Name: Gore Aniket Machhindra

Email: aniket.m.gore.1901@gmail.com

Role: Data Science Intern

Task 2: Stock Market Prediction And Forecasting Using Stacked LSTM

Contents-

Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler

import warnings
warnings.filterwarnings("ignore")
```

Reading the Dataset

```
1 data = pd.read_csv("https://raw.githubusercontent.com/mwitiderrick/stockprice/master/NSE-T
```

```
1 data.head()
```

	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

Data Sorting

```
1 data['Date']=pd.to_datetime(data['Date'])
2 print(type(data.Date[0]))
```

```
<class 'pandas._libs.tslibs.timestamps.Timestamp'>
```

```
1 df=data.sort_values(by='Date')
2 df.head()
```

	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
2034	2010-07-21	122.1	123.00	121.05	121.10	121.55	658666	803.56
2033	2010-07-22	120.3	122.00	120.25	120.75	120.90	293312	355.17
2032	2010-07-23	121.8	121.95	120.25	120.35	120.65	281312	340.31

```
1 df.reset_index(inplace=True)
```

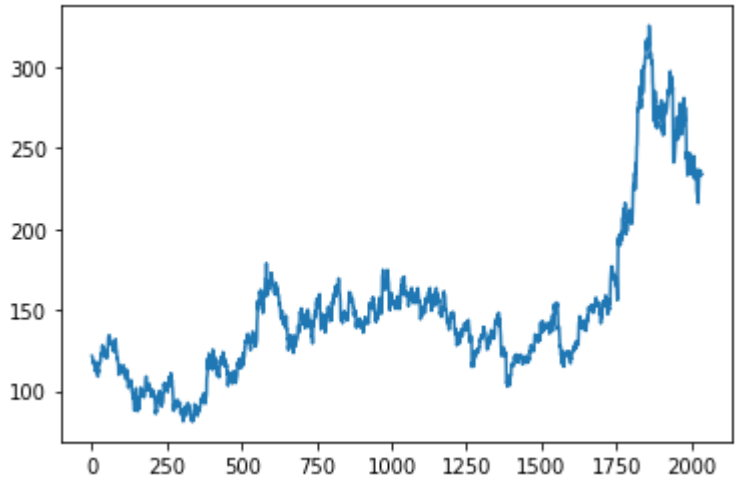
```
1 df.head()
```

	index	Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
0	2034	2010-07-21	122.1	123.00	121.05	121.10	121.55	658666	803.56
1	2033	2010-07-22	120.3	122.00	120.25	120.75	120.90	293312	355.17
2	2032	2010-07-23	121.8	121.95	120.25	120.35	120.65	281312	340.31

Visualization

```
1 plt.plot(df['Close'])
```

```
[<matplotlib.lines.Line2D at 0x7fed7500cad0>]
```




```
1 dff=df['Close']
2 dff
```

```
0       121.55
1       120.90
2       120.65
3       117.60
4       118.65
         ...
2030    233.30
2031    236.10
2032    234.25
2033    233.25
2034    233.75
Name: Close, Length: 2035, dtype: float64
```

Min Max Scaler

```
scaler=MinMaxScaler(feature_range=(0,1))
dff=scaler.fit_transform(np.array(dff).reshape(-1,1))
dff
```

```
array([[0.16584967],
       [0.16319444],
       [0.1621732 ],
       ...,
       [0.62622549],
       [0.62214052],
       [0.62418301]])
```

Spliting the Dataset

```
training_size=int(len(dff)*0.70)
test_size=len(dff)-training_size
train_data,test_data=dff[0:training_size,:],dff[training_size:len(dff),:1]
```

Convert an array of values into a 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 np.array(dataX), np.array(dataY)
```

Spliting Data into Train and Test

```
time_step = 100
X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)
```

```
print(X_train.shape), print(y_train.shape)
```

```
(1323, 100)
(1323,)
(None, None)
```

```
print(X_test.shape), print(ytest.shape)
```

```
print(X_test.shape), print(ytest.shape)
```

```
(510, 100)
(510,)
(None, None)
```

```
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)
```

Creating the LSTM Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

```
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')
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 50)           10400

 lstm_1 (LSTM)               (None, 100, 50)           20200

 lstm_2 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(X_train,y_train,validation_split=0.1,epochs=70,batch_size=64,verbose=1)
```

```
Epoch 41/70
19/19 [==============================] - 4s 187ms/step - loss: 2.9497e-04 - val_loss
Epoch 42/70
19/19 [==============================] - 4s 187ms/step - loss: 2.6368e-04 - val_loss
Epoch 43/70
19/19 [==============================] - 4s 190ms/step - loss: 2.7197e-04 - val_loss
Epoch 44/70
```

```
Epoch 44/70
19/19 [==============================] - 4s 188ms/step - loss: 2.4793e-04 - val_loss
Epoch 45/70
19/19 [==============================] - 4s 191ms/step - loss: 2.4348e-04 - val_loss
Epoch 46/70
19/19 [==============================] - 4s 191ms/step - loss: 2.6380e-04 - val_loss
Epoch 47/70
19/19 [==============================] - 4s 188ms/step - loss: 2.6637e-04 - val_loss
Epoch 48/70
19/19 [==============================] - 4s 187ms/step - loss: 2.6047e-04 - val_loss
Epoch 49/70
19/19 [==============================] - 4s 188ms/step - loss: 2.7958e-04 - val_loss
Epoch 50/70
19/19 [==============================] - 4s 188ms/step - loss: 2.4133e-04 - val_loss
Epoch 51/70
19/19 [==============================] - 4s 189ms/step - loss: 2.3590e-04 - val_loss
Epoch 52/70
19/19 [==============================] - 4s 188ms/step - loss: 2.1302e-04 - val_loss
Epoch 53/70
19/19 [==============================] - 4s 192ms/step - loss: 2.2519e-04 - val_loss
Epoch 54/70
19/19 [==============================] - 4s 191ms/step - loss: 2.1626e-04 - val_loss
Epoch 55/70
19/19 [==============================] - 4s 190ms/step - loss: 2.1162e-04 - val_loss
Epoch 56/70
19/19 [==============================] - 4s 185ms/step - loss: 2.2579e-04 - val_loss
Epoch 57/70
19/19 [==============================] - 4s 187ms/step - loss: 2.0361e-04 - val_loss
Epoch 58/70
19/19 [==============================] - 4s 188ms/step - loss: 1.8802e-04 - val_loss
Epoch 59/70
19/19 [==============================] - 4s 186ms/step - loss: 1.8810e-04 - val_loss
Epoch 60/70
19/19 [==============================] - 4s 187ms/step - loss: 1.8268e-04 - val_loss
Epoch 61/70
19/19 [==============================] - 4s 186ms/step - loss: 2.3597e-04 - val_loss
Epoch 62/70
19/19 [==============================] - 4s 189ms/step - loss: 2.1695e-04 - val_loss
Epoch 63/70
19/19 [==============================] - 4s 188ms/step - loss: 1.7980e-04 - val_loss
Epoch 64/70
19/19 [==============================] - 4s 189ms/step - loss: 1.8647e-04 - val_loss
Epoch 65/70
19/19 [==============================] - 4s 187ms/step - loss: 1.7957e-04 - val_loss
Epoch 66/70
19/19 [==============================] - 4s 186ms/step - loss: 1.8091e-04 - val_loss
Epoch 67/70
19/19 [==============================] - 4s 189ms/step - loss: 1.6304e-04 - val_loss
Epoch 68/70
19/19 [==============================] - 4s 187ms/step - loss: 1.6550e-04 - val_loss
Epoch 69/70
```

Prediction and Checking Performance

```
test_predict=model.predict(X_test)
```

```
test_predicted=scaler.inverse_transform(test_predict)
test_predicted
```

```
array([[140.47122 ],
       [140.55112 ],
       [139.08086 ],
       [135.50777 ],
       [134.66795 ],
       [135.04314 ],
       [137.17647 ],
       [138.7034  ],
       [138.01086 ],
       [136.98502 ],
       [136.9208  ],
       [139.13553 ],
       [139.78525 ],
       [141.66862 ],
       [144.1348  ],
       [139.60512 ],
       [137.0315  ],
       [138.36626 ],
       [139.85718 ],
       [147.14671 ],
       [150.63072 ],
       [150.65312 ],
       [149.76509 ],
       [146.79482 ],
       [148.32158 ],
       [148.54263 ],
       [149.23941 ],
       [151.42966 ],
       [151.27753 ],
       [150.58588 ],
       [151.19287 ],
       [149.23354 ],
       [144.93518 ],
       [138.20634 ],
       [136.55518 ],
       [137.23349 ],
       [137.74792 ],
       [135.81522 ],
       [132.2413  ],
       [127.79882 ],
       [126.24735 ],
       [126.1875  ],
       [123.71038 ],
       [123.4802  ],
       [120.83661 ],
       [117.83272 ],
       [118.286606],
       [118.48905 ],
```

```
[119.03652 ],
[117.39505 ],
[116.40066 ],
[116.326935],
[115.45687 ],
[116.116135],
[118.1997  ],
[120.27875 ],
[122.16918 ],
[122.17144 ],
```

Calculating Performance

```
1 import math
2 from sklearn.metrics import mean_squared_error
```

```
1 performance = math.sqrt(mean_squared_error(ytest,test_predict))
2 performance
```

```
0.04474436443675083
```