SPRINT 4

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))
```

In [ ]:

```
dateparse = lambda x: pd.datetime.strptime(x, '%b %d, %Y')
#Read csv file
```

```
from google.colab import files
uploaded = files.upload()
df = pd.read_csv('BrentOilPrices.csv',parse_dates=['Date'], date_parser=dateparse)
#Sort dataset by column Date
df = df.sort_values('Date')
df = df.groupby('Date')['Price'].sum().reset_index()
df.set_index('Date', inplace=True)
df=df.loc[datetime.date(year=2000,month=1,day=1):]
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving BrentOilPrices.csv to BrentOilPrices (1).csv

In [ ]:

```
df.head()
```

Out[ ]:

```
              Price

      Date

2000-01-04    23.95

2000-01-05    23.72

2000-01-06    23.55

2000-01-07    23.35

2000-01-10    22.77
```

In [ ]:

```
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 [ ]:

```
DfInfo(df)
```

Out[ ]:

	Price
column type	float64
null values (nb)	0
null values (%)	0.0

In [ ]:

```
df.index
```

Out[ ]:

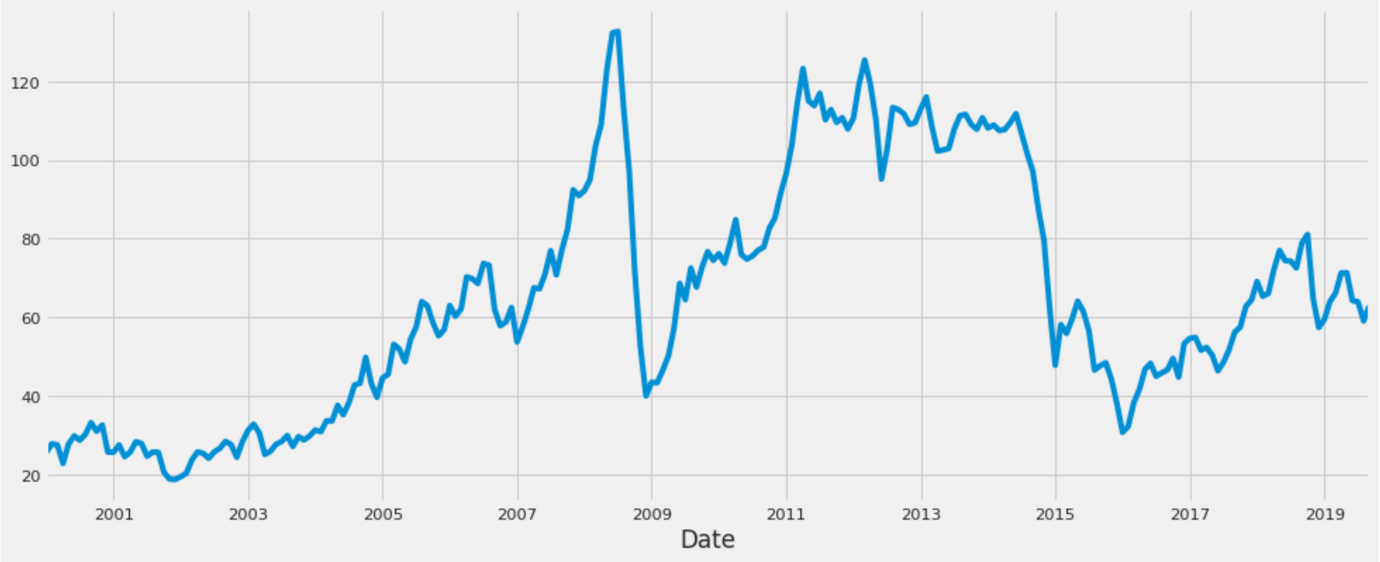
```
DatetimeIndex(['2000-01-04', '2000-01-05', '2000-01-06', '2000-01-07',
               '2000-01-10', '2000-01-11', '2000-01-12', '2000-01-13',
               '2000-01-14', '2000-01-17',
               ...
               '2019-09-17', '2019-09-18', '2019-09-19', '2019-09-20',
               '2019-09-23', '2019-09-24', '2019-09-25', '2019-09-26',
               '2019-09-27', '2019-09-30'],
              dtype='datetime64[ns]', name='Date', length=5016, freq=None)
```

In [ ]:

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

In [ ]:

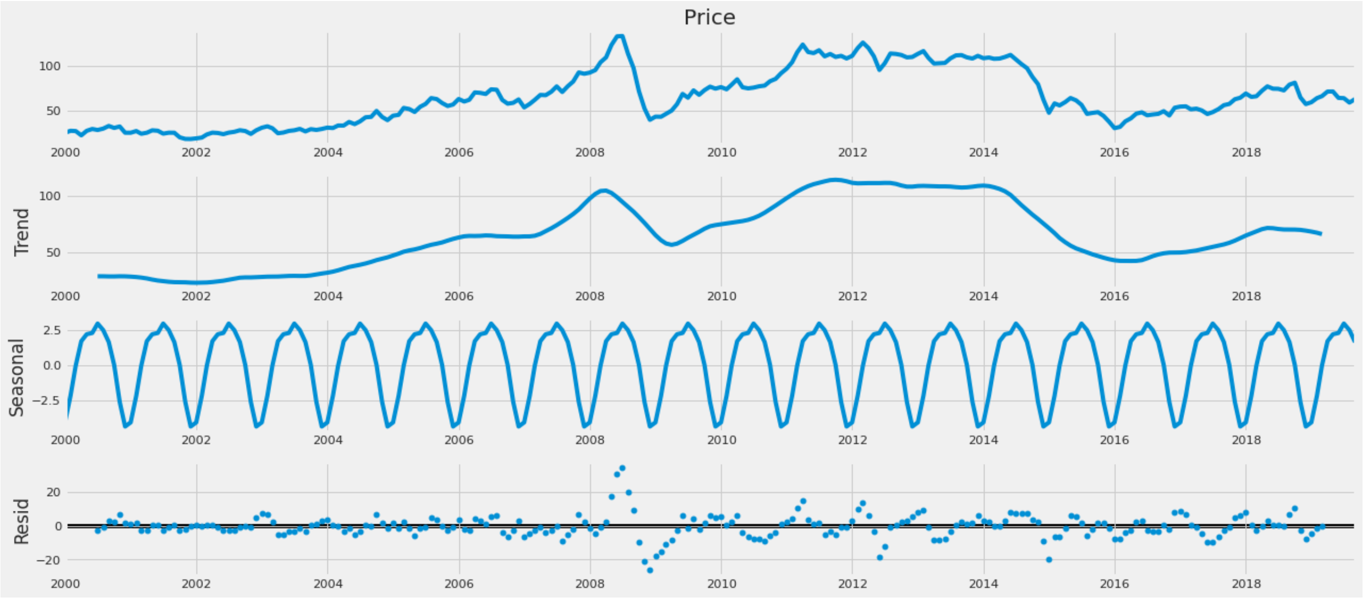
```
y.plot(figsize=(15, 6))
plt.show()
```



In [ ]:

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

```
plt.show()
```



In [ ]:

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

In [ ]:

```
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 [ ]:

```
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 [ ]:

```
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))
```

In [ ]:

```
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
228/228 [==============================] - 28s 101ms/step - loss: 0.0053 - val_loss: 0.0737 - lr:
0.0010
Epoch 2/20
228/228 [==============================] - 22s 96ms/step - loss: 0.0142 - val_loss: 0.1127 - lr: 0
.0010
Epoch 3/20
228/228 [==============================] - 21s 93ms/step - loss: 0.0241 - val_loss: 0.1022 - lr: 0
.0010
Epoch 4/20
228/228 [==============================] - 22s 95ms/step - loss: 0.0191 - val_loss: 0.0532 - lr: 0
.0010
Epoch 5/20
228/228 [==============================] - 22s 96ms/step - loss: 0.0044 - val_loss: 0.0023 - lr: 0
.0010
Epoch 6/20
228/228 [==============================] - 21s 93ms/step - loss: 0.0018 - val_loss: 0.0028 - lr: 0
.0010
Epoch 7/20
228/228 [==============================] - 23s 100ms/step - loss: 0.0016 - val_loss: 0.0040 - lr:
0.0010
Epoch 8/20
228/228 [==============================] - 21s 91ms/step - loss: 0.0015 - val_loss: 0.0040 - lr: 0
.0010
Epoch 9/20
228/228 [==============================] - 21s 92ms/step - loss: 0.0015 - val_loss: 0.0074 - lr: 0
.0010
Epoch 10/20
228/228 [==============================] - 21s 92ms/step - loss: 0.0017 - val_loss: 0.0043 - lr: 0
.0010
Epoch 11/20
228/228 [==============================] - 21s 93ms/step - loss: 0.0014 - val_loss: 4.9764e-04 -
lr: 1.0000e-04
Epoch 12/20
```

```
228/228 [==============================] - 22s 97ms/step - loss: 0.0011 - val_loss: 4.0066e-04 - lr: 1.0000e-04
Epoch 13/20
228/228 [==============================] - 21s 93ms/step - loss: 9.6524e-04 - val_loss: 3.3321e-04 - lr: 1.0000e-04
Epoch 14/20
228/228 [==============================] - 21s 92ms/step - loss: 9.6356e-04 - val_loss: 2.8505e-04 - lr: 1.0000e-04
Epoch 15/20
228/228 [==============================] - 21s 92ms/step - loss: 9.2024e-04 - val_loss: 2.7974e-04 - lr: 1.0000e-04
Epoch 16/20
228/228 [==============================] - 21s 93ms/step - loss: 9.1895e-04 - val_loss: 2.7104e-04 - lr: 1.0000e-04
Epoch 17/20
228/228 [==============================] - 22s 96ms/step - loss: 8.2996e-04 - val_loss: 2.7737e-04 - lr: 1.0000e-04
Epoch 18/20
228/228 [==============================] - 21s 93ms/step - loss: 8.1935e-04 - val_loss: 2.6434e-04 - lr: 1.0000e-04
Epoch 19/20
228/228 [==============================] - 21s 94ms/step - loss: 8.7719e-04 - val_loss: 2.7174e-04 - lr: 1.0000e-05
Epoch 20/20
228/228 [==============================] - 21s 92ms/step - loss: 8.3641e-04 - val_loss: 2.6845e-04 - lr: 1.0000e-05
```

In [ ]:

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

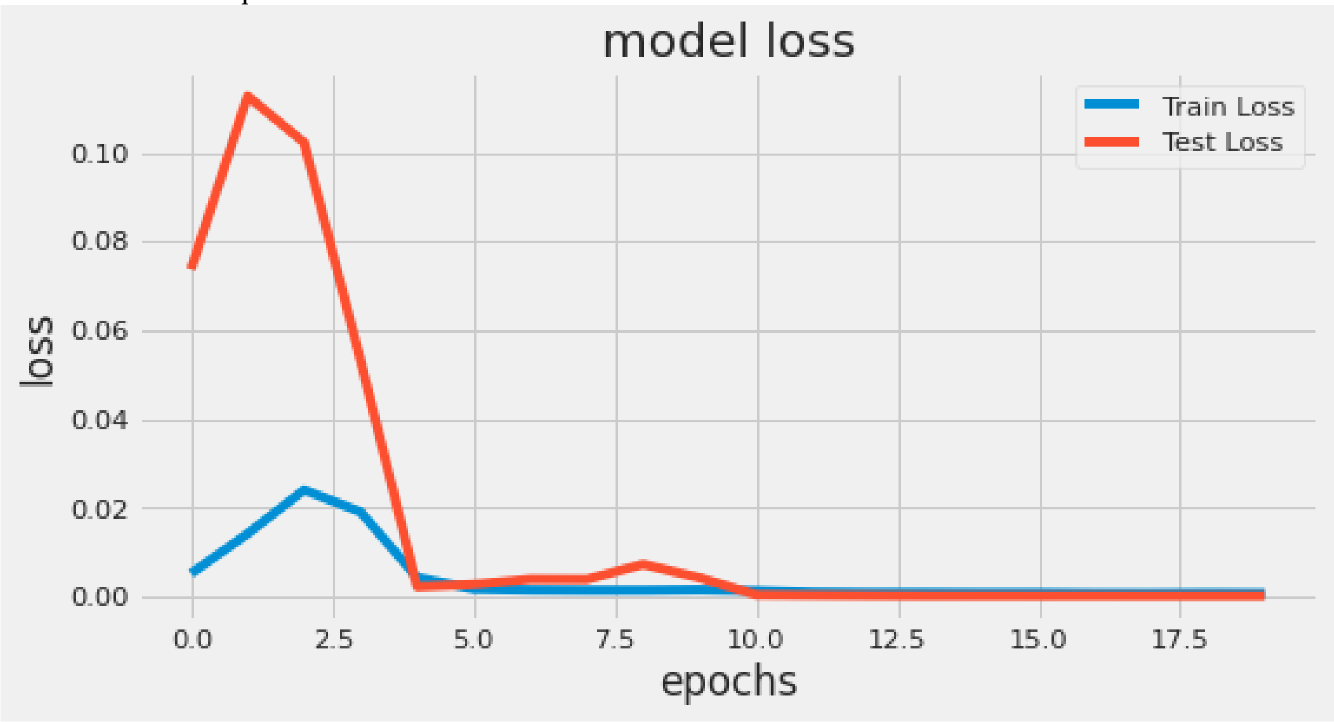
In [ ]:

```
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])
```

In [ ]:

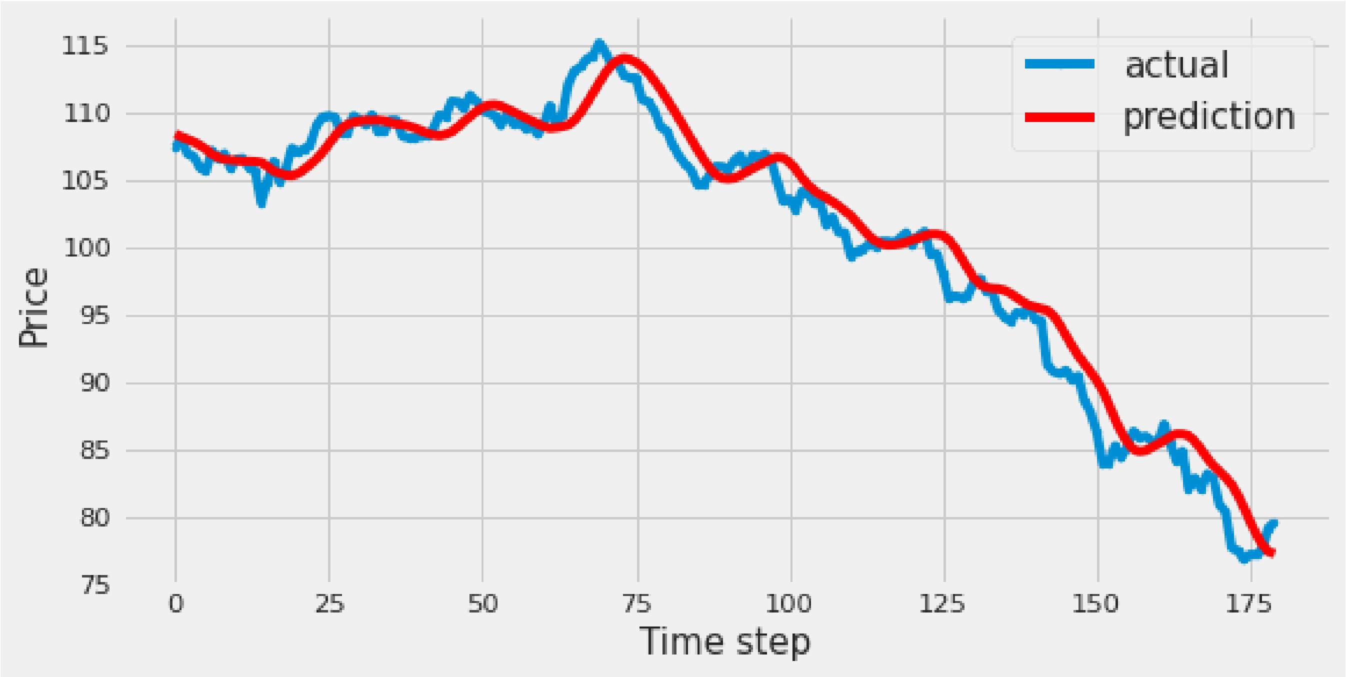
```
print('Train Mean Absolute Error:', mean_absolute_error(Y_train[0], train_predict[:,0]))
print('Train Root Mean Squared Error:',np.sqrt(mean_squared_error(Y_train[0], train_predict[:,0])))
print('Test Mean Absolute Error:', mean_absolute_error(Y_test[0], test_predict[:,0]))
print('Test Root Mean 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: 1.8387156992906715
Train Root Mean Squared Error: 2.4879726857036757
Test Mean Absolute Error: 1.6482952979599061
Test Root Mean Squared Error: 2.0880482671332192
```



In [ ]:

```
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();
```



actual
prediction
Price
115
110
105
100
95
90
85
80
75
0
25
50
75
100
125
150
175
Time step