

MODEL BUILDING- SAVE THE MODEL

Team ID	PNT2022TMID43580
Project Name	Crude Oil Price Prediction

```
In [3]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
In [4]: data=pd.read_excel("/content/Crude Oil Prices Daily.xlsx")
```

```
In [5]: data.isnull().any()
```

```
Out[5]: Date           False
Closing Value        True
dtype: bool
```

```
In [6]: data.isnull().sum()
```

```
Out[6]: Date           0
Closing Value         7
dtype: int64
```

```
In [7]: data.dropna(axis=0,inplace=True)
```

```
In [8]: data.isnull().sum()
```

```
Out[8]: Date           0
Closing Value         0
dtype: int64
```

```
In [9]: data_oil=data.reset_index()['Closing Value']
data_oil
```

```
Out[9]: 0      25.56
1      26.00
2      26.52
3      25.85
```

```
4      25.82
...
8211   73.39
8212   72.19
8213   73.02
8214   74.78
8215   74.93
Name: Closing Value, length: 8216, dtype: float64
```

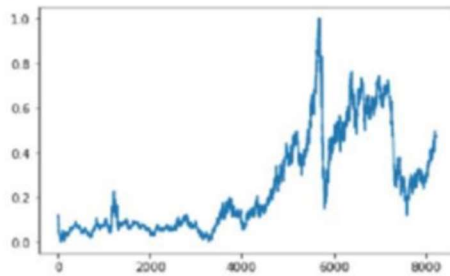
```
In [10]: from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
data_oil=scaler.fit_transform(np.array(data_oil).reshape(-1,1))
```

```
In [11]: data_oil
```

```
Out[11]: array([[0.11115781],
 [0.11661464],
 [0.12053902],
 ...,
 [0.46497853],
 [0.47038353],
 [0.47140415]])
```

```
In [12]: plt.plot(data_oil)
```

```
Out[12]: 11
```



```
In [13]: 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),:]
```

```
In [14]: training_size,test_size
```

```
Out[14]: (5340, 2879)
```

```
Out[15]: (2879, 1)
```

```
In [16]: 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 [17]: time_steps=18
x_train,y_train=create_dataset(train_data,time_step)
x_test,y_test=create_dataset(test_data,time_step)
```

```
In [18]: print(x_train.shape),print(y_train.shape)
```

```
(5320, 18)
```

```
(5320,)
```

```
Out[18]: (None, None)
```

```
In [19]: print(x_test.shape),print(y_test.shape)
```

```
(2865, 18)
```

```
(2865,)
```

```
Out[19]: (None, None)
```

```
In [20]: x_train
```

```
Out[20]: array([[0.11337703, 0.11004494, 0.12002902, ..., 0.10080000, 0.10000000,
0.11054246],
[0.11661404, 0.12051907, 0.11550672, ..., 0.10890000, 0.11054106,
0.10103852],
[0.12053902, 0.11550422, 0.1156528, ..., 0.11054346, 0.10165852,
0.09906700],
...,
[0.26731823, 0.35176058, 0.36080261, ..., 0.36391224, 0.37042706,
0.17047796],
[0.15176950, 0.16080261, 0.15154657, ..., 0.17042796, 0.17042796,
0.37879461],
[0.16001261, 0.15154657, 0.15295674, ..., 0.17042796, 0.17042796,
0.37910462]])
```

```
In [21]: 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 [22]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
```

```
In [23]: model=Sequential()
```

```
In [24]: model.add(LSTM(50,return_sequences=True,input_shape=(10,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
```

```
In [25]: model.add(Dense(1))
```

```
In [26]: 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 [27]: model.compile(loss='mean_squared_error', optimizer='adam')
```

```
In [28]: model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=3, batch_size=64, verbose=1)
```

```
Epoch 1/3  
64/64 [-----] - 10s 49ms/step - loss: 6.0018 - val_loss: 6.0018  
Epoch 2/3  
64/64 [-----] - 3s 30ms/step - loss: 1.2823e-04 - val_loss: 7.6827e-04  
Epoch 3/3  
64/64 [-----] - 1s 32ms/step - loss: 1.2129e-04 - val_loss: 7.7510e-04
```

```
Out[28]:
```

```
In [29]: 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]: 29.147839442169938
```

```
In [30]: from tensorflow.keras.models import load_model
```

```
In [31]: model.save("crude oil.h5")
```

WARNING: Saved model contains 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.

```
In [32]:
```