Task 1 Stock Prediction :

In this task, I will apple stock dataset which I find at Kaggle. I have to make a model using LSTM method for Stock Prediction

1. At first I have to import all libarires which I have to use later in this task

In [45]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
import math
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

2. Load the 'apple stock.csv' dataset and analysis this dataset for further process.

In [5]:
```
data_appl = pd.read_csv('AAPL.csv')
data_appl.head()
```

Out[5]:

	Unnamed: 0	symbol	date	close	high	low	open	volume	adjClos
0	0	AAPL	2015-05-27 00:00:00+00:00	132.045	132.260	130.05	130.34	45833246	121.68255
1	1	AAPL	2015-05-28 00:00:00+00:00	131.780	131.950	131.10	131.86	30733309	121.43835
2	2	AAPL	2015-05-29 00:00:00+00:00	130.280	131.450	129.90	131.23	50884452	120.05606
3	3	AAPL	2015-06-01 00:00:00+00:00	130.535	131.390	130.05	131.20	32112797	120.29105
4	4	AAPL	2015-06-02 00:00:00+00:00	129.960	130.655	129.32	129.86	33667627	119.76118

In [6]:
```
data_appl.tail()
```

Out[6]:

	Unnamed: 0	symbol	date	close	high	low	open	volume	adjClo
1253	1253	AAPL	2020-05-18 00:00:00+00:00	314.96	316.50	310.3241	313.17	33843125	314.
1254	1254	AAPL	2020-05-19 00:00:00+00:00	313.14	318.52	313.0100	315.03	25432385	313.
1255	1255	AAPL	2020-05-20 00:00:00+00:00	319.23	319.52	316.2000	316.68	27876215	319.
1256	1256	AAPL	2020-05-21 00:00:00+00:00	316.85	320.89	315.8700	318.66	25672211	316.
1257	1257	AAPL	2020-05-22 00:00:00+00:00	318.89	319.23	315.3500	315.77	20450754	318.

In [7]:
```
print("Dimension of the dataset: ", data_appl.shape)
```

```
Dimension of the dataset:  (1258, 15)
```

In [8]:
```
data_appl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Unnamed: 0   1258 non-null   int64  
 1   symbol       1258 non-null   object 
 2   date         1258 non-null   object 
 3   close        1258 non-null   float64
 4   high         1258 non-null   float64
 5   low          1258 non-null   float64
 6   open         1258 non-null   float64
 7   volume       1258 non-null   int64  
 8   adjClose     1258 non-null   float64
 9   adjHigh      1258 non-null   float64
 10  adjLow       1258 non-null   float64
 11  adjOpen      1258 non-null   float64
 12  adjVolume    1258 non-null   int64  
 13  divCash      1258 non-null   float64
 14  splitFactor  1258 non-null   float64
dtypes: float64(10), int64(3), object(2)
memory usage: 147.5+ KB
```

In [9]:
```
Y = data_appl['close']
```

In [10]:
```
Y.head()
```

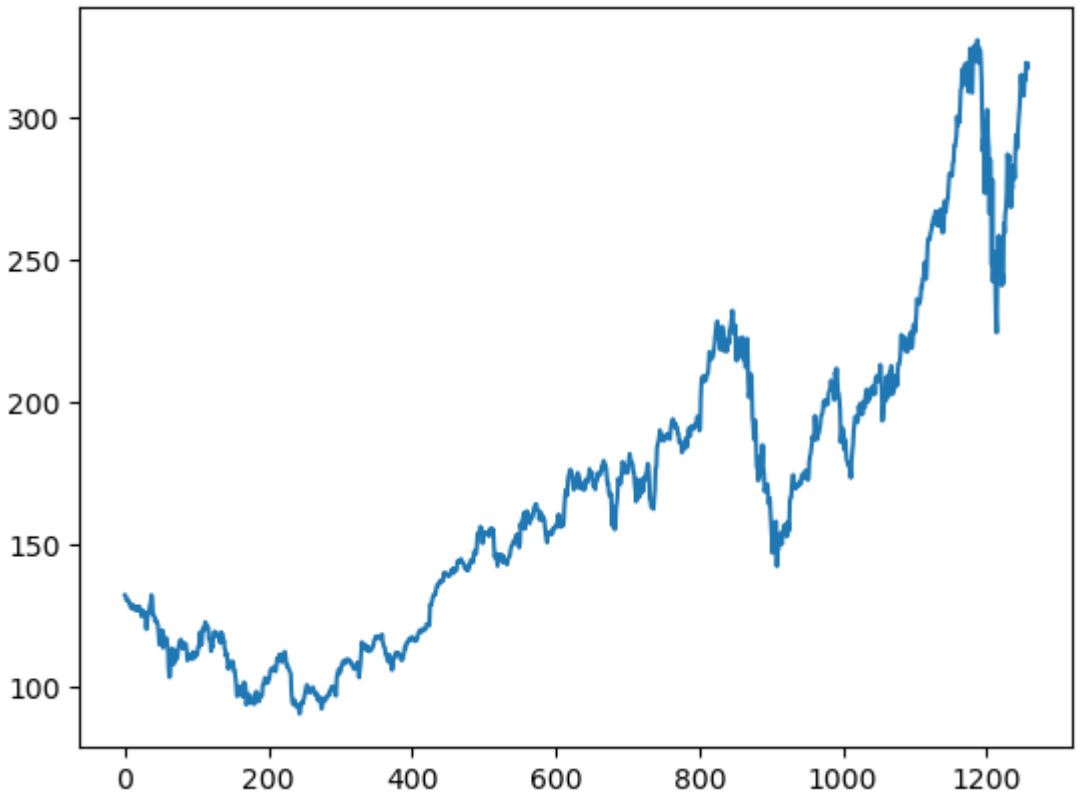
Out[10]:
```
0    132.045
1    131.780
2    130.280
3    130.535
4    129.960
Name: close, dtype: float64
```

In [11]:
```
Y.tail()
```

Out[11]:
```
1253    314.96
1254    313.14
1255    319.23
1256    316.85
1257    318.89
Name: close, dtype: float64
```

In [12]:
```
plt.plot(Y)
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x7ac8582ab3a0>]
```



2.1 Reshaping scale of data as LSTM are sensitive to the scale of the data.

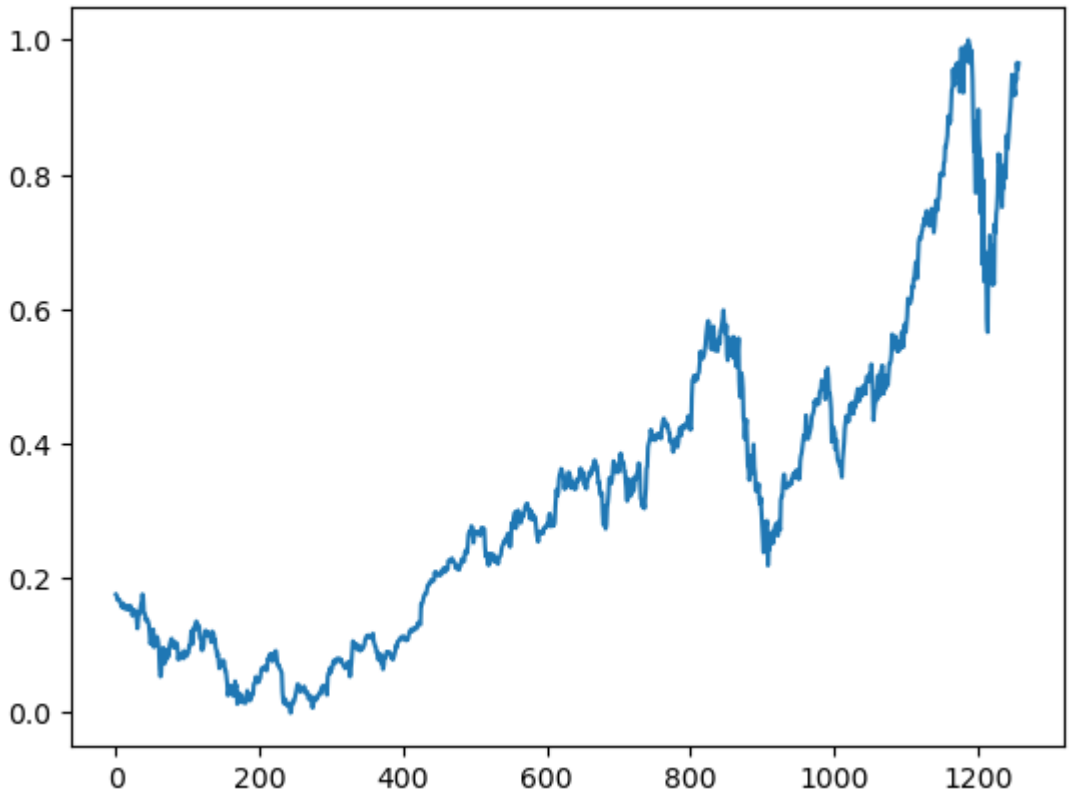
In [13]:
```
scaler=MinMaxScaler(feature_range=(0,1))
Y=scaler.fit_transform(np.array(Y).reshape(-1,1))
```

In [14]:
```
Y
```

Out[14]:
```
array([[0.17607447],
       [0.17495567],
       [0.16862282],
       ...,
       [0.96635143],
       [0.9563033 ],
       [0.96491598]])
```

In [15]:
```
plt.plot(Y)
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x7ac8561a9b10>]
```



2.2 Splitting the dataset into train and test split

In [16]:
```
training_size = int(len(Y) * 0.65)
train_data = Y[:training_size]
test_data = Y[training_size:]
```

In [17]:
```
train_data.shape, test_data.shape
```

Out[17]:
```
((817, 1), (441, 1))
```

In [18]:
```
train_data
```

```
       [0.08975766],
       [0.09055982],
       [0.08388922],
       [0.09085536],
       [0.0873934 ],
       [0.09030651],
       [0.09891919],
       [0.09887697],
       [0.10622309],
       [0.1213375 ],
       [0.10529427],
       [0.10221228],
       [0.12213966],
       [0.12745926],
       [0.1231107 ],
       [0.1302035 ],
       [0.13607194],
       [0.13366546],
       [0.1291058 ],
       [0.12969687],
```

In [19]:
```
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)
```

In [20]:
```
time_step = 100
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)
```

In [21]:
```
print(X_train.shape)
print(y_train.shape)
```

```
(716, 100)
(716,)
```

In [22]:
```
print(X_test.shape)
print(y_test.shape)
```

```
(340, 100)
(340,)
```

In [36]:
```
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='mse',optimizer='adam')
```

In [37]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (None, 100, 50)           10400

 lstm_4 (LSTM)               (None, 100, 50)           20200

 lstm_5 (LSTM)               (None, 50)                20200

 dense_1 (Dense)             (None, 1)                 51

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

In [25]:
```
model.fit(X_train,y_train,validation_data=(X_test,y_test), epochs=100, batc
```

Out[25]:
```
<keras.callbacks.History at 0x7ac848276800>
```

In [26]:
```
### Lets Do the prediction and check performance metrics
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

```
23/23 [==============================] - 1s 8ms/step
11/11 [==============================] - 0s 6ms/step
```

In [27]:
```
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

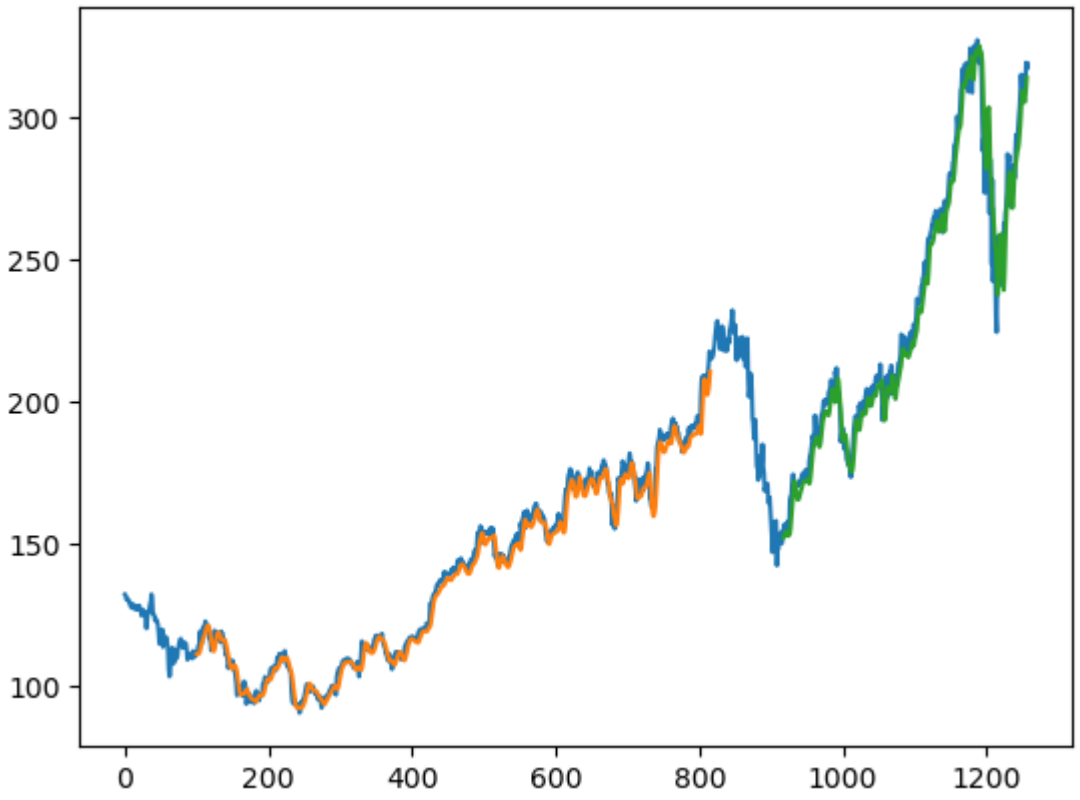
In [28]:
```
math.sqrt(mean_squared_error(y_train,train_predict))
```

Out[28]:
```
140.18044830546
```

In [29]:
```
### Test Data RMSE
math.sqrt(mean_squared_error(y_test,test_predict))
```

Out[29]:
```
235.52831725237826
```

In [30]:
```
### Plotting
# shift train predictions for plotting
look_back=100
trainPredictPlot = numpy.empty_like(Y)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(Y)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(Y)-1, :] = test_pred
# plot baseline and predictions
plt.plot(scaler.inverse_transform(Y))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



In [31]:
```
len(test_data)
```

Out[31]: 441

In [32]:
```
x_input=test_data[341:].reshape(1,-1)
x_input.shape
```

Out[32]: (1, 100)

In [33]:
```
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
```

In [34]:
```
temp_input
```

Out[34]: [0.8583551465000423,
0.8866418981676942,
0.8743139407244789,
0.8843198513890065,
0.8783669678290975,
0.8986321033521913,
0.925821160179009,
0.9287764924427933,
0.9567677108840666,
0.9386979650426415,
0.933040614709111,
0.9495060373216249,
0.9642404796082076,
0.9551211686228154,
0.9598919192772104,
0.9663514312251966,
0.9624672802499368,
0.9229502659799038,
0.9598497002448705,
0.9879253567508233,
0.985941062230854,
0.9253145317909315,
0.9217259140420504,
0.964747107996285,
0.9757240564046274,
0.9915984125643842,
0.9697289538123788,
0.9761462467280253,
0.9679557544541082,
1.0000000000000002,
0.9901629654648318,
0.9905007177235499,
0.9653803934813816,
0.9848855864223593,
0.9708688676855528,
0.9402600692392133,
0.8774803681499621,
0.8348391454867856,
0.8541332432660644,
0.7733682344000676,
0.7726927298826314,
0.8801401671873683,
0.8400743054969182,
0.8967322468969012,
0.8552731571392387,
0.8388499535590646,
0.7423372456303303,
0.8232711306256861,
0.7814320695769654,
0.6665963016127672,
0.7921557037912694,
0.6411804441442204,
0.6861437135860848,
0.6600101325677616,
0.6520307354555435,
0.5864223591995272,
0.5658616904500551,
0.660896732246897,
0.6551549438486872,
0.7097019336316812,
0.664527569028118,

0.6943764248923416,
0.692181035210673,
0.6356919699400492,
0.6526640209406402,
0.637802921557038,
0.7267162036646122,
0.7138816178333194,
0.7419150553069325,
0.7500211095161702,
0.7722283205268936,
0.8304905851557884,
0.8194291986827664,
0.8289706999915563,
0.8125474964113824,
0.7877649244279323,
0.7516254327450818,
0.7842607447437306,
0.7797433082833742,
0.8132652199611587,
0.8141096006079542,
0.7947310647639958,
0.8333614793548934,
0.8589884319851391,
0.8390188296884238,
0.8562864139153934,
0.8748627881448958,
0.887824031073208,
0.9009541501308793,
0.9279321117959978,
0.9485349995778098,
0.9333361479354896,
0.9174617917757326,
0.925441188887951,
0.9177151059697712,
0.9483239044161109,
0.9406400405302711,
0.9663514312251966,
0.9563033015283293,
0.964915984125644]

In [38]:
```
# demonstrate prediction for next 10 days
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)
```

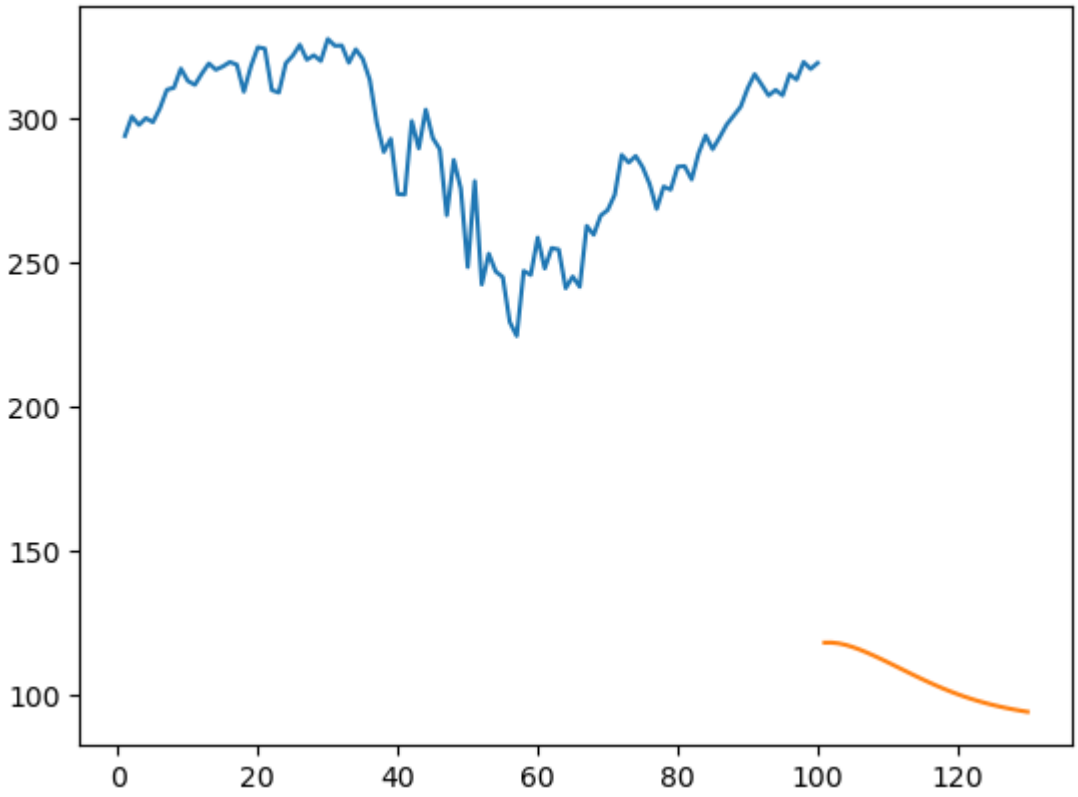
```
[0.11748741]
101
1 day input [0.8866419  0.87431394 0.88431985 0.87836697 0.8986321  0.
92582116
 0.92877649 0.95676771 0.93869797 0.93304061 0.94950604 0.96424048
 0.95512117 0.95989192 0.96635143 0.96246728 0.92295027 0.9598497
 0.98792536 0.98594106 0.92531453 0.92172591 0.96474711 0.97572406
 0.99159841 0.96972895 0.97614625 0.96795575 1.         0.99016297
 0.99050072 0.96538039 0.98488559 0.97086887 0.94026007 0.87748037
 0.83483915 0.85413324 0.77336823 0.77269273 0.88014017 0.84007431
 0.89673225 0.85527316 0.83884995 0.74233725 0.82327113 0.78143207
 0.6665963  0.7921557  0.64118044 0.68614371 0.66001013 0.65203074
 0.58642236 0.56586169 0.66089673 0.65515494 0.70970193 0.66452757
 0.69437642 0.69218104 0.63569197 0.65266402 0.63780292 0.7267162
 0.71388162 0.74191506 0.75002111 0.77222832 0.83049059 0.8194292
 0.8289707  0.8125475  0.78776492 0.75162543 0.78426074 0.77974331
 0.81326522 0.8141096  0.79473106 0.83336148 0.85898843 0.83901883
 0.85628641 0.87486279 0.88782403 0.90095415 0.92793211 0.948535
 0.93333615 0.91746179 0.92544119 0.91771511 0.9483239  0.94064004
```

In [39]:
```
day_new=np.arange(1,101)
day_pred=np.arange(101,131)
```

In [40]:
```
len(Y)
```

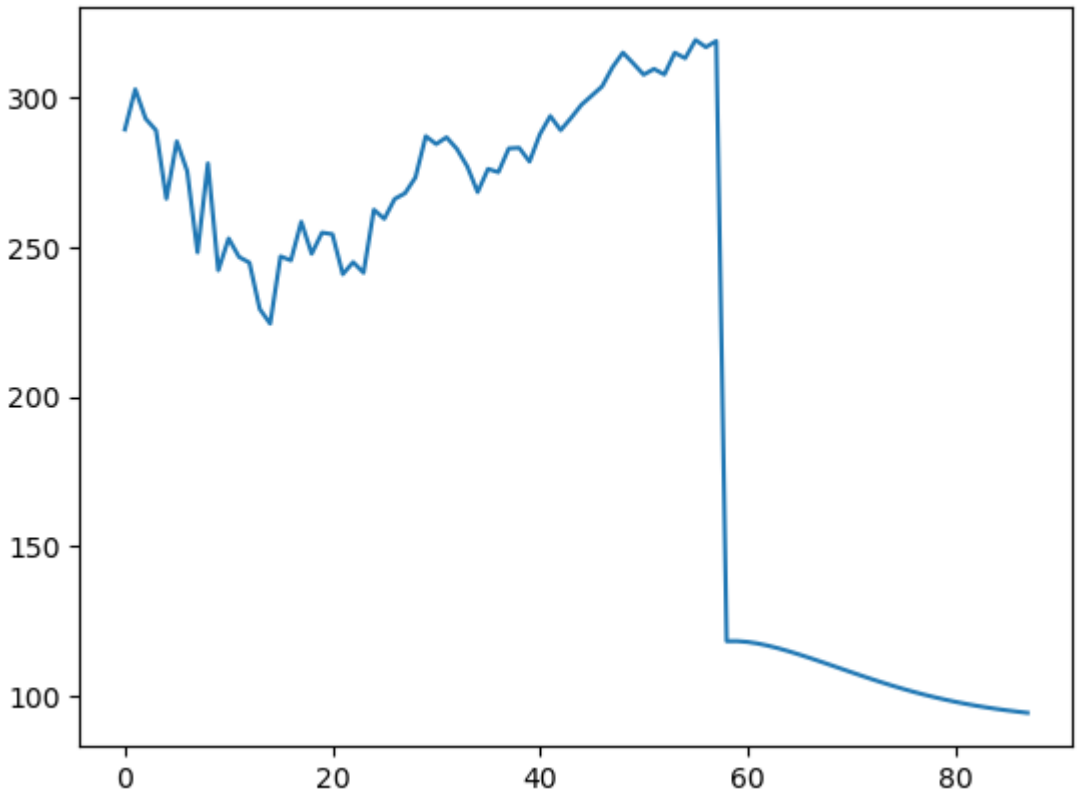
Out[40]: 1258

In [41]:
```
plt.plot(day_new,scaler.inverse_transform(Y[1158:]))
plt.plot(day_pred,scaler.inverse_transform(lst_output))
plt.show()
```



In [42]:
```
df3=Y.tolist()
df3.extend(lst_output)
df3=scaler.inverse_transform(df3).tolist()
plt.plot(df3[1200:])
```

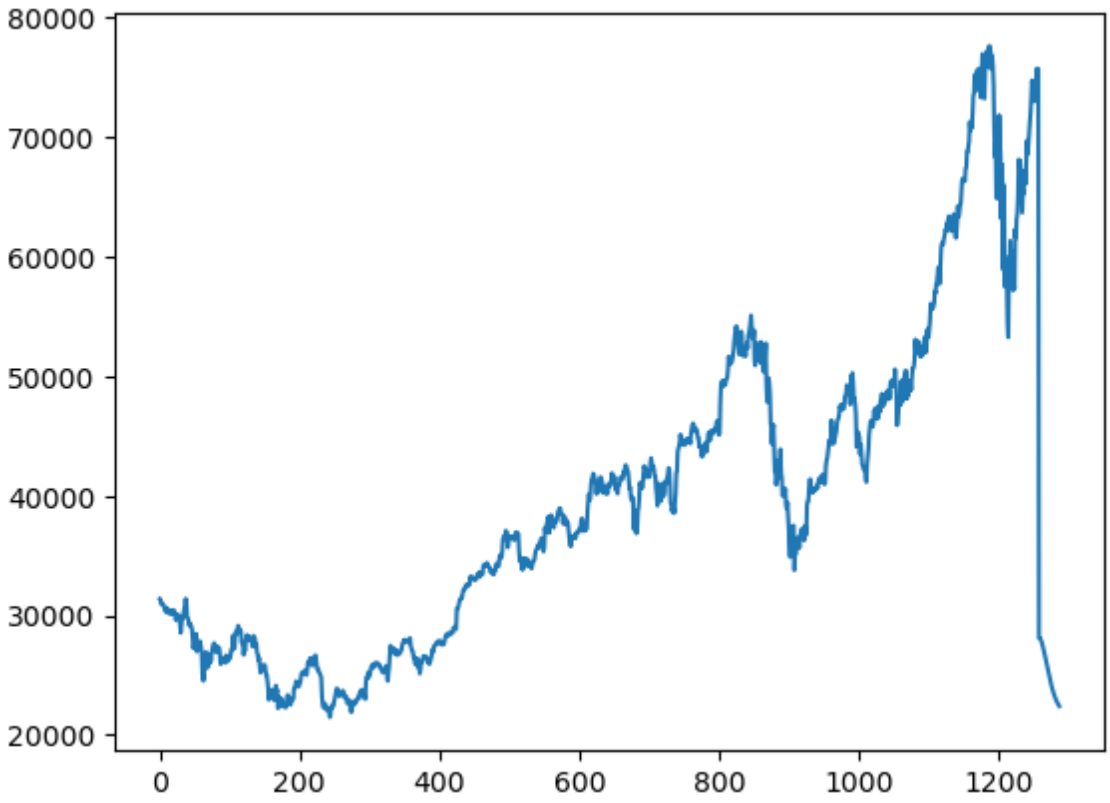
Out[42]:
```
[<matplotlib.lines.Line2D at 0x7ac7e0215000>]
```



In [43]:
```
df3=scaler.inverse_transform(df3).tolist()
```

In [44]:
```
plt.plot(df3)
```

Out[44]:
```
[<matplotlib.lines.Line2D at 0x7ac7e032f070>]
```



In [ ]:

In [ ]: