In [11]:
```
import pandas as pd
import pandas_datareader as pdr
key = 'Put_your_key'
df = pdr.get_data_tiingo('AAPL', api_key = key)
```

```
<ipython-input-11-71508f9ade83>:4: FutureWarning: In a future version of pand
as all arguments of concat except for the argument 'objs' will be keyword-onl
y.
  df = pdr.get_data_tiingo('AAPL', api_key = key)
```

In [12]:
```
df.to_csv('AAPL.csv')
```

In [13]:
```
df = pd.read_csv('AAPL.csv')
```

In [14]:
```
df
```

Out[14]:

	symbol	date	close	high	low	open	volume	adjClose	adjHigh
0	AAPL	2019-03-01 00:00:00+00:00	174.97	175.1500	172.89	174.28	25886167	42.109541	42.15286
1	AAPL	2019-03-04 00:00:00+00:00	175.85	177.7500	173.97	175.69	27436203	42.321328	42.77859
2	AAPL	2019-03-05 00:00:00+00:00	175.53	176.0000	174.54	175.94	19737419	42.244315	42.35742
3	AAPL	2019-03-06 00:00:00+00:00	174.52	175.4900	173.94	174.67	20810384	42.001241	42.23468
4	AAPL	2019-03-07 00:00:00+00:00	172.50	174.4400	172.02	173.87	24796374	41.515093	41.98198
...	...	...	...	...	...	...	...	...	..
1252	AAPL	2024-02-21 00:00:00+00:00	182.32	182.8888	180.66	181.94	41529674	182.320000	182.88880
1253	AAPL	2024-02-22 00:00:00+00:00	184.37	184.9550	182.46	183.48	52292208	184.370000	184.95500
1254	AAPL	2024-02-23 00:00:00+00:00	182.52	185.0400	182.23	185.01	45119677	182.520000	185.04000
1255	AAPL	2024-02-26 00:00:00+00:00	181.16	182.7600	180.65	182.24	40867421	181.160000	182.76000
1256	AAPL	2024-02-27 00:00:00+00:00	182.63	183.9225	179.56	181.10	54318851	182.630000	183.92250

1257 rows × 14 columns

In [15]:

```
df.tail()
```

Out[15]:

	symbol	date	close	high	low	open	volume	adjClose	adjHigh	ac
1252	AAPL	2024-02-21 00:00:00+00:00	182.32	182.8888	180.66	181.94	41529674	182.32	182.8888	1
1253	AAPL	2024-02-22 00:00:00+00:00	184.37	184.9550	182.46	183.48	52292208	184.37	184.9550	1
1254	AAPL	2024-02-23 00:00:00+00:00	182.52	185.0400	182.23	185.01	45119677	182.52	185.0400	1
1255	AAPL	2024-02-26 00:00:00+00:00	181.16	182.7600	180.65	182.24	40867421	181.16	182.7600	1
1256	AAPL	2024-02-27 00:00:00+00:00	182.63	183.9225	179.56	181.10	54318851	182.63	183.9225	1

In [16]:

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

In [17]:

```
df1.shape
```

Out[17]:

```
(1257,)
```

In [18]:

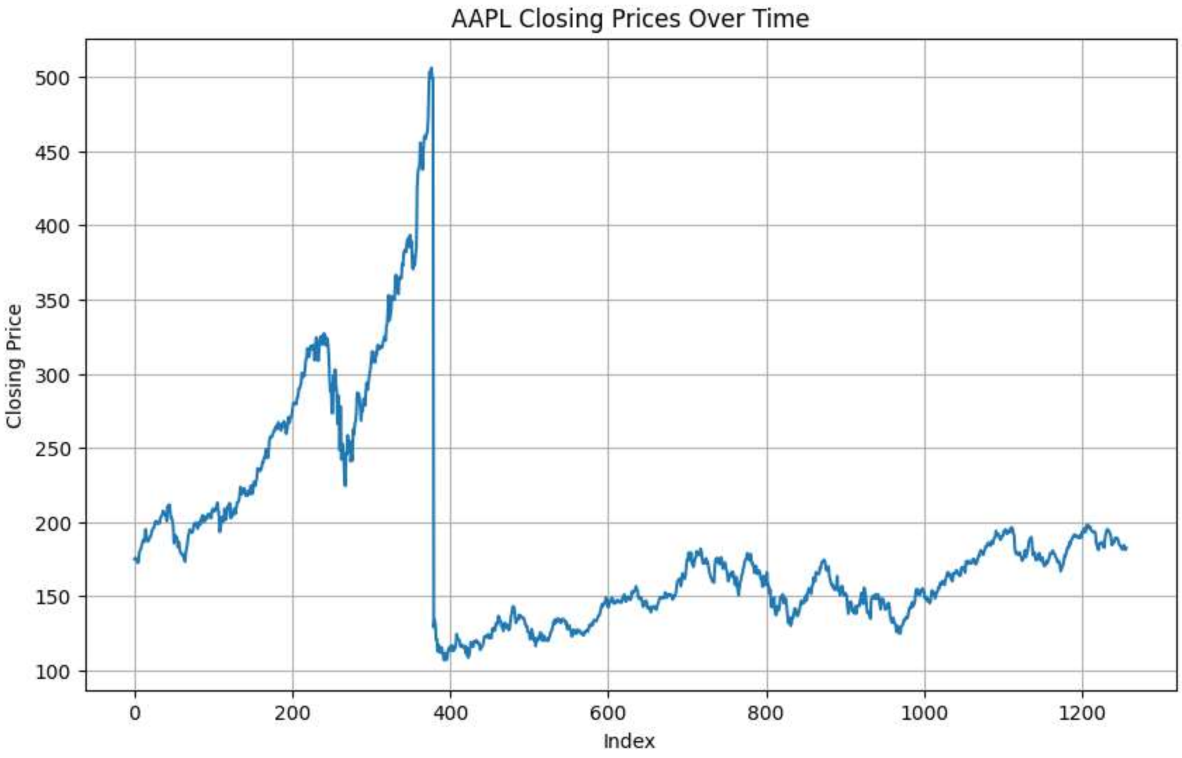
```
df1
```

Out[18]:

```
0       174.97
1       175.85
2       175.53
3       174.52
4       172.50
          ...
1252    182.32
1253    184.37
1254    182.52
1255    181.16
1256    182.63
Name: close, Length: 1257, dtype: float64
```

In [20]:

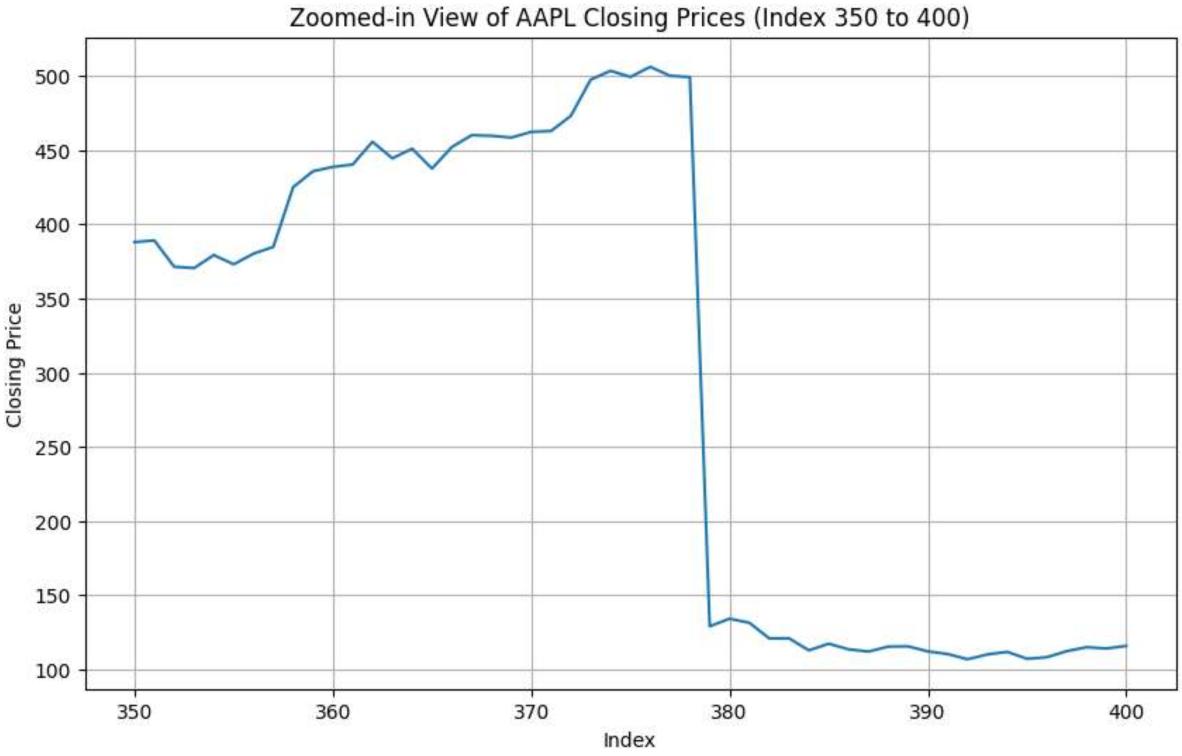
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.plot(df1)
plt.title('AAPL Closing Prices Over Time')
plt.xlabel('Index')
plt.ylabel('Closing Price')
plt.grid(True)
plt.show()
```



In [21]:

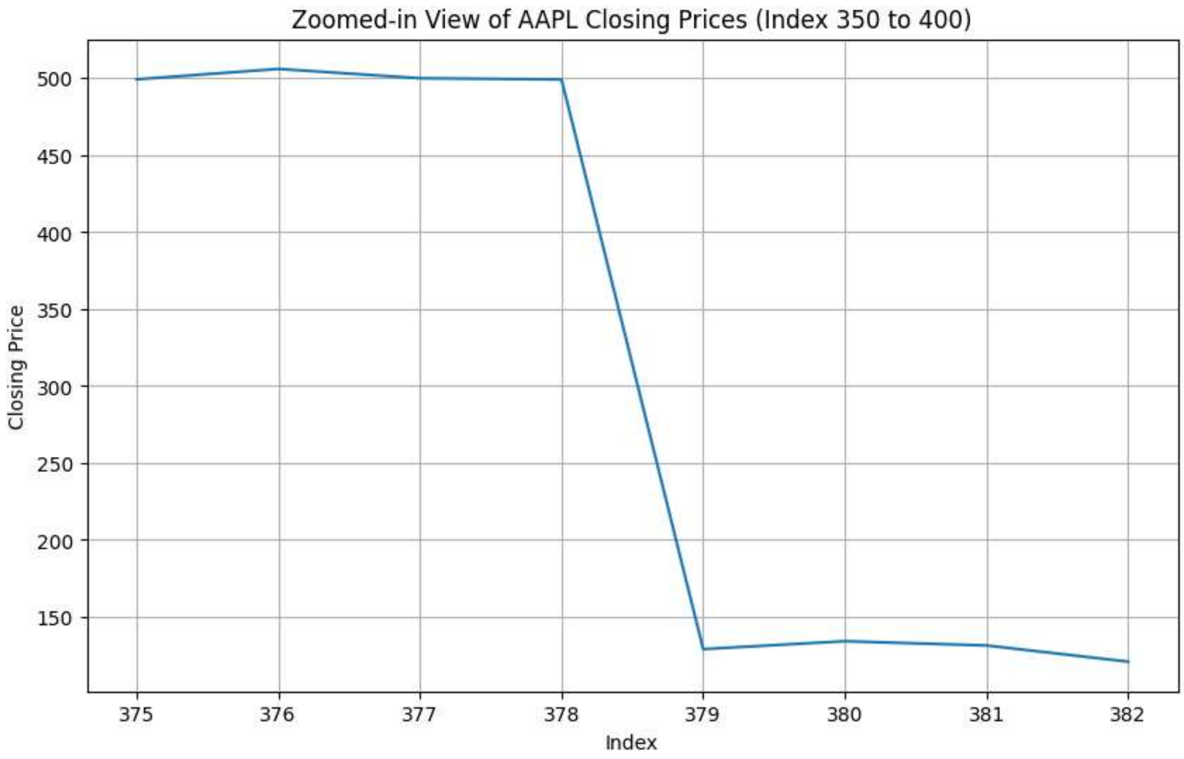
```
subset_df1 = df1[350:401]

# Plot the subset
plt.figure(figsize=(10, 6))
plt.plot(subset_df1)
plt.title('Zoomed-in View of AAPL Closing Prices (Index 350 to 400)')
plt.xlabel('Index')
plt.ylabel('Closing Price')
plt.grid(True)
plt.show()
```



In [22]:
```
subset_df1 = df1[375:383]

# Plot the subset
plt.figure(figsize=(10, 6))
plt.plot(subset_df1)
plt.title('Zoomed-in View of AAPL Closing Prices (Index 350 to 400)')
plt.xlabel('Index')
plt.ylabel('Closing Price')
plt.grid(True)
plt.show()
```



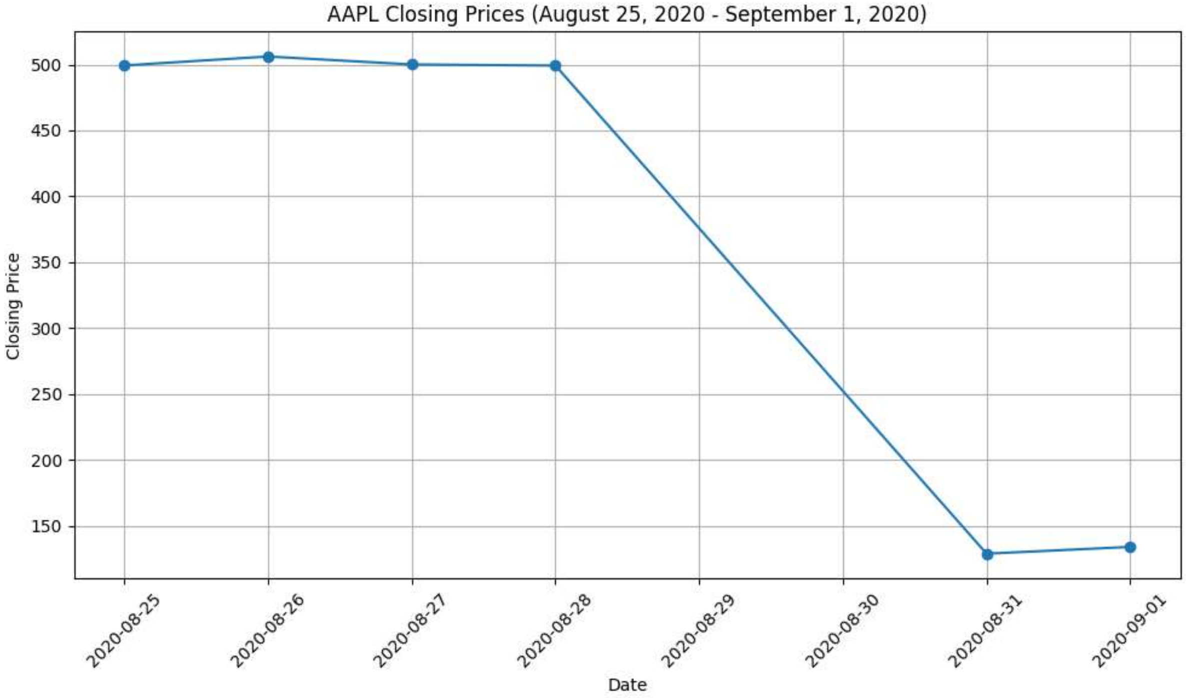
In [23]:

```

df['date'] = pd.to_datetime(df['date'])


subset_df = df[(df['date'] >= '2020-08-25') & (df['date'] <= '2020-09-01')]

plt.figure(figsize=(10, 6))
plt.plot(subset_df['date'], subset_df['close'], marker='o')
plt.title('AAPL Closing Prices (August 25, 2020 - September 1, 2020)')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [24]:

```
import numpy as np
```

In [25]:

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

In [26]:

```
df1.shape
```

Out[26]:

```
(1257, 1)
```

In [27]:

```
df1
```

Out[27]:

```
array([[0.17064496],
       [0.17284909],
       [0.17204759],
       ...,
       [0.18955542],
       [0.18614903],
       [0.18983093]])
```

In [28]:

```
training_size= int(len(df1)*0.65)
test_size = len(df1) - training_size
train_data, test_data = df1[0:training_size,:], df1[training_size:len(df1), :1
```

In [30]:

```
training_size, test_size
```

Out[30]:

```
(817, 440)
```

In [31]:

```

import numpy as np
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)
```

In [32]:

```
time_step = 100
X_train , Y_train = create_dataset(train_data, time_step)
X_test , Y_test = create_dataset(test_data, time_step)
```

In [33]:

```
print(X_train)
```

```
[[0.17064496 0.17284909 0.17204759 ... 0.23982467 0.25142142 0.25547902]
 [0.17284909 0.17204759 0.16951785 ... 0.25142142 0.25547902 0.25505322]
 [0.17204759 0.16951785 0.16445836 ... 0.25547902 0.25505322 0.25092048]
 ...
 [0.18146525 0.18169067 0.17873513 ... 0.10619912 0.08510958 0.07641828]
 [0.18169067 0.17873513 0.17715717 ... 0.08510958 0.07641828 0.07701941]
 [0.17873513 0.17715717 0.18827802 ... 0.07641828 0.07701941 0.09084534]]
```

In [36]:

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

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

Out[36]:

```
(None, None)
```

In [37]:

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

```
(339, 100)
(339,)
```

Out[37]:

```
(None, None)
```

In [38]:

```
#reshape input to be [sample , time steps , featurens]. which is required for L
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 [39]:

```
#create stack LSTM model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

In [40]:

```
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')
```

In [41]:

```
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)
_________________________________________________________________
```

In [42]:

```
model.fit(X_train,Y_train,validation_data = (X_test,Y_test),epochs = 100,batch_
```

```
Epoch 1/100
12/12 [==============================] - 10s 290ms/step - loss: 0.0352 - va
l_loss: 0.0012
Epoch 2/100
12/12 [==============================] - 2s 170ms/step - loss: 0.0165 - val
_loss: 0.0028
Epoch 3/100
12/12 [==============================] - 3s 268ms/step - loss: 0.0101 - val
_loss: 3.4270e-04
Epoch 4/100
12/12 [==============================] - 2s 172ms/step - loss: 0.0090 - val
_loss: 6.7223e-04
Epoch 5/100
12/12 [==============================] - 2s 171ms/step - loss: 0.0083 - val
_loss: 7.2278e-04
Epoch 6/100
12/12 [==============================] - 2s 170ms/step - loss: 0.0075 - val
_loss: 2.6384e-04
Epoch 7/100
```

In [43]:

```
import tensorflow as tf
```

In [44]:

```
tf.__version__
```

Out[44]:

```
'2.15.0'
```

In [45]:

```
#executing and checking the performance mertrics
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
```

```
23/23 [==============================] - 3s 58ms/step
11/11 [==============================] - 1s 88ms/step
```

In [47]:

```
#Transform back to the original form
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
```

In [48]:

```
#calculate the RMSE performance
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(Y_train,train_predict))
```

Out[48]:

```
223.49725799435646
```

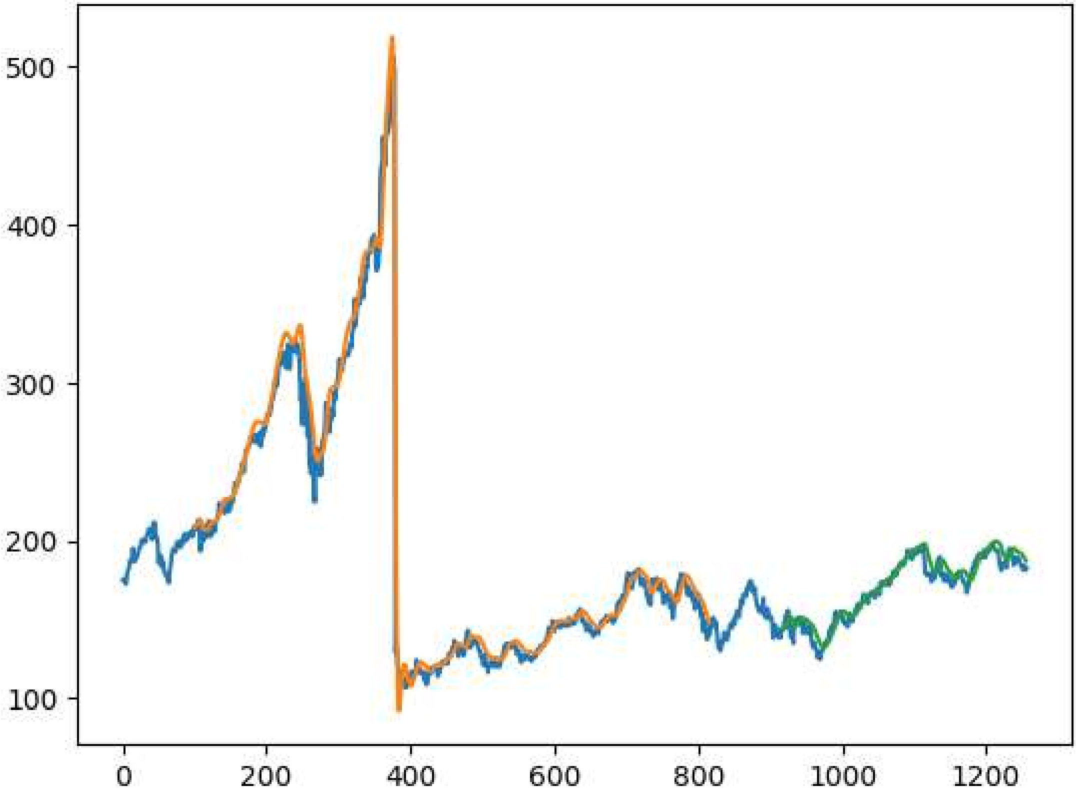
In [49]:

```
#test data RMSE
math.sqrt(mean_squared_error(Y_test,test_predict))
```

Out[49]:

```
173.02751900860065
```

In [52]:
```
###plotting
#shift train predictions for plotting
look_back = 100
trainPredictPlot = np.empty_like(df1)
trainPredictPlot[:,:] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
#shift test prediction for plotting
testPredictPlot = np.empty_like(df1)
testPredictPlot[:,:] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df1)-1, :] = test_predic
#plot baseline and predictions
plt.plot(scaler.inverse_transform(df1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



In [53]:
```
len(test_data)
```

Out[53]:
```
440
```

In [54]:
```
x_input = test_data[340:].reshape(1,-1)
x_input.shape
```

Out[54]:
```
(1, 100)
```

In [55]:
```
temp_input = list(x_input)
temp_input = temp_input[0].tolist()
```

In [56]:

```
temp_input
```

```
Out[56]: [0.1673638071383844,
           0.17049467752035058,
           0.17695679398872882,
           0.18071383844708827,
           0.17921102066374445,
           0.18274264245460237,
           0.1850219160926737,
           0.18036318096430803,
           0.18003757044458357,
           0.17610519724483403,
           0.17282404508453347,
           0.17187226048841575,
           0.1654101440200375,
           0.1657107075767063,
           0.1668127739511584,
           0.16095178459611764,
           0.15040701314965554,
           0.1537382592360676,
           0.15892298058860355,
           0.16012523481527863,
           0.16814026299311202,
           0.17715716969317463,
           0.17485284909204757,
           0.1813149655604257,
           0.18780212899185966,
           0.19048215403882274,
           0.1892798998121477,
           0.19927363807138382,
           0.19526612398246712,
           0.20187852222917968,
           0.20330619912335623,
           0.2075641828428303,
           0.2075140889167188,
           0.21192235441452717,
           0.20989355040701307,
           0.211571696931747,
           0.20821540388227922,
           0.2077645585472761,
           0.2092924232936756,
           0.20671258609893545,
           0.20816530995616772,
           0.2113963681903569,
           0.20686286787726987,
           0.21685660613650587,
           0.21410144020037564,
           0.21898559799624295,
           0.22259236067626798,
           0.21625547902316838,
           0.22008766437069505,
           0.2282279273638071,
           0.22860363180964305,
           0.2272510958046336,
           0.22304320601127103,
           0.22567313713212267,
           0.2203882279273638,
           0.22001252348152783,
           0.21730745147150904,
 0.21592986850344392,
 0.21618033813400123,
 0.2172573575453976,
 0.21462742642454596,
 0.19737006887914832,
 0.19388854101440195,
 0.18802755165936125,
 0.18619912335629302,
 0.1971696931747025,
 0.19611772072636185,
 0.1987476518472135,
 0.19724483406386972,
 0.19807138384470874,
 0.19233562930494674,
 0.18995616781465247,
 0.20485911083281144,
 0.21219787100814025,
 0.21803381340012518,
 0.2212648716343143,
 0.21956167814652466,
 0.2187351283656856,
 0.21435190983093294,
 0.2126236693800876,
 0.2033813400125234,
 0.1942642454602379,
 0.2004257983719474,
 0.19789605510331865,
 0.20247964934251717,
 0.20653725735754536,
 0.20681277395115838,
 0.20408265497808387,
 0.2054101440200375,
 0.20115216030056354,
 0.19586725109580455,
 0.1936380713838447,
 0.19291170945522856,
 0.18902943018159046,
 0.18715090795241074,
 0.18905447714464613,
 0.19418910457107075,
 0.1895554164057608,
 0.18614902943018152,
 0.18983093299937376]
```

In [57]:

```
# demonstrate prediction for next 10 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)
```

```
[0.19933964]
101
1 day input [0.17049468 0.17695679 0.18071384 0.17921102 0.18274264 0.18502
192
 0.18036318 0.18003757 0.1761052  0.17282405 0.17187226 0.16541014
 0.16571071 0.16681277 0.16095178 0.15040701 0.15373826 0.15892298
 0.16012523 0.16814026 0.17715717 0.17485285 0.18131497 0.18780213
 0.19048215 0.1892799  0.19927364 0.19526612 0.20187852 0.2033062
 0.20756418 0.20751409 0.21192235 0.20989355 0.2115717  0.2082154
 0.20776456 0.20929242 0.20671259 0.20816531 0.21139637 0.20686287
 0.21685661 0.21410144 0.2189856  0.22259236 0.21625548 0.22008766
 0.22822793 0.22860363 0.2272511  0.22304321 0.22567314 0.22038823
 0.22001252 0.21730745 0.21592987 0.21618034 0.21725736 0.21462743
 0.19737007 0.19388854 0.18802755 0.18619912 0.19716969 0.19611772
 0.19874765 0.19724483 0.19807138 0.19233563 0.18995617 0.20485911
 0.21219787 0.21803381 0.22126487 0.21956168 0.21873513 0.21435191
 0.21262367 0.20338134 0.19426425 0.2004258  0.19789606 0.20247965
 0.20653726 0.20681277 0.20408265 0.20541014 0.20115216 0.19586725
 0.19363807 0.19291171 0.18902943 0.18715091 0.18905448 0.1941891
 0.18955542 0.18614903 0.18983903 0.19933964]
```

In [58]:
```
day_new = np.arange(1,101)
day_pred = np.arange(101,131)
```

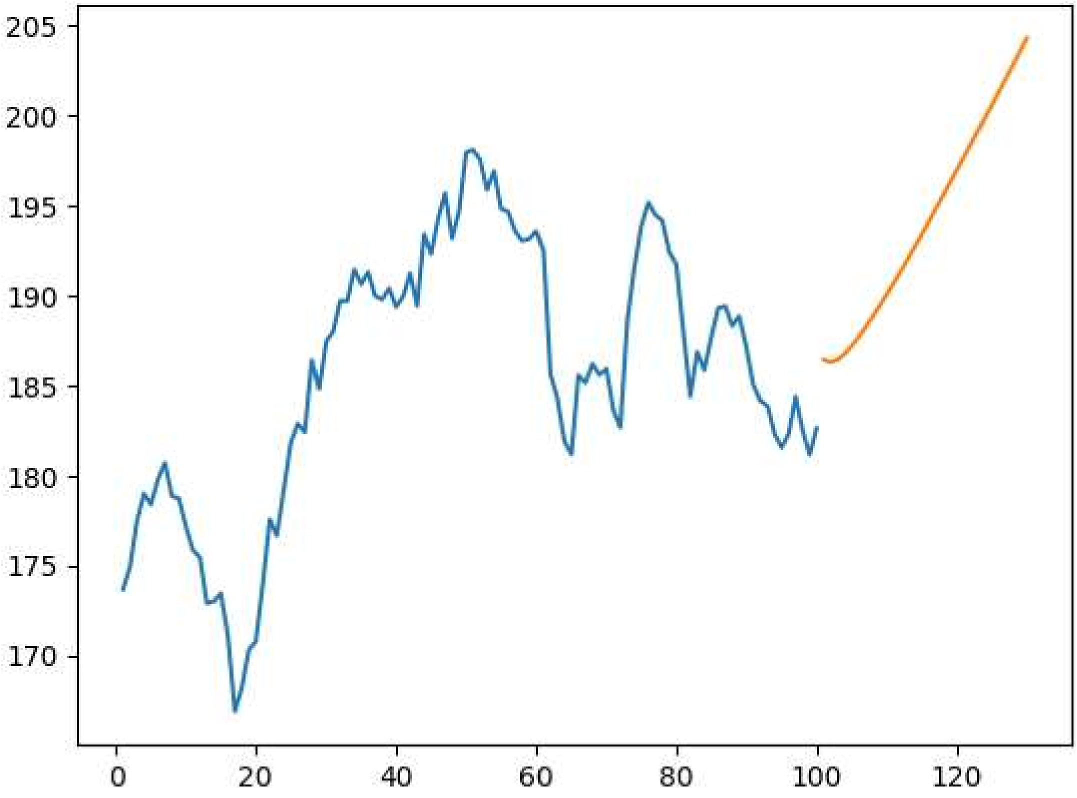
In [59]:
```
len(df1)
```

Out[59]:
```
1257
```

In [61]:
```
df3 = df1.tolist()
df3.extend(lst_output)
```

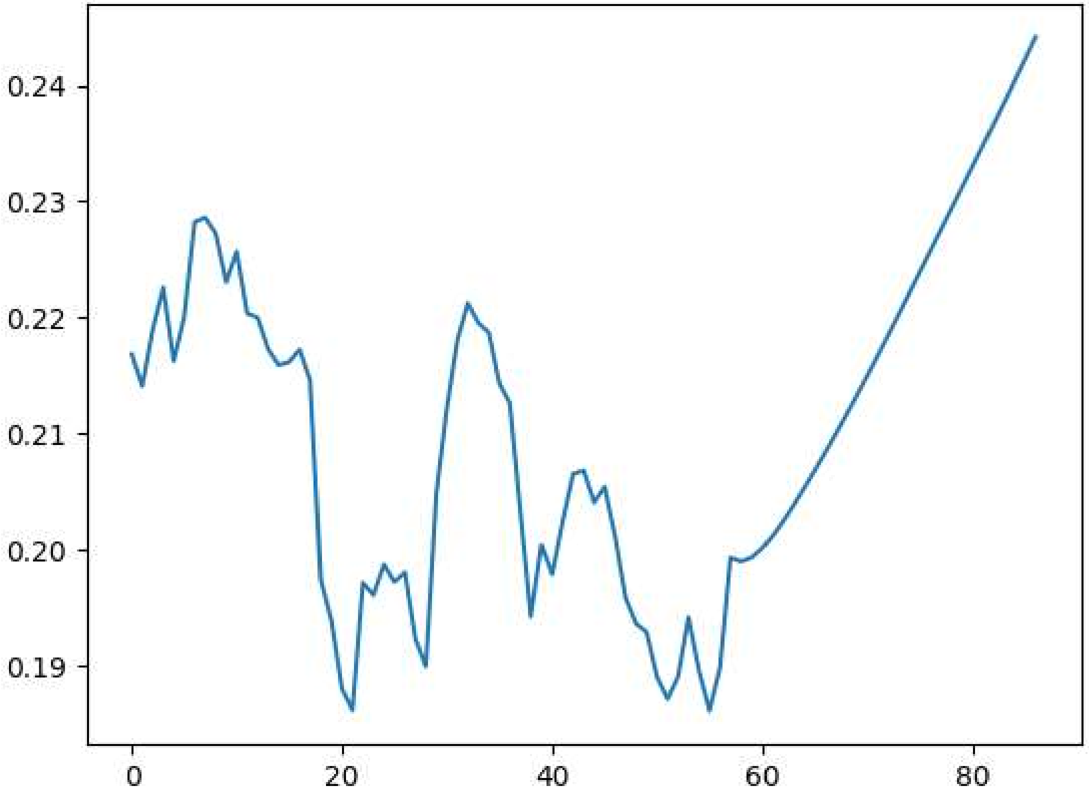
In [62]:
```
plt.plot(day_new,scaler.inverse_transform(df1[1157:]))
plt.plot(day_pred,scaler.inverse_transform(lst_output))
```

Out[62]:
```
[<matplotlib.lines.Line2D at 0x7bcdab79e740>]
```



```
In [64]: plt.plot(df3[1200:])

Out[64]: [<matplotlib.lines.Line2D at 0x7bcdab79fc70>]
```



```
In [ ]:
```