```
import pandas as pd
d=pd.read_csv("/content/AAPL.csv")
df=pd.DataFrame(d)
df.head()
```

	Unnamed: 0	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolume	di
0	0	AAPL	2015-05-27 00:00:00+00:00	132.045	132.260	130.05	130.34	45833246	121.682558	121.880685	119.844118	120.111360	45833246	
1	1	AAPL	2015-05-28 00:00:00+00:00	131.780	131.950	131.10	131.86	30733309	121.438354	121.595013	120.811718	121.512076	30733309	
2	2	AAPL	2015-05-29	130.280	131.450	129.90	131.23	50884452	120.056069	121.134251	119.705890	120.931516	50884452	

```
df.shape
```

```
(1258, 15)
```

```
print(df.columns)
```

```
Index(['Unnamed: 0', 'symbol', 'date', 'close', 'high', 'low', 'open',
       'volume', 'adjClose', 'adjHigh', 'adjLow', 'adjOpen', 'adjVolume',
       'divCash', 'splitFactor'],
      dtype='object')
```

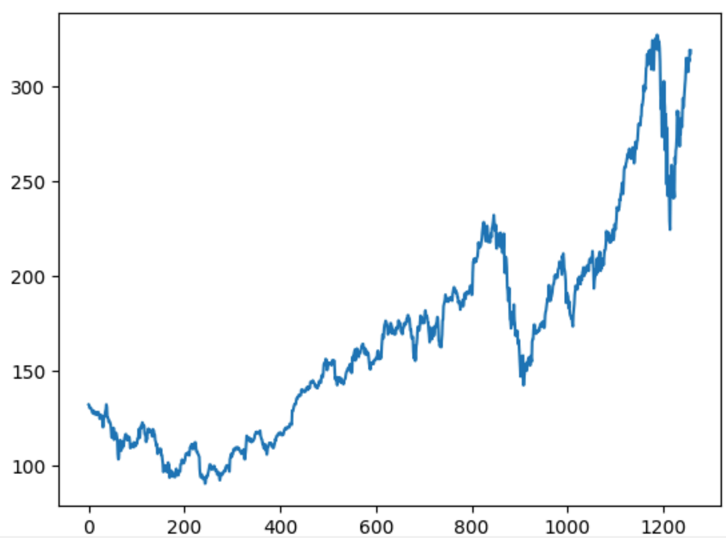
```
df1=df.reset_index()['close']
```

```
df.describe
```

```
pandas.core.generic.NDFrame.describe
def describe(percentiles=None, include=None, exclude=None) -> Self

Generate descriptive statistics.

Descriptive statistics include those that summarize the central
tendency, dispersion and shape of a
dataset's distribution, excluding ``NaN`` values.
```

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

```
[<matplotlib.lines.Line2D at 0x7cd5fc5274f0>]
```




```
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))
```

```
print(df1)
```

```
[[0.17607447]
 [0.17495567]
 [0.16862282]
 ...
 [0.96635143]
 [0.9563033 ]
 [0.96491598]]
```

```
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
```

```
       [0.40754032],
       [0.42176813],
       [0.42848096],
       [0.43472938],
       [0.43755805],
       [0.43536266],
       [0.42793211],
       [0.42594782],
       [0.43038082],
       [0.42371021],
       [0.4241324 ],
       [0.41585747],
       [0.41543528],
       [0.40255847],
       [0.40597821],
       [0.40158744],
       [0.39930761],
       [0.38769737],
       [0.39723888],
       [0.39609896],
       [0.40175631],
       [0.40010977],
       [0.40884911],
       [0.3950857 ],
       [0.40133412],
       [0.41218441],
       [0.42320358],
       [0.42223254],
       [0.41180444],
       [0.42510344],
       [0.42637001],
       [0.42459681],
       [0.42687664],
       [0.42244364],
       [0.42869205],
       [0.42683442],
       [0.42755214],
       [0.43342059],
       [0.44110445],
       [0.43852909],
       [0.42489234],
       [0.42037491],
       [0.42197923],
       [0.46930676],
       [0.49417377],
       [0.49670692],
       [0.50126657],
       [0.49299164],
       [0.49358271],
       [0.50046441],
       [0.49476484],
       [0.50042219],
       [0.50413747],
       [0.5062062 ],
       [0.51920966],
       [0.53719497],
       [0.52824453],
       [0.52647133]])
```

```
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)
```

```
time_step = 100
X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)
```

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

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

```
print(X_test.shape)
print(ytest.shape)
```

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

```
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)
```

```
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')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(**kwargs)
```

```
model.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
lstm_15 (LSTM)	(None, 100, 50)	10,400
lstm_16 (LSTM)	(None, 100, 50)	20,200
lstm_17 (LSTM)	(None, 50)	20,200
dense_5 (Dense)	(None, 1)	51

Total params: 50,851 (198.64 KB)

```
model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=100,batch_size=64,verbose=1)
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 4s 294ms/step - loss: 2.0811e-04 - val_loss: 0.0011
Epoch 79/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 175ms/step - loss: 2.2225e-04 - val_loss: 0.0022
Epoch 80/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 175ms/step - loss: 2.1990e-04 - val_loss: 0.0010
Epoch 81/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 177ms/step - loss: 1.8725e-04 - val_loss: 0.0010
Epoch 82/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 283ms/step - loss: 1.5077e-04 - val_loss: 0.0013
Epoch 83/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 5s 226ms/step - loss: 2.3622e-04 - val_loss: 0.0016
Epoch 84/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 4s 180ms/step - loss: 1.9970e-04 - val_loss: 0.0011
Epoch 85/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 199ms/step - loss: 1.7326e-04 - val_loss: 9.6667e-04
Epoch 86/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 282ms/step - loss: 1.6861e-04 - val_loss: 0.0012
Epoch 87/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 4s 198ms/step - loss: 1.5890e-04 - val_loss: 8.9791e-04
Epoch 88/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 188ms/step - loss: 1.6699e-04 - val_loss: 8.9539e-04
Epoch 89/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 176ms/step - loss: 1.6501e-04 - val_loss: 9.4278e-04
Epoch 90/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 206ms/step - loss: 1.4558e-04 - val_loss: 8.8152e-04
Epoch 91/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 284ms/step - loss: 1.5990e-04 - val_loss: 8.3912e-04
Epoch 92/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 4s 177ms/step - loss: 1.4283e-04 - val_loss: 8.7593e-04
Epoch 93/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 176ms/step - loss: 1.5932e-04 - val_loss: 9.2086e-04
Epoch 94/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 177ms/step - loss: 1.5807e-04 - val_loss: 0.0013
Epoch 95/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 237ms/step - loss: 2.0684e-04 - val_loss: 8.1647e-04
Epoch 96/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 4s 178ms/step - loss: 1.4427e-04 - val_loss: 9.2412e-04
Epoch 97/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 179ms/step - loss: 1.9770e-04 - val_loss: 8.0459e-04
Epoch 98/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 2s 175ms/step - loss: 1.6045e-04 - val_loss: 8.7562e-04
Epoch 99/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 187ms/step - loss: 1.6364e-04 - val_loss: 7.7142e-04
Epoch 100/100
12/12 ━━━━━━━━━━━━━━━━━━━━ 3s 290ms/step - loss: 1.3643e-04 - val_loss: 7.7229e-04
<keras.src.callbacks.history.History at 0x7cd5fc4070d0>
```

```
import numpy as np
from sklearn.metrics import accuracy_score

# Combine training and test datasets
X_combined = np.concatenate((X_train, X_test), axis=0)
y_combined = np.concatenate((y_train, y_test), axis=0)

# Predict on the combined dataset
y_pred = model.predict(X_combined)

# Calculate accuracy
overall_accuracy = accuracy_score(y_combined, y_pred)
print(f"Overall Accuracy: {overall_accuracy * 100:.2f}%")
```

```
Overall Accuracy: 78.00%
```

```
import tensorflow as tf
```

```
tf.__version__
```

```
'2.17.0'
```

```
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

```
23/23 ━━━━━━━━━━━━━━━━━━━━ 3s 89ms/step
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
```

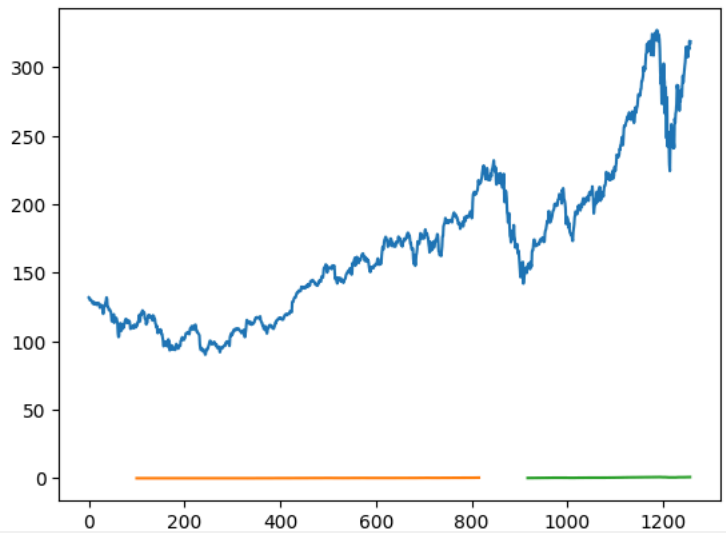
```
import math #Suitable for regression tasks (e.g., predicting the actual price of a stock).
from sklearn.metrics import mean_squared_error #MSE
math.sqrt(mean_squared_error(y_train,train_predict))
```

```
0.01138800274314705
```

```
math.sqrt(mean_squared_error(ytest,test_predict)) #RMSE
```

```
0.02779012270573828
```

```
# shift train predictions for plotting
import matplotlib.pyplot as plt
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[341:].reshape(1,-1)
x_input.shape
```

```
(1, 100)
```

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

```
temp_input
```

```
      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]
```

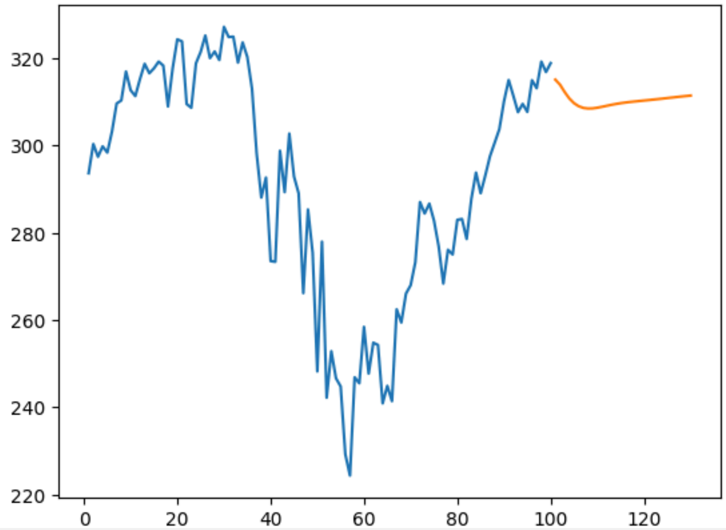
```
from numpy import array #the scaling is different so use inverse scale
lst_output=[]
n_steps=100
i=0
while(i<30):
    if(len(temp_input)>100):
        #print(temp_input)
        x_input=np.array(temp_input[1:])
        print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)
        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)
```

```
3 day input [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 0.96635143 0.9563033
 0.96491598 0.94892615 0.94406486 0.93691605]
3 day output [[0.93067515]]
4 day input [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 0.96635143 0.9563033  0.96491598
 0.94892615 0.94406486 0.93691605 0.93067515]
4 day output [[0.92614037]]
5 day input [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
```

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

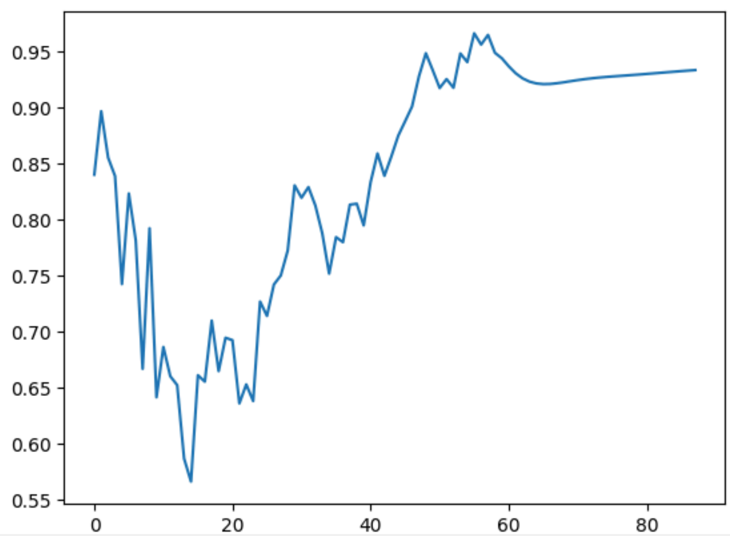
```
import matplotlib.pyplot as plt
plt.plot(day_new,scaler.inverse_transform(df1[1158:]))
plt.plot(day_pred,scaler.inverse_transform(lst_output))
```

```
[<matplotlib.lines.Line2D at 0x7cd5fb8a21d0>]
```




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

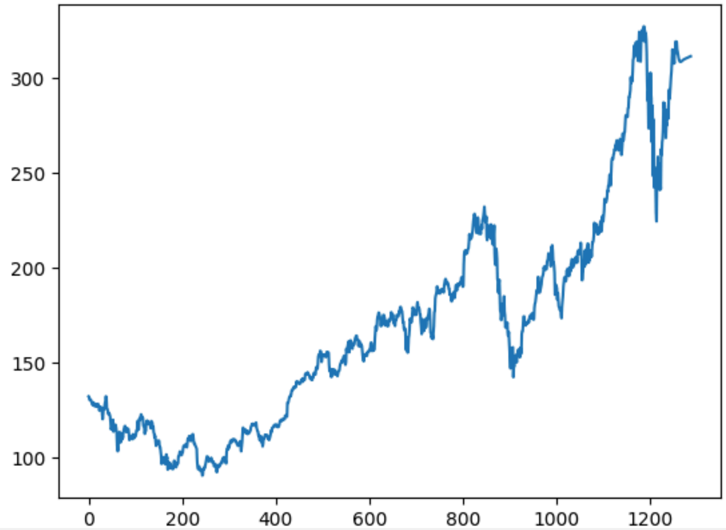
```
[<matplotlib.lines.Line2D at 0x7cd5fb982e90>]
```




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

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

```
[<matplotlib.lines.Line2D at 0x7cd5fb7eaa10>]
```




```
lst_output_test = model.predict(X_test)
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 56ms/step
```

```
!pip install scikit-learn
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

# Assuming you have your features in X and labels in y
# Replace these with your actual data
X = np.random.rand(1000, 10)
y = np.random.randint(0, 2, 1000)

# Feature Scaling
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Define the number of folds
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Lists to store the accuracy for each fold
train_accuracies = []
test_accuracies = []

# Example with RandomForestClassifier and GridSearchCV
def train_model(X_train, y_train):
    model = RandomForestClassifier(random_state=42)

    # Hyperparameter tuning with GridSearchCV
    param_grid = {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 5, 10],
    }
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5)
    grid_search.fit(X_train, y_train)

    return grid_search.best_estimator_

# Perform k-fold cross-validation
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train your model
    model = train_model(X_train, y_train)

    # Predict on training and testing data
    y_pred_train = model.predict(X_train)
```

```
    # Calculate accuracies
    train_accuracy = accuracy_score(y_train, y_pred_train)

    train_accuracies.append(train_accuracy)

# Calculate average accuracies across all folds
avg_train_accuracy = np.mean(train_accuracies)

print(f"Average Training Accuracy: {avg_train_accuracy:.4f}")
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Average Training Accuracy: 0.9235
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_train, y_pred_train)
print("Confusion Matrix:\n", cm)
```

```
Confusion Matrix:
 [[397   0]
 [  0 403]]
```