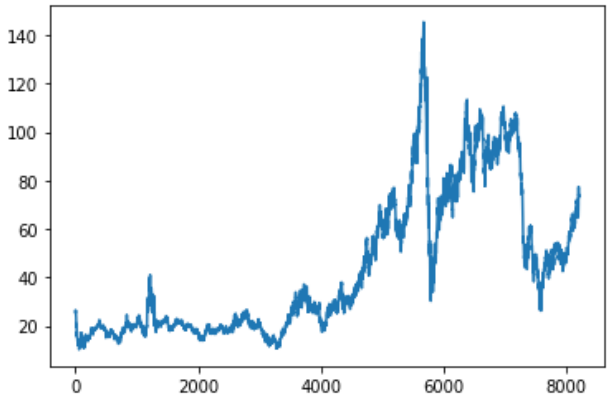
Test The Model

Finally, we can generate predictions using the model for both the train and test to visualize the model.

We must shift the predictions so that they align on the x-axis with the original dataset. Once prepared, the data is plotted, showing the original dataset in blue, the predictions for the training dataset in green, the predictions on the unseen test dataset in orange.

```
### Plotting
# shift train predictions for plotting
look_back=10
trainPredictPlot = np.empty_like(data_oil)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = np.empty_like(data_oil)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(data_oil)-1, :] = test_predict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(data_oil))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



:

Now let us predict the price of crude oil for the next 10 days.

As the length of the test data is 2876, We are taking previous 10 days input i.e., from index 2866 -2876 to predict 2867 th day output

```
len(test_data)
```

```
2876
```

Create the input and reshape it and convert it into list

```
x_input=test_data[2866:].reshape(1,-1)
x_input.shape
```

```
(1, 10)
```

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

We can see temp_input contains last 10 days price list

```
temp_input
```

```
[0.44172960165852215,
 0.48111950244335855,
 0.49726047682511476,
 0.4679401747371539,
 0.4729749740855915,
 0.47119798608026064,
 0.47341922108692425,
 0.4649785280616022,
 0.4703835332444839,
 0.47149415074781587]
```

For predicting next 10 days crude oil prices we consider n_steps=10

We create the input for prediction, index starting from the date 10 days before the first date in the test dataset. Then, reshape the inputs to have only 1 column and predict using model_predict predefined function.

This can be done using the below code

```
lst_output=[]
n_steps=10
i=0
while(i<10):

    if(len(temp_input)>10):
        #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
```

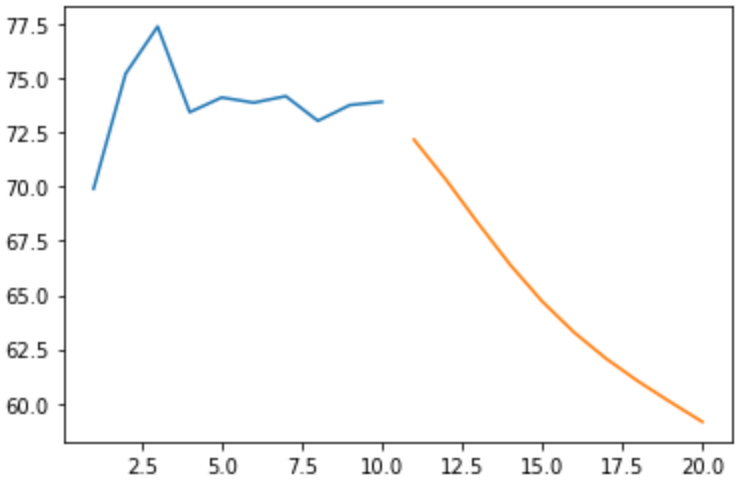
The output is as shown follows

We can infer that it is taking 10 inputs and predicting the day 11 th output.

```
[0.45856044]
11
1 day input [0.4811195  0.49726048 0.46794017 0.47297497 0.47119799 0.47341922
 0.46497853 0.47038353 0.47149415 0.45856044]
1 day output [[0.44484875]]
2 day input [0.49726048 0.46794017 0.47297497 0.47119799 0.47341922 0.46497853
 0.47038353 0.47149415 0.45856044 0.44484875]
2 day output [[0.43000898]]
3 day input [0.46794017 0.47297497 0.47119799 0.47341922 0.46497853 0.47038353
 0.47149415 0.45856044 0.44484875 0.43000898]
3 day output [[0.4157903]]
4 day input [0.47297497 0.47119799 0.47341922 0.46497853 0.47038353 0.47149415
 0.45856044 0.44484875 0.43000898 0.41579029]
4 day output [[0.40325096]]
5 day input [0.47119799 0.47341922 0.46497853 0.47038353 0.47149415 0.45856044
 0.44484875 0.43000898 0.41579029 0.40325096]
5 day output [[0.39260027]]
6 day input [0.47341922 0.46497853 0.47038353 0.47149415 0.45856044 0.44484875
 0.43000898 0.41579029 0.40325096 0.39260027]
6 day output [[0.383632]]
7 day input [0.46497853 0.47038353 0.47149415 0.45856044 0.44484875 0.43000898
 0.41579029 0.40325096 0.39260027 0.383632  ]
7 day output [[0.37586474]]
8 day input [0.47038353 0.47149415 0.45856044 0.44484875 0.43000898 0.41579029
 0.40325096 0.39260027 0.383632   0.37586474]
8 day output [[0.36881918]]
9 day input [0.47149415 0.45856044 0.44484875 0.43000898 0.41579029 0.40325096
 0.39260027 0.383632   0.37586474 0.36881918]
9 day output [[0.3620221]]
[[0.45856043696403503], [0.4448487460613251], [0.43000897765159607], [0.41579028964042664], [0.40325096249580383], [0.39260026812553406],
[0.38363200426101685], [0.37586474418640137], [0.3688191771507263], [0.3620221018791199]]
```

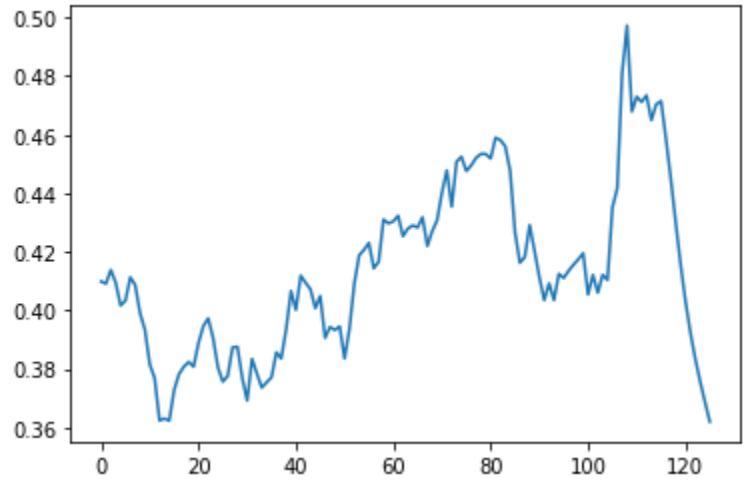
let’s create a visualization plot to easily review the prediction.

Don’t forget, what we have predicted is the scaled values, so we need to reverse the prediction



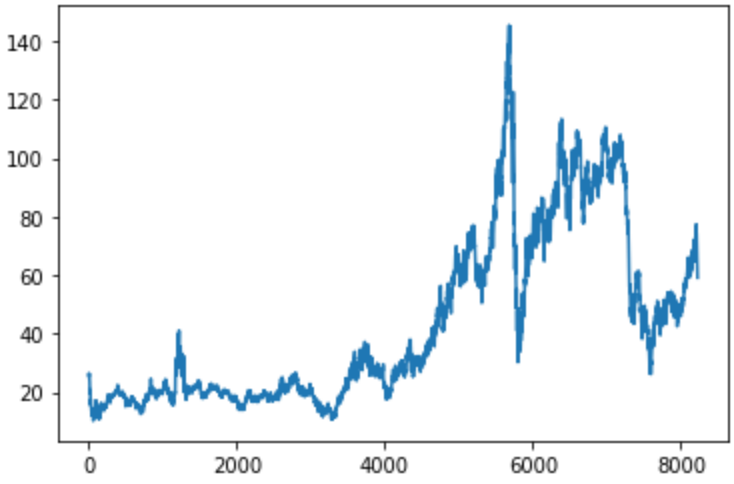
Let us merge the the past data and next 10 days output prediction

```
df3=data_oil.tolist()
df3.extend(lst_output)
plt.plot(df3[8100:])
```



Don’t forget, what we have predicted is the scaled values, so we need to reverse the prediction

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


140
120
100
80
60
40
20
0
2000
4000
6000
8000