**Importing the Libraries :**

**#Importing the Libraries**

**import pandas as PD**

**import NumPy as np**

**%matplotlib inline**

**import matplotlib. pyplot as plt**

**import matplotlib**

**from sklearn. Preprocessing import MinMaxScaler**

**from Keras. layers import LSTM, Dense, Dropout**

**from sklearn.model\_selection import TimeSeriesSplit**

**from sklearn.metrics import mean\_squared\_error, r2\_score**

**import matplotlib. dates as mandates**

**from sklearn. Preprocessing import MinMaxScaler**

**from sklearn import linear\_model**

**from Keras. Models import Sequential**

**from Keras. Layers import Dense**

**import Keras. Backend as K**

**from Keras. Callbacks import EarlyStopping**

**from Keras. Optimisers import Adam**

**from Keras. Models import load\_model**

**from Keras. Layers import LSTM**

**from Keras. utils.vis\_utils import plot\_model**

**Getting to Visualising the Stock Market Prediction Data:**

**#Get the Dataset**

**df=pd.read\_csv(“MicrosoftStockData.csv”,na\_values=[‘null’],index\_col=’Date’,parse\_dates=True,infer\_datetime\_format=True)**

**Checking for Null Values by Printing the DataFrame Shape:**

**#Print the shape of Dataframe  and Check for Null Values**

**print(“Dataframe Shape: “, df. shape)**

**print(“Null Value Present: “, df.IsNull().values.any())**

**Output:**

**>> Dataframe Shape: (7334, 6)**

**>>Null Value Present: False**

**Plotting the True Adjusted Close Value:**

**#Plot the True Adj Close Value**

**df[‘Adj Close’].plot()**

**Setting the Target Variable and Selecting the Features:**

**#Set Target Variable**

**output\_var = PD.DataFrame(df[‘Adj Close’])**

**#Selecting the Features**

**features = [‘Open’, ‘High’, ‘Low’, ‘Volume’]**

**Scaling:**

**#Scaling**

**scaler = MinMaxScaler()**

**feature\_transform = scaler.fit\_transform(df[features])**

**feature\_transform= pd.DataFrame(columns=features, data=feature\_transform, index=df.index)**

**feature\_transform.head()**

**Creating a Training Set and a Test Set for Stock Market Prediction:**

**timesplit= TimeSeriesSplit(n\_splits=10)**

**#Splitting to Training set and Test set**

**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\_var[:len(train\_index)].values.ravel(), output\_var[len(train\_index): (len(train\_index)+len(test\_index))].values.ravel()**

**Data Processing For LSTM:**

**#Process the data for LSTM**

**trainX =np.array(X\_train)**

**testX =np.array(X\_test)**

**X\_train = trainX.reshape(X\_train.shape[0], 1, X\_train.shape[1])**

**X\_test = testX.reshape(X\_test.shape[0], 1, X\_test.shape[1])**

**Building the LSTM Model for Stock Market Prediction:**

**#Building the LSTM Model**

**lstm = Sequential()**

**lstm.add(LSTM(32, input\_shape=(1, trainX.shape[1]), activation=’relu’, return\_sequences=False))**

**lstm.add(Dense(1))**

**lstm.compile(loss=’mean\_squared\_error’, optimizer=’adam’)**

**plot\_model(lstm, show\_shapes=True, show\_layer\_names=True)**

**Training the Stock Market Prediction Model:**

**#Model Training**

**history=lstm.fit(X\_train, y\_train, epochs=100, batch\_size=8, verbose=1, shuffle=False)**

**Eросh 1/100**

**834/834 [==============================] – 3s 2ms/steр – lоss: 67.1211**

**Eросh 2/100**

**834/834 [==============================] – 1s 2ms/steр – lоss: 70.4911**

**Eросh 3/100**

**834/834 [==============================] – 1s 2ms/steр – lоss: 48.8155**

**Eросh 4/100**

**834/834 [==============================] – 1s 2ms/steр – lоss: 21.5447**

**Eросh 5/100**

**834/834 [==============================] – 1s 2ms/steр – lоss: 6.1709**

**Eросh 6/100**

**834/834 [==============================] – 1s 2ms/steр – lоss: 1.8726**

**Eросh 7/100**

**834/834 [==============================] – 1s 2ms/steр – lоss: 0.9380**

**Eросh 8/100**

**834/834 [==============================] – 2s 2ms/steр – lоss: 0.6566**

**Eросh 9/100**

**834/834 [==============================] – 1s 2ms/steр – lоss: 0.5369**

**Eросh 10/100**

**834/834 [==============================] – 2s 2ms/steр – lоss:**

**0.4761**

**.**

**.**

**.**

**.**

**Eросh 95/100**

**834/834 [==============================] – 1s 2ms/steр – lоss: 0.4542**

**Eросh 96/100**

**834/834 [==============================] – 2s 2ms/steр – lоss: 0.4553**

**Eросh 97/100**

**834/834 [==============================] – 1s 2ms/steр – lоss: 0.4565**

**Eросh 98/100**

**834/834 [==============================] – 1s 2ms/steр – lоss: 0.4576**

**Eросh 99/100**

**834/834 [==============================] – 1s 2ms/steр – lоss: 0.4588**

**Eросh 100/100**

**834/834 [==============================] – 1s 2ms/steр – lоss: 0.4599**

**Making the LSTM Prediction:**

**#LSTM Prediction y\_pred= lstm.predict(X\_test)**

**Comparing Predicted vs True Adjusted Close Value – LSTM:**

**#Predicted vs True Adj Close Value – LSTM**

**plt.plot(y\_test, label=’True Value’)**

**plt.plot(y\_pred, label=’LSTM Value’)**

**plt.title(“Prediction by LSTM”)**

**plt.xlabel(‘Time Scale’)**

**plt.ylabel(‘Scaled USD’)**

**plt.legend()**

**plt.show()**

**Time Series Models :**

**# Make sure that you have all these libaries available to run the code successfully**

**from pandas\_datareader import data**

**import matplotlib.pyplot as plt**

**import pandas as pd**

**import datetime as dt**

**import urllib.request, json**

**import os**

**import numpy as np**

**import tensorflow as tf # This code has been tested with TensorFlow 1.6**

**from sklearn.preprocessing import MinMaxScaler**

**Getting Data from Kaggle :**

**data\_source = 'kaggle' # alphavantage or Kaggle**

**if data\_source == 'alphavantage’:**

**# ====================== Loading Data from Alpha Vantage ==================================**

**api\_key = '<your API key>' # American Airlines stock market prices ticker = "AAL“**

**# JSON file with all the stock market data for AAL from the last 20 years**

**url\_string ="https://www.alphavantage.co/query?function=TIME\_SERIES\_DAILY&symbol=%s&outputsize=full&apikey=%s"%(ticker,api\_key)**

**# Save data to this file**

**file\_to\_save = 'stock\_market\_data-%s.csv'%ticker**

**if not os.path.exists(file\_to\_save):**

**with urllib.request.urlopen(url\_string) as url:**

**data = json.loads(url.read().decode())**

**# extract stock market data**

**data = data['Time Series (Daily)’]**

**df = pd.DataFrame(columns=['Date','Low','High','Close','Open']) for k,v in data.items():**

**date = dt.datetime.strptime(k, '%Y-%m-%d’)**

**data\_row = [date.date(),float(v['3. low']),float(v['2. high']), float(v['4. close']),float(v['1. open'])]**

**df.loc[-1,:] = data\_row**

**df.index = df.index + 1**

**print('Data saved to : %s'%file\_to\_save) df.to\_csv(file\_to\_save)**

**else: print('File already exists. Loading data from CSV') df = pd.read\_csv(file\_to\_save)**

**else:**

**# ====================== Loading Data from Kaggle ==================================**

**# You will be using HP's data. Feel free to experiment with other data.**

**# But while doing so, be careful to have a large enough dataset and also pay attention to the data normalization**

**df = pd.read\_csv(os.path.join('Stocks','hpq.us.txt'),delimiter=',',usecols=['Date','Open','High','Low','Close’])**

**print('Loaded data from the Kaggle repository')**

**Data Exploration :**

**# Sort DataFrame by date**

**df = df.sort\_values('Date')**

**# Double check the result**

**df.head()**

**Data Visualization :**

**plt.figure(figsize = (18,9))**

**plt.plot(range(df.shape[0]),(df['Low']+df['High'])/2.0)**

**plt.xticks(range(0,df.shape[0],500),df['Date'].loc[::500],rotation=45)**

**plt.xlabel('Date',fontsize=18)**

**plt.ylabel('Mid Price',fontsize=18)**

**plt.show()**

**Splitting Data into a Training set and a Test set :**

**# First calculate the mid prices from the highest and lowest**

**high\_prices = df.loc[:,'High'].as\_matrix()**

**low\_prices = df.loc[:,'Low'].as\_matrix()**

**mid\_prices = (high\_prices+low\_prices)/2.0**

**train\_data = mid\_prices[:11000]**

**test\_data = mid\_prices[11000:]**