Data Preprocessing

Import the libraries

n [1]:
```
!pip install ibm-cos-sdk | grep -v 'already satisfied'
import ibm_boto3
from ibm_botocore.client import Config
import pandas as pd
import numpy as np
import io, datetime
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from pylab import rcParams
from sklearn.preprocessing import MinMaxScaler
```

Importing the dataset

n [2]:
```
# The code was removed by Watson Studio for sharing.
```

ut[2]:

	date	price
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

Handling missing data

In [3]:
```
df.isnull().any()
```

Out[3]:
```
date     False
price     True
dtype: bool
```

In [4]:
```
df.dropna(axis=0,inplace=True)
df.isnull().any()
```

Out[4]:
```
date     False
price    False
dtype: bool
```

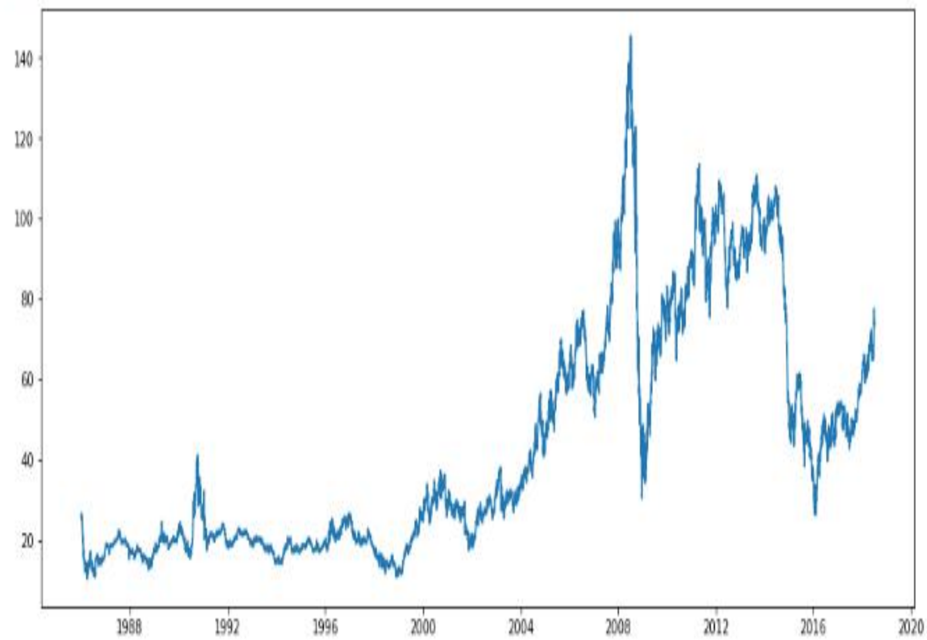
In [5]:
```
df.shape
```

Out[5]: (8216, 2)

Data visualization

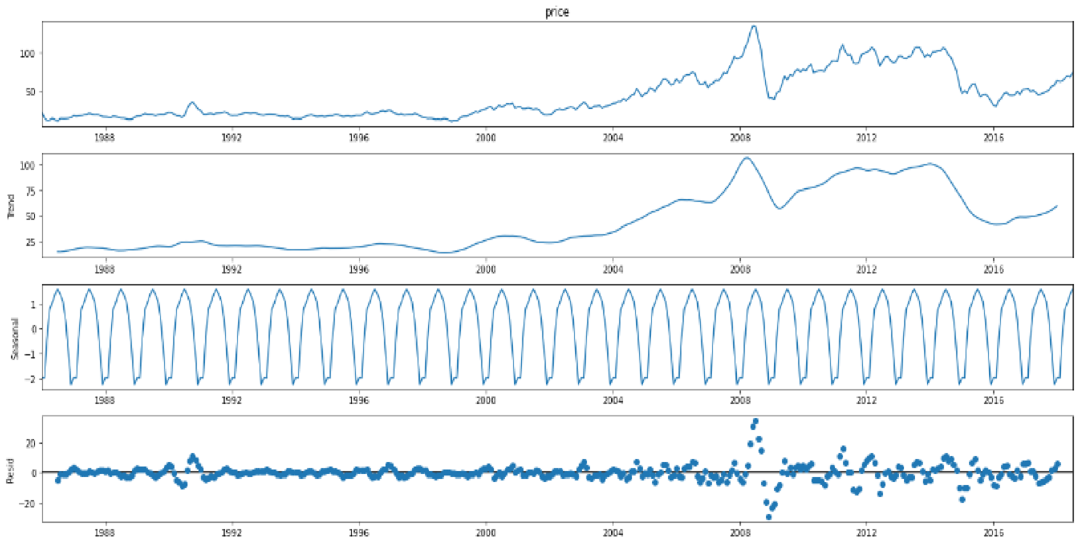
In [6]:
```
plot = plt.figure(figsize=(15, 6))
time = pd.to_datetime(df['date'])
price = list(df['price'])
data = pd.Series(price, time)
plt.plot(data)
```

Out[6]: [1



In [7]:

```
#Decompose the plot
df.set_index('date', inplace=True)
y = df['price'].resample('MS').mean()
y.plot(figsize=(15, 6))
plt.show()
```



Feature Scaling

In [9]:
```python
df1 = df.reset_index()['price']
sc = MinMaxScaler(feature_range = (0, 1))
df1 = sc.fit_transform(np.array(df1).reshape(-1,1))
```

In [10]:
```python
df1.shape
```

Out[10]: (8216, 1)

Train Test Split

In [11]:
```python
train_size = int(len(df1) * 0.80)
test_size = len(df1) - train_size
train, test = df1[0:train_size, :], df1[train_size:len(df1), :]
```

In [12]:
```python
len(test)
```

Out[12]: 1644

Creating Window

In [13]:
```python
def dataset(df, lookback=1):
    data_x, data_y = [], []
    for i in range(len(df) - lookback - 1):
        a = df[i:(i + lookback), 0]
        data_x.append(a)
        data_y.append(df[i + lookback, 0])
    return np.array(data_x), np.array(data_y)

time_step = 10
# Reshape into X=t and Y=t+1
X_train , Y_train = dataset(train,time_step)
X_test , Y_test = dataset(test,time_step)
# Reshape input to be [samples, time steps, features]
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 [14]:
```python
X_train.shape
```

Out[14]: (6561, 10, 1)

Model Building

Import the Model building libraries

In [15]:
```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
```

Model

In [16]:
```
model = Sequential()
model.add(LSTM(units = 10, return_sequences = True, input_shape = (X_train.shape[1], 1)))
model.add(LSTM(units = 10, return_sequences = True))
model.add(LSTM(units = 10))
model.add(Dense(units = 1))
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 10, 10)            480

 lstm_1 (LSTM)               (None, 10, 10)            840

 lstm_2 (LSTM)               (None, 10)                840

 dense (Dense)               (None, 1)                 11

=================================================================
Total params: 2,171
Trainable params: 2,171
Non-trainable params: 0
_________________________________________________________________
```

In [17]:
```
# 1st LSTM Layer
print(4 * 10 * (1 + 10 + 1))
```

```
480
```

In [18]:
```
# 2nd LSTM Layer
parameters = 4 * 10 * (10 + 10 + 1)
print(parameters)
```

```
840
```

In [19]:
```
history = model.fit(X_train, Y_train, epochs = 30, batch_size = 64,validation_data=(X_test, Y_test),verbose=2)
```

```
Epoch 1/30
103/103 - 7s - loss: 0.0106 - val_loss: 5.6487e-04 - 7s/epoch - 69ms/step
Epoch 2/30
103/103 - 2s - loss: 3.9524e-04 - val_loss: 4.6745e-04 - 2s/epoch - 16ms/step
Epoch 3/30
103/103 - 2s - loss: 3.8390e-04 - val_loss: 5.1335e-04 - 2s/epoch - 16ms/step
Epoch 4/30
103/103 - 2s - loss: 3.7019e-04 - val_loss: 4.4842e-04 - 2s/epoch - 16ms/step
Epoch 5/30
103/103 - 2s - loss: 3.6652e-04 - val_loss: 4.8122e-04 - 2s/epoch - 15ms/step
Epoch 6/30
103/103 - 1s - loss: 3.5819e-04 - val_loss: 4.5359e-04 - 1s/epoch - 14ms/step
Epoch 7/30
103/103 - 1s - loss: 3.7374e-04 - val_loss: 4.3644e-04 - 1s/epoch - 14ms/step
Epoch 8/30
103/103 - 2s - loss: 3.5272e-04 - val_loss: 5.2173e-04 - 2s/epoch - 15ms/step
Epoch 9/30
103/103 - 1s - loss: 3.4195e-04 - val_loss: 5.8963e-04 - 1s/epoch - 14ms/step
Epoch 10/30
103/103 - 2s - loss: 3.3386e-04 - val_loss: 8.3337e-04 - 2s/epoch - 15ms/step
Epoch 11/30
103/103 - 2s - loss: 3.2806e-04 - val_loss: 4.3660e-04 - 2s/epoch - 15ms/step
Epoch 12/30
103/103 - 2s - loss: 3.1741e-04 - val_loss: 4.0663e-04 - 2s/epoch - 15ms/step
Epoch 13/30
103/103 - 1s - loss: 3.1387e-04 - val_loss: 3.6922e-04 - 1s/epoch - 14ms/step
Epoch 14/30
103/103 - 2s - loss: 3.1879e-04 - val_loss: 6.4365e-04 - 2s/epoch - 15ms/step
Epoch 15/30
103/103 - 1s - loss: 3.5249e-04 - val_loss: 3.6145e-04 - 1s/epoch - 15ms/step
Epoch 16/30
103/103 - 1s - loss: 2.9288e-04 - val_loss: 3.6011e-04 - 1s/epoch - 13ms/step
Epoch 17/30
103/103 - 1s - loss: 2.9086e-04 - val_loss: 4.4847e-04 - 1s/epoch - 13ms/step
Epoch 18/30
103/103 - 2s - loss: 2.8538e-04 - val_loss: 4.8739e-04 - 2s/epoch - 15ms/step
```

```
Epoch 6/30
103/103 - 1s - loss: 3.5819e-04 - val_loss: 4.5359e-04 - 1s/epoch - 14ms/step
Epoch 7/30
103/103 - 1s - loss: 3.7374e-04 - val_loss: 4.3644e-04 - 1s/epoch - 14ms/step
Epoch 8/30
103/103 - 2s - loss: 3.5272e-04 - val_loss: 5.2173e-04 - 2s/epoch - 15ms/step
Epoch 9/30
103/103 - 1s - loss: 3.4195e-04 - val_loss: 5.8963e-04 - 1s/epoch - 14ms/step
Epoch 10/30
103/103 - 2s - loss: 3.3386e-04 - val_loss: 8.3337e-04 - 2s/epoch - 15ms/step
Epoch 11/30
103/103 - 2s - loss: 3.2806e-04 - val_loss: 4.3660e-04 - 2s/epoch - 15ms/step
Epoch 12/30
103/103 - 2s - loss: 3.1741e-04 - val_loss: 4.0663e-04 - 2s/epoch - 15ms/step
Epoch 13/30
103/103 - 1s - loss: 3.1387e-04 - val_loss: 3.6922e-04 - 1s/epoch - 14ms/step
Epoch 14/30
103/103 - 2s - loss: 3.1879e-04 - val_loss: 6.4365e-04 - 2s/epoch - 15ms/step
Epoch 15/30
103/103 - 1s - loss: 3.5249e-04 - val_loss: 3.6145e-04 - 1s/epoch - 15ms/step
Epoch 16/30
103/103 - 1s - loss: 2.9288e-04 - val_loss: 3.6011e-04 - 1s/epoch - 13ms/step
Epoch 17/30
103/103 - 1s - loss: 2.9086e-04 - val_loss: 4.4847e-04 - 1s/epoch - 13ms/step
Epoch 18/30
103/103 - 2s - loss: 2.8538e-04 - val_loss: 4.8739e-04 - 2s/epoch - 15ms/step
Epoch 19/30
103/103 - 2s - loss: 2.9127e-04 - val_loss: 4.6284e-04 - 2s/epoch - 17ms/step
Epoch 20/30
103/103 - 1s - loss: 2.7772e-04 - val_loss: 3.7055e-04 - 1s/epoch - 14ms/step
Epoch 21/30
103/103 - 1s - loss: 2.8659e-04 - val_loss: 3.6304e-04 - 1s/epoch - 14ms/step
Epoch 22/30
103/103 - 1s - loss: 2.6882e-04 - val_loss: 3.4819e-04 - 1s/epoch - 15ms/step
Epoch 23/30
103/103 - 1s - loss: 2.6004e-04 - val_loss: 3.3309e-04 - 1s/epoch - 14ms/step
Epoch 24/30
103/103 - 1s - loss: 2.5006e-04 - val_loss: 3.7485e-04 - 1s/epoch - 13ms/step
Epoch 25/30
103/103 - 2s - loss: 2.3378e-04 - val_loss: 3.1101e-04 - 2s/epoch - 15ms/step
Epoch 26/30
103/103 - 2s - loss: 2.4116e-04 - val_loss: 3.9566e-04 - 2s/epoch - 17ms/step
Epoch 27/30
103/103 - 2s - loss: 2.3316e-04 - val_loss: 4.5380e-04 - 2s/epoch - 15ms/step
Epoch 28/30
103/103 - 2s - loss: 2.1976e-04 - val_loss: 2.7630e-04 - 2s/epoch - 15ms/step
Epoch 29/30
103/103 - 2s - loss: 2.1748e-04 - val_loss: 2.3854e-04 - 2s/epoch - 15ms/step
Epoch 30/30
103/103 - 1s - loss: 2.1277e-04 - val_loss: 6.7326e-04 - 1s/epoch - 14ms/step
```

Train the model

In [20]:

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

In [21]:

```
# 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])
```

Model evaluation

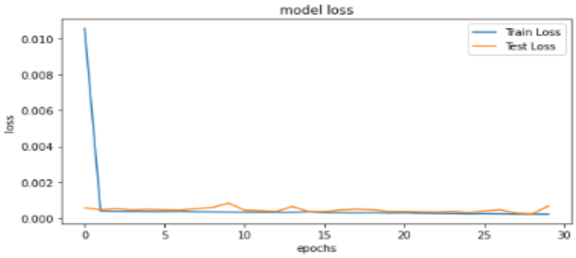
In [22]:

```
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: 1.4820543204355379
Train Root Mean Squared Error: 2.4348844667718192
Test Mean Absolute Error: 2.8878097367388884
Test Root Mean Squared Error: 3.504429960912005
```

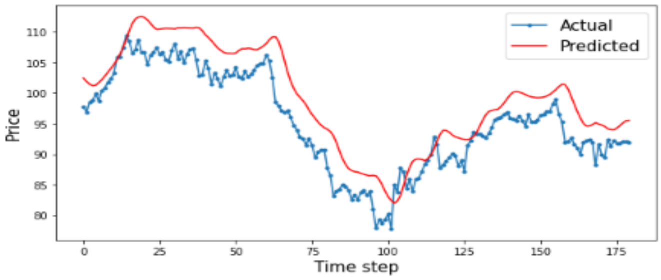
In [23]:

```
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();
```



In [24]:

```
data = [i for i in range(180)]
plt.figure(figsize=(8,4))
plt.plot(data, Y_test[0][:180], marker='.', label="Actual")
plt.plot(data, test_predict[:,0][:180], 'r', label="Predicted")
plt.tight_layout()
plt.subplots_adjust(left=0.07)
plt.ylabel('Price', size=15)
plt.xlabel('Time step', size=15)
plt.legend(fontsize=15)
plt.show();
```



Save the model

In [25]:

```
model.save("model.h5")
!tar -zcvf model.tgz model.h5
```

```
model.h5
```

IBM WATSON Deployment

In [26]:

```
!pip install ibm_watson_machine_learning watson-machine-learning-client | grep -v 'already satisfied'
```

```
Collecting watson-machine-learning-client
  Downloading watson_machine_learning_client-1.0.391-py3-none-any.whl (538 kB)
Installing collected packages: watson-machine-learning-client
Successfully installed watson-machine-learning-client-1.0.391
```

In [27]:

```
# The code was removed by Watson Studio for sharing.
```

In [28]:

```
def guid_from_space_name(client,space_name):
    space = client.spaces.get_details()
    return(next(item for item in space['resources'] if item['entity']["name"] == space_name)['metadata']['id'])
```

In [29]:

```
space_uid = guid_from_space_name(client,'model')
print("Space - {}".format(space_uid))
```

```
Space - 0570a6bf-fbb3-4149-b9fd-3a7d7008a00b
```

In [30]:

```
client.set.default_space(space_uid)
```

Out[30]: 'SUCCESS'

In [31]:

```
client.software_specifications.list()
```

```
----------------------------  ------------------------------------  ----
```

```
NAME                           ASSET_ID                              TYPE
default_py3.6                  0062b8c9-8b7d-44a0-a9b9-46c416adcbd9  base
kernel-spark3.2-scala2.12      020d69ce-7ac1-5e68-ac1a-31189867356a  base
pytorch-onnx_1.3-py3.7-edt     069ea134-3346-5748-b513-49120e15d288  base
scikit-learn_0.20-py3.6        09c5a1d0-9c1e-4473-a344-eb7b665ff687  base
spark-mllib_3.0-scala_2.12     09f4cff0-90a7-5899-b9ed-1ef348aebdee  base
pytorch-onnx_rt22.1-py3.9      0b848dd4-e681-5599-be41-b5f6fccc6471  base
ai-function_0.1-py3.6          0cdb0f1e-5376-4f4d-92dd-da3b69aa9bda  base
shiny-r3.6                     0e6e79df-875e-4f24-8ae9-62dcc2148306  base
tensorflow_2.4-py3.7-horovod   1092590a-307d-563d-9b62-4eb7d64b3f22  base
pytorch_1.1-py3.6              10ac12d6-6b30-4ccd-8392-3e922c096a92  base
tensorflow_1.15-py3.6-ddl      111e41b3-de2d-5422-a4d6-bf776828c4b7  base
autoai-kb_rt22.2-py3.10        125b6d9a-5b1f-5e8d-972a-b251688ccf40  base
runtime-22.1-py3.9             12b83a17-24d8-5082-900f-0ab31fbfd3cb  base
scikit-learn_0.22-py3.6        154010fa-5b3b-4ac1-82af-4d5ee5abbc85  base
default_r3.6                   1b70aec3-ab34-4b87-8aa0-a4a3c8296a36  base
pytorch-onnx_1.3-py3.6         1bc6029a-cc97-56da-b8e0-39c3880dbbe7  base
kernel-spark3.3-r3.6           1c9e5454-f216-59dd-a20e-474a5cdf5988  base
pytorch-onnx_rt22.1-py3.9-edt  1d362186-7ad5-5b59-8b6c-9d0880bde37f  base
tensorflow_2.1-py3.6           1eb25b84-d6ed-5dde-b6a5-3fbdf1665666  base
spark-mllib_3.2                20047f72-0a98-58c7-9ff5-a77b012eb8f5  base
tensorflow_2.4-py3.8-horovod   217c16f6-178f-56bf-824a-b19f20564c49  base
runtime-22.1-py3.9-cuda        26215f05-08c3-5a41-a1b0-da66306ce658  base
do_py3.8                       295addb5-9ef9-547e-9bf4-92ae3563e720  base
autoai-ts_3.8-py3.8            2aa0c932-798f-5ae9-abd6-15e0c2402fb5  base
tensorflow_1.15-py3.6          2b73a275-7cbf-420b-a912-eae7f436e0bc  base
kernel-spark3.3-py3.9          2b7961e2-e3b1-5a8c-a491-482c8368839a  base
pytorch_1.2-py3.6              2c8ef57d-2687-4b7d-acce-01f94976dac1  base
spark-mllib_2.3                2e51f700-bca0-4b0d-88dc-5c6791338875  base
pytorch-onnx_1.1-py3.6-edt     32983cea-3f32-4400-8965-dde874a8d67e  base
spark-mllib_3.0-py37           36507ebe-8770-55ba-ab2a-eafe787600e9  base
spark-mllib_2.4                390d21f8-e58b-4fac-9c55-d7ceda621326  base
autoai-ts_rt22.2-py3.10        396b2e83-0953-5b86-9a55-7ce1628a406f  base
xgboost_0.82-py3.6             39e31acd-5f30-41dc-ae44-60233c80306e  base
pytorch-onnx_1.2-py3.6-edt     40589d0e-7019-4e28-8daa-fb03b6f4fe12  base
pytorch-onnx_rt22.2-py3.10     40e73f55-783a-5535-b3fa-0c8b94291431  base
default_r36py38                41c247d3-45f8-5a71-b065-8580229facf0  base
autoai-ts_rt22.1-py3.9         4269d26e-07ba-5d40-8f66-2d495b0c71f7  base
autoai-obm_3.0                 42b92e18-d9ab-567f-988a-4240ba1ed5f7  base
pmml-3.0_4.3                   493bcb95-16f1-5bc5-bee8-81b8af80e9c7  base
spark-mllib_2.4-r_3.6          49403dff-92e9-4c87-a3d7-a42d0021c095  base
xgboost_0.90-py3.6             4ff8d6c2-1343-4c18-85e1-689c965304d3  base
pytorch-onnx_1.1-py3.6         50f95b2a-bc16-43bb-bc94-b0bed208c60b  base
autoai-ts_3.9-py3.8            52c57136-80fa-572e-8728-a5e7cbb42cde  base
spark-mllib_2.4-scala_2.11     55a70f99-7320-4be5-9fb9-9edb5a443af5  base
spark-mllib_3.0                5c1b0ca2-4977-5c2e-9439-ffd44ea8ffe9  base
autoai-obm_2.0                 5c2e37fa-80b8-5e77-840f-d912469614ee  base
spss-modeler_18.1              5c3cad7e-507f-4b2a-a9a3-ab53a21dee8b  base
cuda-py3.8                     5d3232bf-c86b-5df4-a2cd-7bb870a1cd4e  base
```

Note: Only first 50 records were displayed. To display more use 'limit' parameter.

In [32]:

```
software_spec_uid = client.software_specifications.get_uid_by_name("tensorflow_rt22.1-py3.9")
software_spec_uid
```

Out[32]: 'acd9c798-6974-5d2f-a657-ce06e986df4d'

In [33]:

```
model_details = client.repository.store_model(model = "model.tgz",meta_props={
    client.repository.ModelMetaNames.NAME:"CrudeOil Price Prediction",
    client.repository.ModelMetaNames.TYPE:"tensorflow_2.7",
    client.repository.ModelMetaNames.SOFTWARE_SPEC_UID:software_spec_uid
})
model_id = client.repository.get_model_id(model_details)
```

In [34]:

```
model_id = client.repository.get_model_id(model_details)
model_id
```

Out[34]: 'ee7e58da-643d-4526-b30c-7ae6c3b783a9'

In [35]:

```
client.repository.download(model_id,'ibm_model.h5')
```

Successfully saved model content to file: 'ibm_model.h5'

Out[35]: '/home/wsuser/work/ibm_model.h5'

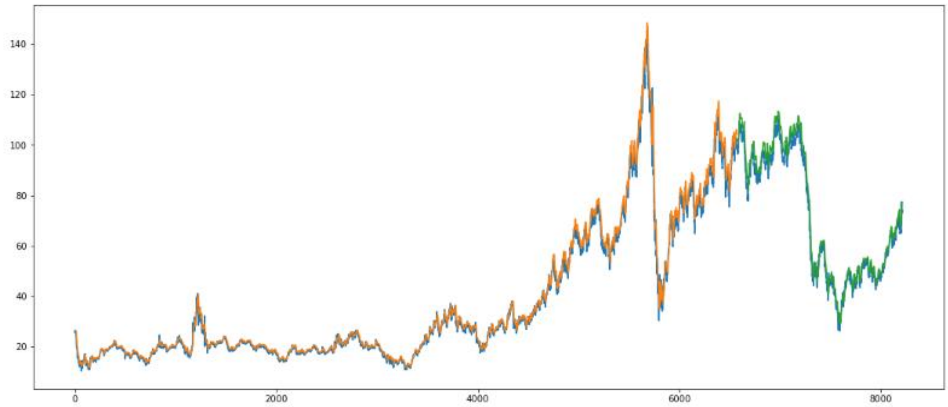
Test the model

In [36]:

```
# Model Testing
look_back = 10
trainPredictPlot = np.empty_like(df1)
trainPredictPlot[:,:] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back,:] = train_predict
testPredictPlot = np.empty_like(df1)
testPredictPlot[:,:] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df1)-1,:] = test_predict
plt.plot(sc.inverse_transform(df1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

Test the model

In [36]:
```
# Model Testing
look_back = 10
trainPredictPlot = np.empty_like(df1)
trainPredictPlot[:,:] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back,:] = train_predict
testPredictPlot = np.empty_like(df1)
testPredictPlot[:,:] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df1)-1,:] = test_predict
plt.plot(sc.inverse_transform(df1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



In [37]:
```
x_input = test[len(test)-10:].reshape(1,-1)
x_input.shape
```

Out[37]: (1, 10)

In [38]:
```
temp_input = list(x_input[0])
temp_list = temp_input[0].tolist()
```

In [39]:
```
temp_input
```

Out[39]:
```
[0.44172960165852215,
 0.48111950244335855,
 0.49726047682511476,
 0.4679401747371539,
 0.4729749740855915,
 0.47119798608026064,
 0.47341922108692425,
 0.4649785280616022,
 0.4703835332444839,
 0.47149415074781587]
```

In [40]:
```
lst_output = []
n_steps = 10
i = 0
while(i<10):
  if(len(temp_input) > 10):
    x_input = np.array(temp_input[1:])
    print("Day {} Input {}".format(i,x_input),end="\n")
    x_input = x_input.reshape(1,-1)
    x_input = x_input.reshape((1,n_steps,1))
    yhat = model.predict(x_input,verbose=0)
    print("Day {} Output {}".format(i,yhat),end="\n")
    temp_input.extend(yhat[0].tolist())
    temp_input = temp_input[1:]
    print("-----------------------",end="\n")
    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("Day {} output {}".format(i,yhat),end="\n")
    temp_input.extend(yhat[0].tolist())
    lst_output.extend(yhat.tolist())
    i = i+1
```

```
Day 0 output [[0.4920351]]
Day 1 Input [0.4811195  0.49726048 0.46794017 0.47297497 0.47119799 0.47341922
 0.46497853 0.47038353 0.47149415 0.49203509]
Day 1 Output [[0.491485]]
------------------------
Day 2 Input [0.49726048 0.46794017 0.47297497 0.47119799 0.47341922 0.46497853
 0.47038353 0.47149415 0.49203509 0.491485  ]
Day 2 Output [[0.4946947]]
------------------------
Day 3 Input [0.46794017 0.47297497 0.47119799 0.47341922 0.46497853 0.47038353
 0.47149415 0.49203509 0.491485   0.49469471]
Day 3 Output [[0.5004281]]
------------------------
Day 4 Input [0.47297497 0.47119799 0.47341922 0.46497853 0.47038353 0.47149415
 0.49203509 0.491485   0.49469471 0.50042808]
Day 4 Output [[0.5058997]]
------------------------
Day 5 Input [0.47119799 0.47341922 0.46497853 0.47038353 0.47149415 0.49203509
 0.491485   0.49469471 0.50042808 0.50589973]
Day 5 Output [[0.5120742]]
------------------------
Day 6 Input [0.47341922 0.46497853 0.47038353 0.47149415 0.49203509 0.491485
 0.49469471 0.50042808 0.50589973 0.51207417]
Day 6 Output [[0.51838446]]
------------------------
Day 7 Input [0.46497853 0.47038353 0.47149415 0.49203509 0.491485   0.49469471
 0.50042808 0.50589973 0.51207417 0.51838446]
Day 7 Output [[0.52484316]]
------------------------
Day 8 Input [0.47038353 0.47149415 0.49203509 0.491485   0.49469471 0.50042808
 0.50589973 0.51207417 0.51838446 0.52484316]
Day 8 Output [[0.5308133]]
------------------------
Day 9 Input [0.47149415 0.49203509 0.491485   0.49469471 0.50042808 0.50589973
 0.51207417 0.51838446 0.52484316 0.53081328]
Day 9 Output [[0.5368741]]
------------------------
```