.38	300.85	302.19	302.19	2774200.0
2	2015-01-06	302.24	303.00	292.38	295.29	295.29	3519000.0
3	2015-01-07	297.50	301.28	295.33	298.42	298.42	2640300.0
4	2015-01-08	300.32	303.14	296.11	300.46	300.46	3088400.0

In [60]: `AMZN.info()`

```
RangeIndex: 1549 entries, 0 to 1548
Data columns (total 7 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Date         1549 non-null   object
1   Open         1549 non-null   float64
2   High         1549 non-null   float64
3   Low          1549 non-null   float64
4   Close        1549 non-null   float64
5   Adj. Close   1549 non-null   float64
6   Volume       1549 non-null   float64
dtypes: float64(6), object(1)
memory usage: 84.8+ KB
```

In [61]: `# Change Dtype of Date column`  
`AMZN["Date"] = pd.to_datetime(AMZN["Date"])`

#### Split the Data into Training and Test set

Training Period: 2015-01-02 - 2020-10-30

Testing Period: 2020-11-02 - 2021-02-26

In [62]: `AMZN_Date = '2020-11-01'`  
`AMZN_Train_X, AMZN_Train_Y, AMZN_Test_X, AMZN_Test_Y = Dataset(AMZN, AMZN_Date)`

## Model Fitting

```
In [63]: AMZN_Model = Model()

In [64]: AMZN_Model.compile(optimizer = tf.keras.optimizers.Adam(), loss = 'mse', metrics = tf.keras.metrics.RootMeanSquaredError())

In [ ]: AMZN_hist = AMZN_Model.fit(AMZN_Train_X, AMZN_Train_Y, epochs = 200, validation_data = (AMZN_Test_X, AMZN_Test_Y), callbacks=[callback])

In [66]: history_dict = AMZN_hist.history

loss = history_dict["loss"]
root_mean_squared_error = history_dict["root_mean_squared_error"]
val_loss = history_dict["val_loss"]
val_root_mean_squared_error = history_dict["val_root_mean_squared_error"]

epochs = range(1, len(loss) + 1)

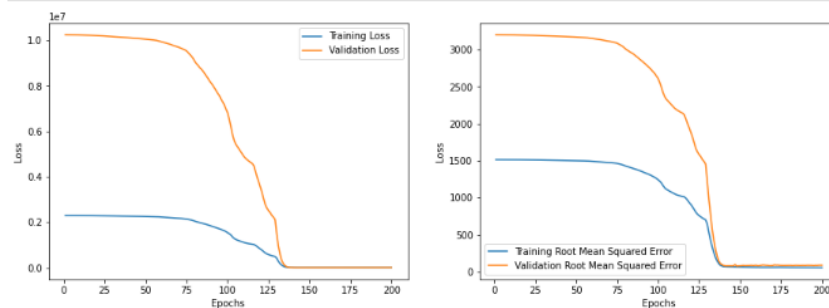
In [67]: fig, (ax1, ax2) = plt.subplots(1, 2)

fig.set_figheight(5)
fig.set_figwidth(15)

ax1.plot(epochs, loss, label = "Training Loss")
ax1.plot(epochs, val_loss, label = "Validation Loss")
ax1.set(xlabel = "Epochs", ylabel = "Loss")
ax1.legend()

ax2.plot(epochs, root_mean_squared_error, label = "Training Root Mean Squared Error")
ax2.plot(epochs, val_root_mean_squared_error, label = "Validation Root Mean Squared Error")
ax2.set(xlabel = "Epochs", ylabel = "Loss")
ax2.legend()

plt.show()
```



## E. Model Prediction :

We predict the stock prices using our model which has been trained for the past 5 years of data, of the test period time and compare it with our data. The results is as follows :

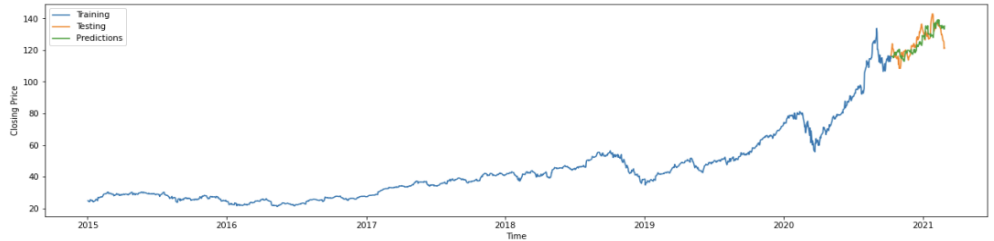
### 1) APPLE :

### Predicting the closing stock price of Apple

```
In [...]: AAPL_prediction = AAPL_Model.predict(AAPL_Test_X)
```

```
In [...]: plt.figure(figsize=(20, 5))
plt.plot(AAPL['Date'][AAPL['Date'] < '2020-10-12'], AAPL['Adj. Close'][AAPL['Date'] < '2020-10-12'], label='Training')
plt.plot(AAPL['Date'][AAPL['Date'] >= '2020-10-09'], AAPL['Adj. Close'][AAPL['Date'] >= '2020-10-09'], label='Testing')
plt.plot(AAPL['Date'][AAPL['Date'] >= '2020-10-12'], AAPL_prediction.reshape(-1), label='Predictions')
plt.xlabel('Time')
plt.ylabel('Closing Price')
plt.legend(loc='best')
```

Out [...]:



```
In [...]: rmse = math.sqrt(mean_squared_error(AAPL_Test_Y.reshape(-1, 5), AAPL_prediction))
mape = np.mean(np.abs(AAPL_prediction - AAPL_Test_Y.reshape(-1, 5))/np.abs(AAPL_Test_Y.reshape(-1, 5)))
print(f'RMSE: {rmse}')
print(f'MAPE: {mape}')
```

RMSE: 4.822911467470171  
MAPE: 0.030390689646776783

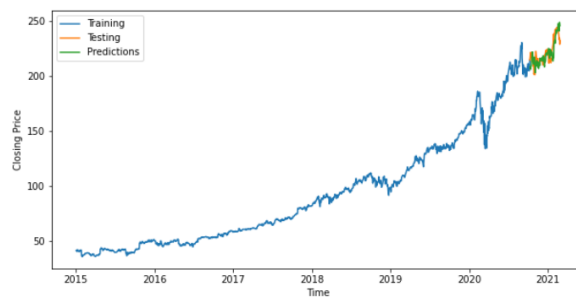
## 2) MICROSOFT :

### Predicting the closing stock price of Microsoft

```
In [56]: MSFT_prediction = MSFT_Model.predict(MSFT_Test_X)
```

```
In [57]: plt.figure(figsize=(10, 5))
plt.plot(MSFT['Date'][MSFT['Date'] < '2020-10-12'], MSFT['Adj. Close'][MSFT['Date'] < '2020-10-12'], label='Training')
plt.plot(MSFT['Date'][MSFT['Date'] >= '2020-10-09'], MSFT['Adj. Close'][MSFT['Date'] >= '2020-10-09'], label='Testing')
plt.plot(MSFT['Date'][MSFT['Date'] >= '2020-10-12'], MSFT_prediction.reshape(-1), label='Predictions')
plt.xlabel('Time')
plt.ylabel('Closing Price')
plt.legend(loc='best')
```

Out[57]:



```
In [58]: rmse = math.sqrt(mean_squared_error(MSFT_Test_Y.reshape(-1, 5), MSFT_prediction))
mape = np.mean(np.abs(MSFT_prediction - MSFT_Test_Y.reshape(-1, 5))/np.abs(MSFT_Test_Y.reshape(-1, 5)))
print(f'RMSE: {rmse}')
print(f'MAPE: {mape}')
```

RMSE: 6.2306768444560205  
MAPE: 0.02214984775145576

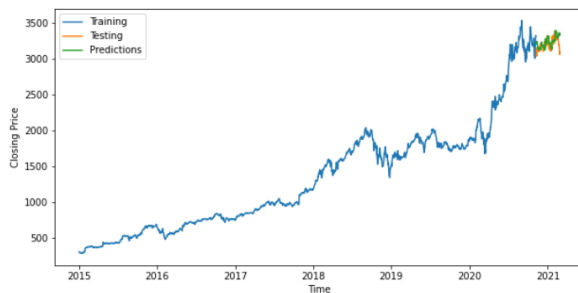
## 3) AMAZON :

### Predicting the closing stock price of Amazon

```
In [ ]: AMZN_prediction = AMZN_Model.predict(AMZN_Test_X)
```

```
In [69]: plt.figure(figsize=(10, 5))
plt.plot(AMZN['Date'][AMZN['Date'] < '2020-11-07'], AMZN['Adj. Close'][AMZN['Date'] < '2020-11-07'], label = 'Training')
plt.plot(AMZN['Date'][AMZN['Date'] >= '2020-11-07'], AMZN['Adj. Close'][AMZN['Date'] >= '2020-11-07'], label = 'Testing')
plt.plot(AMZN['Date'][AMZN['Date'] >= '2020-11-07'], AMZN_prediction.reshape(-1), label = 'Predictions')
plt.xlabel('Time')
plt.ylabel('Closing Price')
plt.legend(loc = 'best')
```

Out[69]:



```
In [70]: rmse = math.sqrt(mean_squared_error(AMZN_Test_Y.reshape(-1, 5), AMZN_prediction))
mape = np.mean(np.abs(AMZN_prediction - AMZN_Test_Y.reshape(-1, 5))/np.abs(AMZN_Test_Y.reshape(-1, 5)))
print(f'RMSE: {rmse}')
print(f'MAPE: {mape}')
```

RMSE: 87.48506342352238  
MAPE: 0.021571786340561845

## F. Result Visualisation :

The above stock price prediction graphs for various firms show that prediction lags behind real values because the model is unable to respond quickly to non-linear changes. However, the model responds well to gradual adjustments. As a result, we draw the conclusion that while the model performs worse than actual prices in prediction segments that have spikes, it performs better in segments that feature gradual variations.

## IV. References :

- 1) [https://www.researchgate.net/publication/47355842\\_Application\\_of\\_Artificial\\_Neural\\_Net\\_work\\_for\\_stock\\_market\\_predictions\\_A\\_review\\_of\\_literature](https://www.researchgate.net/publication/47355842_Application_of_Artificial_Neural_Net_work_for_stock_market_predictions_A_review_of_literature)
- 2) <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

- 3) <https://towardsdatascience.com/time-series-forecasting-with-deep-learning-and-attention-mechanism-2d001fc871fc#:~:text=Recurrent%20Neural%20Networks%20are%20the,when%20applied%20to%20long%20sequences.>
- 4) <https://www.youtube.com/watch?v=b61DPVFX03I>
- 5) [https://www.youtube.com/watch?v=lGaXq\\_Kgr2Y](https://www.youtube.com/watch?v=lGaXq_Kgr2Y)
- 6) <https://towardsdatascience.com/multimodal-deep-learning-ce7d1d994f4>
- 7) [https://github.com/amanjain252002/Stock-Price-Prediction/blob/main/Deep\\_Learning\\_Model.ipynb](https://github.com/amanjain252002/Stock-Price-Prediction/blob/main/Deep_Learning_Model.ipynb)
- 8) [https://github.com/pankush9096/Stock-Prediction-using-LSTM/blob/master/Stock\\_prediction\\_using\\_LSTM.ipynb](https://github.com/pankush9096/Stock-Prediction-using-LSTM/blob/master/Stock_prediction_using_LSTM.ipynb)
- 9) <https://github.com/xingyousong/Deep-Learning-Financial-News-Stock-Movement-Prediction>