

MODEL BUILDING-TEST THE MODEL

Team ID	PNT2022TMID13501
Project Name	Crude Oil Price Prediction

```
In [ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
In [ ]: data=pd.read_excel("/content/Crude Oil Prices Daily.xlsx")
```

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

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

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

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

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

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

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

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

```
Out[ ]: 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
```

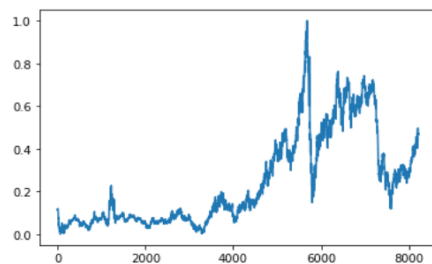
```
In [ ]: from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
data_oil=scaler.fit_transform(np.array(data_oil).reshape(-1,1))
```

```
In [ ]: data_oil
```

```
Out[ ]: array([[0.11335703],
[0.11661484],
[0.12053902],
...,
[0.46497853],
[0.47038353],
[0.47149415]])
```

```
In [ ]: plt.plot(data_oil)
```

```
Out[ ]: []
```



```
In [ ]: 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]
```

```
In [ ]: training_size,test_size
```

```
Out[ ]: (5340, 2876)
```

```
In [ ]: train_data.shape
```

```
Out[ ]: (5340, 1)
```

```
In [ ]: 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)
```

```
In [ ]: time_step=10  
x_train,y_train=create_dataset(train_data,time_step)  
x_test,y_test=create_dataset(test_data,time_step)
```

```
In [ ]: print(x_train.shape),print(y_train.shape)
```

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

```
Out[ ]: (None, None)
```

```
In [ ]: print(x_test.shape),print(y_test.shape)
```

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

```
Out[ ]: (None, None)
```

```
In [ ]: x_train
```

```
Out[ ]: 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]])
```

```
In [ ]: 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)
```

```
In [ ]: from tensorflow.keras.models import Sequential  
        from tensorflow.keras.layers import Dense  
        from tensorflow.keras.layers import LSTM
```

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

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

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

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

```
Model: "sequential"
-----
Layer (type)                Output Shape              Param #
-----
lstm (LSTM)                  (None, 10, 50)           10400
lstm_1 (LSTM)                (None, 10, 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
-----
```

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

```
In [ ]: model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=3,batch_size=64,verbose=1)
```

```
Epoch 1/3
84/84 [=====] - 6s 25ms/step - loss: 0.0017 - val_loss: 0.0011
Epoch 2/3
84/84 [=====] - 1s 16ms/step - loss: 1.2375e-04 - val_loss: 7.8338e-04
Epoch 3/3
84/84 [=====] - 1s 16ms/step - loss: 1.2058e-04 - val_loss: 7.5010e-04
```

```
Out[ ]:
```

```
In [ ]: ##Transformback to original form
train_predict=scaler.inverse_transform(train_data)
test_predict=scaler.inverse_transform(test_data)
### Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(train_data,train_predict))
```

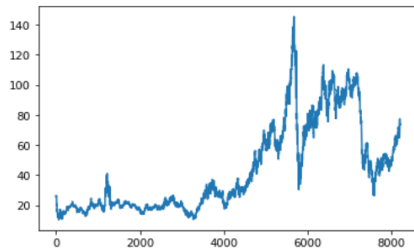
```
Out[ ]: 29.347830443269938
```

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

```
In [ ]: model.save("crude_oil.hs")
```

```
WARNING:absl:Found untraced functions such as lstm_cell_layer_call_fn, lstm_cell_layer_call_and_return_conditional_losses, lstm_cell_1_layer_call_fn, lstm_cell_1_layer_call_and_return_conditional_losses, lstm_cell_2_layer_call_fn while saving (showing 5 of 6). These functions will not be directly callable after loading.
WARNING:absl: has the same name 'LSTMCell' as a built-in Keras object. Consider renaming to avoid naming conflicts when loading with `tf.keras.models.load_model`. If renaming is not possible, pass the object in the `custom_objects` parameter of the load function.
WARNING:absl: has the same name 'LSTMCell' as a built-in Keras object. Consider renaming to avoid naming conflicts when loading with `tf.keras.models.load_model`. If renaming is not possible, pass the object in the `custom_objects` parameter of the load function.
WARNING:absl: has the same name 'LSTMCell' as a built-in Keras object. Consider renaming to avoid naming conflicts when loading with `tf.keras.models.load_model`. If renaming is not possible, pass the object in the `custom_objects` parameter of the load function.
```

```
In [ ]: ### 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[look_back:len(test_predict)+look_back, :] = test_predict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(data_oil))
plt.show()
```



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

```
Out[ ]: 2876
```

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

```
Out[ ]: (1, 10)
```

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

```
In [ ]: temp_input
```

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

```
In [ ]: 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
```

```

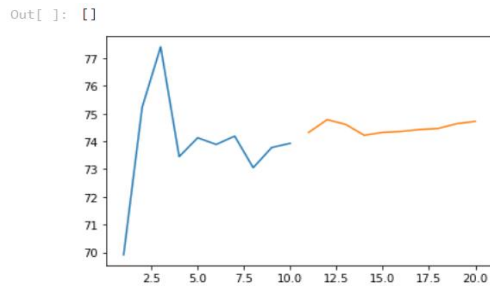
[0.47442466]
11
1 day input [0.4811195  0.49726048 0.46794017 0.47297497 0.47119799 0.47341922
0.46497853 0.47038353 0.47149415 0.47442466]
1 day output [[0.47781762]]
2 day input [0.49726048 0.46794017 0.47297497 0.47119799 0.47341922 0.46497853
0.47038353 0.47149415 0.47442466 0.47781762]
2 day output [[0.47653615]]
3 day input [0.46794017 0.47297497 0.47119799 0.47341922 0.46497853 0.47038353
0.47149415 0.47442466 0.47781762 0.47653615]
3 day output [[0.47364426]]
4 day input [0.47297497 0.47119799 0.47341922 0.46497853 0.47038353 0.47149415
0.47442466 0.47781762 0.47653615 0.47364426]
4 day output [[0.47442248]]
5 day input [0.47119799 0.47341922 0.46497853 0.47038353 0.47149415 0.47442466
0.47781762 0.47653615 0.47364426 0.47442248]
5 day output [[0.47467044]]
6 day input [0.47341922 0.46497853 0.47038353 0.47149415 0.47442466 0.47781762
0.47653615 0.47364426 0.47442248 0.47467044]
6 day output [[0.47518066]]
7 day input [0.46497853 0.47038353 0.47149415 0.47442466 0.47781762 0.47653615
0.47364426 0.47442248 0.47467044 0.47518066]
7 day output [[0.47546706]]
8 day input [0.47038353 0.47149415 0.47442466 0.47781762 0.47653615 0.47364426
0.47442248 0.47467044 0.47518066 0.47546706]
8 day output [[0.4767432]]
9 day input [0.47149415 0.47442466 0.47781762 0.47653615 0.47364426 0.47442248
0.47467044 0.47518066 0.47546706 0.47674319]
9 day output [[0.47736228]]

```

```

In [ ]: day_new=np.arange(1,11)
day_pred=np.arange(11,21)
len(data_oil)
plt.plot(day_new, scaler.inverse_transform(data_oil[8206:]))
plt.plot(day_pred, scaler.inverse_transform(lst_output))

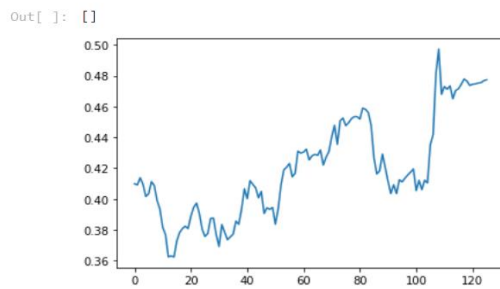
```



```

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

```



```

In [ ]: df3=scaler.inverse_transform(df3).tolist()

```

```

In [ ]: plt.plot(scaler.inverse_transform(data_oil))

```

