SPRINT 3

DATE	8 NOVEMBER 2022
TEAM ID	PNT2022TMID10129
PROJECT TITLE	CRUDE OIL PRICE PREDICTION

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import datetime
from pylab import rcParams
import matplotlib.pyplot as plt
import warnings
import itertools
import statsmodels.api as sm
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from sklearn.metrics import mean_squared_error
from keras.callbacks import ReduceLROnPlateau, EarlyStopping, ModelCheckpoint
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import seaborn as sns
sns.set_context("paper", font_scale=1.3)
sns.set_style('white')
import math
from sklearn.preprocessing import MinMaxScaler
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the
input directory
warnings.filterwarnings("ignore")
plt.style.use('fivethirtyeight')
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

IMPORTING DATA

In [2]:

```
dateparse = lambda x: pd.datetime.strptime(x, '%b %d, %Y')
#Read csv file
from google.colab import files
uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Crude Oil Prices Daily.xlsx to Crude Oil Prices Daily.xlsx

In [8]:

```
import io
df = pd.read_excel(io.BytesIO(uploaded['Crude Oil Prices Daily.xlsx']))
df.head()
df[:10]
```

Out[8]:

	Date	Closing Value
0	1986-01-02	25.56
1	1986-01-03	26.00
2	1986-01-06	26.53
3	1986-01-07	25.85
4	1986-01-08	25.87
5	1986-01-09	26.03
6	1986-01-10	25.65
7	1986-01-13	25.08
8	1986-01-14	24.97
9	1986-01-15	25.18

In [9]:

```
#Sort dataset by column Date
```

```
df = df.sort_values('Date')
df = df.groupby('Date')['Closing Value'].sum().reset_index()
df.set_index('Date', inplace=True)
df=df.loc[datetime.date(year=2000,month=1,day=1):]
```

In [10]:

```
df.head()
```

Out[10]:

	Closing Value
Date	
2000-01-04	25.56
2000-01-05	24.65
2000-01-06	24.79
2000-01-07	24.79
2000-01-10	24.71

DATA PRE-PROCESSING

In [11]:

```
def DfInfo(df_initial):
   # gives some infos on columns types and numer of null values
   tab_info = pd.DataFrame(df_initial.dtypes).T.rename(index={0: 'column type'})
   tab_info = tab_info.append(pd.DataFrame(df_initial.isnull().sum()).T.rename(index={0: 'null
values (nb)'}))
   tab_info = tab_info.append(pd.DataFrame(df_initial.isnull().sum() / df_initial.shape[0] *
100).T.
                         rename(index={0: 'null values (%)'}))
   return tab_info
```

In [12]:

```
DfInfo(df)
```

Out[12]:

	Closing Value
column type	float64
null values (nb)	0
null values (%)	0.0

In [13]:

```
df.index
```

Out[13]:

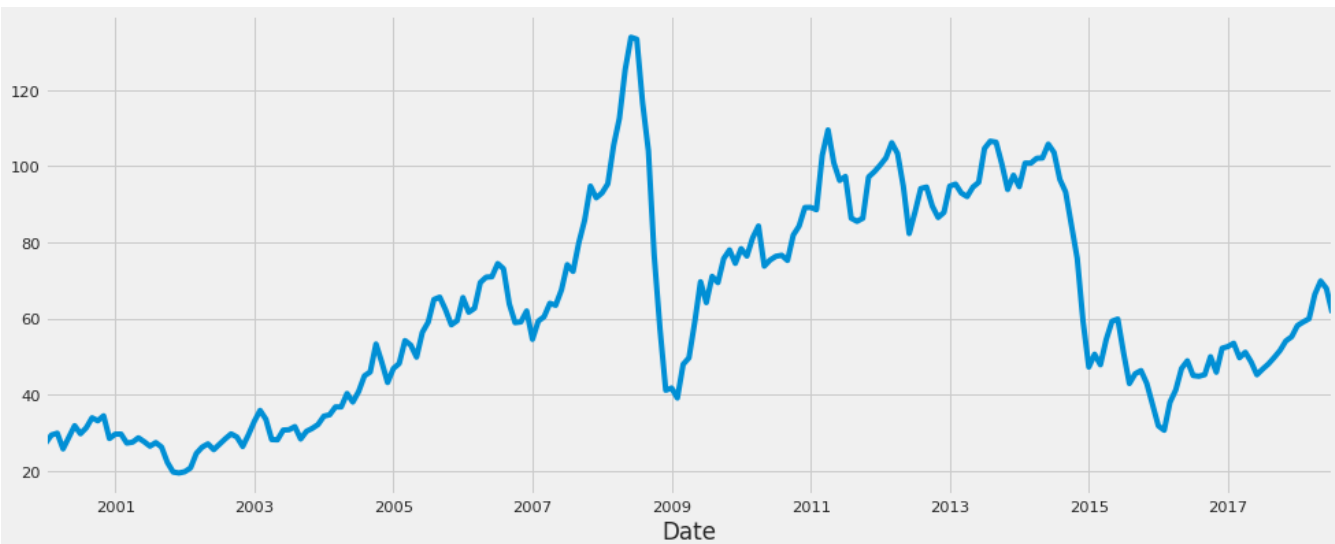
```
DatetimeIndex(['2000-01-04', '2000-01-05', '2000-01-06', '2000-01-07',
               '2000-01-10', '2000-01-11', '2000-01-12', '2000-01-13',
               '2000-01-14', '2000-01-18',
               ...
               '2018-06-26', '2018-06-27', '2018-06-28', '2018-06-29',
               '2018-07-02', '2018-07-03', '2018-07-04', '2018-07-05',
               '2018-07-06', '2018-07-09'],
              dtype='datetime64[ns]', name='Date', length=4673, freq=None)
```

In [14]:

```
y = df['Closing Value'].resample('MS').mean()
```

In [15]:

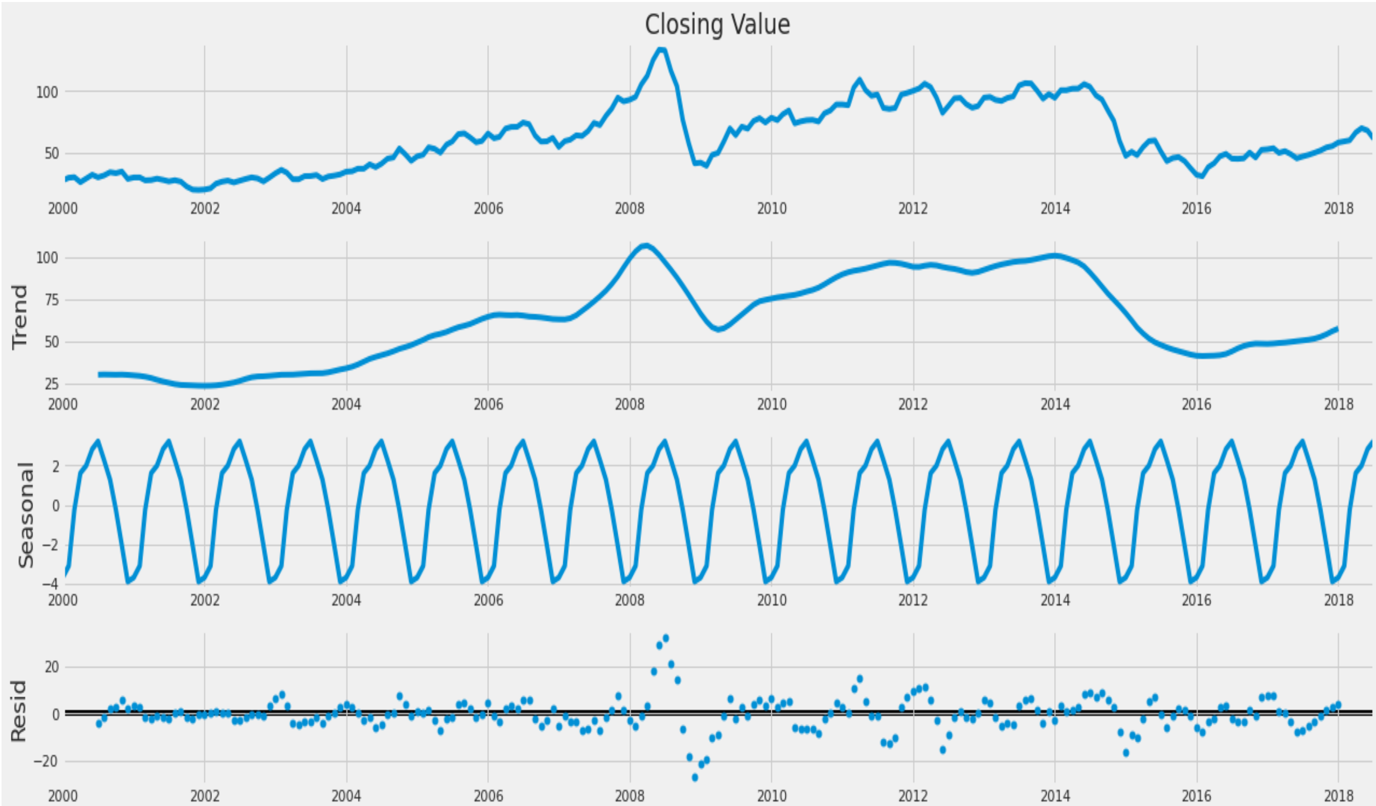
```
y.plot(figsize=(15, 6))
plt.show()
```



In [16]:

```
rcParams['figure.figsize'] = 18, 8
decomposition = sm.tsa.seasonal_decompose(y, model='additive')
fig = decomposition.plot()
```

```
plt.show()
```



In [17]:

```
sc = MinMaxScaler(feature_range = (0, 1))
df = sc.fit_transform(df)
```

TRAINING AND TESTING

In [18]:

```
train_size = int(len(df) * 0.70)
test_size = len(df) - train_size
train, test = df[0:train_size, :], df[train_size:len(df), :]
```

In [19]:

```
def create_data_set(_data_set, _look_back=1):
    data_x, data_y = [], []
    for i in range(len(_data_set) - _look_back - 1):
        a = _data_set[i:(i + _look_back), 0]
        data_x.append(a)
        data_y.append(_data_set[i + _look_back, 0])
    return np.array(data_x), np.array(data_y)
```

In [20]:

```
look_back =90
X_train,Y_train,X_test,Ytest = [],[],[],[]
X_train,Y_train=create_data_set(train,look_back)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test,Y_test=create_data_set(test,look_back)
```

```
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

LSTM LAYER

In [21]:

```
regressor = Sequential()

regressor.add(LSTM(units = 60, return_sequences = True, input_shape = (X_train.shape[1], 1)))
regressor.add(Dropout(0.1))

regressor.add(LSTM(units = 60, return_sequences = True))
regressor.add(Dropout(0.1))

regressor.add(LSTM(units = 60))
regressor.add(Dropout(0.1))

regressor.add(Dense(units = 1))


regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
reduce_lr = ReduceLROnPlateau(monitor='val_loss',patience=5)
history =regressor.fit(X_train, Y_train, epochs = 20, batch_size = 15,validation_data=(X_test,
Y_test), callbacks=[reduce_lr],shuffle=False)
```

```
Epoch 1/20
212/212 [==============================] - 23s 88ms/step - loss: 0.0047 - val_loss: 0.
0251 - lr: 0.0010
Epoch 2/20
212/212 [==============================] - 17s 82ms/step - loss: 0.0122 - val_loss: 0.
0478 - lr: 0.0010
Epoch 3/20
212/212 [==============================] - 17s 82ms/step - loss: 0.0115 - val_loss: 0.
0505 - lr: 0.0010
Epoch 4/20
212/212 [==============================] - 17s 81ms/step - loss: 0.0163 - val_loss: 0.
0461 - lr: 0.0010
Epoch 5/20
212/212 [==============================] - 19s 91ms/step - loss: 0.0193 - val_loss: 0.
0461 - lr: 0.0010
Epoch 6/20
212/212 [==============================] - 17s 82ms/step - loss: 0.0174 - val_loss: 0.
0605 - lr: 0.0010
Epoch 7/20
212/212 [==============================] - 18s 83ms/step - loss: 0.0275 - val_loss: 0.
0047 - lr: 1.0000e-04
Epoch 8/20
212/212 [==============================] - 18s 83ms/step - loss: 0.0040 - val_loss: 0.
0032 - lr: 1.0000e-04
```

```
Epoch 9/20
212/212 [==============================] - 17s 82ms/step - loss: 0.0029 - val_loss: 0.
0021 - lr: 1.0000e-04
Epoch 10/20
212/212 [==============================] - 17s 81ms/step - loss: 0.0023 - val_loss: 0.
0017 - lr: 1.0000e-04
Epoch 11/20
212/212 [==============================] - 17s 83ms/step - loss: 0.0020 - val_loss: 0.
0016 - lr: 1.0000e-04
Epoch 12/20
212/212 [==============================] - 17s 82ms/step - loss: 0.0016 - val_loss: 0.
0015 - lr: 1.0000e-04
Epoch 13/20
212/212 [==============================] - 17s 83ms/step - loss: 0.0014 - val_loss: 0.
0014 - lr: 1.0000e-04
Epoch 14/20
212/212 [==============================] - 18s 83ms/step - loss: 0.0013 - val_loss: 0.
0014 - lr: 1.0000e-04
Epoch 15/20
212/212 [==============================] - 18s 83ms/step - loss: 0.0012 - val_loss: 0.
0013 - lr: 1.0000e-04
Epoch 16/20
212/212 [==============================] - 18s 84ms/step - loss: 0.0011 - val_loss: 0.
0014 - lr: 1.0000e-04
Epoch 17/20
212/212 [==============================] - 18s 86ms/step - loss: 0.0011 - val_loss: 0.
0014 - lr: 1.0000e-04
Epoch 18/20
212/212 [==============================] - 19s 87ms/step - loss: 0.0011 - val_loss: 0.
0015 - lr: 1.0000e-04
Epoch 19/20
212/212 [==============================] - 17s 82ms/step - loss: 0.0011 - val_loss: 0.
0013 - lr: 1.0000e-05
Epoch 20/20
212/212 [==============================] - 18s 83ms/step - loss: 0.0010 - val_loss: 0.
0013 - lr: 1.0000e-05
```

MODEL TRAINING

In [22]:

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

```
100/100 [==============================] - 4s 27ms/step
41/41 [==============================] - 1s 28ms/step
```

In [23]:

```
train_predict = sc.inverse_transform(train_predict)
Y_train = sc.inverse_transform([Y_train])
test_predict = sc.inverse_transform(test_predict)
```

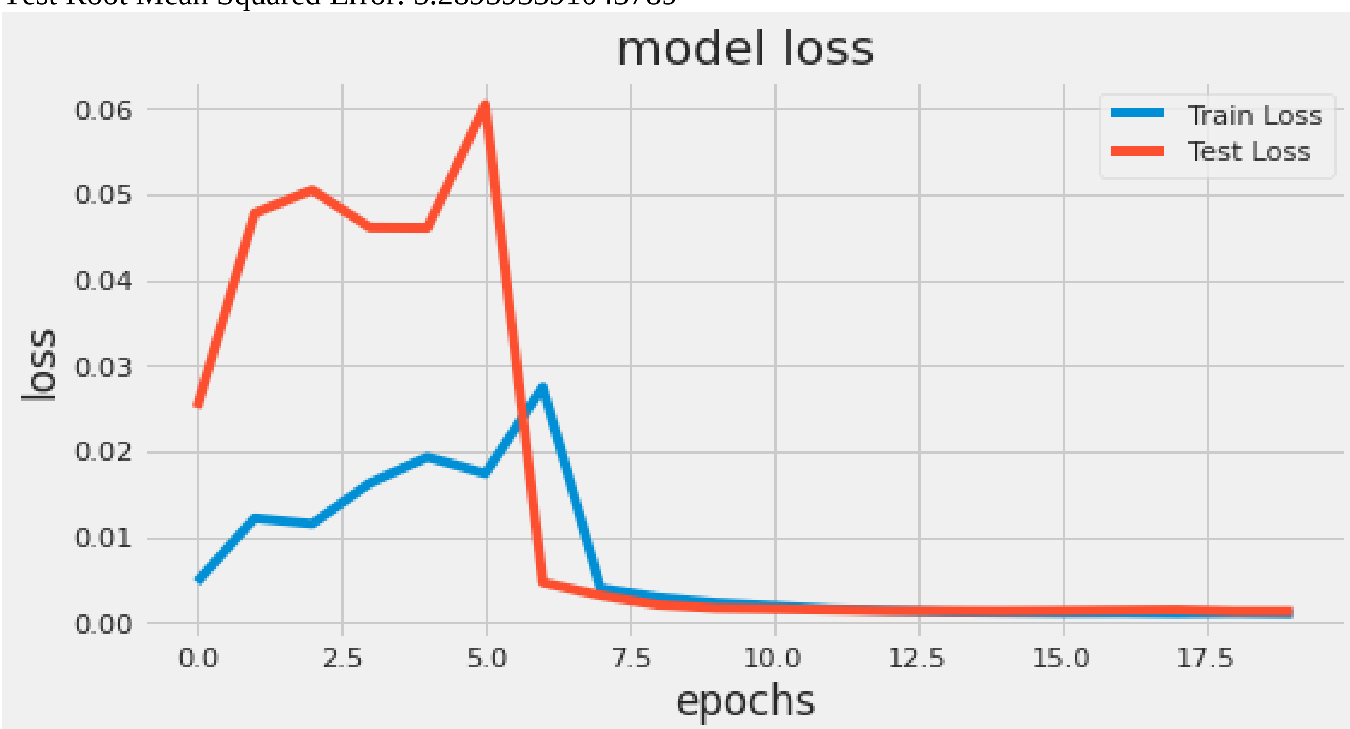
Y_test = sc.inverse_transform([Y_test])

PREDICTION

In [24]:

```
print('Train Mean Absolute Error:', mean_absolute_error(Y_train[0], train_predict[:,0]))
print('Train Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_train[0],
train_predict[:,0])))
print('Test Mean Absolute Error:', mean_absolute_error(Y_test[0], test_predict[:,0]))
print('Test Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_test[0], test_predict[:,0])))
plt.figure(figsize=(8,4))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Test Loss')
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')
plt.legend(loc='upper right')
plt.show();
```

Train Mean Absolute Error: 2.3165036988408305
Train Root Mean Squared Error: 3.285617879896689
Test Mean Absolute Error: 2.3989636110004624
Test Root Mean Squared Error: 5.289593391043789



In [25]:

```
aa=[x for x in range(180)]
plt.figure(figsize=(8,4))
plt.plot(aa, Y_test[0][:180], marker='.', label="actual")
plt.plot(aa, test_predict[:,0][:180], 'r', label="prediction")
```

```
plt.tight_layout()
sns.despine(top=True)
plt.subplots_adjust(left=0.07)
plt.ylabel('Price', size=15)
plt.xlabel('Time step', size=15)
plt.legend(fontsize=15)
plt.show();
```