```
# load data from yahoo finnce for microsoft
import pandas as pd
df = pd.read_csv('sample_data/MSFT.csv')
df.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2017-01-18	62.669998	62.700001	62.119999	62.500000	58.111137	19670100
1	2017-01-19	62.240002	62.980000	62.200001	62.299999	57.925179	18451700
2	2017-01-20	62.669998	62.820000	62.369999	62.740002	58.334286	30213500
3	2017-01-23	62.700001	63.119999	62.570000	62.959999	58.538826	23097600
4	2017-01-24	63.200001	63.740002	62.939999	63.520000	59.059505	24672900

```
# viewing current date
df.tail()
```

	Date	Open	High	Low	Close	Adj Close	Volume
1254	2022-01-10	309.489990	314.720001	304.690002	314.269989	314.269989	44289500
1255	2022-01-11	313.380005	316.609985	309.890015	314.980011	314.980011	29386800
1256	2022-01-12	319.670013	323.410004	317.079987	318.269989	318.269989	34372200
1257	2022-01-13	320.470001	320.880005	304.000000	304.799988	304.799988	45366000
1258	2022-01-14	304.250000	310.820007	303.750000	310.200012	310.200012	39823500

```
# we are going to use the column close for our stacked LSTM time series forecasting
df = df.reset_index()['Close']
df.shape
```

```
(1259,)
```

```
# plotting the dataset
import matplotlib.pyplot as plt
plt.plot(df)
```

```
[<matplotlib.lines.Line2D at 0x7fe568c43450>]
```


```
# LSTM are sensitive to the scale of the data so we apply MinMax scaler
import numpy as np
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range = (0, 1))
df1 = scaler.fit_transform(np.array(df).reshape(-1, 1))
df1.shape
```

```
(1259, 1)
```

```
# splitting dataset into train and test data
# we are going to use a different approach since this is a time series prediction. The data i
training_size = int(len(df1)*0.65)     # data is split into 65% train and 35% test data
test_size = len(df1) - training_size
train_data, test_data = df1[0:training_size,:], df1[training_size:len(df1),:1]
```

```
training_size, test_size
```

```
(818, 441)
```

```
# converting the array values into dataset matrix. By so doing we are creating our dependent
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)
```

```
# reshape into X = t, t + 1, t + 2, t + 3 and y = t + y but here we are doing a 100 steps
time_step = 100
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)
```

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

```
(717, 100)
(717,)
(None, None)
```

```
# reshaping input to be [sample, time_step, features] which is required for LSTM. We are maki
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: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

```
# model fitting for training
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs = 100, batch_size= 64, v
```

```
Epoch 74/100
12/12 [==============================] - 2s 140ms/step - loss: 9.9701e-05 - val_lo
Epoch 75/100
12/12 [==============================] - 2s 139ms/step - loss: 8.4664e-05 - val_lo
Epoch 76/100
12/12 [==============================] - 2s 138ms/step - loss: 8.9422e-05 - val_lo
Epoch 77/100
12/12 [==============================] - 2s 139ms/step - loss: 1.1212e-04 - val_lo
Epoch 78/100
12/12 [==============================] - 2s 137ms/step - loss: 1.2591e-04 - val_lo
Epoch 79/100
12/12 [==============================] - 2s 138ms/step - loss: 9.8966e-05 - val_lo
Epoch 80/100
12/12 [==============================] - 2s 136ms/step - loss: 1.0590e-04 - val_lo
Epoch 81/100
```

```
12/12 [==============================] - 2s 136ms/step - loss: 9.5559e-05 - val_lo
Epoch 82/100
12/12 [==============================] - 2s 137ms/step - loss: 9.6459e-05 - val_lo
Epoch 83/100
12/12 [==============================] - 2s 137ms/step - loss: 9.1205e-05 - val_lo
Epoch 84/100
12/12 [==============================] - 2s 140ms/step - loss: 8.2973e-05 - val_lo
Epoch 85/100
12/12 [==============================] - 2s 139ms/step - loss: 8.1032e-05 - val_lo
Epoch 86/100
12/12 [==============================] - 2s 135ms/step - loss: 7.7550e-05 - val_lo
Epoch 87/100
12/12 [==============================] - 2s 138ms/step - loss: 1.0194e-04 - val_lo
Epoch 88/100
12/12 [==============================] - 2s 138ms/step - loss: 9.0021e-05 - val_lo
Epoch 89/100
12/12 [==============================] - 2s 137ms/step - loss: 8.4112e-05 - val_lo
Epoch 90/100
12/12 [==============================] - 2s 139ms/step - loss: 7.8377e-05 - val_lo
Epoch 91/100
12/12 [==============================] - 2s 135ms/step - loss: 7.9667e-05 - val_lo
Epoch 92/100
12/12 [==============================] - 2s 140ms/step - loss: 8.7688e-05 - val_lo
Epoch 93/100
12/12 [==============================] - 2s 137ms/step - loss: 7.8642e-05 - val_lo
Epoch 94/100
12/12 [==============================] - 2s 138ms/step - loss: 8.6807e-05 - val_lo
Epoch 95/100
12/12 [==============================] - 2s 138ms/step - loss: 1.1788e-04 - val_lo
Epoch 96/100
12/12 [==============================] - 2s 138ms/step - loss: 8.2956e-05 - val_lo
Epoch 97/100
12/12 [==============================] - 2s 138ms/step - loss: 8.9248e-05 - val_lo
Epoch 98/100
12/12 [==============================] - 2s 138ms/step - loss: 1.0453e-04 - val_lo
Epoch 99/100
12/12 [==============================] - 2s 139ms/step - loss: 7.9692e-05 - val_lo
Epoch 100/100
12/12 [==============================] - 2s 139ms/step - loss: 8.9039e-05 - val_lo
<keras.callbacks.History at 0x7fe4ee922ad0>
```

```
# we do our prediction and performance metrics
import tensorflow as tf
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
```

```
# we transform our X features back to their original form using the MinMaxScaler inverse
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
```

```
# calculating the RMSE for performance metrice with train predict
from sklearn.metrics import mean_squared_error
import math
math.sqrt(mean_squared_error(y_train, train_predict))
```

```
116.00648639670915
```

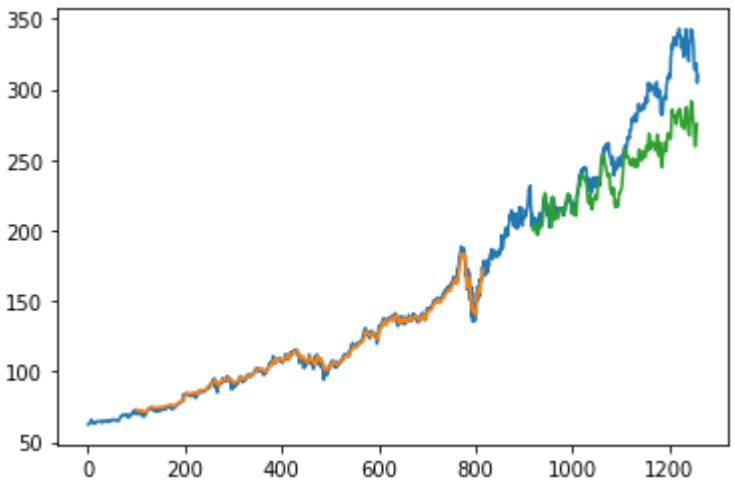
```
# RMSE for test predict
math.sqrt(mean_squared_error(y_test, test_predict))
```

```
241.27221470769246
```

```
# plotting
# shift train prediction for plotting
look_back = 100
trainPredict = np.empty_like(df1)
trainPredict[:, :] = np.nan
trainPredict[look_back:len(train_predict) + look_back, :] = train_predict

# shift test prediction for plotting
testPredict = np.empty_like(df1)
testPredict[:, :] = np.nan
testPredict[len(train_predict) + (look_back * 2) + 1 : len(df1) - 1, :] = test_predict

# plotting baseline and predictions. The green line representing the predicted test data, Blu
plt.plot(scaler.inverse_transform(df1))
plt.plot(trainPredict)
plt.plot(testPredict)
plt.show()
```




```
# predicting into the future. We will be doing short and long term
# we will take 100 days as the cap of days we can predict into
len(test_data)   # find the size of our test data
```

```
441
```

```
# let's take the first 100
x_input = test_data[341:].reshape(1, -1)
```

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

```
# Reshaping input data and predicting
from numpy import array
lst_output = []
n_steps = 100
i = 0
while (i < 30):
  if (len(temp_input) > 100):
    x_input = np.array(temp_input[1:])   # shifting position to 1
    print("{} day input {}".format(i, x_input))
    x_input = x_input.reshape(1, -1)
    x_input = x_input.reshape((1, n_steps, 1))
    yhat = model.predict(x_input, verbose=0)
    print("{} day output {}".format(i, yhat))
    temp_input.extend(yhat[0].tolist())
    temp_input = temp_input[1:]
    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)
```

```
27 day input [0.78633238 0.80645284 0.82194365 0.82814009 0.82814009 0.82593221
 0.82112467 0.83333931 0.85627297 0.86147218 0.87244052 0.87578799
 0.87286784 0.88479763 0.87909981 0.87543185 0.88248281 0.92899123
 0.93319333 0.95908269 0.95107015 0.96446002 0.96755819 0.97624734
 0.9748941  0.97820592 0.97450243 0.95616254 0.96196719 0.97724446
 0.97492974 0.98718003 0.98579114 0.99344754 1.         0.98831951
 0.98066311 0.98148221 0.95217409 0.97692397 0.95541473 0.95359852
 0.95149747 0.92842144 0.93974579 0.97083447 0.97101248 0.96435319
 0.99797024 0.98678825 0.94740219 0.9698729  0.93515191 0.93123465
 0.91738192 0.94366306 0.96470933 0.97001537 0.99764975 0.99337636
 0.99586919 0.98650341 0.97582003 0.97022903 0.9497882  0.90481115
 0.89590833 0.89647813 0.89729711 0.89982559 0.91154162 0.86357324
 0.88280341 0.72162986 0.61623812 0.50051391 0.41371679 0.36756322
 0.35524639 0.3600902  0.36205426 0.34223384 0.29698128 0.24892293
 0.22074546 0.21219905 0.2160362  0.22795361 0.24530023 0.26601014
 0.28832319 0.31069398 0.3317841  0.35051063 0.36611572 0.37823218
```

```
 0.38691884 0.3926411  0.39618686 0.39852914]
27 day output [[0.40066522]]
28 day input [0.80645284 0.82194365 0.82814009 0.82814009 0.82593221 0.82112467
 0.83333931 0.85627297 0.86147218 0.87244052 0.87578799 0.87286784
 0.88479763 0.87909981 0.87543185 0.88248281 0.92899123 0.93319333
 0.95908269 0.95107015 0.96446002 0.96755819 0.97624734 0.9748941
 0.97820592 0.97450243 0.95616254 0.96196719 0.97724446 0.97492974
 0.98718003 0.98579114 0.99344754 1.         0.98831951 0.98066311
 0.98148221 0.95217409 0.97692397 0.95541473 0.95359852 0.95149747
 0.92842144 0.93974579 0.97083447 0.97101248 0.96435319 0.99797024
 0.98678825 0.94740219 0.9698729  0.93515191 0.93123465 0.91738192
 0.94366306 0.96470933 0.97001537 0.99764975 0.99337636 0.99586919
 0.98650341 0.97582003 0.97022903 0.9497882  0.90481115 0.89590833
 0.89647813 0.89729711 0.89982559 0.91154162 0.86357324 0.88280341
 0.72162986 0.61623812 0.50051391 0.41371679 0.36756322 0.35524639
 0.3600902  0.36205426 0.34223384 0.29698128 0.24892293 0.22074546
 0.21219905 0.2160362  0.22795361 0.24530023 0.26601014 0.28832319
 0.31069398 0.3317841  0.35051063 0.36611572 0.37823218 0.38691884
 0.3926411  0.39618686 0.39852914 0.40066522]
28 day output [[0.40347376]]
29 day input [0.82194365 0.82814009 0.82814009 0.82593221 0.82112467 0.83333931
 0.85627297 0.86147218 0.87244052 0.87578799 0.87286784 0.88479763
 0.87909981 0.87543185 0.88248281 0.92899123 0.93319333 0.95908269
 0.95107015 0.96446002 0.96755819 0.97624734 0.9748941  0.97820592
 0.97450243 0.95616254 0.96196719 0.97724446 0.97492974 0.98718003
 0.98579114 0.99344754 1.         0.98831951 0.98066311 0.98148221
 0.95217409 0.97692397 0.95541473 0.95359852 0.95149747 0.92842144
 0.93974579 0.97083447 0.97101248 0.96435319 0.99797024 0.98678825
 0.94740219 0.9698729  0.93515191 0.93123465 0.91738192 0.94366306
 0.96470933 0.97001537 0.99764975 0.99337636 0.99586919 0.98650341
 0.97582003 0.97022903 0.9497882  0.90481115 0.89590833 0.89647813
 0.89729711 0.89982559 0.91154162 0.86357324 0.88280341 0.72162986
 0.61623812 0.50051391 0.41371679 0.36756322 0.35524639 0.3600902
 0.36205426 0.34223384 0.29698128 0.24892293 0.22074546 0.21219905
 0.2160362  0.22795361 0.24530023 0.26601014 0.28832319 0.31069398
 0.3317841  0.35051063 0.36611572 0.37823218 0.38691884 0.3926411
 0.39618686 0.39852914 0.40066522 0.40347376]
29 day output [[0.4076126]]
[[0.7216298580169678], [0.6162381172180176], [0.5005139112472534], [0.413716793060
```

```
# plotting
# identifying which range to plot with our 100 days
day_new = np.arange(1, 101)
day_pred = np.arange(101, 131)
```

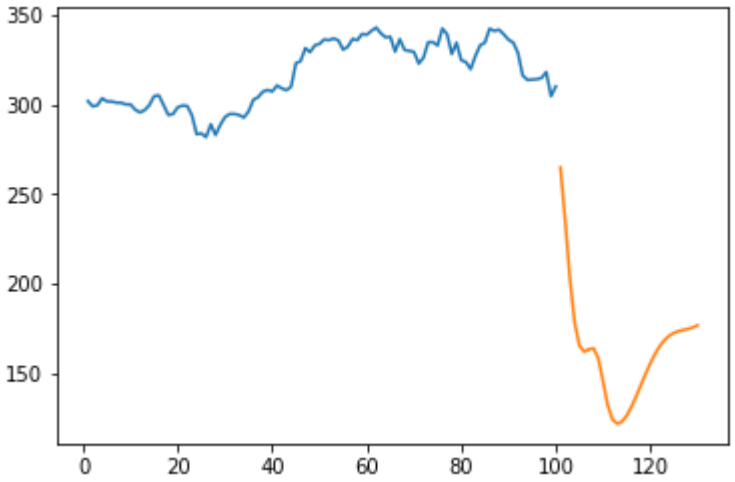
```
# find the size of the dataset in general
len(df1)
```

```
1259
```

```
# convert df1 to list and extend to 1st_output
df2 = df1.tolist()
df2.extend(lst_output)
```

```
# taking of 100 days from the total numbers of days for the dataset
plt.plot(day_new, scaler.inverse_transform(df1[1159:]))   # 100 days
plt.plot(day_pred, scaler.inverse_transform(lst_output))
```

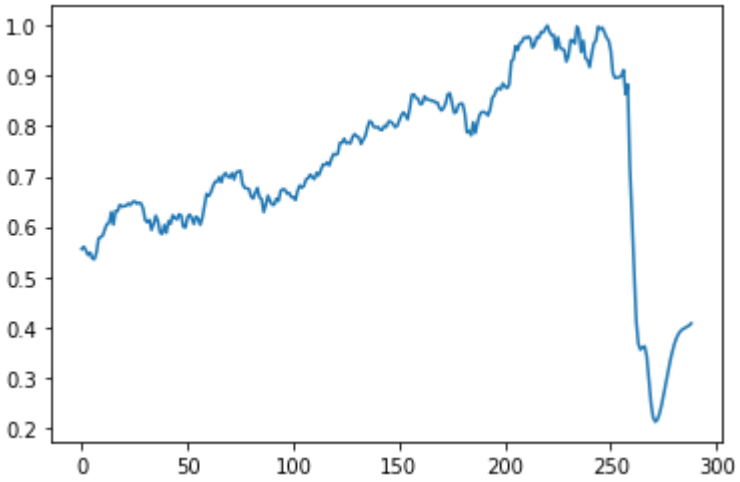
```
[<matplotlib.lines.Line2D at 0x7fe4eaa6c390>]
```



- The orange linge line is the future 30 days prediction

```
# complete output
df3 = df1.tolist()
df3.extend(lst_output)
plt.plot(df3[1000:])
```

```
[<matplotlib.lines.Line2D at 0x7fe4eaa66950>]
```




```
# prediction for the next 60 days
# Reshaping input data and predicting
from numpy import array
```

```
output = []
n_steps = 100
i = 0
while (i < 60):
  if (len(temp_input) > 100):
    x_input = np.array(temp_input[1:])   # shifting position to 1
    print("{} day input {}".format(i, x_input))
    x_input = x_input.reshape(1, -1)
    x_input = x_input.reshape((1, n_steps, 1))
    yhat = model.predict(x_input, verbose=0)
    print("{} day output {}".format(i, yhat))
    temp_input.extend(yhat[0].tolist())
    temp_input = temp_input[1:]
    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))
    output.extend(yhat.tolist())
    i = i + 1
print(output)
```

```
57 day input [0.99586919 0.98650341 0.97582003 0.97022903 0.9497882  0.90481115
 0.89590833 0.89647813 0.89729711 0.89982559 0.91154162 0.86357324
 0.88280341 0.72162986 0.61623812 0.50051391 0.41371679 0.36756322
 0.35524639 0.3600902  0.36205426 0.34223384 0.29698128 0.24892293
 0.22074546 0.21219905 0.2160362  0.22795361 0.24530023 0.26601014
 0.28832319 0.31069398 0.3317841  0.35051063 0.36611572 0.37823218
 0.38691884 0.3926411  0.39618686 0.39852914 0.40066522 0.40347376
 0.40761259 0.41347146 0.42115927 0.43050814 0.44108087 0.45218146
 0.46288151 0.47208312 0.4786393  0.48152891 0.48005831 0.47403777
 0.46387783 0.4505662  0.43551132 0.42027697 0.40627018 0.39446795
 0.38525462 0.37840456 0.37317896 0.36848825 0.36306617 0.35564074
 0.34510052 0.3306914  0.31226256 0.29055506 0.26740146 0.24553785
 0.22781397 0.21613327 0.21090618 0.21134549 0.2161303  0.22389011
 0.23341681 0.24372745 0.25403342 0.2637715  0.27255258 0.28016612
 0.28655574 0.29178286 0.29600343 0.29943126 0.30229801 0.30482423
 0.3072032  0.30958134 0.31205252 0.31466228 0.31740966 0.32025662
 0.32313776 0.32596934 0.32865873 0.3311128 ]
57 day output [[0.33324414]]
58 day input [0.98650341 0.97582003 0.97022903 0.9497882  0.90481115 0.89590833
 0.89647813 0.89729711 0.89982559 0.91154162 0.86357324 0.88280341
 0.72162986 0.61623812 0.50051391 0.41371679 0.36756322 0.35524639
 0.3600902  0.36205426 0.34223384 0.29698128 0.24892293 0.22074546
 0.21219905 0.2160362  0.22795361 0.24530023 0.26601014 0.28832319
 0.31069398 0.3317841  0.35051063 0.36611572 0.37823218 0.38691884
 0.3926411  0.39618686 0.39852914 0.40066522 0.40347376 0.40761259
 0.41347146 0.42115927 0.43050814 0.44108087 0.45218146 0.46288151
 0.47208312 0.4786393  0.48152891 0.48005831 0.47403777 0.46387783
 0.4505662  0.43551132 0.42027697 0.40627018 0.39446795 0.38525462
 0.37840456 0.37317896 0.36848825 0.36306617 0.35564074 0.34510052
```

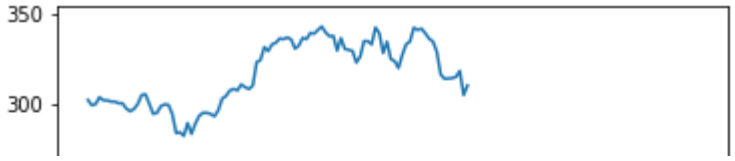
```
 0.3306914  0.31226256 0.29055506 0.26740146 0.24553785 0.22781397
 0.21613327 0.21090618 0.21134549 0.2161303  0.22389011 0.23341681
 0.24372745 0.25403342 0.2637715  0.27255258 0.28016612 0.28655574
 0.29178286 0.29600343 0.29943126 0.30229801 0.30482423 0.3072032
 0.30958134 0.31205252 0.31466228 0.31740966 0.32025662 0.32313776
 0.32596934 0.32865873 0.3311128  0.33324414]
58 day output [[0.33497798]]
59 day input [0.97582003 0.97022903 0.9497882  0.90481115 0.89590833 0.89647813
 0.89729711 0.89982559 0.91154162 0.86357324 0.88280341 0.72162986
 0.61623812 0.50051391 0.41371679 0.36756322 0.35524639 0.3600902
 0.36205426 0.34223384 0.29698128 0.24892293 0.22074546 0.21219905
 0.2160362  0.22795361 0.24530023 0.26601014 0.28832319 0.31069398
 0.3317841  0.35051063 0.36611572 0.37823218 0.38691884 0.3926411
 0.39618686 0.39852914 0.40066522 0.40347376 0.40761259 0.41347146
 0.42115927 0.43050814 0.44108087 0.45218146 0.46288151 0.47208312
 0.4786393  0.48152891 0.48005831 0.47403777 0.46387783 0.4505662
 0.43551132 0.42027697 0.40627018 0.39446795 0.38525462 0.37840456
 0.37317896 0.36848825 0.36306617 0.35564074 0.34510052 0.3306914
 0.31226256 0.29055506 0.26740146 0.24553785 0.22781397 0.21613327
 0.21090618 0.21134549 0.2161303  0.22389011 0.23341681 0.24372745
 0.25403342 0.2637715  0.27255258 0.28016612 0.28655574 0.29178286
 0.29600343 0.29943126 0.30229801 0.30482423 0.3072032  0.30958134
 0.31205252 0.31466228 0.31740966 0.32025662 0.32313776 0.32596934
 0.32865873 0.3311128  0.33324414 0.33497798]
59 day output [[0.33625454]]
[[0.4134714603424072], [0.4211592674255371], [0.4305081367492676], [0.441080868244
```

```
# plotting
# identifying which range to plot with our 100 days
day_new1 = np.arange(1, 101)
day_pred1 = np.arange(101, 161)
```

```
# convert df1 to list and extend to 1st_output
df2 = df1.tolist()
df2.extend(output)
```

```
# taking of 100 days from the total numbers of days for the dataset
plt.plot(day_new1, scaler.inverse_transform(df1[1159:]))   # 100 days
plt.plot(day_pred1, scaler.inverse_transform(output))
```

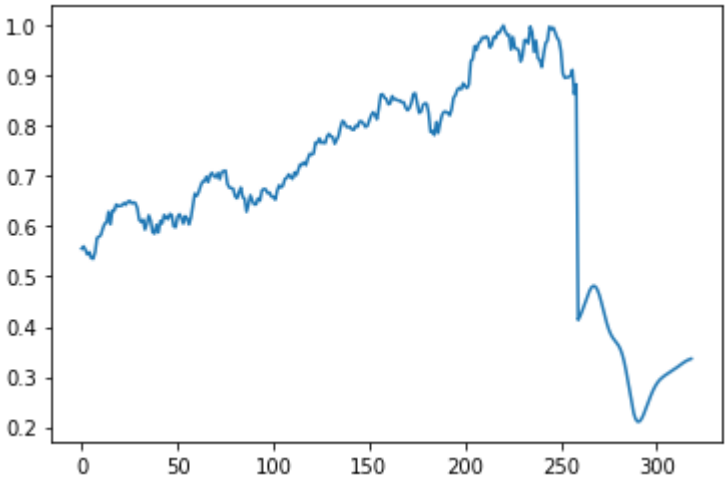
```
[<matplotlib.lines.Line2D at 0x7fe4ea88c990>]
```



- The orange line is the 60 days prediction.

```
# complete output
df3 = df1.tolist()
df3.extend(output)
plt.plot(df3[1000:])
```

```
[<matplotlib.lines.Line2D at 0x7fe4ea97f8d0>]
```



✓ 0s completed at 01:58