CB4247 Statistics & Computational Inference to Big Data for Chemical and Biomedical Engineers

Individual Project

on

Forecasting Brent Crude Oil Prices

by

Parth Bharihoke (U1923552D)

Table of Contents

List of Tables

List of Figures

1. Introduction and Literature Review

Known as the “industrial blood” and “black gold”, crude oil is the basis of industrial activity in all countries and is an indispensable energy resource for the functioning of the global economy. Crude oil is a global commodity that trades in markets around the world, both as spot oil and via derivatives contracts. There are two major oil contracts that are closely watched by oil market participants. In North America, the benchmark for oil futures is West Texas Intermediate (WTI) crude, which trades on the New York Mercantile Exchange (NYMEX). In Europe, Africa, and the Middle East, the benchmark is North Sea Brent Crude, which trades on the Intercontinental Exchange (ICE) [1].

Many economists view crude oil as the single most important commodity in the world, as it is currently the primary source of energy production and hence accurate crude oil price forecasting is critical to the economy’s long-term stability. Research has been conducted in a number of areas related to oil price modelling and forecasting. Research indicates that supply and demand determine the price of crude oil. Additionally, some unforeseen severe events, such natural disasters and geopolitical wars, have an impact on price movement.

Proper determination of the country's economic structure and price-influencing components is necessary for the development of an appropriate price forecasting model. The U.S. economy largely impacts the dynamics of the global oil price since it is one of the major participants in the global energy market with respect to energy supply and consumption. Any changes, news, and shocks in the U.S. economy immediately affect the global oil prices dynamics. These catalysts which are primarily determined by financial and macroeconomic variables shape the dynamics of supply and demand in the crude oil market and, as a result, affect how investors behave. To modify their investing behaviour, investors, for instance, primarily concentrate on shifting macroeconomic variables like the FED rate, industrial production, or unemployment rate. Conversely, with regard to the U.S. economy, variables like the cost of refining crude oil and the volume of oil produced impact the supply of crude oil, which in turn dictates the dynamics of the Brent crude price. The dynamics of oil prices are greatly impacted by world events, conflicts, crises, and OPEC actions. Considering all of this, the study's motivation stems from the historical crude oil prices' non-linearity, uncertainty, and dynamics, all of which make predicting challenging. The uncertain

prediction results are doomed to cause significant uncertainty in relevant investors' returns and the economic system's stable development and as a result, developing a reliable crude oil price prediction model is critical [2]. Therefore, there is an attempt to build the reliable prediction of Brent Crude Oil prices in this project which includes the factors mentioned above.

Figure 1. ICE Brent Crude Oil Front Month Prices

Oil prices fluctuate a lot over time, making it challenging to model and as Figure 1's historical graph illustrates, prices can vary significantly in a short period of time. Following the academic research over the past several years, one can observe that the study of the dynamics of oil prices has progressively evolved from linear econometric models to non-linear ones. Recently, there is an increasing interest in research on methodologies based on machine learning (ML) and artificial intelligence (AI). [3] forecasted the crude oil indices using a Recurrent Neural Network (RNN) and [4] suggested a Neural Network model that was optimized using Genetic Algorithms to predict fluctuations in the price of crude oil. While current research have complicated neural network structures, this project aims to train simple machine learning algorithms to provide a price prediction with R-squared value larger than 0.95.

2. Data Sources

The Bloomberg terminal is used as the primary data source for this project, including the most prominently used global crude oil fundamental demand and supply, inventory and flows, refinery, utilisation data, S&P 500 to measure the optimism in the economy, US, Government 10-year bond yields to measure the interest rate and the Bloomberg Spot Index, the track the performance against a basket of 10 leading global currencies. The only other source used as the Global political risk index provided by Caldara and Iacoviello which reflects automated text-search results of the electronic archives of 10 newspapers and counting the number of articles related to adverse geopolitical events in each newspaper for each month (as a share of the total number of news articles) [5]. Table 1 illustrates the category of data, ticker code and description of the feature variable used within this project.

Table 1: Key Characteristics of the Dataset

Category	Ticker	Description
Inventory	DOESCRUD Index	DOE Crude Oil Total Inventory Data (excluding Strategic Petroleum Reserve)
	DOEASCRD Index	DOE Total Change in Crude Oil Inventories Data
	DOESSPR Index	DOE Strategic Petroleum Reserve (SPR) Total Inventory Data
	DOESCROK Index	DOE Cushing Oklahoma Crude Oil Total Stocks Data
Production	DOETCRUD Index	United States DOE Crude Oil Total Production Data
	BAKEOIL Index	Baker Hughes United States Crude Oil Rotary Rig Count Data
	BAKMWRLD Index	Baker Hughes Total World Oil And Gas Rotary Rig Count Data
	DOESUNCR Index	US DOE Crude Oil Supply Adjustment
	DOETCL48 Index	DOE Crude Oil Lower 48 States Production Data
	OPCRTOTL Index	Bloomberg Total OPEC Crude Oil Production Output Data
	CROMUS Index	DOE EIA US Crude Oil Production (MBPD)
	OPCRSAUD Index	Bloomberg OPEC Crude Oil Production Output Data/Saudi Arabia
	DWOPRUSS Index	DOE Russia Crude Oil Production Data
	ST14WO Index	World Crude Oil & Liquid Fuels Production Total World Supply
	DWOPWRLD Index	DOE World Crude Oil Production Data
	OPCRNIGE Index	Bloomberg OPEC Crude Oil Production Output Data/Nigeria
	CHENRCOL Index	China Industrial Output - Crude Oil Processed
	OPCRANGO Index	Bloomberg OPEC Crude Oil Production Angola Output Data
	OPCRIRAN Index	Bloomberg OPEC Crude Oil Production Output Data/Iran
	OPCRVENZ Index	Bloomberg OPEC Crude Oil Production Output Data/Venezuela
	OPQUTOTL Index	Bloomberg OPEC Total Crude Oil Production Quota/OPEC
	DWOPSAUD Index	DOE Saudi Arabia Crude Oil Production Data
	PSMUFPCR Index	DOE PSM US Field Production of Crude Oil
	OPCRIRAQ Index	Bloomberg OPEC Crude Oil Production Output Data/Iraq
	DWOPBRAZ Index	DOE Brazil Crude Oil Production Data
	DWOPCANA Index	DOE Canada Crude Oil Production Data

	DWOPNIGE Index	DOE Nigeria Crude Oil Production Data
	ERCBSTOT Index	Canada Crude Oil Oil Sands Total Production Monthly
	OPCRLIBY Index	Bloomberg OPEC Crude Oil Production Output Data/Libya
	OPCRKUWA Index	Bloomberg OPEC Crude Oil Production Output Data/Kuwait
	DWOPANGO Index	DOE Angola Crude Oil Production Data
	DOEPCRIN Index	DOE Crude Oil Total Refinery Input Data
	ST14WC Index	World Crude Oil & Liquid Fuels Consumption Total World Consumption
	DOEPCRP3 Index	DOE Crude Oil Refinery Input Data/PADD 3
	DOEPCRP2 Index	DOE Crude Oil Refinery Input Data/PADD 2
	DOEPCRP1 Index	DOE Crude Oil Refinery Input Data/PADD 1
	ST14WC Z24 Index	World Crude Oil & Liquid Fuels Consumption Total World Consumption
	ST14OE Index	World Crude Oil & Liquid Fuels Consumption OECD
	CNIVCRUO Index	China Import Commodity Volume Crude Petroleum Oil
	PEIMCRUD Index	India Crude Oil Imports Quantity Monthly
	SKIMCRTT Index	South Korea Crude Oil Imports Total
Demand	JPCITOTL Index	Japan Crude Oil Import Data/Total
	DOEPPERC Index	DOE Percent Utilization Refinery Operable Capacity Data
	PSMURFUR Index	DOE PSM Total US Operable Refinery Utilization Rate Monthly
	INCTRFTT Index	India Refineries Crude Runs Monthly
	CHPZGALP Index	China Shandong Local Refineries Utilization Rate Weekly
	ZTCNIM BBGE Index	China Supertanker Crude Imports
Refinery	CUIQEX BBGE Index	Iraq Crude Oil Total Exports
	CUIREX BBGE Index	Iran Crude Oil Total Exports
	LOSDQIBT Index	Qua Iboe Crude Oil Loading
	LOSDNGAT Index	Nigeria Crude Oil Loading
	CUNGEX BBGE Index	Nigeria Crude Oil Exports
	CUVEEX BBGE Index	Venezuela Crude Exports
	LOSDPRDS Index	Primorsk Diesel Loading
	LOSDESPO Index	ESPO Crude Oil Loading
	LOSDPRDS Index	Primorsk Diesel Loading
	LOSDLBYT Index	Libya Crude Oil Loading Total
	CUIQEXCN BBGE Index	Iraq Total Crude Oil Exports to China
	FZWWFST VTXA Index	Global Crude Oil Floating Storage
	FZWWWST VTXA Index	Global Crude Oil on Water
Flows	FZWWTST VTXA Index	Global Crude Oil in Transit
	USGG10YR Index	US Government 10 year yields
	SPX Index	S&P 500 Index
	BBDXY Index	Bloomberg Dollar Spot Index
Economy	S5ENRS Index	S&P 500 Integrated Oil & Gas Sub Industry
Geopolitical Risk Index	GPRI	7 day average
	CO1 Comdty	Brent Crude Oil Futures Front Month Price price
Price	HVOLCRUD Index	Crude Oil 60 Day Volatility

This feature variables were divided into two different data sets depending on how many years worth of data is available for the particular feature and modelling was run separately on each dataset. The first data set contained 14 years of data, including 54 feature variables and 3615 observations. Meanwhile, the second data set contained eight years' worth of data, 69 features and 1808 observations. This was due to data collection of some features having begun more recently starting in 2017. The correlation coefficient between the oil price and the feature variable are demonstrated, in Figure 1. and Figure 2.

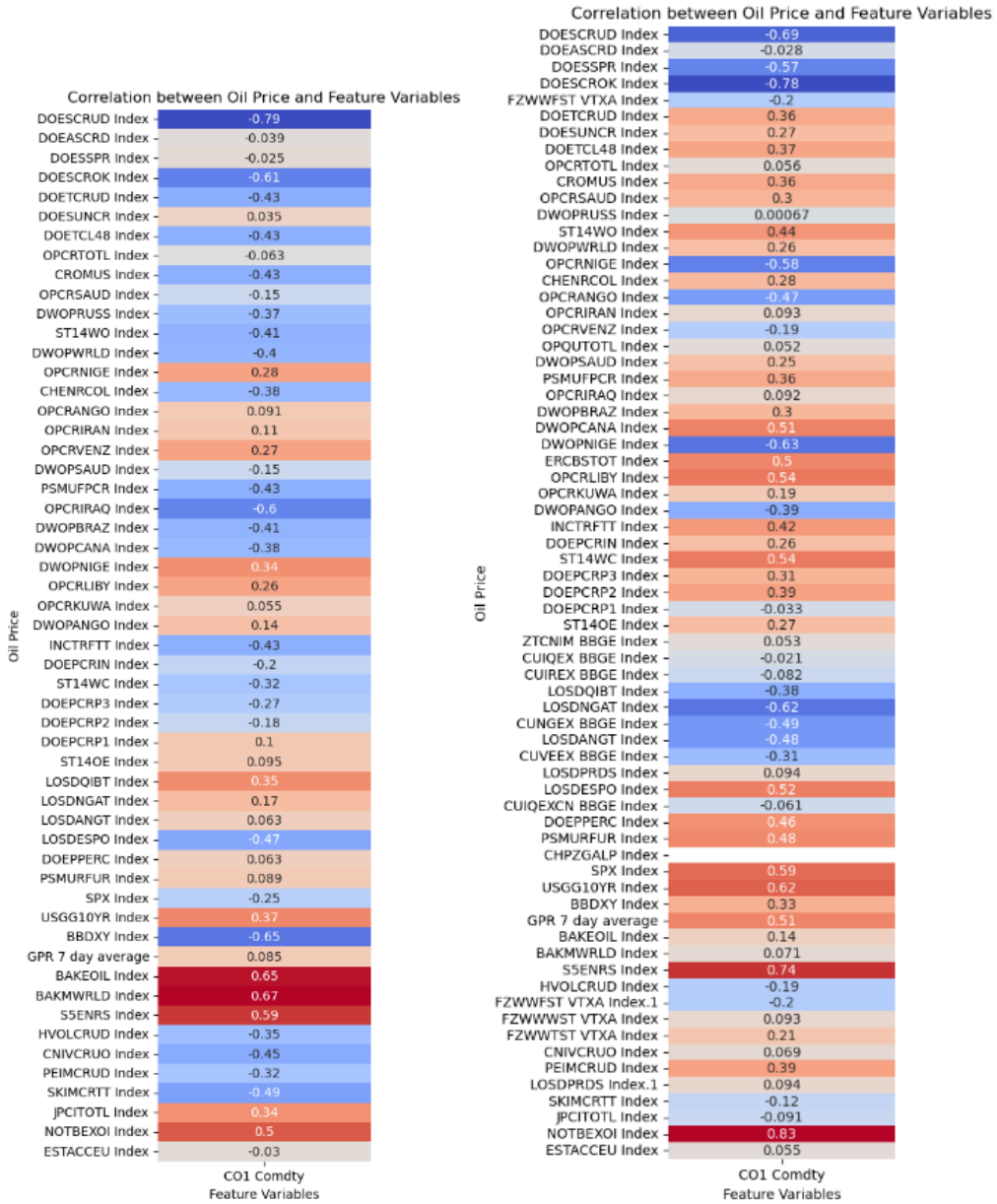

Figure 2. (left) demonstrates correlation coefficients between 54 features and oil prices used for historical data used from 2010-2023 and **Figure 3.** (right) demonstrates correlation coefficients 69 features and oil prices used for historical data from 2017-2023.

3. Data Cleaning and Pre-processing

The data was cleaned and pre-processed within Microsoft Excel since Bloomberg Excel add-in was used to import raw data using the tickers. Since several features of the data were updated on a weekly or on a monthly basis, to ensure the array integrity, the observation was repeated for days until the next observation was available for that particular feature. At the same time, a column containing the price CO1 COMDTY was altered to reflect the next day prices are as a result of current day observations that is, the column was moved by one cell upward. The features and prices were then stored in a .csv file for use in regression and modelling analysis within Python.

4. Regression and Modelling Algorithms

Given that the each dataset has more than 50 feature variables and 1 output variable (oil price) with 1500 observations, below is an analysis of the suitability of each model and why it is selected:

Table 2. Machine Learning Methods' Suitability and Selection

Method	Suitability	Selected?
Linear Least-Squares Regression	This model is a good starting point due to its simplicity and interpretability. However, it assumes a linear relationship between features and the target variable, which may not capture complex nonlinear relationships within the data.	Yes, for starting point.
Weighted Least Squares Regression	This method is useful when there is heteroscedasticity in the data, meaning the variance of the errors is not constant across observations. If the variance of errors changes with different feature values, this method might be beneficial.	No
Nonlinear Regression	If the relationship between features and the target is nonlinear, consider nonlinear regression techniques such as polynomial regression or regression with basis functions. These models can capture more	Yes, since a non-linear relationship has been expected between features and oil prices

	complex relationships compared to linear regression.	
Principal Component Analysis (PCA) based Regression	PCA is useful for dimensionality reduction if there is multicollinearity among your features or there is need to reduce the number of features while preserving most of the variance in the data. However, it is not be the best choice to interpret the relationship between individual features and the target variable.	No, since it cannot help to to interpret the relationship between individual features and the oil price.
Partial Least Squares (PLS)	Suitable with large number of predictors and multicollinearity among them. It is a dimensionality reduction technique that aims to find the directions in the feature space that explain the maximum variance in both the predictors and the response variable and can help improve prediction accuracy.	No, dimensionality reduction methods are not suitable for use with the current dataset.
Kernel Ridge Regression	Extends Ridge regression by using kernel methods to capture nonlinear relationships between features and the target variable.	Yes
Gaussian Process Regression	Suitable for modelling complex nonlinear relationships and can provide uncertainty estimates along with predictions.	Yes
Feedforward Neural Networks	Powerful for capturing complex patterns in data, but they require careful tuning and may be prone to overfitting with a small dataset.	Yes, commonly used Feedforward Neural Network is chosen.

5. Modelling Process

Two separate coding algorithms are used for each dataset, first of which applies Linear Regression, Nonlinear Regression (using statsmodels), Kernel Ridge Regression, and Gaussian Process Regression models together and another which applies the Feedforward Neural Networks (using LTSM) respectively.

Table 3. The modelling process for the first coding algorithm (Also see Appendix)

Step	Function
Loading the Dataset	The pd.read_csv() function loads the dataset from a CSV file into a DataFrame called data.
Splitting the Data	The train_test_split() function splits the dataset into training and validation sets. Here, 80% of the data is used for training, and 20% for validation.
Model Training and Prediction	We train the Linear Regression, Nonlinear Regression (using statsmodels), Kernel Ridge Regression, and Gaussian Process Regression models on the training data.
Model Evaluation	We evaluate the models using Mean Squared Error (MSE), R-squared, Adjusted R-squared, and Sum Squared Error.
Visualization	We visualize the predictions versus the actual values for each model on the validation set using scatter plots.

Table 4. The modelling process for the second coding algorithm (Also see Appendix)

Step	Function
Loading the Dataset	The pd.read_csv() function loads the dataset from a CSV file into a DataFrame called data.
Splitting the Data	The train_test_split() function splits the dataset into training and validation sets. Here, 80% of the data is used for training, and 20% for validation.
Define the Neural Network Architecture	5 hidden layers using the ReLu activation functions and 1 output layer have been specified.
Choosing a loss function and an optimizer for training the model	MSE Loss and Adam optimizer were chosen for the model
Model Training and Prediction	Loop through the dataset for a number of epochs, perform forward and backward passes, and update the model parameters.
Model Evaluation	By evaluating the trained model on the test set, we calculate the test loss and find Mean Squared Error (MSE), Mean Absolute Error (MAE) and R-squared.
Visualization	We visualize the predictions versus the actual values for each model on the validation set using scatter plots.

Residual Analysis	Analysing the distribution of residuals (the differences between predicted and actual values) to check for any patterns or systematic errors.

6. Modelling Outcomes and Results

Dataset 1 (14 years)

First coding algorithm

Linear Regression Metrics: {'MSE': 26.5001901743684, 'R-squared': 0.9585994959414958, 'Adjusted R-squared': 0.9552527486074252, 'Sum Squared Error': 19159.637496068353}
Nonlinear Regression Metrics: {'MSE': 37.15587148212258, 'R-squared': 0.9419524238138968, 'Adjusted R-squared': 0.9372599550802896, 'Sum Squared Error': 26863.695081574624}
Kernel Ridge Regression Metrics: {'MSE': 5472.17784348724, 'R-squared': -7.54903000799124, 'Adjusted R-squared': -8.240119260134245, 'Sum Squared Error': 3956384.580841275}
Gaussian Process Regression Metrics: {'MSE': 1034.871906299114, 'R-squared': -0.616751361966011, 'Adjusted R-squared': -0.7474468313464968, 'Sum Squared Error': 748212.3882542595}

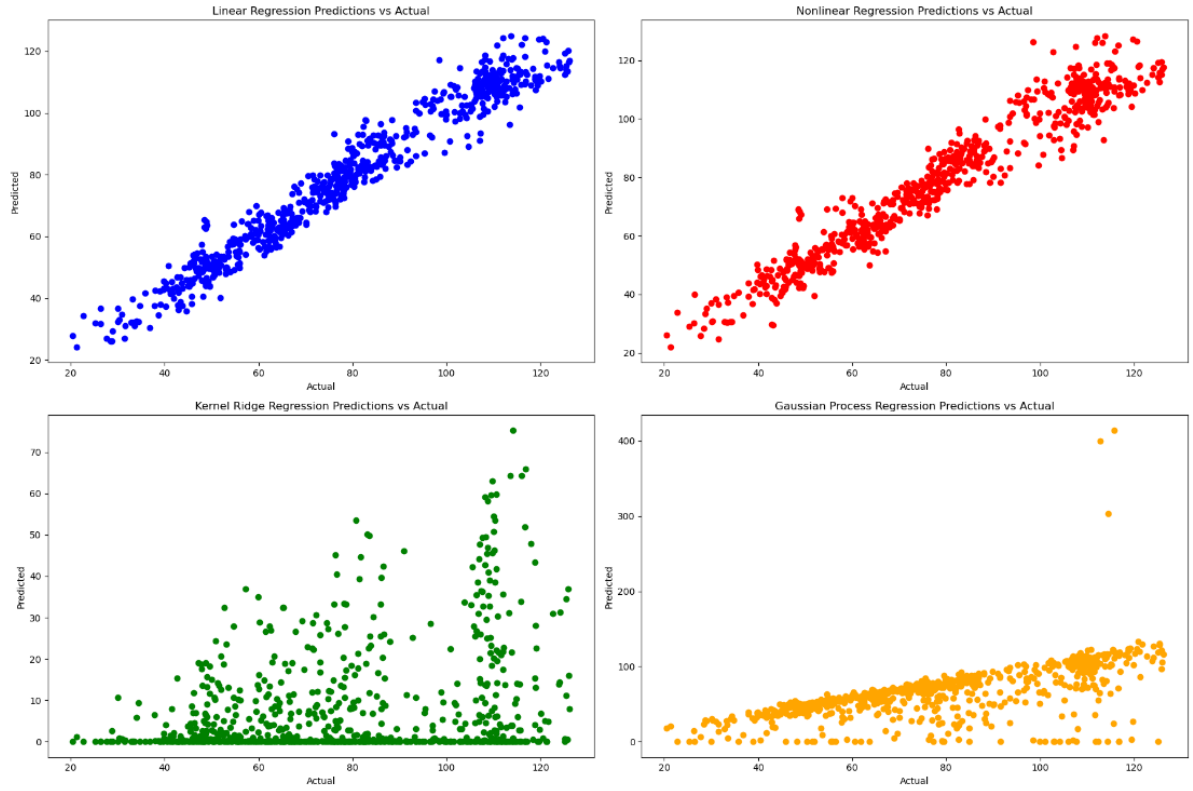

Figure 4. Predictions vs Actual Scatter Plots for Algorithm 1 for Dataset 1

Second coding algorithm

Mean Squared Error (MSE): 7.0591
Mean Absolute Error (MAE): 2.0495
R-squared: 0.9890

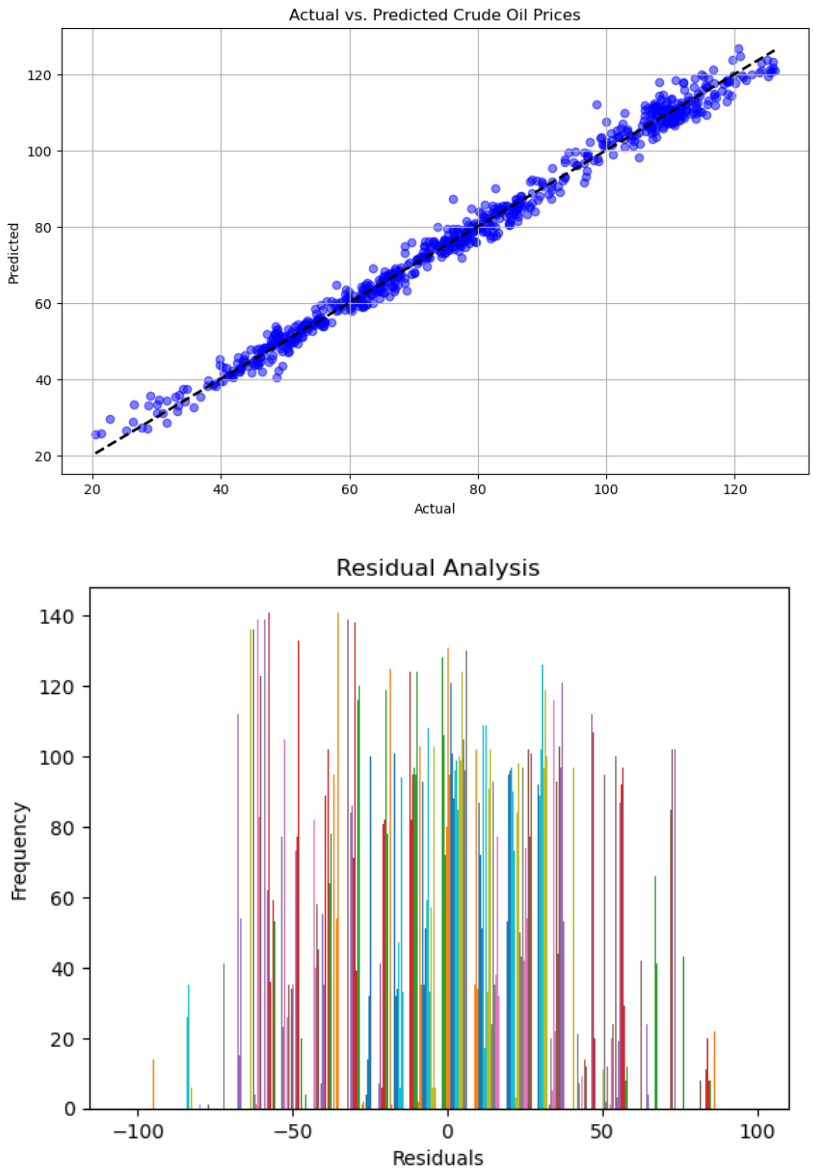

Figure 5. Predictions vs Actuals and Residual Analysis for Algorithm 2 for Dataset 1

Dataset 2 (8 years)

First coding algorithm

Linear Regression Metrics: {'MSE': 9.889596404157938, 'R-squared': 0.9720599423979265, 'Adjusted R-squared': 0.965457668512505, 'Sum Squared Error': 3580.0338983051734}
Nonlinear Regression Metrics: {'MSE': 12.097638254918929, 'R-squared': 0.9658217892947207, 'Adjusted R-squared': 0.9577454312855965, 'Sum Squared Error': 4379.345048280652}
Kernel Ridge Regression Metrics: {'MSE': 4963.73866189401, 'R-squared': -13.02353933034606, 'Adjusted R-squared': -16.337320884434686, 'Sum Squared Error': 1796873.3956056316}
Gaussian Process Regression Metrics: {'MSE': 702.0787827086218, 'R-squared': -0.9835108358744764, 'Adjusted R-squared': -1.4522171635297467, 'Sum Squared Error': 254152.5193405211}

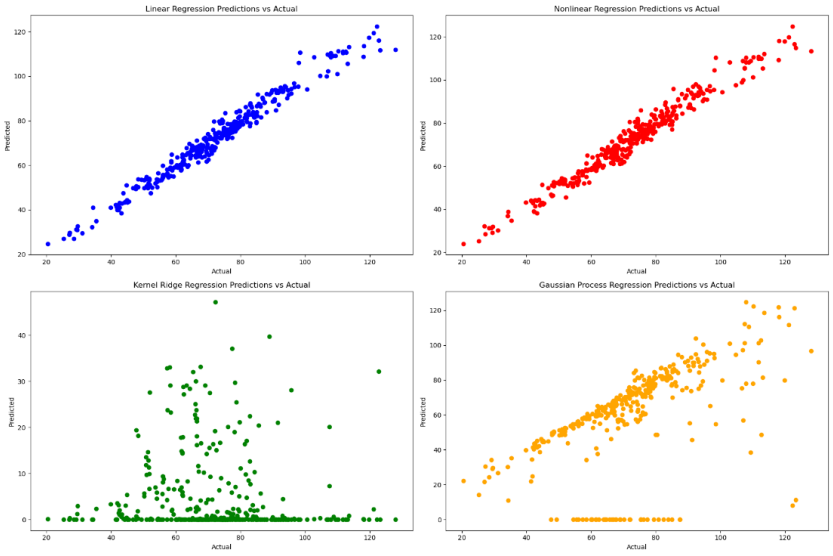

Figure 6. Predictions vs Actual Scatter Plots for Algorithm 1 for Dataset 2

Second coding algorithm

Mean Squared Error (MSE): 7.1623
Mean Absolute Error (MAE): 1.9763
R-squared: 0.9798

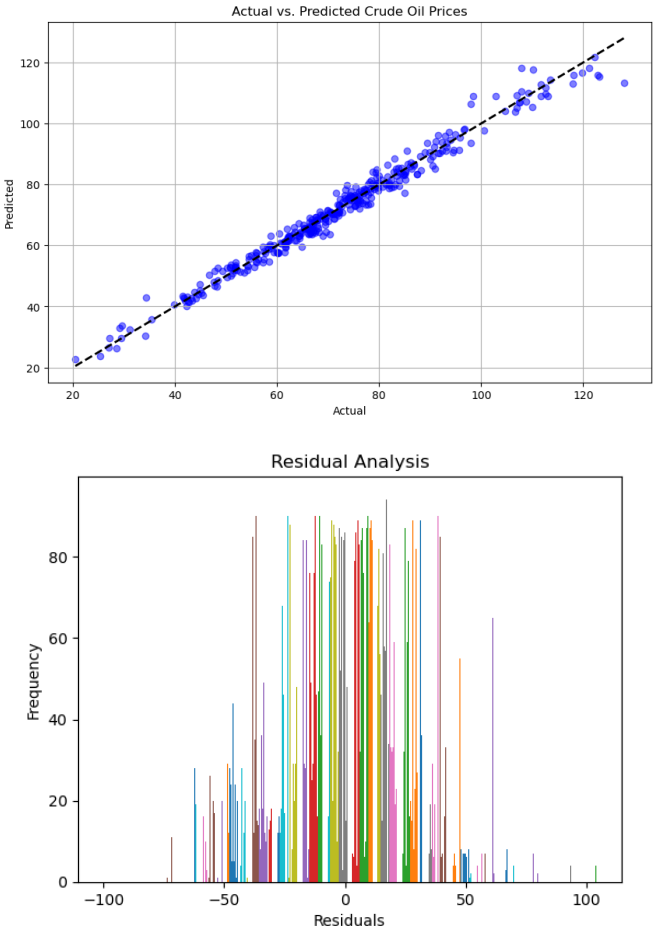

Figure 7. Predictions vs Actuals and Residual Analysis for Algorithm 2 for Dataset 2

7. Discussion of Results

Dataset 1

	Linear Regression	Nonlinear Regression	Kernel Ridge Regression	Gaussian Process Regression	Feedforward Neural Network
MSE	26.5002	37.1559	5472.1778	1034.8719	7.0591
R-squared	0.9586	0.9420	-7.5490	-0.6168	0.9890

Table 5. Model Evaluation Metric Summary for Dataset 1

Dataset 2

Model/Metric	Linear Regression	Nonlinear Regression	Kernel Ridge Regression	Gaussian Process Regression	Feedforward Neural Network
MSE	9.8896	12.0976	4963.7387	702.0788	7.1623
R-squared	0.9721	0.9658	-13.0235	-0.9835	0.9798

Table 6. Model Evaluation Metric Summary for Dataset 2

As seen in Table 5 and Table 6 results, it is clear that using LTSM neural networks to forecast Brent crude oil prices for the next day using feature variables from the previous day is the most accurate model applied with the highest R-squared value and lowest MSE value, thus meeting a better level of accuracy than the linear regression and non-linear methods applied. LSTMs (Long Short-Term Memory networks) and feedforward neural networks are often considered better than traditional regression techniques like Linear Regression, Nonlinear Regression, Kernel Ridge Regression, and Gaussian Process Regression for certain types of datasets due to their ability to capture complex temporal dependencies and nonlinear relationships. Kernel Ridge regression and Gaussian ridge regression on the other hand, resulted in large MSE value and negative R-squared value which make it infeasible for predictions. Dataset 1 which includes more observations than Dataset 2, which has more features than Dataset 1, has performed better across all metrics. Nonetheless, as seen in Figure 5 and 7, from the residual analysis, it is clear that Dataset 1 has higher variance in terms of residuals compared to Dataset 2.

There are several reasons why neural networks are preferred since they are designed to capture temporal dependencies in sequential data, learn from past observations and make predictions based on the sequence of historical data points. Neural networks also have the ability to automatically learn relevant features from the data, whereas traditional regression techniques often require manual feature engineering. This is advantageous when dealing with high-dimensional datasets or when the relationships between features and the target variable are not well understood [6].

Improving the accuracy of the neural network model involves a combination of various techniques, including hyperparameter tuning and model evaluation. By iteratively applying the techniques and carefully evaluating the model's performance, it is possible to improve accuracy and reduce bias, leading to a more effective and reliable model. Below are three techniques to improve the model:

1. Hyperparameter Tuning: Parameters such as learning rate, batch size, number of epochs and optimizer choice can significantly impact model performance [7].
2. Bias-Variance Tradeoff Analysis: Analysing the bias and variance of the model to understand its generalization performance and identify areas for improvement.
3. Feature Importance: Investigating which features have the most impact on predictions and focus on improving the quality of those features.

This model may still not however meet requirements of commercial use within a commodity trading company since it predicts the absolute price of the next day, instead of change in price from today to tomorrow. Further Analysis (Figure 8, Figure 9) has been conducted using a new column CO1 Comdty Delta = Tomorrow's price – Today's price and as we can see the current R-squared value is negative across all models tested on. To explain how measurements of predicted price are still largely inaccurate, due to real time pricing of feature variables into the CO1 Comdty price rather than each day. Therefore, to make the model powerful for commercial use, not only do we need thousands of intra-day observations for each feature variable, but the model needs to be trained continuously in real-time, for it to increase the R-squared value and reduce the MSE value in order to identify inefficiencies and make profitable trades.

CO1 Comdty Delta Modelling Outcomes and Results

First coding algorithm

Linear Regression Metrics: {'MSE': 2.4864199695754006, 'R-squared': -0.025126705598896404, 'Adjusted R-squared': -0.10799622970419653, 'Sum Squared Error': 1797.6816380030145}
Nonlinear Regression Metrics: {'MSE': 2.504354478409105, 'R-squared': -0.03252092869161438, 'Adjusted R-squared': -0.11598818939422983, 'Sum Squared Error': 1810.6482878897828}
Kernel Ridge Regression Metrics: {'MSE': 2.4792998487640907, 'R-squared': -0.022191149224640494, 'Adjusted R-squared': -0.10482336787453672, 'Sum Squared Error': 1792.5337906564375}
Gaussian Process Regression Metrics: {'MSE': 2.4257980636237897, 'R-squared': -0.0001328849669322 0075, 'Adjusted R-squared': -0.08098195051815127, 'Sum Squared Error': 1753.8519999999999}

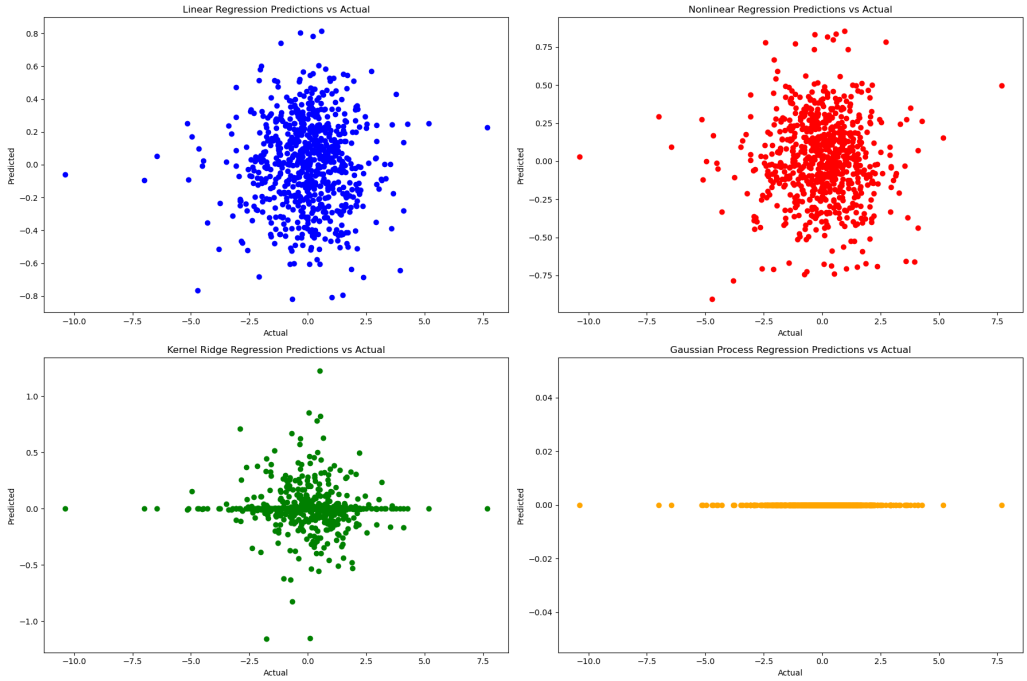

Figure 8. Predictions vs Actual Scatter Plots for Algorithm 1 for CO1 Comdty Delta dataset

Second coding algorithm

Mean Squared Error (MSE): 4.3662
Mean Absolute Error (MAE): 1.4282
R-squared: -0.8001

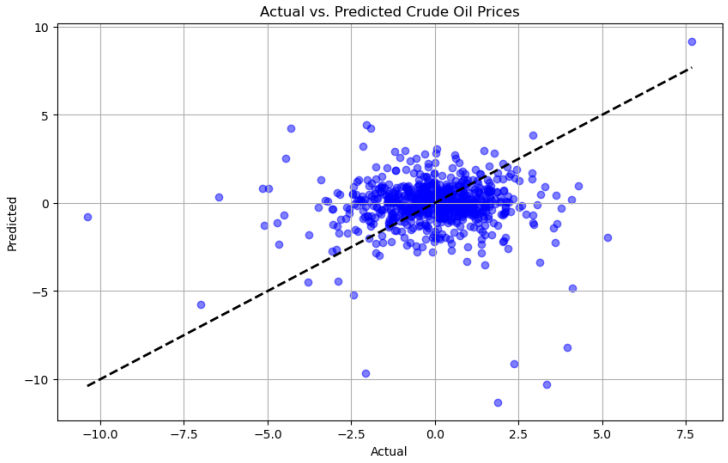

Figure 9. Predictions vs Actuals for Algorithm 2 for CO1 Comdty Delta dataset

8. Conclusion

Crude oil, known as "black gold," is vital for global economic activity and its price, influenced by supply, demand, and unpredictable events like conflicts or disasters, all affect the economy's stability. Forecasting crude oil prices accurately is crucial but challenging due to non-linearity and uncertainty nonetheless, recent research has shifted from linear econometric models to AI and ML-based approaches like Recurrent Neural Networks (RNN). For this project, a dataset with an array of relevant variables is acquired from Bloomberg. Following this, the dataset is partitioned into training and validation sets, facilitating model building and cross-validation. Various regression algorithms, encompassing both linear and nonlinear options were then deployed to model the relationships between the independent and dependent variables. Through use of evaluation metrics like sum squared error and R-squared, the performance of the models was then scrutinized. This project achieved to develop a reliable crude oil price prediction model using Feedforward neural network method on Dataset 1 with an R-squared value exceeding 0.95, at 0.989. Finally, the pitfalls of using this algorithm and dataset in commercial applications was explained using price delta dataset, showcasing the need for thousands of intra-day observations for each feature variable that the model will need and for it to be trained continuously in real-time, for it to increase the R-squared value and reduce the MSE value of the delta.

9. References

[1] What Is Crude Oil, and Why Is It Important to Investors? https://www.investopedia.com/terms/c/crude-oil.asp

[2] H. Guliyev and E. Mustafayev, "Predicting the changes in the WTI crude oil price dynamics using machine learning models," Resources policy, vol. 77, pp. 102664-, 2022, doi: 10.1016/j.resourpol.2022.102664.

[3] J. Wang, J. Wang, Forecasting energy market indices with recurrent neural networks: Case study of crude oil price fluctuations, Energy 102 (2016) 365–374.

[4] H. Chiroma, S. Abdulkareem, T. Herawan, Evolutionary neural network model for west texas intermediate crude oil price prediction, Applied Energy 142 (2015) 266 – 273.

[5] Caldara, Dario and Matteo Iacoviello (2022), "Measuring Geopolitical Risk," American Economic Review, April, 112(4), pp.1194-1225. https://www.matteoiacoviello.com/gpr.htm

[6] Why Deep Learning over Traditional Machine Learning? https://towardsdatascience.com/the-advantages-of-neural-networks-over-traditional-machine-learning-algorithms-7f594464d716

[7] How to Grid Search Hyperparameters for Deep Learning Models in Python with Keras https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/

10. Appendix

First Coding Algorithm

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.kernel_ridge import KernelRidge
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm
import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset
file_path = 'crudeoil8 (1).csv'  # Change this to the path of your dataset
data = pd.read_csv(file_path)

# Calculate correlations
correlation_matrix = data.corr()

# Extract correlations with the oil price
oil_price_correlation = correlation_matrix.iloc[:, -1]

# Plot heatmap
plt.figure(figsize=(3, 15))
sns.heatmap(oil_price_correlation[:-1].to_frame(), annot=True, cmap='coolwarm', cbar=False)
plt.title('Correlation between Oil Price and Feature Variables')
plt.xlabel('Feature Variables')
plt.ylabel('Oil Price')
plt.show()

# Load the dataset
# Load the dataset
file_path = 'crudeoil8 (1).csv'  # Change this to the path of your dataset
data = pd.read_csv(file_path)

y_index = -1  # Assuming oil price is in the last column
X = data.iloc[:, :-1]  # Features
y = data.iloc[:, y_index]  # Target variable (oil prices)

# Finding the dimensions of X and y
num_samples, num_features = X.shape
num_samples_y = y.shape[0]

print("Dimension of X:", num_samples, "samples x", num_features, "features")
print("Dimension of y:", num_samples_y, "samples")

# Split the data into training and validation sets
def split_data(X, y):
    return train_test_split(X, y, test_size=0.2, random_state=42)

# Linear Least-Squares Regression
def linear_regression(X_train, X_val, y_train):
    linear_model = LinearRegression()
    linear_model.fit(X_train, y_train)
```

```
    linear_preds = linear_model.predict(X_val)
    return linear_preds

# Nonlinear Regression (Polynomial of degree 2)
def nonlinear_regression(X_train, X_val, y_train):
    X_train_poly = sm.add_constant(X_train)
    X_val_poly = sm.add_constant(X_val)
    poly_reg = sm.OLS(y_train, sm.add_constant(X_train_poly**2))
    poly_result = poly_reg.fit()
    poly_preds = poly_result.predict(X_val_poly**2)
    return poly_preds

# Kernel Ridge Regression
def kernel_ridge_regression(X_train, X_val, y_train):
    kernel_ridge_model = KernelRidge(kernel='rbf')
    kernel_ridge_model.fit(X_train, y_train)
    kernel_ridge_preds = kernel_ridge_model.predict(X_val)
    return kernel_ridge_preds

# Gaussian Process Regression
def gaussian_process_regression(X_train, X_val, y_train):
    gaussian_kernel = RBF(length_scale=1.0)
    gaussian_process_model = GaussianProcessRegressor(kernel=gaussian_kernel)
    gaussian_process_model.fit(X_train, y_train)
    gaussian_process_preds, _ = gaussian_process_model.predict(X_val, return_std=True)
    return gaussian_process_preds

# Evaluate the models
def evaluate_model(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    adj_r2 = 1 - (1 - r2) * (len(y_true) - 1) / (len(y_true) - X.shape[1] - 1)
    sse = np.sum((y_true - y_pred)**2)
    return {'MSE': mse, 'R-squared': r2, 'Adjusted R-squared': adj_r2, 'Sum Squared Error': sse}

    # Split the data into training and validation sets
    X_train, X_val, y_train, y_val = split_data(X, y)

    # Perform linear least-squares regression
    linear_preds = linear_regression(X_train, X_val, y_train)

    # Perform nonlinear regression
    poly_preds = nonlinear_regression(X_train, X_val, y_train)

    # Perform kernel ridge regression
    kernel_ridge_preds = kernel_ridge_regression(X_train, X_val, y_train)

    # Perform Gaussian process regression
    gaussian_process_preds = gaussian_process_regression(X_train, X_val, y_train)

 #Evaluate the models
    linear_metrics = evaluate_model(y_val, linear_preds)
    poly_metrics = evaluate_model(y_val, poly_preds)
    kernel_ridge_metrics = evaluate_model(y_val, kernel_ridge_preds)
    gaussian_process_metrics = evaluate_model(y_val, gaussian_process_preds)

# Print evaluation metrics
    print("Linear Regression Metrics:", linear_metrics)
    print("Nonlinear Regression Metrics:", poly_metrics)
```

```
    print("Kernel Ridge Regression Metrics:", kernel_ridge_metrics)
    print("Gaussian Process Regression Metrics:", gaussian_process_metrics)

# Visualize predictions vs actual values for each model
    plt.figure(figsize=(18, 12))

    plt.subplot(2, 2, 1)
    plt.scatter(y_val, linear_preds, color='blue')
    plt.xlabel('Actual')
    plt.ylabel('Predicted')
    plt.title('Linear Regression Predictions vs Actual')

    plt.subplot(2, 2, 2)
    plt.scatter(y_val, poly_preds, color='red')
    plt.xlabel('Actual')
    plt.ylabel('Predicted')
    plt.title('Nonlinear Regression Predictions vs Actual')

    plt.subplot(2, 2, 3)
    plt.scatter(y_val, kernel_ridge_preds, color='green')
    plt.xlabel('Actual')
    plt.ylabel('Predicted')
    plt.title('Kernel Ridge Regression Predictions vs Actual')

    plt.subplot(2, 2, 4)
    plt.scatter(y_val, gaussian_process_preds, color='orange')
    plt.xlabel('Actual')
    plt.ylabel('Predicted')
    plt.title('Gaussian Process Regression Predictions vs Actual')

    plt.tight_layout()
    plt.show()
```

Second Coding Algorithm

```
import pandas as pd

# Load data from CSV file
data = pd.read_csv("crudeoil8 (1).csv")

# Assuming 'crude_oil_price' is the column name for crude oil prices
# and other columns are feature variables
X = data.drop(columns=['CO1 Comdty']).values
y = data['CO1 Comdty'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32)

class NeuralNet(nn.Module):
    def __init__(self, input_size):
        super(NeuralNet, self).__init__()
```

```
        self.fc1 = nn.Linear(input_size, 256)  # Input layer to first hidden layer
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, 32)
        self.fc5 = nn.Linear(32, 16)
        self.fc6 = nn.Linear(16, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))  # ReLU activation function for first hidden layer
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.relu(self.fc4(x))
        x = torch.relu(self.fc5(x))
        x = self.fc6(x)            # Output layer, no activation function
        return x

input_size = X_train.shape[1]
model = NeuralNet(input_size)

criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

num_epochs = 1000
for epoch in range(num_epochs):
    # Forward pass
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor.unsqueeze(1))  # Unsqueezing y_train_tensor to match shape

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch+1) % 100 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

with torch.no_grad():
    model.eval()
    y_pred = model(X_test_tensor)
    test_loss = criterion(y_pred, y_test_tensor.unsqueeze(1))
    print(f'Test Loss: {test_loss.item():.4f}')

# Convert predictions back to a numpy array for further analysis
y_pred_numpy = y_pred.detach().numpy()

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute evaluation metrics
mse = mean_squared_error(y_test, y_pred_numpy)
mae = mean_absolute_error(y_test, y_pred_numpy)
r2 = r2_score(y_test, y_pred_numpy)

print(f'Mean Squared Error (MSE): {mse:.4f}')
print(f'Mean Absolute Error (MAE): {mae:.4f}')
print(f'R-squared: {r2:.4f}')

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred_numpy, color='blue', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
```

```
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs. Predicted Crude Oil Prices')
plt.grid(True)
plt.show()

def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

mape = mean_absolute_percentage_error(y_test, y_pred_numpy)
print(f'Mean Absolute Percentage Error (MAPE): {mape:.4f}%')

rmse = np.sqrt(mean_squared_error(y_test, y_pred_numpy))
print(f'Root Mean Squared Error (RMSE): {rmse:.4f}')

mae = mean_absolute_error(y_test, y_pred_numpy)
print(f'Mean Absolute Error (MAE): {mae:.4f}')

r2 = r2_score(y_test, y_pred_numpy)
print(f'R-squared: {r2:.4f}')

residuals = y_test - y_pred_numpy
plt.hist(residuals, bins=20)
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Residual Analysis')
plt.show()
```