Stock prediction using LSTM

```
import pandas_datareader as pdr
key="0036233318361195d558408515d2f7360204d926"
```

```
#data collection from pandas datareader from Tiingo API
df=pdr.get_data_tiingo('GOOG',api_key=key)
```

```
<ipython-input-3-8c7baefdb045>:2: FutureWarning: In a future version of pandas all arguments of concat except for the argument 'obj
  df=pdr.get_data_tiingo('GOOG',api_key=key)
```

```
df.to_csv('GOOG.csv')
```

```
import pandas as pd
df=pd.read_csv('GOOG.csv')
```

```
df.head()
```

	symbol	date	close	high	low	open	volume	adjClose	adj
0	GOOG	2018-08-13 00:00:00+00:00	1235.01	1249.273	1233.641	1236.98	997346	61.7505	62.4
1	GOOG	2018-08-14 00:00:00+00:00	1242.10	1245.870	1225.110	1235.19	1348194	62.1050	62.2
2	GOOG	2018-08-15 00:00:00+00:00	1214.38	1235.240	1209.510	1229.26	1828814	60.7190	61.7

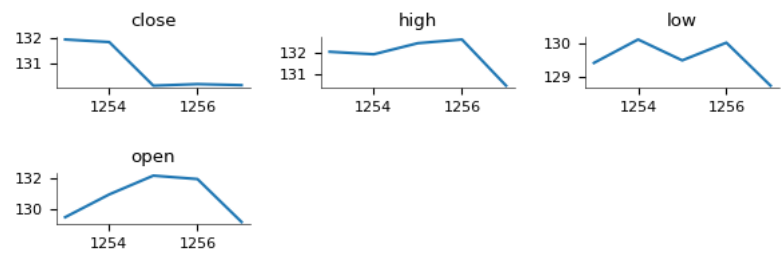
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   symbol       1258 non-null   object
 1   date         1258 non-null   object
 2   close        1258 non-null   float64
 3   high         1258 non-null   float64
 4   low          1258 non-null   float64
 5   open         1258 non-null   float64
 6   volume       1258 non-null   int64
 7   adjClose     1258 non-null   float64
 8   adjHigh      1258 non-null   float64
 9   adjLow       1258 non-null   float64
 10  adjOpen      1258 non-null   float64
 11  adjVolume    1258 non-null   int64
 12  divCash      1258 non-null   float64
 13  splitFactor  1258 non-null   float64
dtypes: float64(10), int64(2), object(2)
memory usage: 137.7+ KB
```

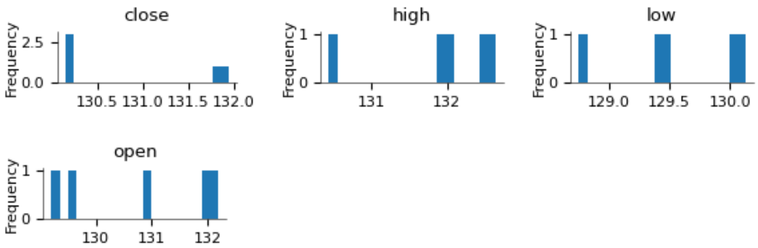
```
df.tail()
```

	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	a
1253	GOOG	2023-08-07 00:00:00+00:00	131.94	132.060	129.430	129.510	17621041	131.94	132.060	129.430	1
1254	GOOG	2023-08-08 00:00:00+00:00	131.84	131.940	130.130	130.980	16835952	131.84	131.940	130.130	1
1255	GOOG	2023-08-09 00:00:00+00:00	130.15	132.470	129.505	132.190	17745218	130.15	132.470	129.505	1
1256	GOOG	2023-08-10 00:00:00+00:00	130.21	132.647	130.035	131.970	17855681	130.21	132.647	130.035	1
1257	GOOG	2023-08-11 00:00:00+00:00	130.17	130.440	128.750	129.202	15205465	130.17	130.440	128.750	1

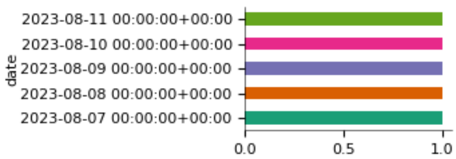
Values



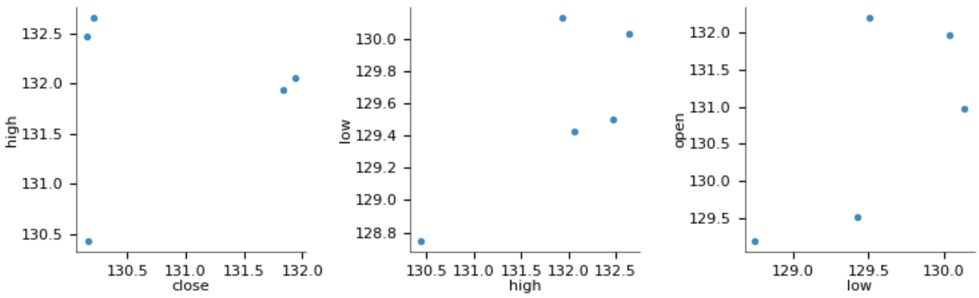
Distributions



Categorical distributions



2-d distributions



Faceted distributions

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

```
df1
```

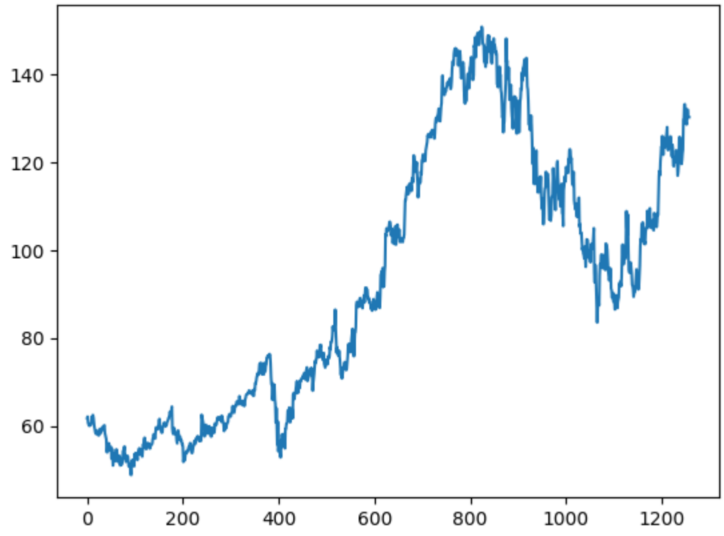
```
0        61.7505
1        62.1050
2        60.7190
3        60.3245
4        60.0480
           ...
1253    131.9400
1254    131.8400
1255    130.1500
1256    130.2100
1257    130.1700
Name: adjClose, Length: 1258, dtype: float64
```

```
df1.shape
```

```
(1258,)
```

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

```
[<matplotlib.lines.Line2D at 0x78d8e23f1f90>]
```




```
### 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(df1).reshape(-1,1))
```

```
df1
```

```
array([[0.12698483],
       [0.1304638 ],
       [0.11686196],
       ...,
       [0.79823942],
       [0.79882824],
       [0.79843569]])
```

```
##splitting dataset into train and test split
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]
```

```
training_size,test_size
```

```
(817, 441)
```

```
train_data
```

```
           [0.1013121 ],
           [0.10551238],
           [0.10466839],
           [0.08590453],
           [0.09454062],
           [0.0940254 ],
           [0.09558578],
           [0.10629747],
           [0.10396671],
           [0.09429528],
           [0.10068402],
           [0.1153899 ],
           [0.11222497],
           [0.11196981],
           [0.11275   ],
           [0.11970304],
           [0.12661191],
           [0.12921745],
           [0.12516438],
           [0.1241094 ],
           [0.12570904],
           [0.12880037],
           [0.12449214],
           [0.12650395],
           [0.11901117],
           [0.13263263],
           [0.13011541],
           [0.12211722],
           [0.11912893],
           [0.11230839],
           [0.09833853],
           [0.10383423],
           [0.11422207],
           [0.11357436],
           [0.10447212],
```

```
# convert an array of values into a dataset matrix
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=0,1,2,3,...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,y_test=create_dataset(test_data,time_step)
```

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

```
(716, 100)
(716,)
(None, None)
```

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

```
(340, 100)
(340,)
(None, None)
```

```
# reshape input to be [samples, time steps, features] which is 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)
```

```
### Create 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.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=100,batch_size=64,verbose=1)
```

```
Epoch 1/100
12/12 [==============================] - 11s 342ms/step - loss: 0.0458 - val_loss: 0.0212
Epoch 2/100
12/12 [==============================] - 3s 213ms/step - loss: 0.0086 - val_loss: 0.0163
Epoch 3/100
12/12 [==============================] - 3s 223ms/step - loss: 0.0026 - val_loss: 0.0085
Epoch 4/100
12/12 [==============================] - 3s 290ms/step - loss: 0.0020 - val_loss: 0.0044
Epoch 5/100
12/12 [==============================] - 3s 241ms/step - loss: 0.0015 - val_loss: 0.0040
Epoch 6/100
12/12 [==============================] - 3s 212ms/step - loss: 0.0014 - val_loss: 0.0063
Epoch 7/100
12/12 [==============================] - 3s 214ms/step - loss: 0.0013 - val_loss: 0.0081
Epoch 8/100
12/12 [==============================] - 3s 217ms/step - loss: 0.0014 - val_loss: 0.0057
Epoch 9/100
12/12 [==============================] - 4s 328ms/step - loss: 0.0013 - val_loss: 0.0050
Epoch 10/100
12/12 [==============================] - 3s 219ms/step - loss: 0.0012 - val_loss: 0.0052
Epoch 11/100
12/12 [==============================] - 3s 214ms/step - loss: 0.0013 - val_loss: 0.0052
Epoch 12/100
12/12 [==============================] - 3s 217ms/step - loss: 0.0012 - val_loss: 0.0057
Epoch 13/100
12/12 [==============================] - 3s 273ms/step - loss: 0.0012 - val_loss: 0.0054
Epoch 14/100
12/12 [==============================] - 3s 263ms/step - loss: 0.0012 - val_loss: 0.0052
Epoch 15/100
12/12 [==============================] - 3s 221ms/step - loss: 0.0012 - val_loss: 0.0063
Epoch 16/100
12/12 [==============================] - 3s 217ms/step - loss: 0.0012 - val_loss: 0.0058
Epoch 17/100
12/12 [==============================] - 3s 216ms/step - loss: 0.0013 - val_loss: 0.0059
Epoch 18/100
12/12 [==============================] - 4s 326ms/step - loss: 0.0012 - val_loss: 0.0050
Epoch 19/100
12/12 [==============================] - 3s 220ms/step - loss: 0.0011 - val_loss: 0.0042
Epoch 20/100
12/12 [==============================] - 3s 285ms/step - loss: 0.0011 - val_loss: 0.0056
Epoch 21/100
12/12 [==============================] - 4s 332ms/step - loss: 0.0011 - val_loss: 0.0040
Epoch 22/100
12/12 [==============================] - 4s 288ms/step - loss: 0.0012 - val_loss: 0.0038
Epoch 23/100
12/12 [==============================] - 3s 219ms/step - loss: 0.0011 - val_loss: 0.0037
Epoch 24/100
12/12 [==============================] - 3s 214ms/step - loss: 0.0011 - val_loss: 0.0038
Epoch 25/100
12/12 [==============================] - 3s 221ms/step - loss: 0.0011 - val_loss: 0.0047
Epoch 26/100
12/12 [==============================] - 4s 312ms/step - loss: 0.0012 - val_loss: 0.0044
Epoch 27/100
12/12 [==============================] - 3s 222ms/step - loss: 0.0010 - val_loss: 0.0035
Epoch 28/100
12/12 [==============================] - 3s 220ms/step - loss: 0.0011 - val_loss: 0.0034
Epoch 29/100
12/12 [==============================] - 3s 217ms/step - loss: 0.0011 - val_loss: 0.0045
```

```
import tensorflow as tf
```

```
tf.__version__
```

```
'2.12.0'
```

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

```
23/23 [==============================] - 4s 63ms/step
11/11 [==============================] - 1s 47ms/step
```

```
##Transformback to original form
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

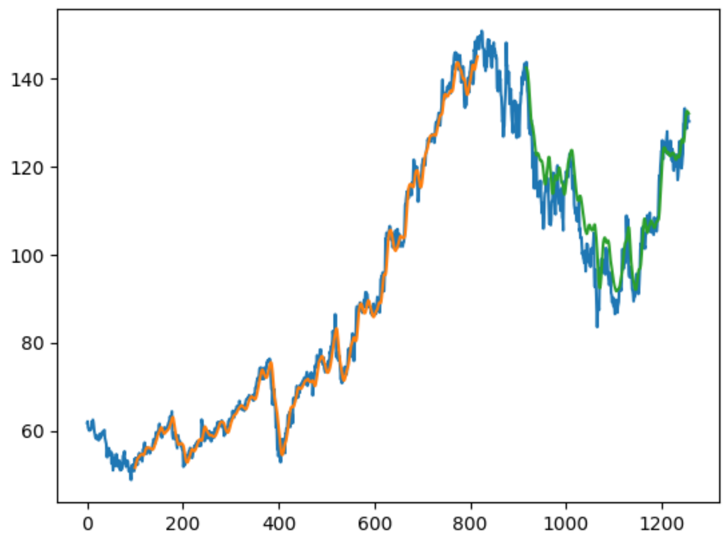
```
### Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

```
86.91708079182273
```

```
### Test Data RMSE
math.sqrt(mean_squared_error(y_test,test_predict))
```

```
112.024519641827
```

```
### Plotting
# 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)
```

```
441
```

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

```
(1, 101)
```

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

```
# 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 day input [0.55966751 0.54376926 0.56378928 0.56182653 0.53238533 0.51570198
 0.52100139 0.51530943 0.54161024 0.55054074 0.55260162 0.55093329
 0.58969754 0.57056076 0.56241536 0.55358299 0.58272979 0.59519323
 0.56535948 0.55260162 0.55162025 0.56025633 0.56035447 0.56889242
 0.54759662 0.54602642 0.58449626 0.5830242  0.5780192  0.56104143
 0.56241536 0.55348486 0.56334766 0.58322048 0.58027635 0.62286797
 0.66820742 0.67821743 0.66879625 0.69951324 0.71315433 0.73317435
 0.73052464 0.75623663 0.73091719 0.71472453 0.74131975 0.75191859
 0.74416573 0.73170229 0.74151603 0.74995584 0.76369507 0.77625665
 0.72748238 0.72483268 0.72679542 0.74131975 0.74210485 0.74161416
 0.75545153 0.73847377 0.73641288 0.71099531 0.73660916 0.72826748
 0.6896995  0.6889144  0.70922884 0.69872814 0.70814933 0.70412569
 0.72444013 0.70775678 0.70000393 0.66791301 0.67615655 0.69490078
 0.74603034 0.75456829 0.7482875  0.73867004 0.72591219 0.69401755
 0.70167226 0.71707982 0.72601032 0.79343069 0.79549157 0.8263067
 0.82728807 0.81531532 0.78342068 0.78469646 0.7824393  0.815806
 0.81482463 0.79823942 0.79882824 0.79843569]
0 day output [[0.8101604]]
1 day input [0.54376926 0.56378928 0.56182653 0.53238533 0.51570198 0.52100139
 0.51530943 0.54161024 0.55054074 0.55260162 0.55093329 0.58969754
 0.57056076 0.56241536 0.55358299 0.58272979 0.59519323 0.56535948
 0.55260162 0.55162025 0.56025633 0.56035447 0.56889242 0.54759662
 0.54602642 0.58449626 0.5830242  0.5780192  0.56104143 0.56241536
 0.55348486 0.56334766 0.58322048 0.58027635 0.62286797 0.66820742
 0.67821743 0.66879625 0.69951324 0.71315433 0.73317435 0.73052464
 0.75623663 0.73091719 0.71472453 0.74131975 0.75191859 0.74416573
 0.73170229 0.74151603 0.74995584 0.76369507 0.77625665 0.72748238
 0.72483268 0.72679542 0.74131975 0.74210485 0.74161416 0.75545153
 0.73847377 0.73641288 0.71099531 0.73660916 0.72826748 0.6896995
 0.6889144  0.70922884 0.69872814 0.70814933 0.70412569 0.72444013
 0.70775678 0.70000393 0.66791301 0.67615655 0.69490078 0.74603034
 0.75456829 0.7482875  0.73867004 0.72591219 0.69401755 0.70167226
 0.71707982 0.72601032 0.79343069 0.79549157 0.8263067  0.82728807
 0.81531532 0.78342068 0.78469646 0.7824393  0.815806   0.81482463
 0.79823942 0.79882824 0.79843569 0.8101604 ]
1 day output [[0.80800366]]
2 day input [0.56378928 0.56182653 0.53238533 0.51570198 0.52100139 0.51530943
 0.54161024 0.55054074 0.55260162 0.55093329 0.58969754 0.57056076
 0.56241536 0.55358299 0.58272979 0.59519323 0.56535948 0.55260162
 0.55162025 0.56025633 0.56035447 0.56889242 0.54759662 0.54602642
 0.58449626 0.5830242  0.5780192  0.56104143 0.56241536 0.55348486
 0.56334766 0.58322048 0.58027635 0.62286797 0.66820742 0.67821743
 0.66879625 0.69951324 0.71315433 0.73317435 0.73052464 0.75623663
 0.73091719 0.71472453 0.74131975 0.75191859 0.74416573 0.73170229
 0.74151603 0.74995584 0.76369507 0.77625665 0.72748238 0.72483268
 0.72679542 0.74131975 0.74210485 0.74161416 0.75545153 0.73847377
 0.73641288 0.71099531 0.73660916 0.72826748 0.6896995  0.6889144
 0.70922884 0.69872814 0.70814933 0.70412569 0.72444013 0.70775678
 0.70000393 0.66791301 0.67615655 0.69490078 0.74603034 0.75456829
 0.7482875  0.73867004 0.72591219 0.69401755 0.70167226 0.71707982
 0.72601032 0.79343069 0.79549157 0.8263067  0.82728807 0.81531532
 0.78342068 0.78469646 0.7824393  0.815806   0.81482463 0.79823942
```

```
 0.79882824 0.79843569 0.8101604  0.80800366]
2 day output [[0.8067274]]
3 day input [0.56182653 0.53238533 0.51570198 0.52100139 0.51530943 0.54161024
 0.55054074 0.55260162 0.55093329 0.58969754 0.57056076 0.56241536
```

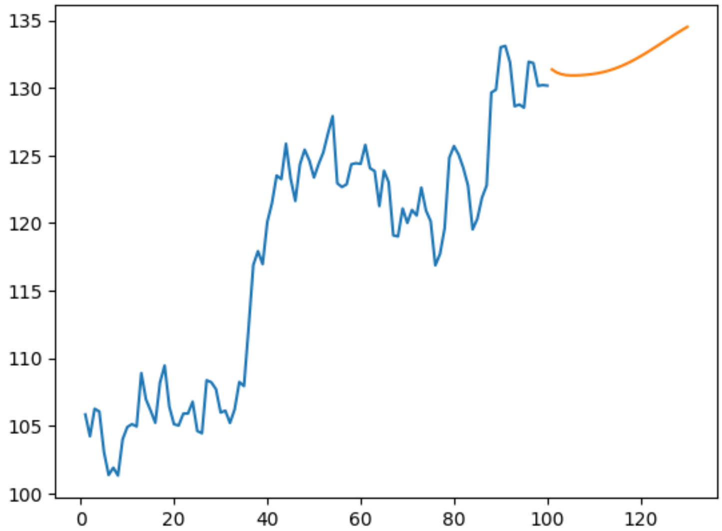
```
day_new=np.arange(1,101)
day_pred=np.arange(101,131)
```

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

```
1258
```

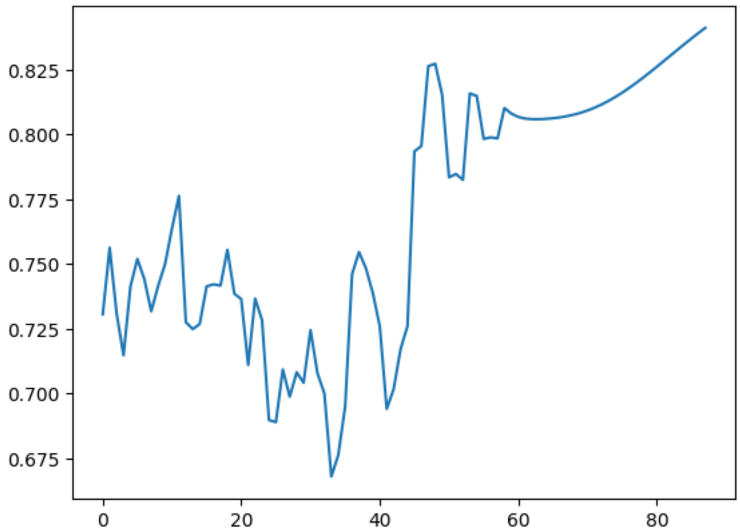
```
plt.plot(day_new,scaler.inverse_transform(df1[1158:]))
plt.plot(day_pred,scaler.inverse_transform(lst_output))
```

```
[<matplotlib.lines.Line2D at 0x78d87dbc0f40>]
```




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

```
[<matplotlib.lines.Line2D at 0x78d87ee3cca0>]
```




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

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

```
[<matplotlib.lines.Line2D at 0x78d87edaab90>]
```