Crude Oil Price Prediction Using IBM Watson Studio

1. INTRODUCTION

1.1 Overview

Oil demand is inelastic, therefore the rise in price is good news for producers because they will see an increase in their revenue. Oil importers, however, will experience increased costs of purchasing oil. Because oil is the largest traded commodity, the effects are quite significant. A rising oil price can even shift economic/political power from oil importers to oil exporters. The crude oil price movements are subject to diverse influencing factors.

1.2 Purpose

This Project mainly focuses on applying Neural Networks to predict the Crude Oil Price. This decision helps us to buy crude oil at the proper time. Time series analysis is the best option for this kind of prediction because we are using the Previous history of crude oil prices to predict future crude oil. So we would be implementing RNN(Recurrent Neural Network) with LSTM(Long Short Term Memory) to achieve the task.

2. LITERATURE SURVEY

2.1 Existing problem

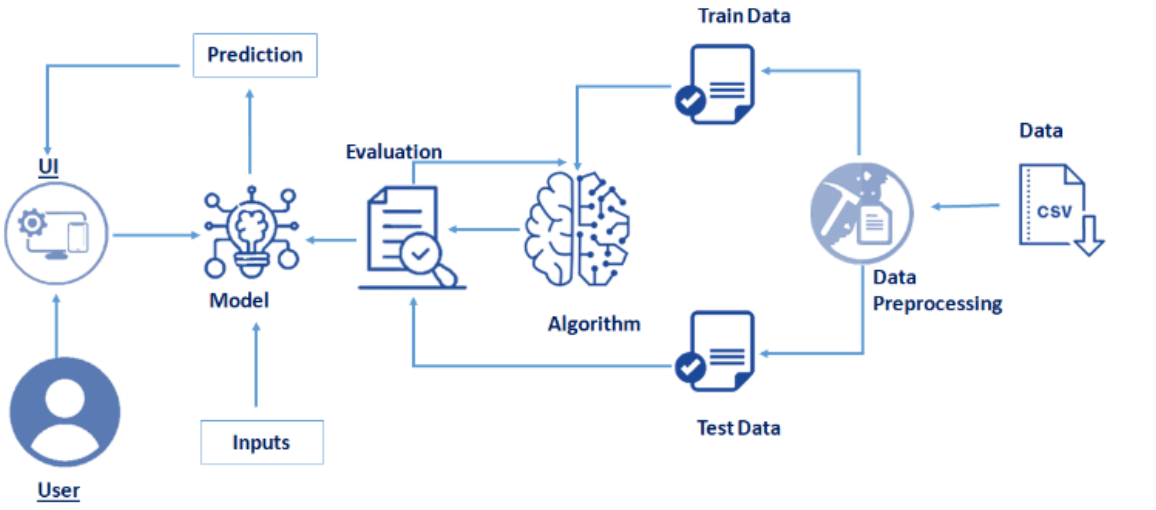
The internal and external environment of the oil market is changing, and the influencing factors have become diverse and complex. As the factors affecting international oil prices become more and more complex, it becomes difficult to capture practical factors and predict oil prices. Oil demand is inelastic, therefore the rise in price is good news for producers because they will see an increase in their revenue. Oil importers, however, will experience increased costs of purchasing oil. Because oil is the largest traded commodity, the effects are quite significant.

2.2 Proposed solution

This Project mainly focuses on applying Neural Networks to predict the Crude Oil Price. This decision helps us to buy crude oil at the proper time. Time series analysis is the best option for this kind of prediction because we are using the Previous history of crude oil prices to predict future crude oil. So we would be implementing RNN(Recurrent Neural Network) with LSTM(Long Short Term Memory) to achieve the task.

3. THEORITICAL ANALYSIS

3.1 Block Diagram



3.2 Hardware / Software designing

Software Requirements:

- Anaconda Navigator
- Tensor flow
- Keras
- Flask

Hardware Requirements:

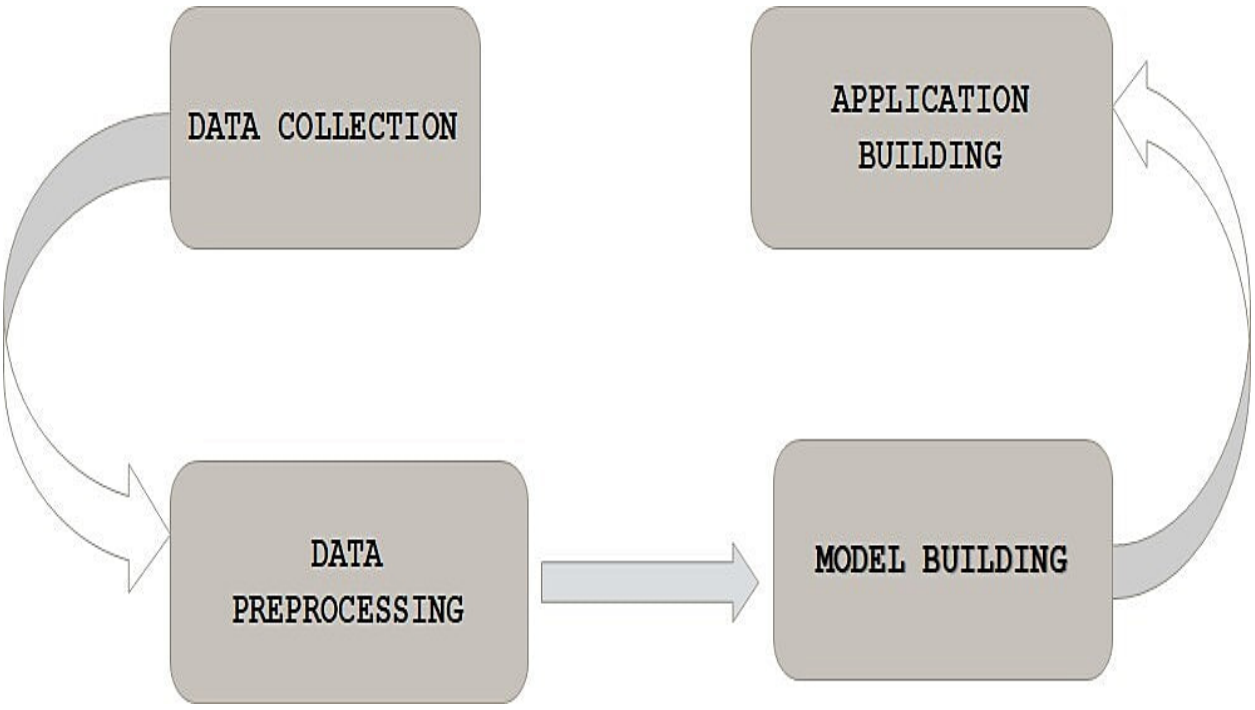
- Processor : Intel Core i3
- Hard Disk Space : Min 100 GB

- Ram : 4 GB
- Display : 14.1 “Color Monitor(LCD, CRT or LED)
- Clock Speed : 1.67 GHz

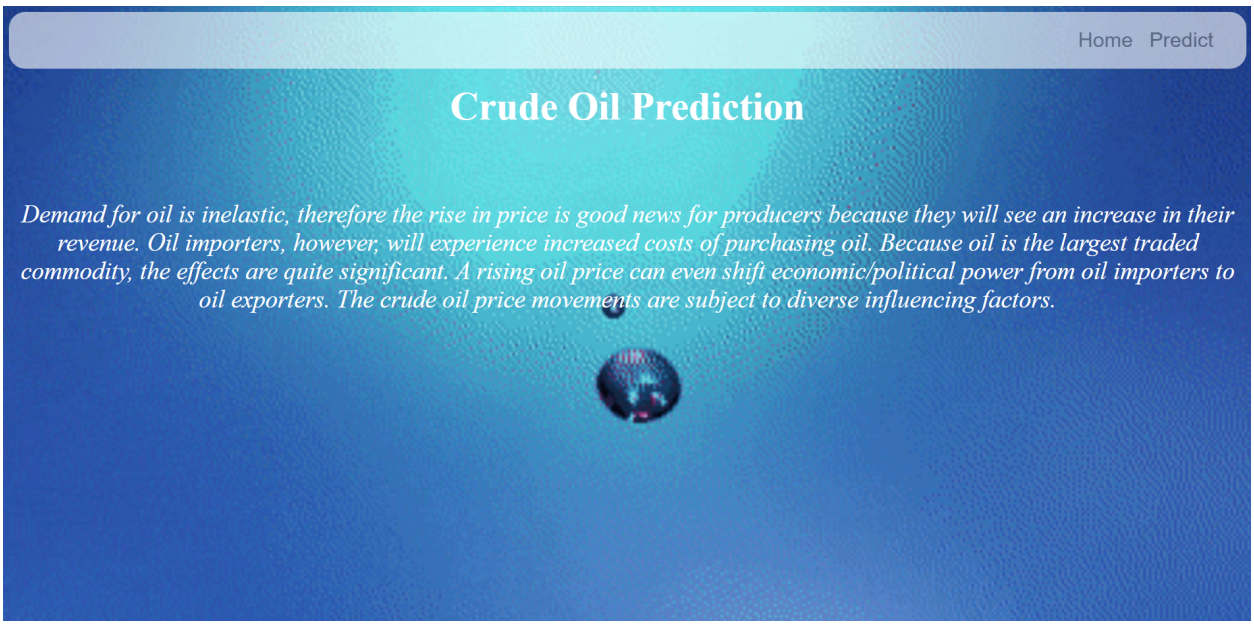
4. EXPERIMENTAL INVESTIGATIONS

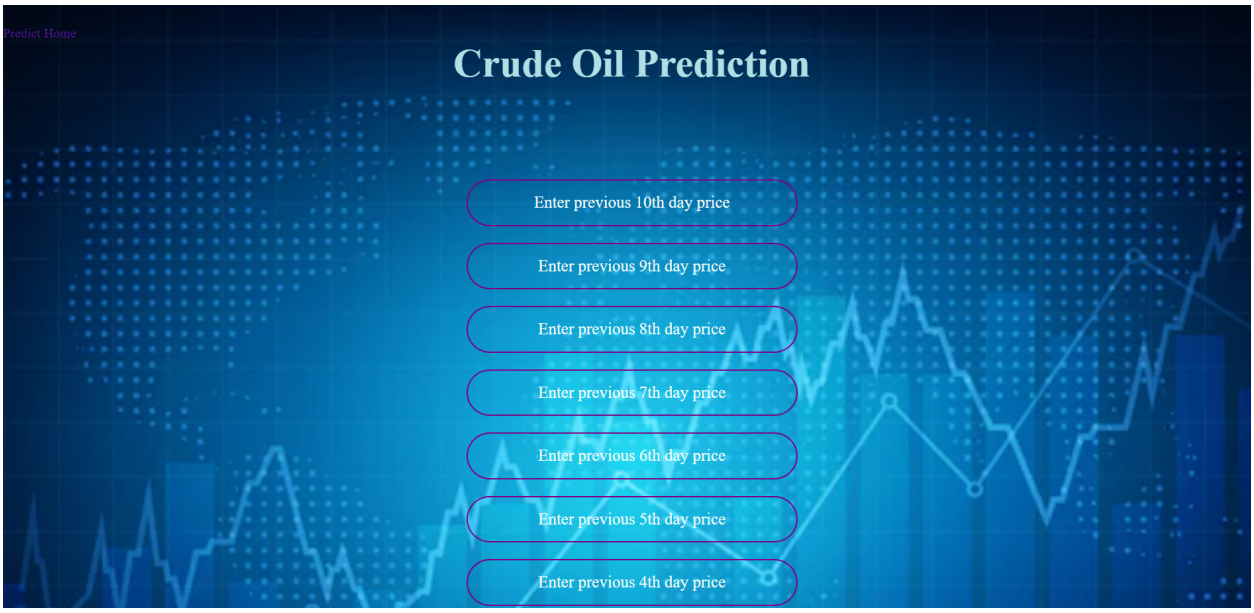
Study shows that it provide with different price of crude Oil from past days, the model detects, and prediction of given prices. When click in to the predict button then it will shows the predicted output.

5. FLOWCHART



6. RESULT


Home Predict
Crude Oil Prediction
Demand for oil is inelastic, therefore the rise in price is good news for producers because they will see an increase in their revenue. Oil importers, however, will experience increased costs of purchasing oil. Because oil is the largest traded commodity, the effects are quite significant. A rising oil price can even shift economic/political power from oil importers to oil exporters. The crude oil price movements are subject to diverse influencing factors.


Predict Home
Crude Oil Prediction
Enter previous 10th day price
Enter previous 9th day price
Enter previous 8th day price
Enter previous 7th day price
Enter previous 6th day price
Enter previous 5th day price
Enter previous 4th day price

7. ADVANTAGES & DISADVANTAGES

Advantages:

- Increased accuracy for crude oil price prediction.
- Reduce the time complexity.

Disadvantages:

- Data mining techniques does not help to provide effective decision making.

8. APPLICATIONS

- Neural Network technology is considered as one of the key technology used to predict the crude oil price.
- It using the Previous history of crude oil prices to predict future crude oil.

9. CONCLUSION

In this project, we are using the Previous history of crude oil prices to predict future crude oil based on the IBM Watson Studio. This Project mainly focuses on applying Neural Networks to predict the Crude Oil Price. This decision helps us to buy crude oil at the proper time.

10. FUTURE SCOPE

The project can be further enhanced by deploying the deep learning model obtained using a web application and larger dataset cloud be used for prediction to give higher accuracy and produce better result.

11. BIBILOGRAPHY

- Abdullah, S. N. and Zeng, X. (2010) „„Machine learning approach for crude oil price prediction with Artificial Neural Networks-Quantitative (ANN-Q) model“ Proceedings

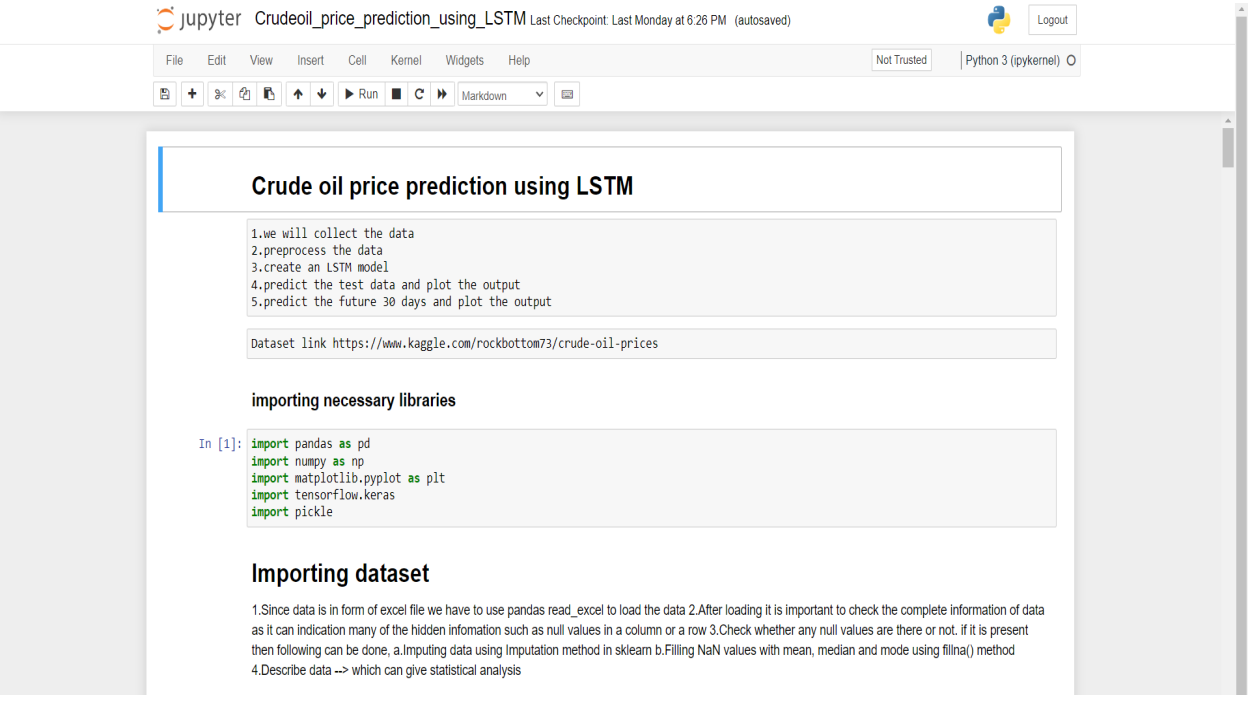
of the International Joint Conference on Neural Networks (IJCNN’2010), 1-8.

- Alizadeh, A. and Mafinezhad, K (2010) „„Monthly Brent Oil Price Forecasting Using Artificial Neural Networks and A Crisis Index““ Proceedings of the International Conference On Electronics And Information Engineering (ICEIE’2010), 2, 465-468

- Aloui, C. Hamdi, M., Mensi, W. and Nguyen, D. Y. (2012) „„Further evidence on the timevarying efficiency of crude oil markets““ Energy Studies Review 19(2), 38-51.

12.APPENDIX

Source Code



In [2]:

```
import os, types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share the notebook.
client_2c12594bc3394700b42a33e64f17b2b8 = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='7Rww8Deqh1HRA9GcH7XU8ibz37EVJsEnptxOho83yN15',
    ibm_auth_endpoint="https://iam.cloud.ibm.com/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3.private.us.cloud-object-storage.appdomain.cloud')

body = client_2c12594bc3394700b42a33e64f17b2b8.get_object(Bucket='crudeoil-donotdelete-pr-h4uozk1hs6snux',Key='Crude Oil Prices D

data = pd.read_excel(body.read())
data.head()
```

Out[2]:

	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

Data Preprocessing

Checking or null values

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

Out[5]:
```
Date            0
Closing Value    7
dtype: int64
```

In [6]:
```
data.dropna(axis=0,inplace=True)
```

In [7]:
```
data.isnull().sum()
```

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

In [8]:
```
data.shape
```

Out[8]:
```
(8216, 2)
```

Selecting Closing value column for prediction

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

In [10]:
```
data_oil
```

Out[10]:
```
0       25.56
1       26.00
```

Selecting Closing value column for prediction

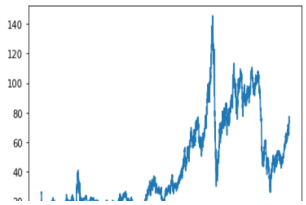
In [9]:
```
data_oil=data.reset_index()['Closing Value']
```

In [10]:
```
data_oil
```

Out[10]:
```
0       25.56
1       26.00
2       26.53
3       25.85
4       25.87
        ...
8211    73.89
8212    74.19
8213    73.05
8214    73.78
8215    73.93
Name: Closing Value, Length: 8216, dtype: float64
```

In [11]:
```
plt.plot(data_oil)
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0x7f9f1c75d0d0>]
```



Create the Stacked LSTM model

In [23]:
```
#tensorflow :open source used for both ML and DL for computation
from tensorflow.keras.models import Sequential#it is a plain stack of layers
from tensorflow.keras.layers import Dense#Dense layer is the regular deeply connected neural network layer
from tensorflow.keras.layers import LSTM #Long Short Trem Memory
```

In [24]:
```
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(10,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')
```

In [25]:
```
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 [26]:
```
#Training the model
model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=50,batch_size=8)
```

```
Epoch 1/50
667/667 [==============================] - 18s 18ms/step - loss: 264.9510 - val_loss: 3117.7952
Epoch 2/50
667/667 [==============================] - 11s 16ms/step - loss: 171.1041 - val_loss: 2526.8098
Epoch 3/50
667/667 [==============================] - 11s 17ms/step - loss: 57.7536 - val_loss: 1433.7810
Epoch 4/50
667/667 [==============================] - 11s 16ms/step - loss: 17.5767 - val_loss: 971.1276
Epoch 5/50
667/667 [==============================] - 11s 17ms/step - loss: 6.2937 - val_loss: 737.6458
Epoch 6/50
667/667 [==============================] - 11s 16ms/step - loss: 3.1081 - val_loss: 605.8196
Epoch 7/50
667/667 [==============================] - 11s 17ms/step - loss: 1.8348 - val_loss: 516.5406
Epoch 8/50
667/667 [==============================] - 11s 17ms/step - loss: 1.2188 - val_loss: 449.3754
Epoch 9/50
667/667 [==============================] - 11s 16ms/step - loss: 1.1672 - val_loss: 407.3385
Epoch 10/50
667/667 [==============================] - 11s 16ms/step - loss: 0.8811 - val_loss: 378.0981
Epoch 11/50
667/667 [==============================] - 11s 16ms/step - loss: 0.8022 - val_loss: 360.6334
Epoch 12/50
667/667 [==============================] - 11s 17ms/step - loss: 0.7800 - val_loss: 341.1000
Epoch 13/50
667/667 [==============================] - 11s 17ms/step - loss: 0.7666 - val_loss: 331.3139
Epoch 14/50
667/667 [==============================] - 11s 17ms/step - loss: 0.7724 - val_loss: 318.3911
Epoch 15/50
667/667 [==============================] - 11s 17ms/step - loss: 0.8337 - val_loss: 309.7038
Epoch 16/50
667/667 [==============================] - 11s 17ms/step - loss: 0.7343 - val_loss: 304.2457
Epoch 17/50
```

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

In [28]:
```
a1 = model.predict([[46,47,48,50,51,52,54,43,44,47]])
```

In [29]:
```
np.round(a1[0][0],2)
```

Out[29]:
```
46.35
```

model evaluation

In [30]:
```
### Calculate RMSE performance metrics
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

Out[30]:
```
0.9475632335882856
```

In [31]:
```
### Test Data RMSE
math.sqrt(mean_squared_error(ytest,test_predict))
```

Out[31]:
```
16.716945182249823
```

In [32]:
```
model.save("crude_oil.h5")
```

In [33]:
```
!tar -zcvf Crude_oil_ibm.tgz crude_oil.h5
```

```
crude_oil.h5
```

In [34]:
```
ls-1
```

```
crude_oil.h5
Crude_oil_ibm.tgz
```

IBM Deployment

In [35]:
```
!pip install ibm_watson_machine_learning
```

```
Requirement already satisfied: ibm_watson_machine_learning in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (1.0.232)
Requirement already satisfied: pandas<1.4.0,>=0.24.2 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm_watson
_machine_learning) (1.3.4)
Requirement already satisfied: requests in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm_watson_machine_lear
ning) (2.26.0)
Requirement already satisfied: packaging in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm_watson_machine_lea
rning) (21.3)
Requirement already satisfied: lomond in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm_watson_machine_learni
ng) (0.3.3)
Requirement already satisfied: tabulate in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm_watson_machine_lear
ning) (0.8.9)
Requirement already satisfied: urllib3 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm_watson_machine_learn
ing) (1.26.7)
Requirement already satisfied: ibm-cos-sdk==2.11.* in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm_watson_m
achine_learning) (2.11.0)
Requirement already satisfied: importlib-metadata in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm_watson_ma
chine_learning) (4.8.2)
Requirement already satisfied: certifi in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm_watson_machine_learn
ing) (2022.6.15)
Requirement already satisfied: ibm-cos-sdk-s3transfer==2.11.0 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from i
bm-cos-sdk==2.11.*->ibm_watson_machine_learning) (2.11.0)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-cos-s
dk==2.11.*->ibm_watson_machine_learning) (0.10.0)
Requirement already satisfied: ibm-cos-sdk-core==2.11.0 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-cos
-sdk==2.11.*->ibm_watson_machine_learning) (2.11.0)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from ibm-
cos-sdk-core==2.11.0->ibm-cos-sdk==2.11.*->ibm_watson_machine_learning) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from pandas<1.4.0,>=0.2
4.2->ibm_watson_machine_learning) (2021.3)
Requirement already satisfied: numpy>=1.17.3 in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from pandas<1.4.0,>=0.2
4.2->ibm_watson_machine_learning) (1.20.3)
```

```
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
autoai-kb_3.1-py3.7            632d4b22-10aa-5180-88f0-f52dfb6444d7  base
pytorch-onnx_1.7-py3.8         634d3cdc-b562-5bf9-a2d4-ea90a478456b  base
spark-mllib_2.3-r_3.6          6586b9e3-ccd6-4f92-900f-0f8cb2bd6f0c  base
tensorflow_2.4-py3.7           65e171d7-72d1-55d9-8ebb-f813d620c9bb  base
spss-modeler_18.2              687eddc9-028a-4117-b9dd-e57b36f1efa5  base
pytorch-onnx_1.2-py3.6         692a6a4d-2c4d-45ff-a1ed-b167ee55469a  base
spark-mllib_2.3-scala_2.11     7963efe5-bbec-417e-92cf-0574e21b4e8d  base
----------------------------  ------------------------------------  ----
Note: Only first 50 records were displayed. To display more use 'limit' parameter.
```

In [42]:
```
software_space_uid = client.software_specifications.get_uid_by_name('tensorflow_rt22.1-py3.9')
software_space_uid
```

Out[42]:
```
'acd9c798-6974-5d2f-a657-ce06e986df4d'
```

In [43]:
```
model_details = client.repository.store_model(model='Crude_oil_ibm.tgz',
                                              meta_props={
                                                  client.repository.ModelMetaNames.NAME:'crude_oil_prediction',
                                                  client.repository.ModelMetaNames.SOFTWARE_SPEC_UID:software_space_uid,
                                                  client.repository.ModelMetaNames.TYPE:'tensorflow_2.7'
                                              })
```

In [44]:
```
model_id = client.repository.get_model_id(model_details)
model_id
```

Out[44]:
```
'cdcf1594-c497-4877-92d9-d0dbcdd729cd'
```

In [ ]: