Semester 7 project

Group A-02

Stock Market Prediction And Forecasting Using Stacked LSTM

```
import pandas as pd
```

```
df = pd.read_csv("./Tesla.csv - Tesla.csv.csv")
```

```
df.head()
```

	Date	Open	High	Low	Close	Volume	Adj Close
0	6/29/2010	19.000000	25.00	17.540001	23.889999	18766300	23.889999
1	6/30/2010	25.790001	30.42	23.299999	23.830000	17187100	23.830000
2	7/1/2010	25.000000	25.92	20.270000	21.959999	8218800	21.959999
3	7/2/2010	23.000000	23.10	18.709999	19.200001	5139800	19.200001
4	7/6/2010	20.000000	20.00	15.830000	16.110001	6866900	16.110001

```
df.tail()
```

	Date	Open	High	Low	Close	Volume	Adj Close
1687	3/13/2017	244.820007	246.850006	242.779999	246.169998	3010700	246.169998
1688	3/14/2017	246.110001	258.119995	246.020004	258.000000	7575500	258.000000
1689	3/15/2017	257.000000	261.000000	254.270004	255.729996	4816600	255.729996
1690	3/16/2017	262.399994	265.750000	259.059998	262.049988	7100400	262.049988
1691	3/17/2017	264.000000	265.329987	261.200012	261.500000	6475900	261.500000

Checking close values & plotting graph

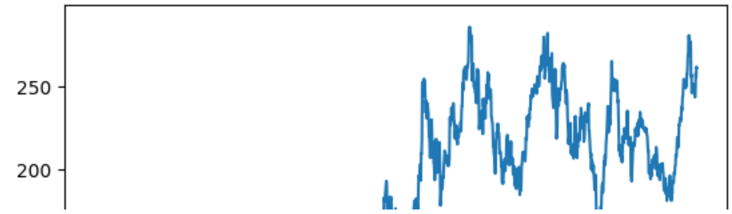
```
df1 = df.reset_index()['Close']
```

```
df1
```

```
0        23.889999
1        23.830000
2        21.959999
3        19.200001
4        16.110001
           ...
1687    246.169998
1688    258.000000
1689    255.729996
1690    262.049988
1691    261.500000
Name: Close, Length: 1692, dtype: float64
```

```
import matplotlib.pyplot as plt
plt.plot(df1)
```

```
[<matplotlib.lines.Line2D at 0x78210727e410>]
```




```
import numpy as np
```

```
df1
```

```
0        23.889999
1        23.830000
2        21.959999
3        19.200001
4        16.110001
           ...
1687    246.169998
1688    258.000000
1689    255.729996
1690    262.049988
1691    261.500000
Name: Close, Length: 1692, dtype: float64
```

Scaling between 0-1 ~ MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))
```

```
print(df1)
```

```
[[0.02993635]
 [0.02971433]
 [0.02279455]
 ...
 [0.88784039]
 [0.91122698]
 [0.9091918 ]]
```

Splitting dataset into train and test

```
training_size = int(len(df1)*0.70)
test_size = len(df1)-training_size

train_data,test_data = df1[0:training_size,:], df1[training_size:len(df1),:1]
```

```
training_size,test_size
```

```
(1184, 508)
```

```
train_data
```

```
array([[0.02993635],
       [0.02971433],
       [0.02279455],
       ...,
       [0.64579633],
       [0.65845174],
       [0.64857164]])
```

```
import numpy

# 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]   ###i=0, 0,1,2,3-----99   100
        dataX.append(a)
```

```
        dataY.append(dataset[i + time_step, 0])

    return numpy.array(dataX), numpy.array(dataY)
```

```
# reshape into X=t,t+1,t+2,t+3 and Y=t+4
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)
```

```
(1083, 100)
(1083,)
(None, None)
```

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

```
(407, 100)
(407,)
(None, None)
```

Reshape input to be [samples, time steps, features] ~ 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)
```

Creating the Stacked 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: 50851 (198.64 KB)
Trainable params: 50851 (198.64 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
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: 50851 (198.64 KB)
```

```
Trainable params: 50851 (198.64 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.fit(X_train, y_train, validation_data=(X_test,ytest), epochs=100, batch_size=64, verbose=1)
```

```
Epoch 1/100
17/17 [==============================] - 15s 88ms/step - loss: 0.0308 - val_loss: 0.0132
Epoch 2/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0041 - val_loss: 0.0054
Epoch 3/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0023 - val_loss: 0.0056
Epoch 4/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0020 - val_loss: 0.0041
Epoch 5/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0019 - val_loss: 0.0040
Epoch 6/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0018 - val_loss: 0.0039
Epoch 7/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0018 - val_loss: 0.0040
Epoch 8/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0017 - val_loss: 0.0036
Epoch 9/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0019 - val_loss: 0.0045
Epoch 10/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0018 - val_loss: 0.0034
Epoch 11/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0017 - val_loss: 0.0033
Epoch 12/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0016 - val_loss: 0.0034
Epoch 13/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0015 - val_loss: 0.0040
Epoch 14/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0015 - val_loss: 0.0040
Epoch 15/100
17/17 [==============================] - 0s 19ms/step - loss: 0.0015 - val_loss: 0.0028
Epoch 16/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0013 - val_loss: 0.0025
Epoch 17/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0013 - val_loss: 0.0025
Epoch 18/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0014 - val_loss: 0.0026
Epoch 19/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0013 - val_loss: 0.0025
Epoch 20/100
17/17 [==============================] - 0s 23ms/step - loss: 0.0012 - val_loss: 0.0026
Epoch 21/100
17/17 [==============================] - 0s 24ms/step - loss: 0.0011 - val_loss: 0.0025
Epoch 22/100
17/17 [==============================] - 0s 24ms/step - loss: 0.0011 - val_loss: 0.0021
Epoch 23/100
17/17 [==============================] - 0s 24ms/step - loss: 0.0011 - val_loss: 0.0020
Epoch 24/100
17/17 [==============================] - 0s 24ms/step - loss: 0.0011 - val_loss: 0.0022
Epoch 25/100
17/17 [==============================] - 0s 24ms/step - loss: 0.0011 - val_loss: 0.0024
Epoch 26/100
17/17 [==============================] - 0s 16ms/step - loss: 0.0010 - val_loss: 0.0020
Epoch 27/100
17/17 [==============================] - 0s 17ms/step - loss: 0.0010 - val_loss: 0.0018
Epoch 28/100
17/17 [==============================] - 0s 16ms/step - loss: 9.9095e-04 - val_loss: 0.0022
Epoch 29/100
17/17 [==============================] - 0s 17ms/step - loss: 9.6073e-04 - val_loss: 0.0017
```

```
import tensorflow as tf
```

Prediction and check performance metrics

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

```
34/34 [==============================] - 1s 8ms/step
13/13 [==============================] - 0s 6ms/step
```

Transformback to original form

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

Calculate RMSE performance metrics

```
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

```
131.34307791688525
```

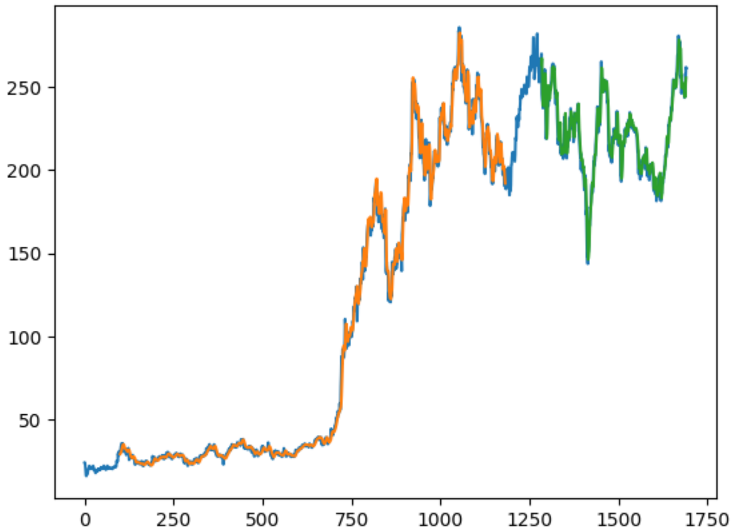
Test Data RMSE

```
math.sqrt(mean_squared_error(ytest,test_predict))
```

```
222.40967937559176
```

Plotting GRAPH

```
# shift train predictions for plotting
look_back=100
trainPredictPlot = numpy.empty_like(df1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(df1)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df1)-1, :] = test_predict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(df1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```




```
len(test_data)
```

```
508
```

```
x_input=test_data[len(test_data)-100:].reshape(1,-1)
x_input.shape
```

```
(1, 100)
```

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

```
temp_input
```

```
[0.6918294433597358,
 0.6902752730444144,
 0.6899052649158253,
 0.6964549612637113,
 0.6815053096005484,
```

```
        0.6732163630145528,
        0.6475354765104379,
        0.6372853695397857,
        0.6350650987433915,
        0.6466843997181779,
        0.6564905309783349,
        0.6628922292553653,
        0.6448341925565877,
        0.6274052706977227,
        0.6392835710718171,
        0.6129736215335901,
        0.621558608666269,
        0.6221506342534202,
        0.6396536347066211,
        0.6261841265702445,
        0.6243339194086542,
        0.648941652455318,
        0.6562314723723977,
        0.6692199081446892,
        0.667258692253818,
        0.6430210228419583,
        0.6423919043016313,
        0.6145648292958723,
        0.613047644621711,
        0.6327708603650912,
        0.6292554778593128,
        0.6562684580135578,
        0.6530860979952084,
        0.6526790524196585,
        0.6536041560004536,
        0.6747705296294598,
        0.6767687829672918,
        0.6726613230685616,
        0.6908303684966204,
        0.6917184346304547,
        0.714142934327685,
        0.7101094975170756,
        0.7128848082594609,
        0.7309798305993989,
        0.7538854063611284,
        0.7546625155714822,
        0.7359383746912175,
        0.7322749978149978,
        0.7444863761827362,
        0.7814905194145402,
        0.7806024014749052,
        0.7889653193432213,
        0.7973652746584982,
        0.7921476756611564,
        0.7916296213563255,
        0.7911115633510803,
        0.8213069590298896,
```

```
len(temp_input)
```

```
100
```

Making Prediction for next 30 days

```
from numpy import array

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)
```

```
[[0.8828132152557373], [0.8823336958885193], [0.8818637132644653], [0.8814046382904053], [0.8809574842453003], [0.8805240392684937],
```

```
day_new = np.arange(1,101)
day_pred = np.arange(101,131)
```

```
import matplotlib.pyplot as plt
```

```
len(df1)
```

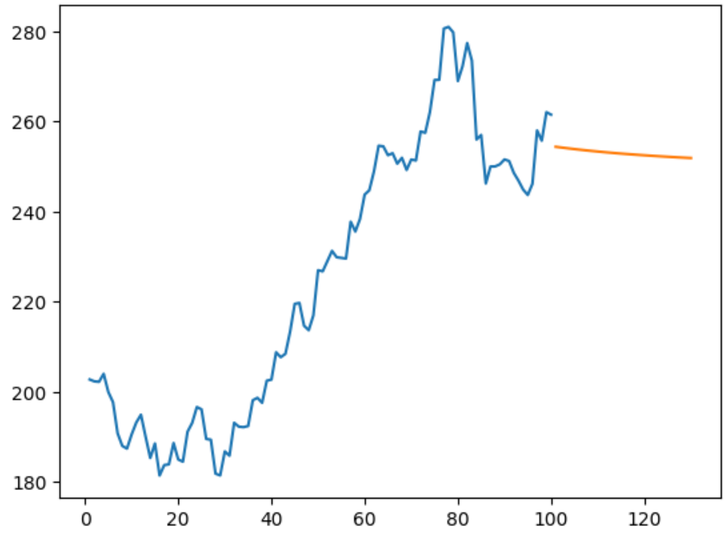
```
1692
```

```
len(df1) - 100
```

```
1592
```

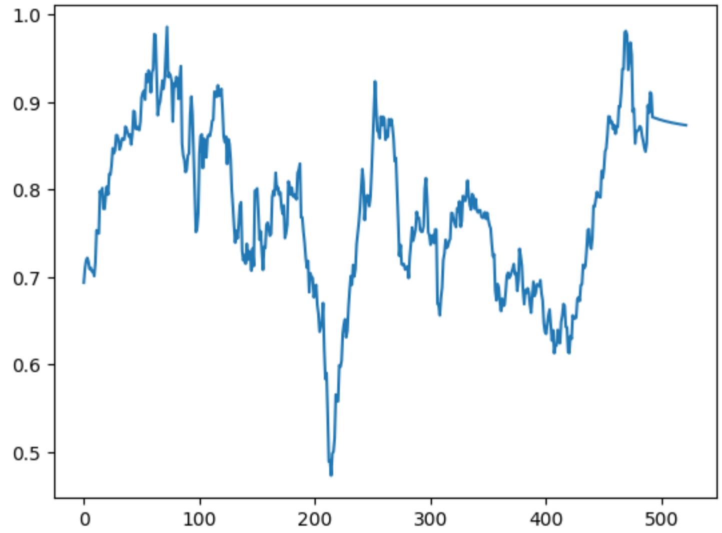
```
plt.plot(day_new, scaler.inverse_transform(df1[1592:]))
plt.plot(day_pred, scaler.inverse_transform(lst_output))
```

```
[<matplotlib.lines.Line2D at 0x782018fc6170>]
```




```
df3=df1.tolist()
df3.extend(lst_output)
plt.plot(df3[1200:])
```

```
[<matplotlib.lines.Line2D at 0x782018f959f0>]
```



```
df3=scaler.inverse_transform(df3).tolist()
```

```
plt.plot(df3)
```

```
[<matplotlib.lines.Line2D at 0x782028369c00>]
```