# Stock Market Prediction Using Stacked LSTM

**Anurag Kujur - M210691CA**

**Saurav Kumar - M210682CA**

In [129]:
```
# The goal of time series forecasting is to predict future values of a variable based on its historical behavior.
# We are using an LSTM (Long Short-Term Memory) neural network to predict the future stock prices of Tata Motors.
# LSTMs are a type of recurrent neural network (RNN) that are specifically designed for time series prediction.
```

In [130]:
```
# Data Collection
import pandas_datareader as pdr
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import pandas as pd
```

In [131]:
```
df=pd.read_csv('TataMotors.csv')
```

In [132]:
```
df.head()
```

Out[132]:

| | Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|---|
| **0** | 2010-01-04 | 156.303482 | 164.040497 | 156.184769 | 163.535919 | 153.140778 | 27906448 |
| **1** | 2010-01-05 | 162.259598 | 167.147186 | 153.246277 | 160.686478 | 150.472473 | 23669317 |
| **2** | 2010-01-06 | 162.200241 | 162.932388 | 159.311234 | 160.933823 | 150.704086 | 14990820 |
| **3** | 2010-01-07 | 161.171280 | 161.923218 | 154.344498 | 155.432831 | 145.552765 | 22722030 |
| **4** | 2010-01-08 | 156.323273 | 159.251862 | 155.343781 | 156.283707 | 146.349579 | 16495776 |

In [133]:
```
df.tail()
```

Out[133]:

| | Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|---|
| **3273** | 2023-04-06 | 426.500000 | 439.299988 | 423.750000 | 437.649994 | 437.649994 | 10907492 |
| **3274** | 2023-04-10 | 452.049988 | 473.299988 | 452.000000 | 461.299988 | 461.299988 | 50462653 |
| **3275** | 2023-04-11 | 463.750000 | 463.750000 | 455.799988 | 458.700012 | 458.700012 | 14495222 |
| **3276** | 2023-04-12 | 459.350006 | 468.600006 | 458.200012 | 465.500000 | 465.500000 | 13552440 |
| **3277** | 2023-04-13 | 464.950012 | 472.000000 | 463.250000 | 469.500000 | 469.500000 | 12733670 |

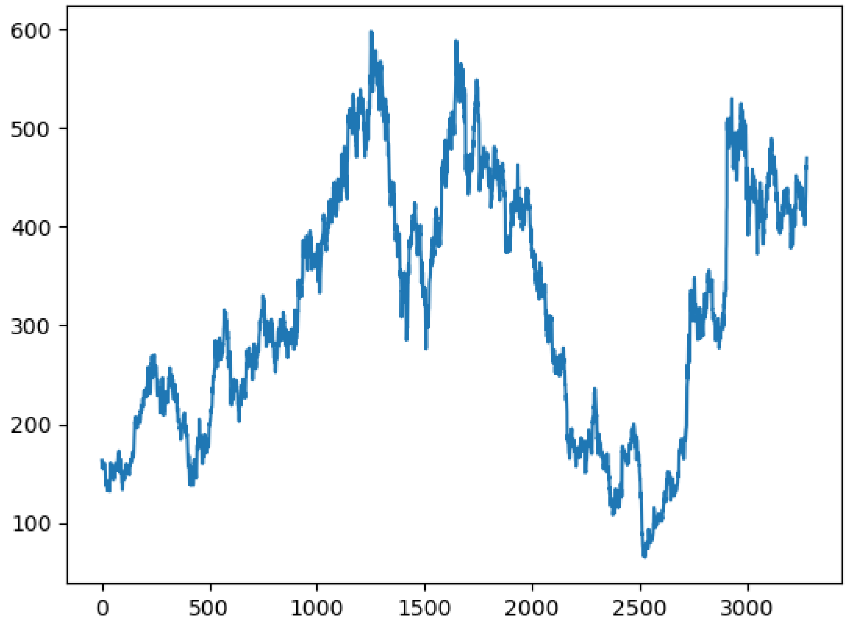
In [134]:
```
df1=df.reset_index()['Close']     #Select the 'Close' column
```

In [135]:
```
df1
```

Out[135]:
```
0       163.535919
1       160.686478
2       160.933823
3       155.432831
4       156.283707
           ...
3273    437.649994
3274    461.299988
3275    458.700012
3276    465.500000
3277    469.500000
Name: Close, Length: 3278, dtype: float64
```

In [136]:
```
plt.plot(df1)
```

Out[136]: [<matplotlib.lines.Line2D at 0x185fe299100>]




In [137]:
```
# LSTM are sensitive to the scale of the data.So we apply MinMax scaler and scale our values between 0 to 1
```

In [138]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))      #scale down in the range 0,1
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))
```

In [139]:
```
print(df1)
```

```
[[0.18436482]
 [0.17901711]
 [0.17948132]
 ...
 [0.73831572]
 [0.75107763]
 [0.75858466]]
```

In [140]:
```
# splitting dataset into train and test split
training_size=int(len(df1)*0.65)    #65% training size
test_size=len(df1)-training_size     #35% test size
train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]
```

In [141]:
```
training_size,test_size
```

Out[141]: (2130, 1148)

In [142]:
```
train_data
```

Out[142]:
```
array([[0.18436482],
       [0.17901711],
       [0.17948132],
       ...,
       [0.34419699],
       [0.34438467],
       [0.34907656]])
```

In [143]:
```
# DATA PREPROCESSING to be fed into LSTM model
# create_dataset to convert the time series data into a format that can be fed into the LSTM network
# This function creates a sliding window of time steps and associates each window with a corresponding output valu
# The sliding window of time steps is created using a user-defined variable called 'time_step'.
# Input data is a sequence of 100 stock prices, and the output data is the stock price that follows that sequence.
import numpy
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)                               #append in X_train
        dataY.append(dataset[i + time_step, 0])       #append in Y_train
    return numpy.array(dataX), numpy.array(dataY)
```

In [144]:
```
# The training and test data are then converted into the required format using the create_dataset function.
time_step = 100
X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)
```

In [145]:
```
#X_train== 100 days data value (100 columns=100 features=independent variables)
#Y_train== Value after 100 days (dependent variable)
print(X_train.shape), print(y_train.shape)
```

```
(2029, 100)
(2029,)
```

Out[145]: (None, None)

In [146]:
```
print(X_test.shape), print(ytest.shape)
```

```
(1047, 100)
(1047,)
```

Out[146]: (None, None)

In [147]:
```
# reshape input to be [samples, time steps, features] which is required for LSTM
# Input to the LSTM network should be a 3D array, so the X_train and X_test arrays are reshaped accordingly.
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)
```

In [148]:
```
### Create the Stacked LSTM model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

In [149]:
```
# The program creates an LSTM model using the Sequential class from the keras.models module.
# The model consists of three LSTM layers with 50 units each, followed by a dense output layer with one unit.
# The model is then compiled using the mean squared error as the loss function and the Adam optimizer.

model=Sequential()   #creates an instance of a Sequential model.

model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
# adds an LSTM layer to the model with 50 units.
# The input_shape=(100,1) argument specifies the shape of the input data that the model will expect

model.add(LSTM(50,return_sequences=True))
# adds another LSTM layer with 50 units and return_sequences=True.

model.add(LSTM(50))
# adds a final LSTM layer with 50 units. Since return_sequences is not specified,
# it defaults to False, meaning that this layer will only return the last output in the sequence.

model.add(Dense(1))
# adds a Dense layer with 1 unit. This is the output layer of the model.

model.compile(loss='mean_squared_error',optimizer='adam')
# compiles the model and specifies the loss function and optimizer to use during training
```

In [150]:
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 lstm_9 (LSTM)               (None, 100, 50)           10400     
                                                                 
 lstm_10 (LSTM)              (None, 100, 50)           20200     
                                                                 
 lstm_11 (LSTM)              (None, 50)                20200     
                                                                 
 dense_3 (Dense)             (None, 1)                 51        
                                                                 
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

In [151]:

```python
# We train the model using the fit function of the model object.
#The training process takes 50 epochs, and the batch size is set to 64.
model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=50,batch_size=64,verbose=1)
#model.save_weights('model.h5')
```

```
Epoch 1/50
32/32 [==============================] - 10s 177ms/step - loss: 0.0411 - val_loss: 0.0075
Epoch 2/50
32/32 [==============================] - 5s 143ms/step - loss: 0.0035 - val_loss: 0.0025
Epoch 3/50
32/32 [==============================] - 5s 145ms/step - loss: 0.0017 - val_loss: 0.0022
Epoch 4/50
32/32 [==============================] - 5s 158ms/step - loss: 0.0016 - val_loss: 0.0022
Epoch 5/50
32/32 [==============================] - 5s 161ms/step - loss: 0.0015 - val_loss: 0.0021
Epoch 6/50
32/32 [==============================] - 5s 166ms/step - loss: 0.0014 - val_loss: 0.0020
Epoch 7/50
32/32 [==============================] - 5s 150ms/step - loss: 0.0014 - val_loss: 0.0019
Epoch 8/50
32/32 [==============================] - 5s 156ms/step - loss: 0.0013 - val_loss: 0.0018
Epoch 9/50
32/32 [==============================] - 5s 172ms/step - loss: 0.0013 - val_loss: 0.0019
Epoch 10/50
32/32 [==============================] - 5s 170ms/step - loss: 0.0013 - val_loss: 0.0015
Epoch 11/50
32/32 [==============================] - 5s 160ms/step - loss: 0.0011 - val_loss: 0.0015
Epoch 12/50
32/32 [==============================] - 5s 160ms/step - loss: 0.0012 - val_loss: 0.0014
Epoch 13/50
32/32 [==============================] - 5s 163ms/step - loss: 0.0011 - val_loss: 0.0014
Epoch 14/50
32/32 [==============================] - 5s 165ms/step - loss: 0.0010 - val_loss: 0.0013
Epoch 15/50
32/32 [==============================] - 5s 166ms/step - loss: 0.0010 - val_loss: 0.0012
Epoch 16/50
32/32 [==============================] - 5s 165ms/step - loss: 9.6733e-04 - val_loss: 0.0012
Epoch 17/50
32/32 [==============================] - 5s 169ms/step - loss: 0.0010 - val_loss: 0.0012
Epoch 18/50
32/32 [==============================] - 5s 170ms/step - loss: 9.9545e-04 - val_loss: 0.0011
Epoch 19/50
32/32 [==============================] - 5s 166ms/step - loss: 8.5790e-04 - val_loss: 0.0010
Epoch 20/50
32/32 [==============================] - 5s 163ms/step - loss: 8.3325e-04 - val_loss: 0.0010
Epoch 21/50
32/32 [==============================] - 5s 162ms/step - loss: 8.0302e-04 - val_loss: 9.7463e-04
Epoch 22/50
32/32 [==============================] - 5s 165ms/step - loss: 7.7385e-04 - val_loss: 0.0011
Epoch 23/50
32/32 [==============================] - 5s 162ms/step - loss: 8.7011e-04 - val_loss: 0.0013
Epoch 24/50
32/32 [==============================] - 5s 170ms/step - loss: 7.6847e-04 - val_loss: 9.6155e-04
Epoch 25/50
32/32 [==============================] - 5s 168ms/step - loss: 7.0657e-04 - val_loss: 8.4388e-04
Epoch 26/50
32/32 [==============================] - 5s 157ms/step - loss: 7.1441e-04 - val_loss: 0.0011
Epoch 27/50
32/32 [==============================] - 5s 155ms/step - loss: 7.5057e-04 - val_loss: 9.1495e-04
Epoch 28/50
32/32 [==============================] - 5s 159ms/step - loss: 8.3262e-04 - val_loss: 8.8486e-04
Epoch 29/50
32/32 [==============================] - 5s 159ms/step - loss: 7.8260e-04 - val_loss: 8.7046e-04
Epoch 30/50
32/32 [==============================] - 5s 164ms/step - loss: 6.4850e-04 - val_loss: 8.4889e-04
Epoch 31/50
32/32 [==============================] - 6s 174ms/step - loss: 5.9416e-04 - val_loss: 7.2489e-04
Epoch 32/50
32/32 [==============================] - 5s 162ms/step - loss: 5.8179e-04 - val_loss: 7.4798e-04
Epoch 33/50
32/32 [==============================] - 5s 158ms/step - loss: 5.4378e-04 - val_loss: 6.8241e-04
Epoch 34/50
32/32 [==============================] - 5s 158ms/step - loss: 5.5800e-04 - val_loss: 6.5913e-04
Epoch 35/50
32/32 [==============================] - 5s 162ms/step - loss: 5.4258e-04 - val_loss: 6.9035e-04
Epoch 36/50
32/32 [==============================] - 5s 161ms/step - loss: 5.0505e-04 - val_loss: 6.1601e-04
Epoch 37/50
32/32 [==============================] - 5s 156ms/step - loss: 5.1911e-04 - val_loss: 6.0619e-04
Epoch 38/50
32/32 [==============================] - 5s 171ms/step - loss: 4.8358e-04 - val_loss: 6.6314e-04
Epoch 39/50
32/32 [==============================] - 5s 164ms/step - loss: 5.1714e-04 - val_loss: 8.6075e-04
Epoch 40/50
32/32 [==============================] - 5s 173ms/step - loss: 6.0393e-04 - val_loss: 6.1163e-04
Epoch 41/50
```

```
32/32 [==============================] - 5s 161ms/step - loss: 4.8957e-04 - val_loss: 5.8789e-04
Epoch 42/50
32/32 [==============================] - 5s 159ms/step - loss: 4.7156e-04 - val_loss: 5.5701e-04
Epoch 43/50
32/32 [==============================] - 5s 160ms/step - loss: 4.7209e-04 - val_loss: 5.4443e-04
Epoch 44/50
32/32 [==============================] - 5s 156ms/step - loss: 4.6022e-04 - val_loss: 5.7703e-04
Epoch 45/50
32/32 [==============================] - 5s 157ms/step - loss: 5.5106e-04 - val_loss: 5.9827e-04
Epoch 46/50
32/32 [==============================] - 5s 154ms/step - loss: 4.6157e-04 - val_loss: 5.5548e-04
Epoch 47/50
32/32 [==============================] - 5s 158ms/step - loss: 4.8892e-04 - val_loss: 5.4797e-04
Epoch 48/50
32/32 [==============================] - 5s 168ms/step - loss: 5.6560e-04 - val_loss: 7.3042e-04
Epoch 49/50
32/32 [==============================] - 5s 161ms/step - loss: 4.1903e-04 - val_loss: 5.2369e-04
Epoch 50/50
32/32 [==============================] - 5s 163ms/step - loss: 4.2514e-04 - val_loss: 5.0664e-04
```

Out[151]:
```
<keras.callbacks.History at 0x185f73171f0>
```

In [152]:
```python
#To load the already trained model
#model.load_weights('model.h5')
```

In [153]:
```python
# After training the model, we use it to make predictions on the training and testing datasets.
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

```
64/64 [==============================] - 3s 31ms/step
33/33 [==============================] - 1s 32ms/step
```

In [154]:
```python
# The predictions are scaled back to their original range using the inverse_transform method of the scaler object.
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

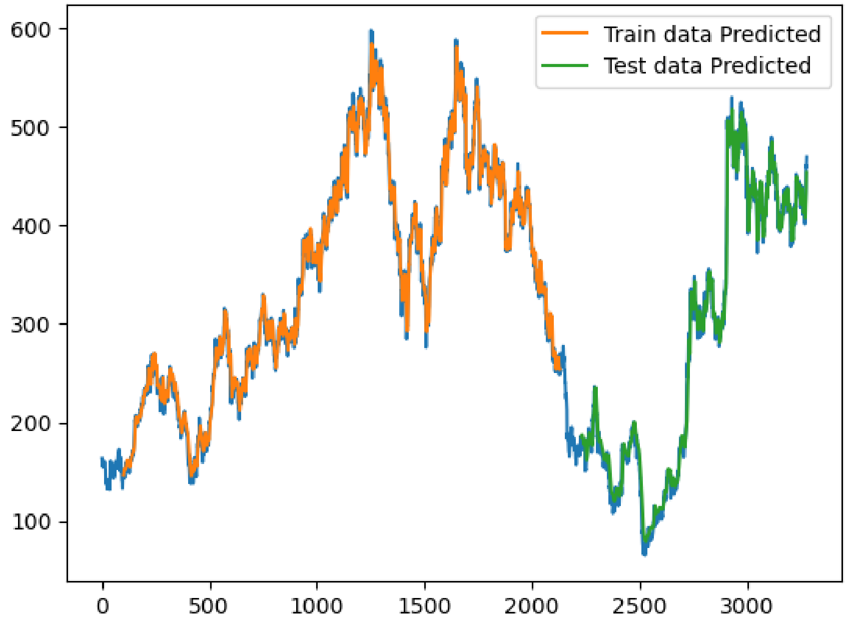
In [155]:
```python
# Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error

#Train data RMSE
math.sqrt(mean_squared_error(y_train,train_predict))
```

Out[155]:
```
367.1638465658042
```

In [156]:
```python
# Test Data RMSE
math.sqrt(mean_squared_error(ytest,test_predict))
```

Out[156]:
```
312.9890655220657
```

In [157]:
```python
# Plot the actual and predicted stock prices for both the training and test data
# 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,label='Train data Predicted')
plt.plot(testPredictPlot,label='Test data Predicted')
plt.legend()
plt.show()
```




In [158]:
```python
len(test_data)
```

Out[158]: 1148

In [159]:
```python
#Taking the last 100 days test_data in x_input
x_input=test_data[1048:].reshape(1,-1)
x_input.shape
```

Out[159]: (1, 100)

In [160]:
```python
# Converting x_input into a list
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
```

In [161]:
```python
# We generate predictions for the next 30 days using the last 100 days of data from the test dataset.
# The predictions are stored in a list called lst_output
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)
```

```
[0.75324076]
101
1 day input [0.66774969 0.67450602 0.67328609 0.67206621 0.69045845 0.69055226
 0.69036457 0.7020943  0.69993604 0.69458727 0.68210686 0.67901022
 0.66587293 0.66043035 0.65311096 0.65470621 0.66287011 0.66174405
 0.65949198 0.66868807 0.66193174 0.64785607 0.63302971 0.61773416
 0.58751838 0.59962342 0.6171711  0.61182234 0.60168787 0.60553525
 0.61839098 0.61670191 0.60112486 0.60356462 0.59436853 0.60835038
 0.65236027 0.66230711 0.65114039 0.64973282 0.65254796 0.65686447
 0.64391487 0.6295577  0.6340619  0.64391487 0.66972026 0.66390231
 0.71373021 0.71007051 0.72592912 0.71570078 0.71222877 0.7134487
 0.70697387 0.69468115 0.70340805 0.69712091 0.71419939 0.70519093
 0.70425255 0.71100889 0.70622318 0.70303267 0.70885063 0.69665172
 0.68342061 0.69045845 0.6802301  0.66183792 0.666999   0.67694578
 0.66652981 0.68069929 0.70340805 0.70190661 0.68858169 0.69543184
 0.67018945 0.6593981  0.64945132 0.65761522 0.66380849 0.64832526
 0.65170339 0.65836591 0.66408999 0.6591166  0.65076502 0.63115295
 0.64541631 0.66718663 0.67366146 0.67797797 0.69880997 0.74319524
 0.73831572 0.75107763 0.75858466 0.75324076]
```

In [162]:
```python
day_new=np.arange(1,101)     #100 days input
day_pred=np.arange(101,131)  #Next 30 days output
```

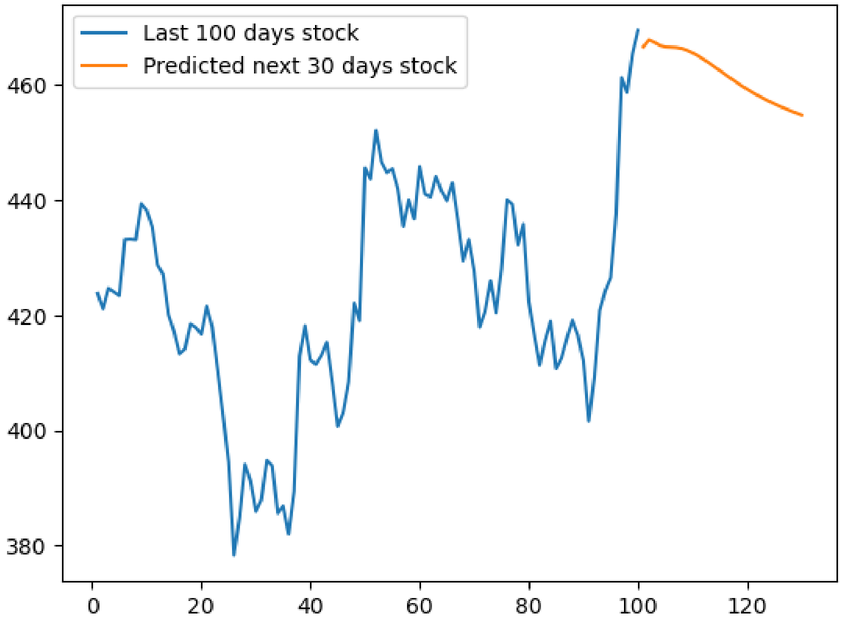
In [163]:
```python
import matplotlib.pyplot as plt
```

In [164]:
```python
len(df1)
```

Out[164]: 3278

In [170]:
```
plt.plot(day_new,scaler.inverse_transform(df1[3178:]),label='Last 100 days stock')
plt.plot(day_pred,scaler.inverse_transform(lst_output),label='Predicted next 30 days stock')
plt.legend()
```

Out[170]: <matplotlib.legend.Legend at 0x18595ebf9d0>




In [166]:
```
df3=df1.tolist()
df3.extend(lst_output)
```

In [167]:
```
# The output of the model is then transformed back to the original scale,
# Plot predicted values for the next 30 days against the actual values for the last 100 days of the test dataset
df3=scaler.inverse_transform(df3).tolist()
```

In [168]:
```
plt.plot(df3)
plt.show()
#Final Graph
```