WTI_Oil-Prices_LTSM_Autoencoder

April 18, 2024

0.1 Anomaly Detection in West Texas Intermediate (WTI) Crude Oil Prices using Autoencoders

0.1.1 The Anomaly Detection in West Texas Intermediate (WTI) Crude Oil Prices has a Real-World Application:

The ability to detect anomalies in crude oil prices has practical application in the financial and energy sectors. Detecting abnormal price movements early can help to make informed decisions and mitigate potential risks.

0.1.2 It is important to consider the following:

0.1.3 * March 8, 2020 a price conflict begins between Saudi Arabia and Russia, and the starting of the coronavirus pandemic around the world

0.1.4 * WTI on April 20, 2020 prices fell below zero for the first time in history

(Source: Wikipedia)

0.1.5 This makes this case of study very attractive with a direct impact and different implications. First, it is easy to evaluate the performance of our model based on the anomaly detection of the two catastrophic scenarios of the Spring of 2020 in the WTI Oil Prices. And second, by incorporating anomaly detection techniques, financial and energy market participants can enhance their risk management and decision making strategies.

[ ]:

```

```

[1]:

```
# Import libraries

import tensorflow as tf
import plotly.graph_objects as go
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt


np.random.seed(1)
tf.random.set_seed(1)
```

```python
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, LSTM, Dropout, RepeatVector,␣
↪TimeDistributed, Dense
```

[2]:
```python
# Fetch historical data for West Texas Intermediate (WTI) crude oil prices
df = yf.download("CL=F", period="max")[['Close']].reset_index()

full_dates = df['Date'].values
full_close_prices = df['Close'].values
```

```
[*********************100%%**********************]  1 of 1 completed
```

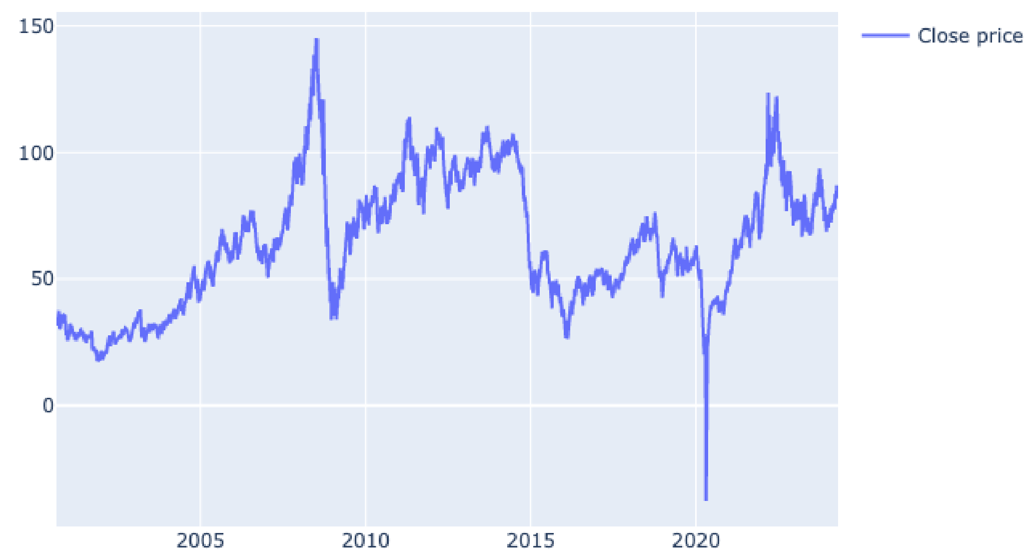
[3]:
```python
df.head()
```

[3]:
```
         Date      Close
0 2000-08-23  32.049999
1 2000-08-24  31.629999
2 2000-08-25  32.049999
3 2000-08-28  32.869999
4 2000-08-29  32.720001
```

[ ]:
```python

```

[4]:
```python
fig = go.Figure()
fig.add_trace(go.Scatter(x=df['Date'], y=df['Close'], name='Close price'))
fig.update_layout(showlegend=True, title='WTI Oil Price 2000-2024')
fig.show()
# Guardar la imagen en formato PNG
fig.write_image('plot.png')
```

WTI Oil Price 2000-2024



[ ]:

[5]:
```
# Identifying the shape of the data
df.shape
```

[5]: (5939, 2)

[ ]:

[6]:
```
# Are there any duplicates
df.duplicated().sum().any()
```

[6]: False

[7]:
```
# Checking for True or False for any nulls in the dataset
df.isnull().values.any()
```

[7]: False

[8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5939 entries, 0 to 5938
Data columns (total 2 columns):
```

```
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Date    5939 non-null   datetime64[ns]
 1   Close   5939 non-null   float64
dtypes: datetime64[ns](1), float64(1)
memory usage: 92.9 KB
```

[ ]:

```

```

[9]:

```python
# Preprocessing
#Train test split
train, test = df.loc[df['Date'] <= '2021-01-01'], df.loc[df['Date'] >␣
 ↪'2021-01-01']

train.shape, test.shape
```

[9]: ((5110, 2), (829, 2))

[10]:

```python
# Scaling
scaler = StandardScaler()
train['Close'] = scaler.fit_transform(train[['Close']])
test['Close'] = scaler.transform(test[['Close']])
```

```
<ipython-input-10-0b756a53dc53>:3: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

<ipython-input-10-0b756a53dc53>:4: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[11]:

```python
# Create sequences for the LSTM
sequence_length = 30
X_train = np.array([train['Close'].values[i:i+sequence_length] for i in␣
 ↪range(len(train) - sequence_length)])
```

```
X_test = np.array([test['Close'].values[i:i+sequence_length] for i in␣
↪range(len(test) - sequence_length)])
```

[12]:
```
# Define and compile the model
model = keras.models.Sequential([
    keras.layers.LSTM(64, activation='relu', input_shape=(X_train.shape[1], 1),␣
↪return_sequences=True),
    keras.layers.Dropout(0.25),
    keras.layers.LSTM(32, activation='relu', return_sequences=False),
    keras.layers.Dropout(0.25),
    keras.layers.RepeatVector(sequence_length),
    keras.layers.LSTM(32, activation='relu', return_sequences=True),
    keras.layers.Dropout(0.25),
    keras.layers.LSTM(64, activation='relu', return_sequences=True),
    keras.layers.Dropout(0.25),
    keras.layers.TimeDistributed(keras.layers.Dense(1))
])

model.compile(optimizer='adam', loss='mse')
```

[13]:
```
# Train the model
history = model.fit(X_train, X_train, epochs=100, batch_size=32,␣
↪validation_data=(X_test, X_test))
```

```
Epoch 1/100
159/159 [==============================] - 10s 40ms/step - loss: 0.4445 -
val_loss: 0.1295
Epoch 2/100
159/159 [==============================] - 6s 38ms/step - loss: 0.1315 -
val_loss: 0.0594
Epoch 3/100
159/159 [==============================] - 6s 35ms/step - loss: 0.0877 -
val_loss: 0.0441
Epoch 4/100
159/159 [==============================] - 5s 34ms/step - loss: 0.0859 -
val_loss: 0.0541
Epoch 5/100
159/159 [==============================] - 5s 34ms/step - loss: 0.0617 -
val_loss: 0.0429
Epoch 6/100
159/159 [==============================] - 5s 34ms/step - loss: 0.0523 -
val_loss: 0.0374
Epoch 7/100
159/159 [==============================] - 5s 34ms/step - loss: 0.0480 -
val_loss: 0.0340
Epoch 8/100
159/159 [==============================] - 6s 35ms/step - loss: 0.0436 -
```

```
val_loss: 0.0420
Epoch 9/100
159/159 [==============================] - 6s 36ms/step - loss: 0.0417 -
val_loss: 0.0349
Epoch 10/100
159/159 [==============================] - 7s 42ms/step - loss: 0.0426 -
val_loss: 0.0388
Epoch 11/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0389 -
val_loss: 0.0424
Epoch 12/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0385 -
val_loss: 0.0323
Epoch 13/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0394 -
val_loss: 0.0437
Epoch 14/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0380 -
val_loss: 0.0497
Epoch 15/100
159/159 [==============================] - 7s 42ms/step - loss: 0.0365 -
val_loss: 0.0418
Epoch 16/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0367 -
val_loss: 0.0404
Epoch 17/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0349 -
val_loss: 0.0403
Epoch 18/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0344 -
val_loss: 0.0333
Epoch 19/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0337 -
val_loss: 0.0382
Epoch 20/100
159/159 [==============================] - 6s 36ms/step - loss: 0.0325 -
val_loss: 0.0405
Epoch 21/100
159/159 [==============================] - 6s 36ms/step - loss: 0.0322 -
val_loss: 0.0414
Epoch 22/100
159/159 [==============================] - 6s 37ms/step - loss: 0.0315 -
val_loss: 0.0343
Epoch 23/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0308 -
val_loss: 0.0391
Epoch 24/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0303 -
```

```
val_loss: 0.0277
Epoch 25/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0306 -
val_loss: 0.0420
Epoch 26/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0306 -
val_loss: 0.0290
Epoch 27/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0295 -
val_loss: 0.0276
Epoch 28/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0279 -
val_loss: 0.0354
Epoch 29/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0276 -
val_loss: 0.0317
Epoch 30/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0269 -
val_loss: 0.0306
Epoch 31/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0273 -
val_loss: 0.0427
Epoch 32/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0266 -
val_loss: 0.0388
Epoch 33/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0261 -
val_loss: 0.0231
Epoch 34/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0260 -
val_loss: 0.0241
Epoch 35/100
159/159 [==============================] - 6s 36ms/step - loss: 0.0254 -
val_loss: 0.0308
Epoch 36/100
159/159 [==============================] - 6s 36ms/step - loss: 0.0251 -
val_loss: 0.0379
Epoch 37/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0259 -
val_loss: 0.0254
Epoch 38/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0251 -
val_loss: 0.0384
Epoch 39/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0245 -
val_loss: 0.0247
Epoch 40/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0251 -
```

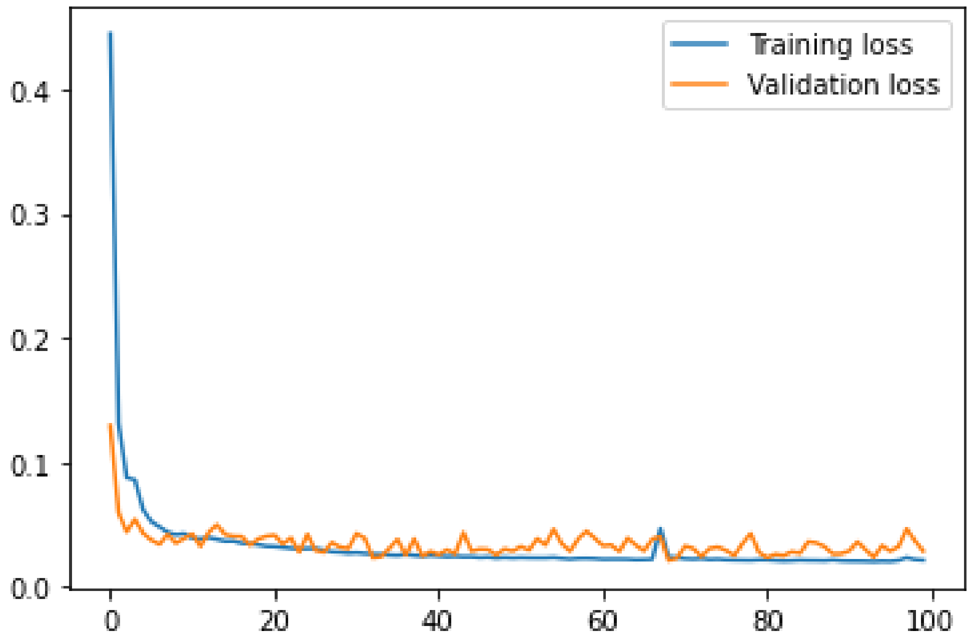
```
val_loss: 0.0283
Epoch 41/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0246 -
val_loss: 0.0257
Epoch 42/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0241 -
val_loss: 0.0296
Epoch 43/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0243 -
val_loss: 0.0263
Epoch 44/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0239 -
val_loss: 0.0435
Epoch 45/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0240 -
val_loss: 0.0286
Epoch 46/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0234 -
val_loss: 0.0301
Epoch 47/100
159/159 [==============================] - 6s 36ms/step - loss: 0.0237 -
val_loss: 0.0299
Epoch 48/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0229 -
val_loss: 0.0254
Epoch 49/100
159/159 [==============================] - 7s 45ms/step - loss: 0.0235 -
val_loss: 0.0303
Epoch 50/100
159/159 [==============================] - 7s 45ms/step - loss: 0.0230 -
val_loss: 0.0285
Epoch 51/100
159/159 [==============================] - 8s 49ms/step - loss: 0.0233 -
val_loss: 0.0321
Epoch 52/100
159/159 [==============================] - 7s 42ms/step - loss: 0.0231 -
val_loss: 0.0294
Epoch 53/100
159/159 [==============================] - 7s 43ms/step - loss: 0.0231 -
val_loss: 0.0384
Epoch 54/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0231 -
val_loss: 0.0335
Epoch 55/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0235 -
val_loss: 0.0461
Epoch 56/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0227 -
```

```
val_loss: 0.0345
Epoch 57/100
159/159 [==============================] - 6s 36ms/step - loss: 0.0221 -
val_loss: 0.0285
Epoch 58/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0227 -
val_loss: 0.0385
Epoch 59/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0228 -
val_loss: 0.0446
Epoch 60/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0225 -
val_loss: 0.0389
Epoch 61/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0220 -
val_loss: 0.0329
Epoch 62/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0221 -
val_loss: 0.0335
Epoch 63/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0221 -
val_loss: 0.0282
Epoch 64/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0220 -
val_loss: 0.0390
Epoch 65/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0215 -
val_loss: 0.0334
Epoch 66/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0217 -
val_loss: 0.0286
Epoch 67/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0219 -
val_loss: 0.0381
Epoch 68/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0467 -
val_loss: 0.0402
Epoch 69/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0241 -
val_loss: 0.0209
Epoch 70/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0233 -
val_loss: 0.0227
Epoch 71/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0224 -
val_loss: 0.0325
Epoch 72/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0221 -
```

```
val_loss: 0.0307
Epoch 73/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0225 -
val_loss: 0.0242
Epoch 74/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0218 -
val_loss: 0.0306
Epoch 75/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0220 -
val_loss: 0.0318
Epoch 76/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0215 -
val_loss: 0.0291
Epoch 77/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0213 -
val_loss: 0.0250
Epoch 78/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0212 -
val_loss: 0.0347
Epoch 79/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0211 -
val_loss: 0.0427
Epoch 80/100
159/159 [==============================] - 6s 37ms/step - loss: 0.0215 -
val_loss: 0.0274
Epoch 81/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0214 -
val_loss: 0.0233
Epoch 82/100
159/159 [==============================] - 6s 38ms/step - loss: 0.0210 -
val_loss: 0.0259
Epoch 83/100
159/159 [==============================] - 6s 37ms/step - loss: 0.0206 -
val_loss: 0.0249
Epoch 84/100
159/159 [==============================] - 6s 37ms/step - loss: 0.0209 -
val_loss: 0.0281
Epoch 85/100
159/159 [==============================] - 6s 37ms/step - loss: 0.0213 -
val_loss: 0.0268
Epoch 86/100
159/159 [==============================] - 6s 39ms/step - loss: 0.0211 -
val_loss: 0.0362
Epoch 87/100
159/159 [==============================] - 6s 37ms/step - loss: 0.0210 -
val_loss: 0.0352
Epoch 88/100
159/159 [==============================] - 6s 37ms/step - loss: 0.0208 -
```

```
val_loss: 0.0321
Epoch 89/100
159/159 [==============================] - 6s 36ms/step - loss: 0.0214 -
val_loss: 0.0263
Epoch 90/100
159/159 [==============================] - 6s 36ms/step - loss: 0.0208 -
val_loss: 0.0264
Epoch 91/100
159/159 [==============================] - 6s 36ms/step - loss: 0.0206 -
val_loss: 0.0286
Epoch 92/100
159/159 [==============================] - 6s 37ms/step - loss: 0.0205 -
val_loss: 0.0360
Epoch 93/100
159/159 [==============================] - 7s 42ms/step - loss: 0.0204 -
val_loss: 0.0297
Epoch 94/100
159/159 [==============================] - 7s 42ms/step - loss: 0.0202 -
val_loss: 0.0235
Epoch 95/100
159/159 [==============================] - 7s 42ms/step - loss: 0.0205 -
val_loss: 0.0330
Epoch 96/100
159/159 [==============================] - 7s 47ms/step - loss: 0.0203 -
val_loss: 0.0285
Epoch 97/100
159/159 [==============================] - 7s 46ms/step - loss: 0.0211 -
val_loss: 0.0322
Epoch 98/100
159/159 [==============================] - 7s 46ms/step - loss: 0.0234 -
val_loss: 0.0464
Epoch 99/100
159/159 [==============================] - 7s 41ms/step - loss: 0.0217 -
val_loss: 0.0367
Epoch 100/100
159/159 [==============================] - 6s 40ms/step - loss: 0.0212 -
val_loss: 0.0286
```

[14]:
```python
# Plot the training and validation loss
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.legend()
plt.show()
```



[15]:
```
# Make predictions
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)
```

```
159/159 [==============================] - 2s 9ms/step
25/25 [==============================] - 0s 8ms/step
```

[ ]:

[ ]:

[16]:
```
from sklearn.metrics import mean_absolute_error

# Calculate Mean Absolute Error (MAE) for training set
train_mae = np.mean(np.abs(train_predictions.reshape(train_predictions.
↪shape[0], -1) - X_train), axis=1)

# Calculate Mean Absolute Error (MAE) for test set
test_mae = np.mean(np.abs(test_predictions.reshape(test_predictions.shape[0],␣
↪-1) - X_test), axis=1)

# Print the mean absolute errors
print(f'Mean Absolute Error (MAE) for training set: {np.mean(train_mae)}')
print(f'Mean Absolute Error (MAE) for test set: {np.mean(test_mae)}')
```

```
Mean Absolute Error (MAE) for training set: 0.1068832055852849
```

```
Mean Absolute Error (MAE) for test set: 0.1305671237562117
```

The MAE values for the training and test sets are quite close, it suggests that the model is not overfitting.

This indicates that the model is generalizing well to unseen data and is performing consistently on both the

training and test sets.

[ ]:

```

```

[17]:

```
# Calculate reconstruction error
train_mse = np.mean(np.square(train_predictions.reshape(train_predictions.
 ↪shape[0], -1) - X_train), axis=1)
test_mse = np.mean(np.square(test_predictions.reshape(test_predictions.
 ↪shape[0], -1) - X_test), axis=1)
```

[18]:

```
# Concatenate data for anomaly detection visualization
full_data = np.concatenate([train['Close'], test['Close']])
full_mse = np.concatenate([train_mse, test_mse])
```

[19]:

```
# Set threshold for anomaly detection
threshold = np.mean(full_mse) + 2 * np.std(full_mse)
threshold
```

[19]:

```
0.07977542261337182
```

[ ]:

```

```

[ ]:

```

```

[20]:

```
# Identify anomalies
anomalies = np.where(full_mse > threshold, 1, 0)
```

[ ]:

```

```

[ ]:

```

```

[21]:

```
# Ensure lengths match
min_length = min(len(full_dates), len(full_close_prices), len(anomalies))
full_dates = full_dates[:min_length]
full_close_prices = full_close_prices[:min_length]
anomalies = anomalies[:min_length]

# Create a DataFrame with all dates, close prices, and anomalies
anomalies_df = pd.DataFrame({'Date': full_dates, 'Close': full_close_prices,␣
 ↪'Anomaly': anomalies})
```

```python
# Filter anomalies to get the detected anomalies
detected_anomalies = anomalies_df[anomalies_df['Anomaly'] == 1]

# Print the number of detected anomalies
print(f'Number of detected anomalies: {len(detected_anomalies)}')

# Print the detected anomalies
print(detected_anomalies)
```

```
Number of detected anomalies: 265
           Date       Close  Anomaly
1862 2008-02-06   87.139999        1
1863 2008-02-07   88.110001        1
1864 2008-02-08   91.769997        1
1876 2008-02-27   99.639999        1
1877 2008-02-28  102.589996        1
…           …           …       …
5429 2022-04-08   98.260002        1
5430 2022-04-11   94.290001        1
5431 2022-04-12  100.599998        1
5432 2022-04-13  104.250000        1
5433 2022-04-14  106.949997        1

[265 rows x 3 columns]
```

[ ]:

[ ]:

[22]:
```python
# Visualize anomalies
fig = go.Figure()
fig.add_trace(go.Scatter(x=df['Date'], y=full_data, name='Close Price'))
fig.add_trace(go.Scatter(x=df['Date'][sequence_length:], y=full_mse,␣
↪mode='lines', name='Reconstruction Error'))
fig.add_trace(go.Scatter(x=df['Date'][sequence_length:], y=np.where(anomalies,␣
↪full_mse, None), mode='markers', name='Anomalies', marker=dict(color='red',␣
↪size=8)))
fig.update_layout(title='Anomaly Detection using LSTM Autoencoder',
                  xaxis_title='Date',
                  yaxis_title='Value')
fig.show()
```

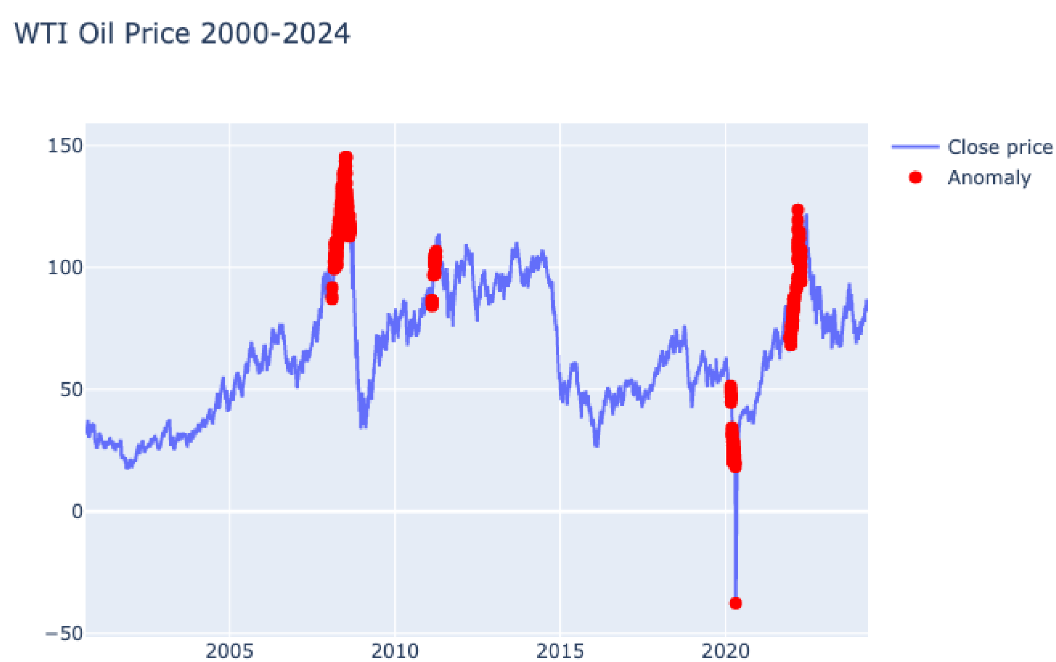
[ ]:

[ ]:

[ ]:

[23]:

```
import plotly.graph_objects as go

# Create a figure
fig = go.Figure()

# Add the close price data
fig.add_trace(go.Scatter(x=df['Date'], y=df['Close'], mode='lines', name='Close␣
↪price'))

# Add markers for anomalies
anomalies_dates = detected_anomalies['Date']
anomalies_prices = detected_anomalies['Close']
fig.add_trace(go.Scatter(x=anomalies_dates, y=anomalies_prices, mode='markers',␣
↪name='Anomaly', marker=dict(color='red', size=8)))

# Update layout
fig.update_layout(showlegend=True, title='WTI Oil Price 2000-2024')

# Show plot
fig.show()
fig.write_image('plot-anomalies.png')
```



[ ]:

Visualize the reconstruction errors or reconstructed data to get a qualitative understanding of how well the model is performing. Look for patterns or anomalies in the reconstructed data.

[ ]:

0.2 Conclusions

0.2.1 The LSTM autoencoder model was able to effectively detect anomalies in the WTI crude oil price data. The model detected anomalies related to low prices in the period of March and April 2020. It is important to highligh that corresponds to the following:

0.2.2 * The price of WTI oil hit an all-time high in July 2008. However, the price began to fall sharply due to the global economic slowdown and the 2008 financial crisis in general, with a drastic decline in the price of WTI oil in late 2008.

(Source: New York Post)

0.2.3 * March 8, 2020 a price conflict begins between Saudi Arabia and Russia, and the starting of the coronavirus pandemic around the world

0.2.4 * WTI on April 20, 2020 prices fell below zero for the first time in history

(Source: The New York Times)

0.2.5 * January 2022, the WTI reached 84 USD and is at its highest price in more than seven years

(Source: EFE)

0.2.6 While the LSTM autoencoder model showed promising results, there is always room for improvement. Fine-tuning the model architecture, exploring different hyperparameters, and incorporating additional features could enhance the model's performance and accuracy in anomaly detection.

[ ]:

[ ]:

[24]:

```python
# Save the model
model.save('anomaly_detection_LTSM_autoencoder_model.h5')
```

[ ]: