CRUDE OIL PROCE PREDICTION

November 15, 2024

1 DATA PREPROCESSING

1.1 Importing the libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as pyt
import seaborn as sns
import tensorflow as tf
```

[2]:
```
pwd
```

[2]: 'C:\\Users\\lenovo\\Desktop'

[3]:
```
import pandas as pd
data= pd.read_csv(r"C:\Users\lenovo\Downloads\Crude Oil Prices Daily.csv")
print(data.head())
```

```
       Date  Closing Value
0  1/2/1986          25.56
1  1/3/1986          26.00
2  1/6/1986          26.53
3  1/7/1986          25.85
4  1/8/1986          25.87
```

1.2 Analyze the Data

[4]:
```
data.head()
```

[4]:

	Date	Closing Value
0	1/2/1986	25.56
1	1/3/1986	26.00
2	1/6/1986	26.53
3	1/7/1986	25.85
4	1/8/1986	25.87

[11]:
```
data.tail()
```

[11]:
```
            Date  Closing Value
8218  7/3/2018          74.19
8219  7/4/2018            NaN
8220  7/5/2018          73.05
8221  7/6/2018          73.78
8222  7/9/2018          73.93
```

[12]:
```
data.describe()
```

[12]:
```
       Closing Value
count    8216.000000
mean       43.492139
std        29.616804
min        10.250000
25%        19.577500
50%        29.610000
75%        63.402500
max       145.310000
```

data.info()

1.3 Handling missing values

[14]:
```
data.isnull().any()
```

[14]:
```
Date             False
Closing Value     True
dtype: bool
```

[15]:
```
data.isnull().sum()
```

[15]:
```
Date             0
Closing Value    7
dtype: int64
```

[16]:
```
data.dropna(axis=0,inplace=True)
```

[17]:
```
data.isnull().sum()
```

[17]:
```
Date             0
Closing Value    0
dtype: int64
```

[18]:
```
data_oil = data.reset_index()['Closing Value']
```

[19]:
```
data_oil
```

[19]:
```
0       25.56
1       26.00
2       26.53
3       25.85
4       25.87
         …
8211    73.89
8212    74.19
8213    73.05
8214    73.78
8215    73.93
Name: Closing Value, Length: 8216, dtype: float64
```

[20]:
```python
print(data_oil.isnull().sum())
print(data_oil.shape)
```

```
0
(8216,)
```

[21]:
```python
data_oil.dropna(inplace=True)
print(data_oil.isnull().sum())
print(data_oil.shape)
```

```
0
(8216,)
```

[22]:
```python
print(data_oil.isnull().any())
```

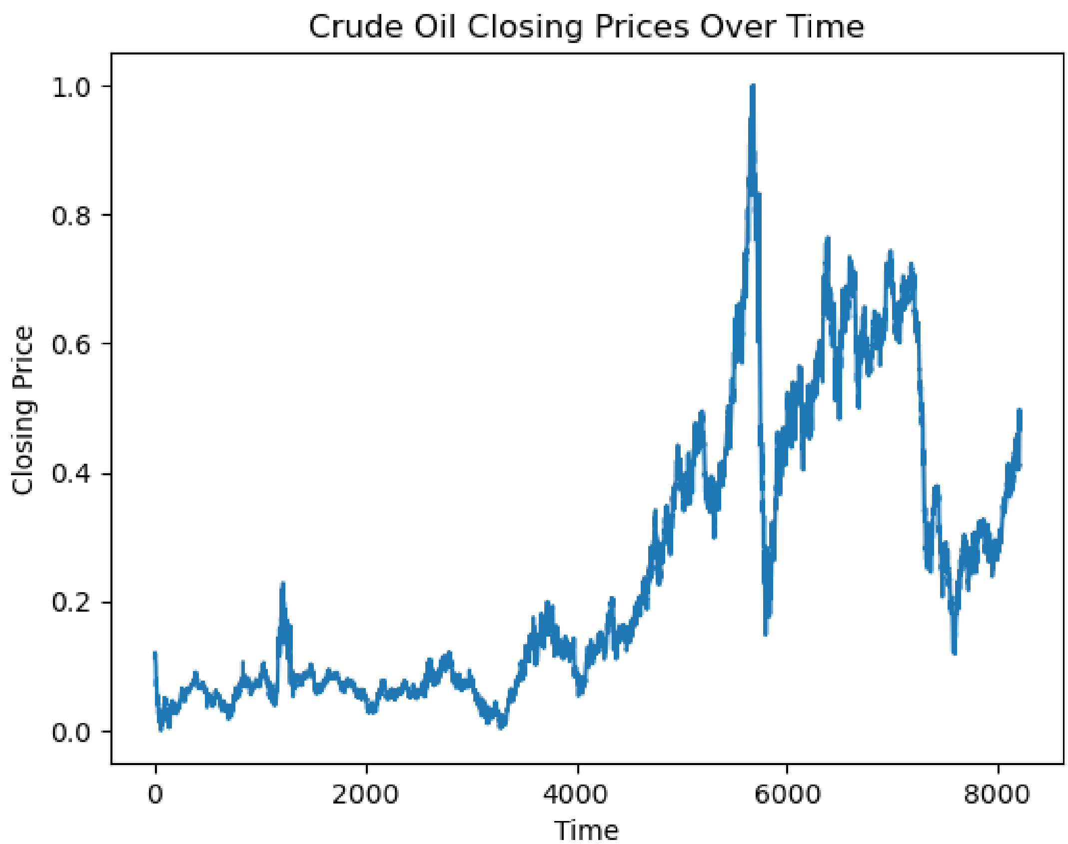
```
False
```

1.4 Feature Scaling

[23]:
```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
data_oil = scaler.fit_transform(np.array(data_oil).reshape(-1, 1))
```

1.5 Data Visualization

[24]:
```python
import matplotlib.pyplot as plt  # Use plt as the standard alias
plt.title('Crude Oil Closing Prices Over Time')
plt.plot(data_oil)
plt.xlabel('Time')
plt.ylabel('Closing Price')
plt.show()
```



1.6 Splitting Data into Train and Test

[25]:
```
training_size = int(len(data_oil)*0.65)
test_size = len(data_oil) - training_size
train_data,test_data = data_oil[0:training_size,:],data_oil[training_size:
 ↪len(data_oil),:1]
```

[26]:
```
training_size, test_size
(5340, 2876)
```

[26]: (5340, 2876)

[27]:
```
train_data.shape
(5340)
```

[27]: 5340

[28]:
```
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)
```

[29]:
```
time_step = 10
X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)
```

[30]:
```
print(X_train.shape,y_train.shape)
```

```
(5329, 10) (5329,)
```

[31]:
```
print(X_test.shape,y_train.shape)
```

```
(2865, 10) (5329,)
```

[32]:
```
X_train
```

[32]:
```
array([[0.11335703, 0.11661484, 0.12053902, …, 0.10980305, 0.1089886 ,
        0.11054346],
       [0.11661484, 0.12053902, 0.11550422, …, 0.1089886 , 0.11054346,
        0.10165852],
       [0.12053902, 0.11550422, 0.1156523 , …, 0.11054346, 0.10165852,
        0.09906708],
       …,
       [0.36731823, 0.35176958, 0.36080261, …, 0.36391234, 0.37042796,
        0.37042796],
       [0.35176958, 0.36080261, 0.35354657, …, 0.37042796, 0.37042796,
        0.37879461],
       [0.36080261, 0.35354657, 0.35295424, …, 0.37042796, 0.37879461,
        0.37916482]])
```

[33]:
```
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)
```

2 Model Building

2.1 Importing the Model Building Libraries

[38]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense,Input
```

2.2 Initializing the model

[39]:
```
model = Sequential()
```

2.3 Adding LSTM Layers

[40]:
```
model.add(LSTM(50, return_sequences=True, input_shape=(10,1)))
model.add(LSTM(50, return_sequences=True))
model.add(LSTM(50))
```

2.4 Adding output Layers

[41]:
```
model.add(Dense(1))
```

[42]:
```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm_3 (LSTM)	(None, 10, 50)	10,400
lstm_4 (LSTM)	(None, 10, 50)	20,200
lstm_5 (LSTM)	(None, 50)	20,200
dense_1 (Dense)	(None, 1)	51

Total params: 50,851 (198.64 KB)

Trainable params: 50,851 (198.64 KB)

Non-trainable params: 0 (0.00 B)

2.5 Configure The Learning Process

[43]:
```
model.compile(loss='mean_squared_error',optimizer='adam')
```

2.6 Train The model

[44]:
```
model.fit(X_train, y_train, validation_data=(X_test,ytest), epochs=10,␣
↪batch_size=64, verbose=1)
```

```
Epoch 1/10
84/84               12s 43ms/step -
loss: 0.0055 - val_loss: 8.8414e-04
Epoch 2/10
84/84               3s 30ms/step -
loss: 1.3258e-04 - val_loss: 7.6096e-04
Epoch 3/10
84/84               2s 28ms/step -
loss: 1.1566e-04 - val_loss: 8.1974e-04
Epoch 4/10
84/84               3s 31ms/step -
loss: 1.1804e-04 - val_loss: 7.6955e-04
Epoch 5/10
84/84               3s 29ms/step -
loss: 1.1955e-04 - val_loss: 0.0011
Epoch 6/10
84/84               2s 29ms/step -
loss: 1.1475e-04 - val_loss: 7.9172e-04
Epoch 7/10
84/84               2s 28ms/step -
loss: 1.3041e-04 - val_loss: 8.3297e-04
Epoch 8/10
84/84               2s 28ms/step -
loss: 1.1645e-04 - val_loss: 7.7667e-04
Epoch 9/10
84/84               3s 29ms/step -
loss: 1.1570e-04 - val_loss: 9.2034e-04
Epoch 10/10
84/84               3s 32ms/step -
loss: 1.0593e-04 - val_loss: 0.0027
```

[44]:
```
<keras.src.callbacks.history.History at 0x1aaadb4f590>
```

[45]:
```
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
```

```
167/167             3s 15ms/step
90/90               1s 8ms/step
```

2.7 Model Evaluation

[47]:
```
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

[48]:
```
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

[48]: 3827.0821056981763

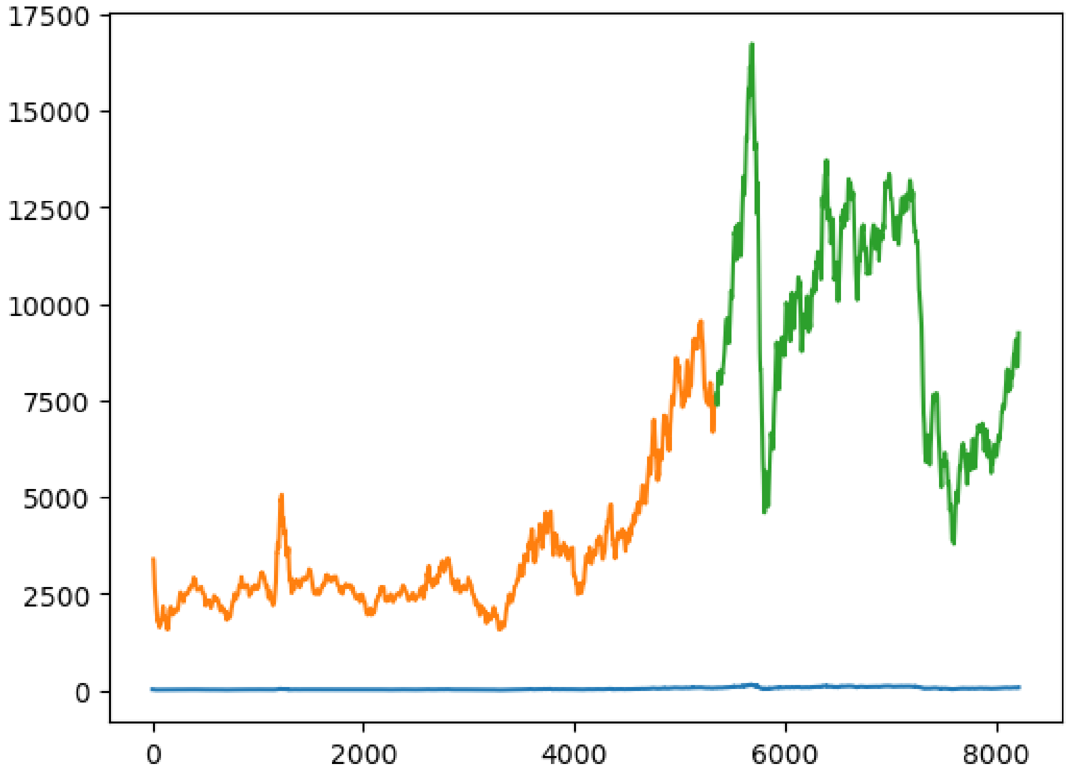
2.8 Save the Model

[49]:
```
from tensorflow.keras.models import load_model
```

[52]:
```
model.save('Crude_oil_price_prediction.keras')
```

2.9 Test the Model

[55]:
```
look_back=10
trainPredictPlot = np.empty_like(data_oil)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back: len(train_predict)+look_back, : ] = train_predict
testPredictPlot = np.empty_like(data_oil)
testPredictPlot[:, :] = np.nan
testPredictPlot [len (train_predict)+(look_back*2)+1:len(data_oil)-1, :] =␣
 ↪test_predict
plt.plot(scaler.inverse_transform(data_oil))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



[56]:
```
len(test_data)
```

[56]: 2876

[57]:
```
x_input = test_data[2866:].reshape(1,-1)
x_input.shape
```

[57]: (1, 10)

[58]:
```
temp_input = list(x_input)
temp_input = temp_input[0].tolist()
```

[59]:
```
temp_input
```

[59]:
```
[0.44172960165852215,
 0.48111950244335855,
 0.49726047682511476,
 0.4679401747371539,
 0.4729749740855915,
 0.47119798608026064,
 0.47341922108692425,
 0.4649785280616022,
```

```
 0.4703835332444839,
 0.47149415074781587]
```

[61]:
```python
first_output = []
n_steps = 10
i = 0

while i < 10:
    if len(temp_input) > 10:
        x_input = np.array(temp_input[1:])
        print(f"{i} day input {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(f"{i} day output {yhat}")

        temp_input.extend(yhat[0].tolist())
        temp_input = temp_input[1:]

        first_output.extend(yhat.tolist())
        i += 1
    else:
        x_input = np.array(temp_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))
        first_output.extend(yhat.tolist())
        i += 1
```

```
[0.43688264]
11
1 day input [0.4811195  0.49726048 0.46794017 0.47297497 0.47119799 0.47341922
 0.46497853 0.47038353 0.47149415 0.43688264]
1 day output [[0.43722245]]
2 day input [0.49726048 0.46794017 0.47297497 0.47119799 0.47341922 0.46497853
 0.47038353 0.47149415 0.43688264 0.43722245]
2 day output [[0.4339377]]
3 day input [0.46794017 0.47297497 0.47119799 0.47341922 0.46497853 0.47038353
 0.47149415 0.43688264 0.43722245 0.4339377 ]
3 day output [[0.42893893]]
4 day input [0.47297497 0.47119799 0.47341922 0.46497853 0.47038353 0.47149415
 0.43688264 0.43722245 0.4339377  0.42893893]
4 day output [[0.42501116]]
5 day input [0.47119799 0.47341922 0.46497853 0.47038353 0.47149415 0.43688264
 0.43722245 0.4339377  0.42893893 0.42501116]
```

```
5 day output [[0.42015254]]
6 day input [0.47341922 0.46497853 0.47038353 0.47149415 0.43688264 0.43722245
 0.4339377  0.42893893 0.42501116 0.42015254]
6 day output [[0.41498962]]
7 day input [0.46497853 0.47038353 0.47149415 0.43688264 0.43722245 0.4339377
 0.42893893 0.42501116 0.42015254 0.41498962]
7 day output [[0.4094336]]
8 day input [0.47038353 0.47149415 0.43688264 0.43722245 0.4339377  0.42893893
 0.42501116 0.42015254 0.41498962 0.4094336 ]
8 day output [[0.40418002]]
9 day input [0.47149415 0.43688264 0.43722245 0.4339377  0.42893893 0.42501116
 0.42015254 0.41498962 0.4094336  0.40418002]
9 day output [[0.39845008]]
```

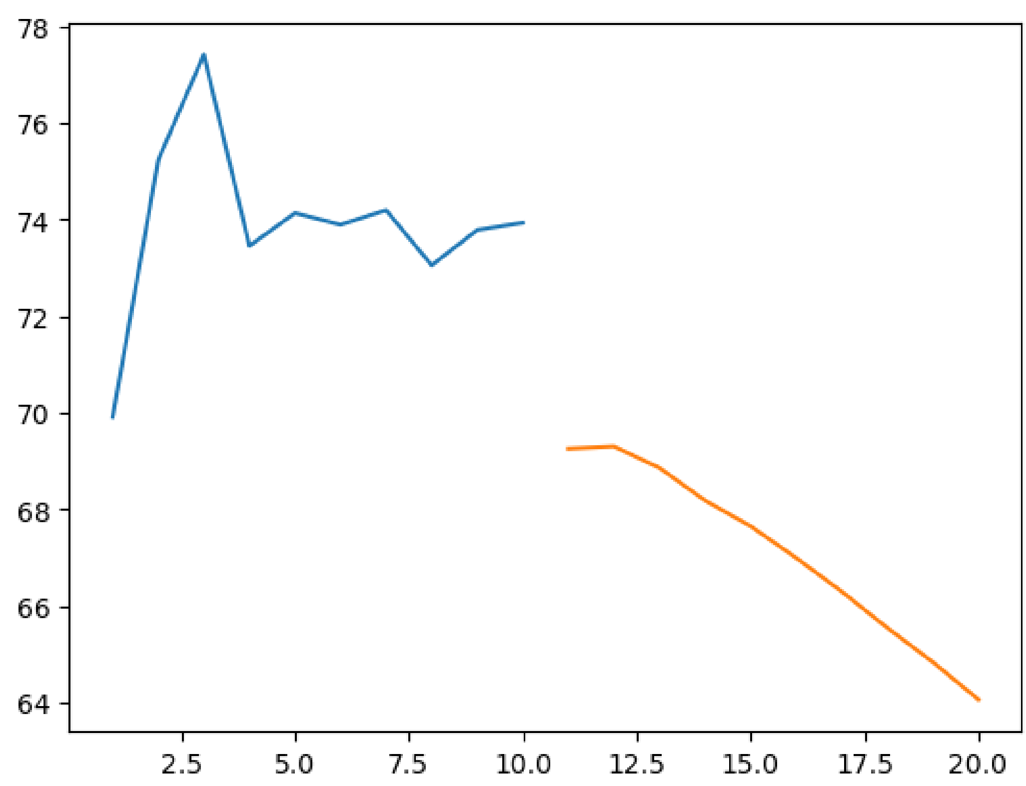
[65]:
```
day_new = np.arange(1, 11)
day_pred = np.arange(11, 21)
```

[66]:
```
len(data_oil)
```

[66]: 8216

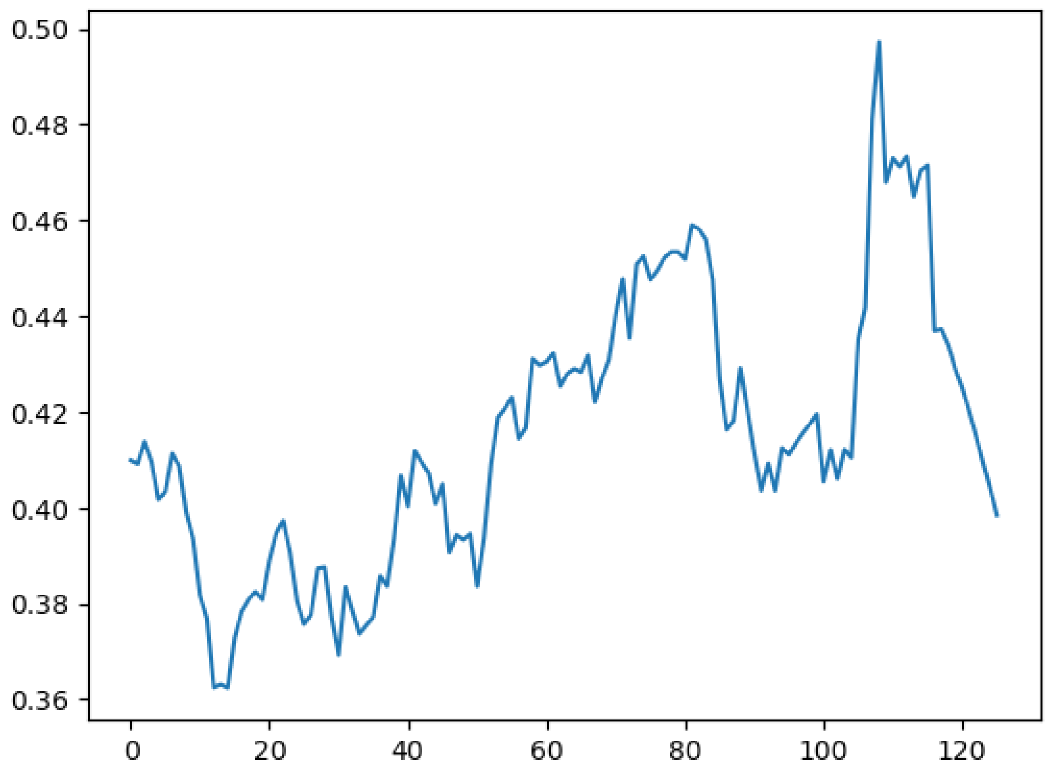
[68]:
```
plt.plot(day_new,scaler.inverse_transform(data_oil[8206:]))
plt.plot(day_pred,scaler.inverse_transform(first_output))
```

[68]: [<matplotlib.lines.Line2D at 0x1aab4c23350>]



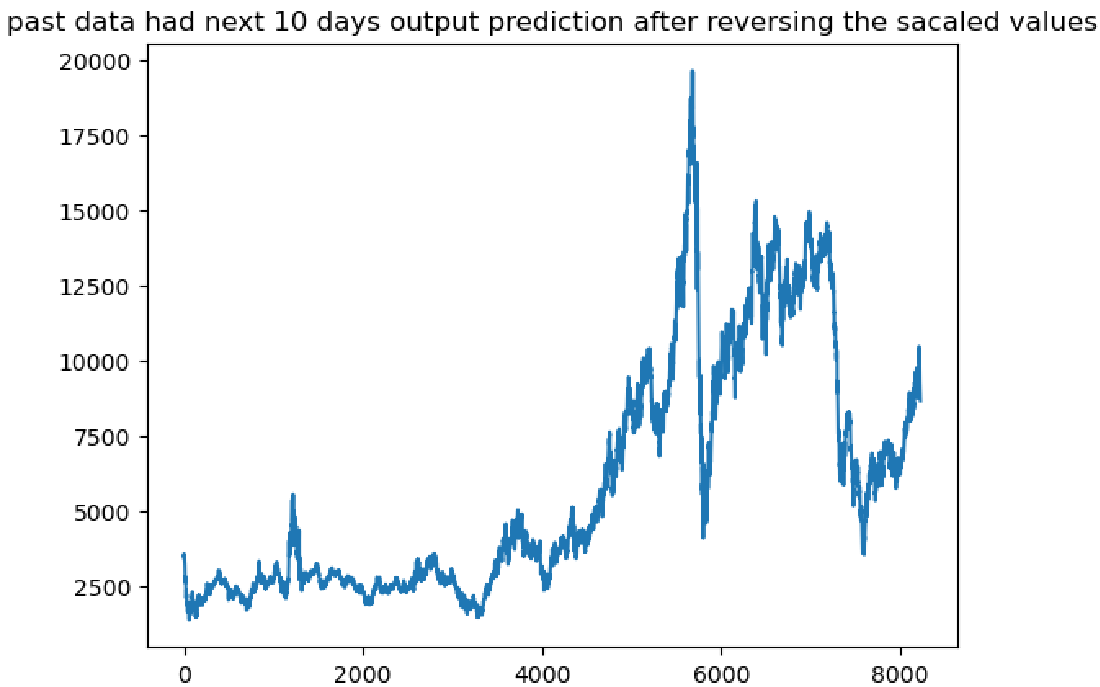
[71]:
```
df3 = data_oil.tolist()
df3.extend(first_output)
plt.plot(df3[8100:])
```

[71]: [<matplotlib.lines.Line2D at 0x1aab4ee61b0>]



[73]:
```
df3 = scaler.inverse_transform(df3).tolist()
plt.title('past data had next 10 days output prediction after reversing the␣
  ↪sacaled values')
plt.plot(df3)
```

[73]: [<matplotlib.lines.Line2D at 0x1aab501e8a0>]


past data had next 10 days output prediction after reversing the sacaled values
20000
17500
15000
12500
10000
7500
5000
2500
0
2000
4000
6000
8000