```
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from datetime 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 GRU
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))

# Any results you write to the current directory are saved as output.

import pandas as pd
from datetime import datetime

# Read the CSV file
df = pd.read_csv(r'BrentOilPrices1.csv', parse_dates=['Date'])

# Convert date columns to specific format
df['Date'] = pd.to_datetime(df['Date'], format='%d-%b-%y')

# Sort dataset by column Date
```

```
df = df.sort_values('Date')

# Group by Date and sum Prices
df = df.groupby('Date')['Price'].sum().reset_index()

# Set Date as index
df.set_index('Date', inplace=True)

# Filter the DataFrame for dates after 2000-01-01
df = df.loc[datetime.strptime('2000-01-01', "%Y-%m-%d"):]


# Print some data rows.
df.head()
```

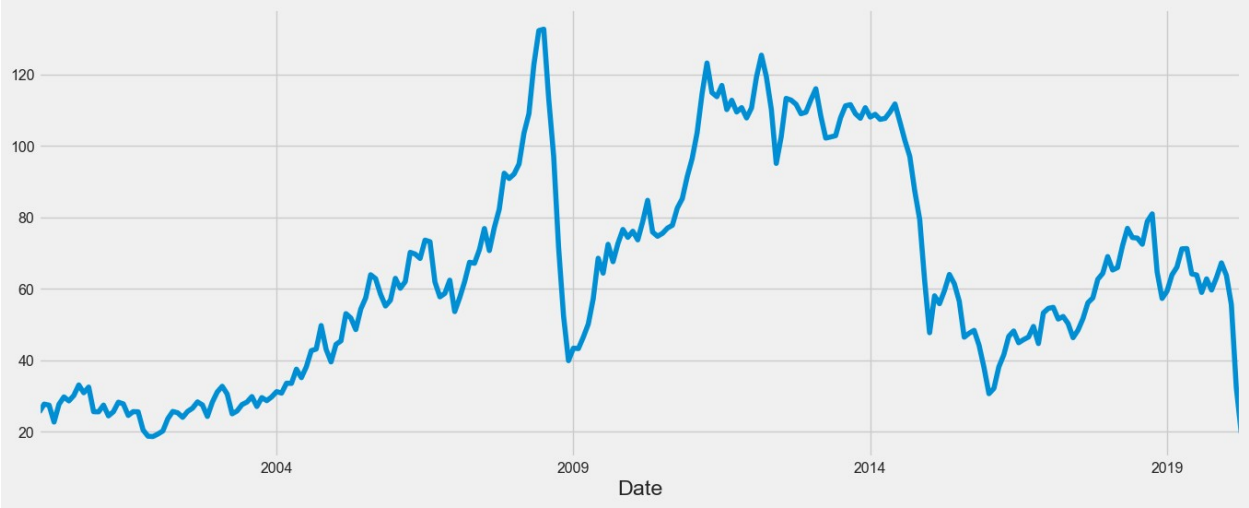
```
            Price
Date
2000-01-04  23.95
2000-01-05  23.72
2000-01-06  23.55
2000-01-07  23.35
2000-01-10  22.77
```

```
#Read dataframe info
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

df.index
```

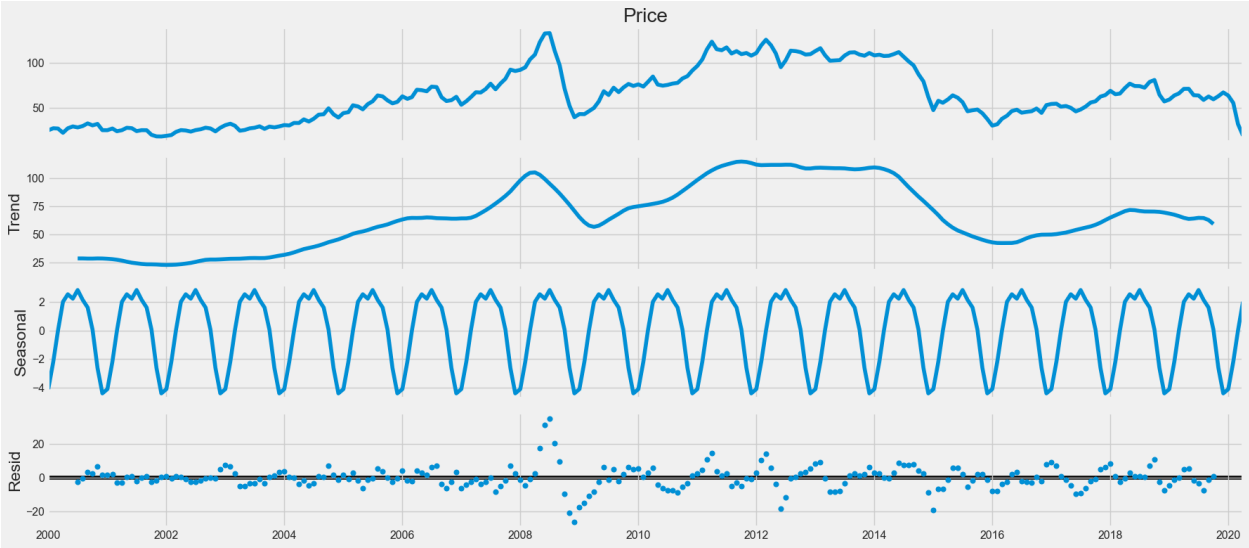
```
DatetimeIndex(['2000-01-04', '2000-01-05', '2000-01-06', '2000-01-07',
               '2000-01-10', '2000-01-11', '2000-01-12', '2000-01-13',
               '2000-01-14', '2000-01-17',
               ...
               '2020-04-06', '2020-04-07', '2020-04-08', '2020-04-09',
               '2020-04-14', '2020-04-15', '2020-04-16', '2020-04-17',
               '2020-04-20', '2020-04-21'],
              dtype='datetime64[ns]', name='Date', length=5160,
freq=None)
```

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

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




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




```
# normalize the data_set
sc = MinMaxScaler(feature_range = (0, 1))
df = sc.fit_transform(df)

# split into train and test sets
train_size = int(len(df) * 0.80)
test_size = len(df) - train_size
train, test = df[0:train_size, :], df[train_size:len(df), :]
```

```
# convert an array of values into a data_set matrix def
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)

# reshape into X=t and Y=t+1
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))

# create and fit the LSTM network regressor = Sequential()
regressor = Sequential()

regressor.add(GRU(units = 60, return_sequences = True, input_shape =
(X_train.shape[1], 1)))
# regressor.add(GRU(4, return_sequences=True, return_state=True)
# regressor.add(Dropout(0.1))

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

regressor.add(GRU(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 =
20,validation_data=(X_test, Y_test),
callbacks=[reduce_lr],shuffle=False)

Epoch 1/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 39s 140ms/step - loss: 0.0042 - val_loss:
0.0260 - learning_rate: 0.0010
Epoch 2/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 28s 138ms/step - loss: 0.0043 - val_loss:
0.0227 - learning_rate: 0.0010
Epoch 3/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 28s 138ms/step - loss: 0.0055 - val_loss:
0.0192 - learning_rate: 0.0010
Epoch 4/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 29s 145ms/step - loss: 0.0055 - val_loss:
```

```
0.0163 - learning_rate: 0.0010
Epoch 5/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 30s 149ms/step - loss: 0.0058 - val_loss:
0.0147 - learning_rate: 0.0010
Epoch 6/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 28s 141ms/step - loss: 0.0047 - val_loss:
0.0110 - learning_rate: 0.0010
Epoch 7/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 28s 138ms/step - loss: 0.0035 - val_loss:
0.0086 - learning_rate: 0.0010
Epoch 8/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 30s 146ms/step - loss: 0.0022 - val_loss:
0.0015 - learning_rate: 0.0010
Epoch 9/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 30s 148ms/step - loss: 2.6799e-04 -
val_loss: 9.9408e-04 - learning_rate: 0.0010
Epoch 10/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 28s 136ms/step - loss: 1.9317e-04 -
val_loss: 0.0010 - learning_rate: 0.0010
Epoch 11/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 29s 144ms/step - loss: 1.8746e-04 -
val_loss: 9.1422e-04 - learning_rate: 0.0010
Epoch 12/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 40s 140ms/step - loss: 1.7405e-04 -
val_loss: 8.9549e-04 - learning_rate: 0.0010
Epoch 13/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 29s 145ms/step - loss: 1.7506e-04 -
val_loss: 9.1547e-04 - learning_rate: 0.0010
Epoch 14/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 28s 139ms/step - loss: 1.8443e-04 -
val_loss: 9.5748e-04 - learning_rate: 0.0010
Epoch 15/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 28s 138ms/step - loss: 2.0428e-04 -
val_loss: 1.5533e-04 - learning_rate: 1.0000e-04
Epoch 16/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 28s 141ms/step - loss: 1.1454e-04 -
val_loss: 1.4115e-04 - learning_rate: 1.0000e-04
Epoch 17/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 29s 143ms/step - loss: 9.5814e-05 -
val_loss: 1.4007e-04 - learning_rate: 1.0000e-04
Epoch 18/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 28s 136ms/step - loss: 9.1937e-05 -
val_loss: 1.3950e-04 - learning_rate: 1.0000e-04
Epoch 19/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 28s 137ms/step - loss: 8.8829e-05 -
val_loss: 1.3936e-04 - learning_rate: 1.0000e-04
Epoch 20/20
202/202 ━━━━━━━━━━━━━━━━━━━━ 28s 138ms/step - loss: 8.6300e-05 -
val_loss: 1.3965e-04 - learning_rate: 1.0000e-04
```

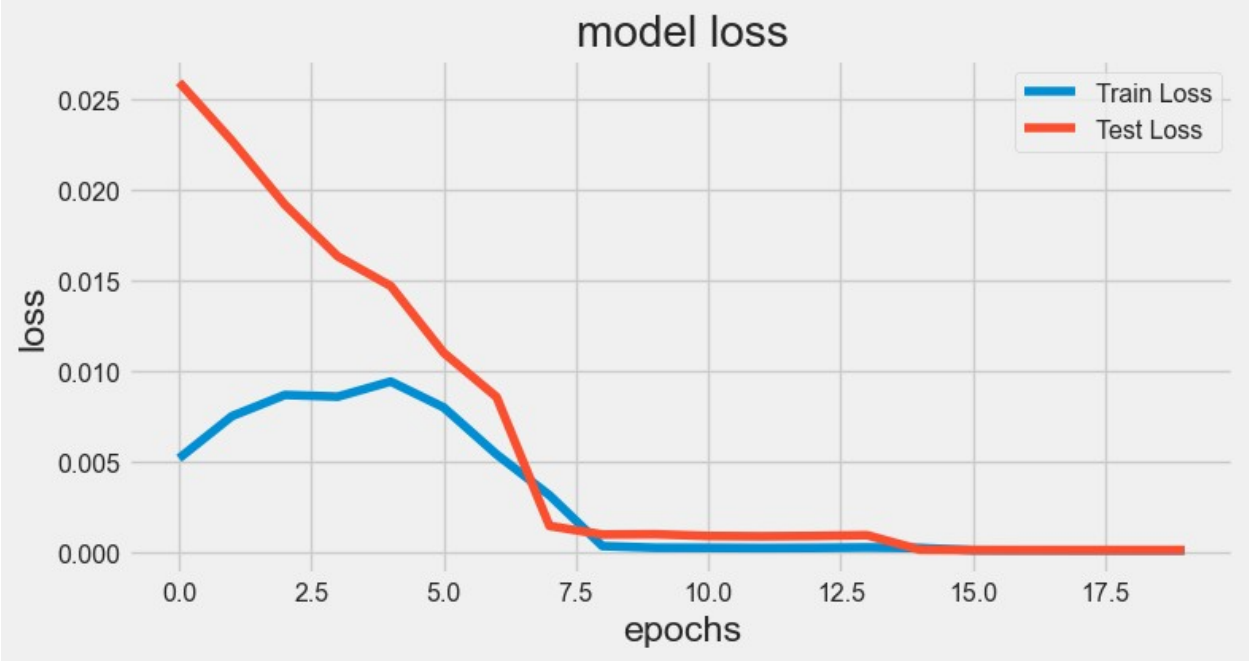
```
train_predict = regressor.predict(X_train)
test_predict = regressor.predict(X_test)
```

```
127/127 ━━━━━━━━━━━━━━━━━━━━ 9s 61ms/step
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step
```

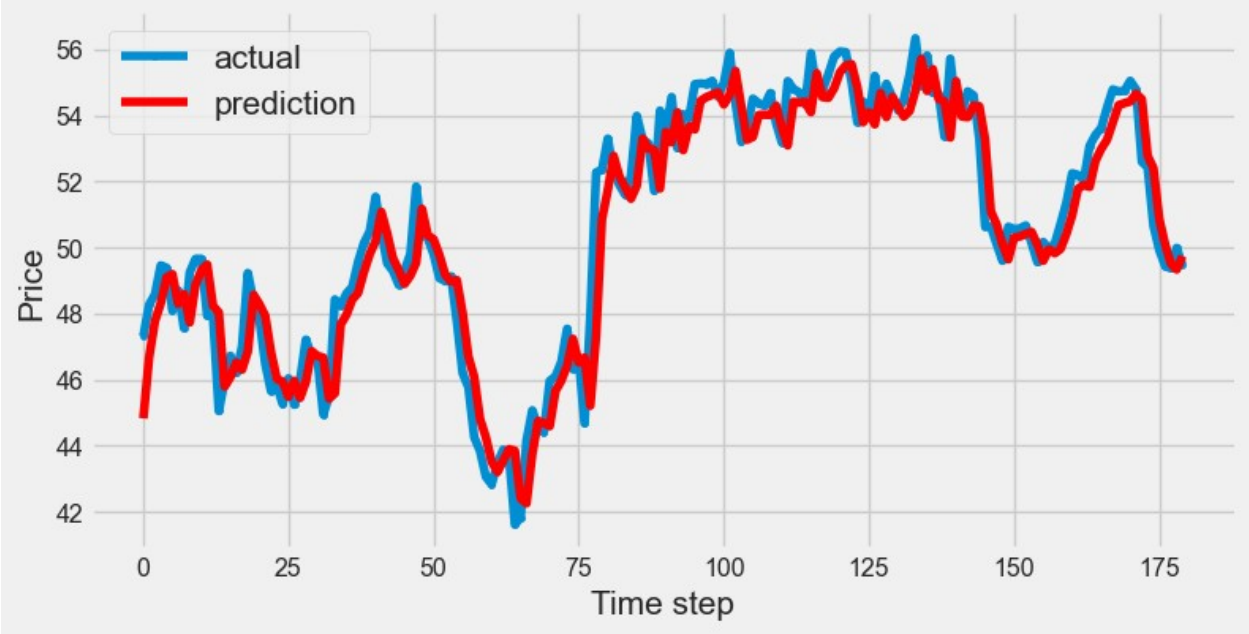
```
# invert predictions
train_predict = sc.inverse_transform(train_predict)
Y_train = sc.inverse_transform([Y_train])
test_predict = sc.inverse_transform(test_predict)
Y_test = sc.inverse_transform([Y_test])

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])))
```

```
Train Mean Absolute Error: 1.3122656631375231
Train Root Mean Squared Error: 1.6542822303001408
Test Mean Absolute Error: 1.0776522967412538
Test Root Mean Squared Error: 1.4699781183192673
```

```
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();
```




```
#Compare Actual vs. Prediction
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(p)
```


actual
prediction
Price
Time step
56
54
52
50
48
46
44
42
0
25
50
75
100
125
150
175