Final Project

Forecasting of Crude oil prices through Decision Tree & KNN

Jules Maheust & Lisa Manguin

Introduction

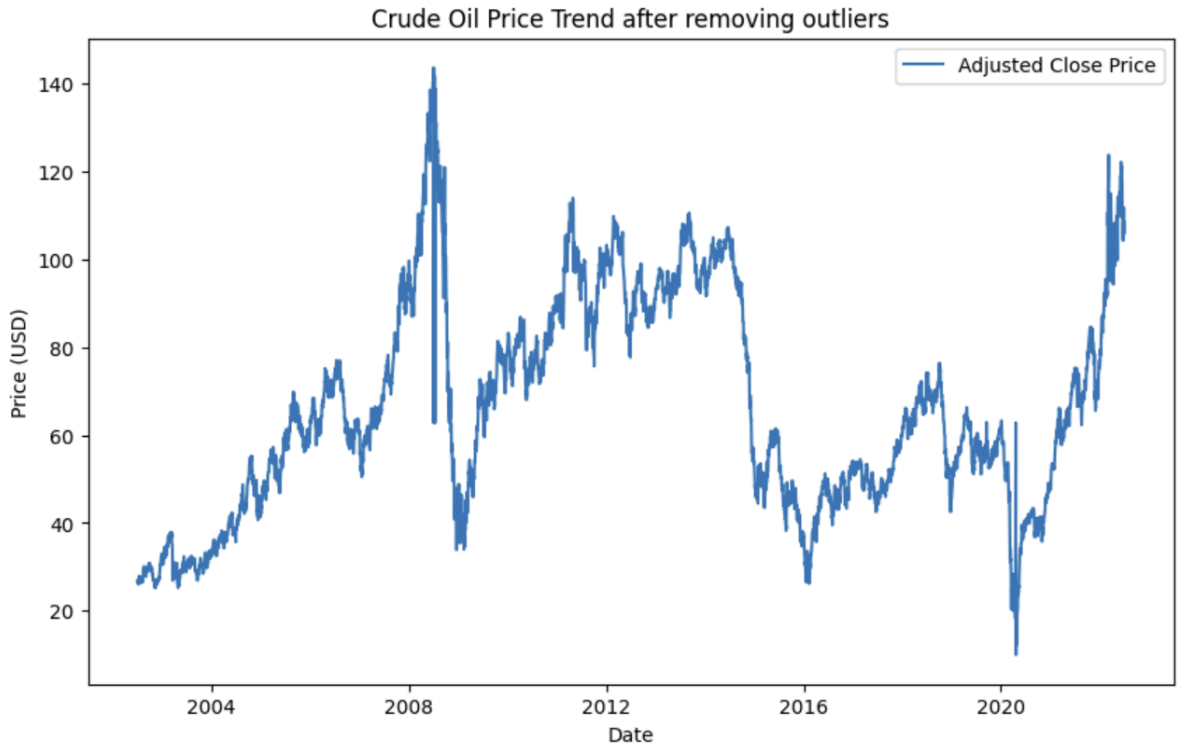
Crude oil price forecasting is critical for effective risk management, strategic optimization, and operational planning in industries heavily reliant on oil. This machine learning (ML) project aims to provide accurate short-term predictions and long-term trend analysis by leveraging historical data and market dynamics. The primary objectives include supporting data-driven decision-making, identifying key market drivers such as seasonality and anomalies, and benefiting stakeholders like producers, traders, and policymakers. Our approach encompasses implementing foundational ML methods, optimizing their performance, and advancing to state-of-the-art models like Long Short-Term Memory (LSTM) networks while incorporating real-time forecasting capabilities.

The dataset used in this project spans 20 years of crude oil price data scraped from Yahoo Finance (available on Kaggle). It contains *the date, opening price, highest and lowest daily prices, traded volume, and the adjusted closing price*, which serves as the prediction target. Our goal was to systematically identify the most effective forecasting method by experimenting with a variety of models and optimization techniques.

The project report is divided into three main parts:

- Implementation of Standard Solutions**:** Initial exploration and baseline models to set performance benchmarks.
- Improving Standard Solutions**:** Iterative optimization of the foundational models to enhance accuracy and reliability.
- Implementation of Advanced Solutions**:** Introducing cutting-edge techniques like Long Short-Term Memory (LSTM) networks for superior time-series forecasting.

Preprocessing step in this project was outlier detection and handling, particularly for the Adjusted Close (Adj Close**) column, as outliers can significantly distort machine learning model performance. Using the interquartile range (IQR) method, the Volume column was first cleaned by removing commas and converting values to numeric types for consistency. The date column was set as the index to facilitate time-series operations.
Outliers were identified as data points falling outside the bounds defined by Q1 - 1.5 × IQR and Q3 + 1.5 × IQR, calculated from the 25th and 75th percentiles of the Adj Close** column. These outliers were replaced with the median value of non-outlier entries to preserve the data's overall integrity while mitigating the influence of extreme values.

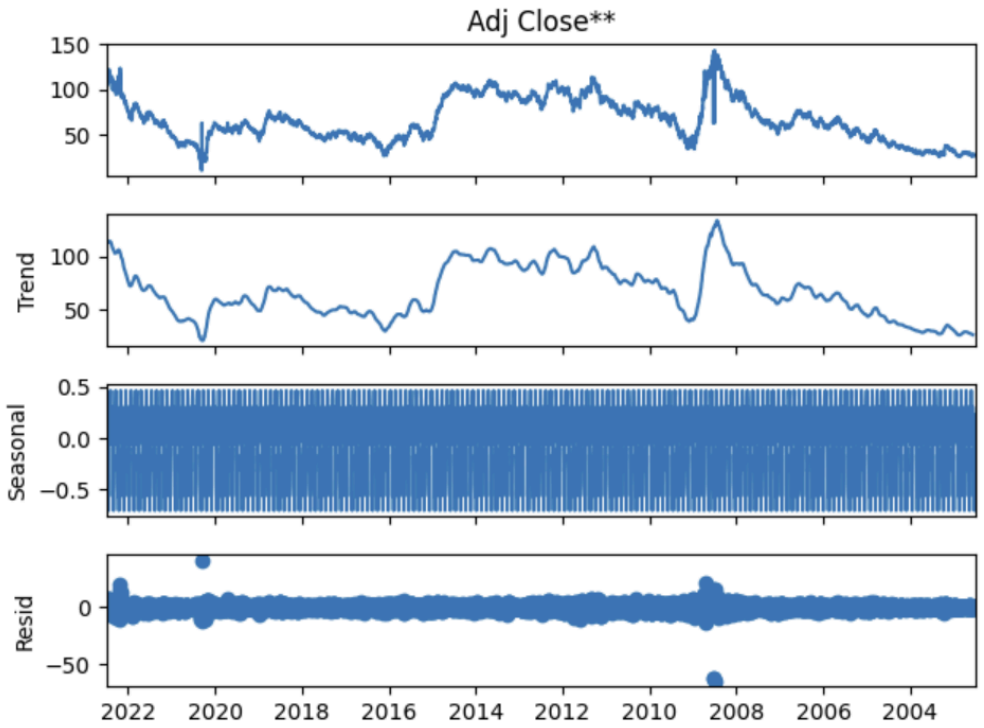


Outlier detection and treatment were particularly vital for ensuring the stability and reliability of models, especially sensitive ones like K-Nearest Neighbors (KNN). By replacing outliers with the median value rather than removing them entirely, we avoided unnecessary data loss and maintained continuity in the dataset. This preprocessing improved model robustness and ensured accurate predictions across different forecasting techniques.
By systematically addressing data quality issues and ensuring the suitability of input features for various machine learning models, this preprocessing step laid the foundation for effective implementation and optimization of forecasting methodologies.

Implementation of standards solution

Time-serie decomposition and analysis
We conducted a time series decomposition and analysis to assess the presence of stationarity or seasonality in crude oil price data. Given the nature of crude oil markets, seasonal patterns such as increased winter heating demand or summer driving season trends were evaluated to understand their potential impact on price fluctuations.

We decided to decompose our price using seasonal_decompose from statsmodels.tsa.seasonal. It is used to decompose a time series into its components: trend, seasonal, and residual.We choose the model='additive', assuming that the time series can be decomposed into components as follows: $Y(t) = T(t) + S(t) + R(t)$
where Y(t) is the observed value, T(t) is the trend, S(t) is the seasonal component, and R(t) is the residual or noise.
We set period=30, which Indicates the period of the seasonality, in this case assumes a monthly pattern for the data (e.g., daily data with a 30-day cycle).



Trend: The long-term movement shows a general decline in crude oil prices, with notable spikes and drops, particularly during economic events or market crises.

Seasonal: The oscillatory pattern highlights recurring short-term fluctuations, possibly tied to trading cycles or periodic market behaviors.

Residual: The remaining noise captures unpredictable variations, including anomalies and unexplained events, with more pronounced fluctuations in certain periods.

To evaluate the residuals from the time series decomposition, we applied several statistical and visual diagnostic tests. First, the Ljung-Box test was performed to assess the presence of autocorrelation in the residuals. The test statistic (2509.26) and p-value (0.0) strongly reject the null hypothesis of no autocorrelation, indicating that the residuals are not white noise and still contain patterns.

Next, we examined the distribution of residuals using a histogram and kernel density estimation, complemented by the Shapiro-Wilk test for normality. The Shapiro-Wilk test yielded a statistic of 0.72 and a p-value of approximately 4.05×10^{-68}, rejecting the null hypothesis of normality and confirming that the residuals deviate significantly from a normal distribution. These results collectively indicate that the decomposition model does not fully capture the patterns in the time series, leaving significant unexplained structure in the residuals, necessitating further refinement of the model or additional transformations.

Stationarity testing

We evaluated stationarity using the Augmented Dickey-Fuller (ADF) test to assess whether the Adjusted Close Price series exhibits a consistent mean and variance over time.
Initial Results:

ADF Statistic: -2.573
p-value: 0.0987

The p-value exceeded the 0.05 threshold, indicating that the series is non-stationary and contains trends or seasonality. While strict stationarity is not a requirement for machine learning models, transforming the data to a stationary format can improve performance, especially for models sensitive to trends.

We applied differencing to the Adjusted Close Price, effectively removing trends and producing a second target variable.
Post-Transformation Results:

p-value: 2.22e-23

After differencing, the p-value dropped significantly below 0.05, confirming that the transformed series is now stationary.

We will evaluate the performance of models trained on both the original and differenced data in the next stage to determine the impact of stationarization on forecasting accuracy. This approach ensures we understand the trade-offs between preserving raw trends and optimizing model performance.

Feature engineering and lagged values

To enhance predictive modeling, we engineered features and incorporated lagged values from historical data. We created :

- Temporal Features,Extracted time-based attributes such as weekday, month, quarter, and day of the month to explore potential temporal patterns (e.g *oil_data['month'] = oil_data.index.month, oil_data['quarter'] = oil_data.index.quarter*)
- Price-Related Features, rolling statistics to capture trends and fluctuations (e.g 30-day Moving Average (*7_day_ma*), 7-day Rolling Standard Deviation (*rolling_std_7*))
- Lagged Features, lagged versions of key variables like Adjusted Close Price, Open, High, Low, and Volume to capture temporal dependencies (e.g *oil_data['Adj Close Lag2'] = oil_data['Adj Close**'].shift(2)*). Missing values from rolling and shifting operations were filled using backfill (bfill).

To select the features, we used correlation analysis. We computed correlations between engineered features and the target variable (Adj Close) to identify predictive features. Lagged features and rolling statistics exhibited high correlations with Adj Close**. Temporal features (e.g., weekday, month) had negligible correlation, consistent with the absence of seasonality observed in time series analysis.

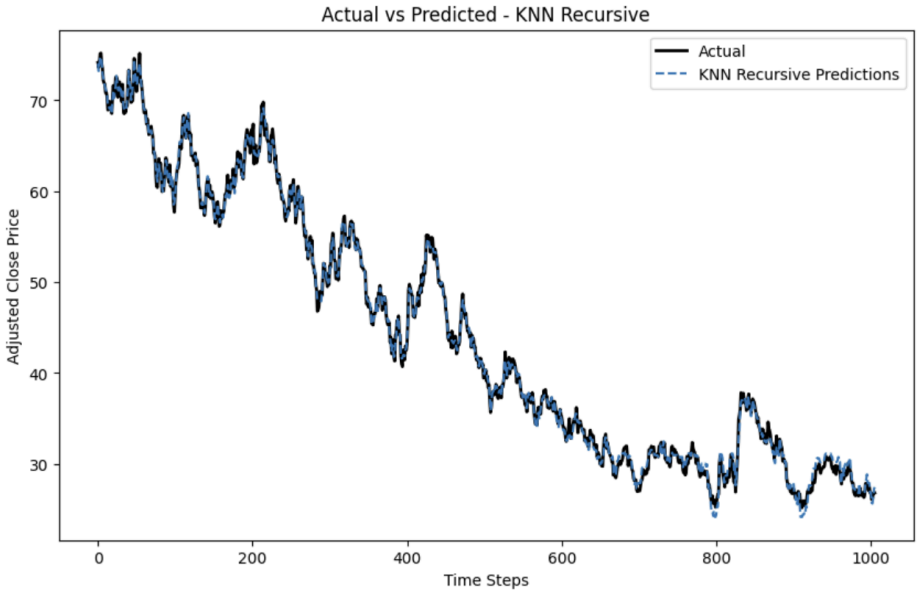
Based on the correlation results, we dropped irrelevant features, including temporal features (weekday, month, quarter, etc.) and poorly correlated ones (Volume, Adj Close_diff**). We retained the most predictive lagged and rolling features. The final selected columns include:

['Open', 'High', 'Low', '7_day_ma', '30_day_ma','rolling_std_7', 'Adj Close Lag1', 'Adj Close Lag2', 'Adj Close Lag7','Open Lag1', 'High Lag1', 'Low Lag1']
In the next stage, we will perform more robust feature selection using machine learning methods, instead of relying solely on correlation.

We are now going to explore the performance of K-Nearest Neighbors (KNN) and Decision Tree models in forecasting crude oil prices, comparing recursive and direct approaches while integrating ensemble methods like bagging and boosting. The analysis highlights how different methodologies impact prediction accuracy, computational efficiency, and robustness, with ensemble techniques proving crucial in enhancing model performance. Below, we delve into the specific results and insights for each model and forecasting approach.

K-Nearest Neighbors

- Recursive Forecasting: This approach predicted sequentially over the forecast horizon by using prior outputs as inputs. The recursive_knn_forecasting function trained a KNN model on X_train and y_train and iteratively updated test data by incorporating predictions as lagged values. Results were averaged across the horizon.
- Direct Forecasting: This method avoided sequential dependencies by training a separate KNN model for each forecast step. The direct_knn_forecasting function adjusted training data (y_train) for each step and stored the forecasts for averaging.
- Bagging with KNN: The bagging ensemble used BaggingRegressor with multiple KNN models to reduce variance and improve stability. Predictions were aggregated, and the mean squared error (MSE) was calculated.

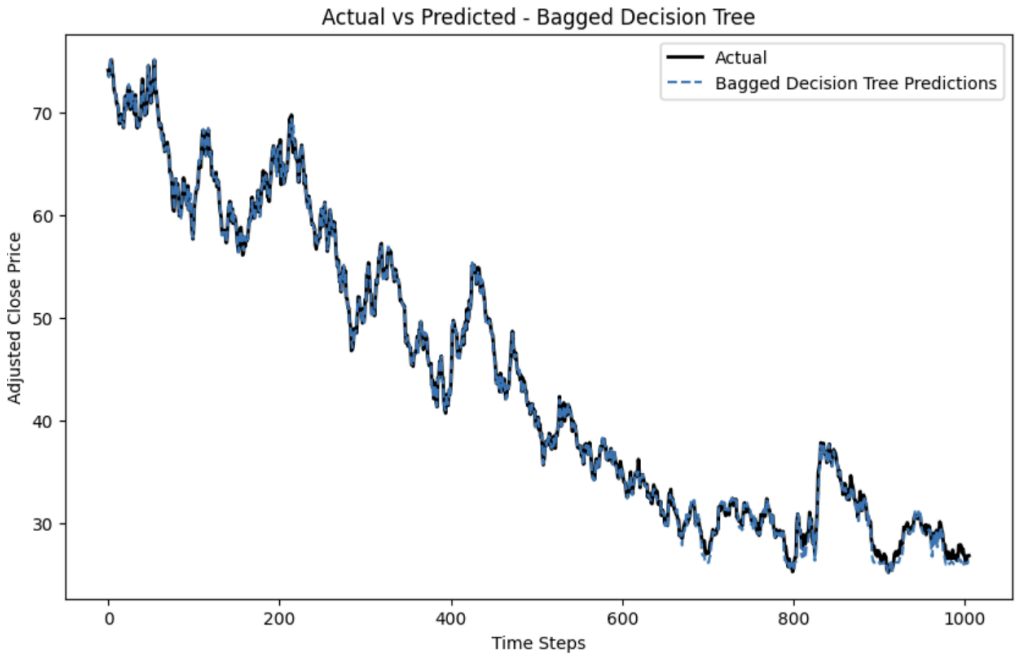


The recursive approach delivered reasonable results (MSE = 0.3259) but required significant computational resources to handle sequential updates. In contrast, direct KNN forecasting struggled significantly (MSE = 2.6581) due to increased complexity in training individual models for each forecast step. Bagging enhanced KNN's performance (MSE = 0.3120) by

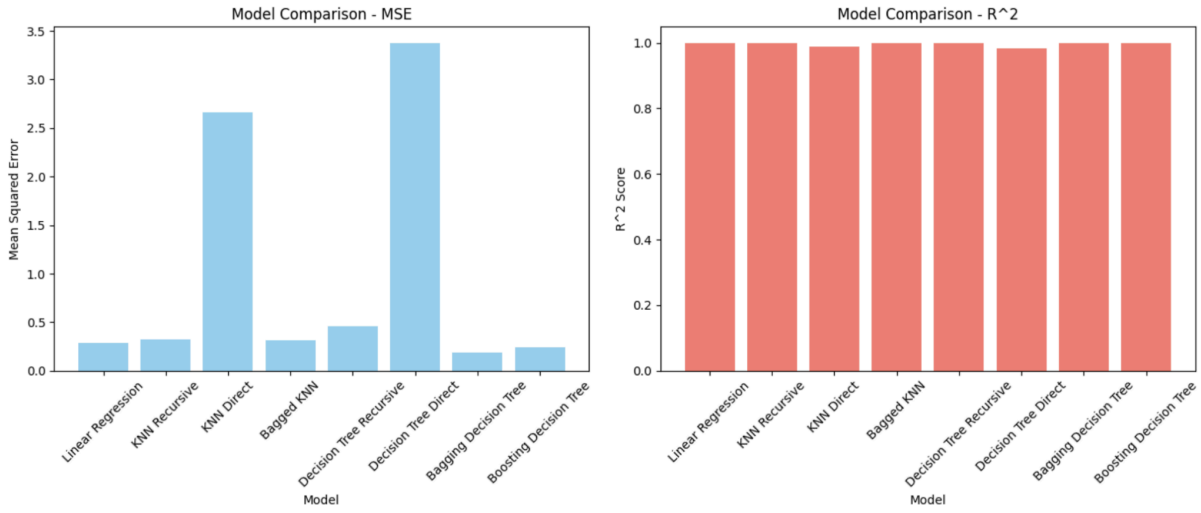
reducing variability and improving stability, though its limitations in handling extreme volatility persisted.

Decision Tree Models

- Recursive Forecasting: This approach used a single decision tree model to iteratively generate predictions for subsequent steps via the recursive_tree_forecasting function.
- Direct Forecasting: Independent decision tree models were trained for each forecast step to avoid error propagation, as implemented in the direct_tree_forecasting function.
- Bagging and Boosting:
    Bagging: Combined multiple decision tree regressors using BaggingRegressor to reduce overfitting and improve stability.
    Boosting: Utilized GradientBoostingRegressor to sequentially optimize residuals, enhancing weak decision trees for better performance.



Decision Trees consistently outperformed KNN, especially when enhanced with ensemble methods. The recursive Decision Tree model performed moderately well (MSE = 0.4560), but bagging emerged as the best-performing approach, achieving the lowest MSE of 0.1902. Bagging reduced variance and stabilized predictions by aggregating multiple models, making it highly effective. Boosting also delivered strong results (MSE = 0.2396) by capturing complex patterns through sequential error correction, though it came with higher computational demands and sensitivity to overfitting. Decision Trees, particularly with bagging and boosting, demonstrated superior accuracy and robustness compared to KNN.



Model	MSE	R^2
Linear Regression	0.289298	0.998610
KNN Recursive	0.325903	0.998434
KNN Direct	2.658150	0.987230
Bagged KNN	0.312001	0.998501
Decision Tree Recursive	0.456048	0.997809
Decision Tree Direct	3.374391	0.983789
Bagging Decision Tree	0.190195	0.999086
Boosting Decision Tree	0.239610	0.998849

Recursive vs. Direct Forecasting

Direct forecasting methods generally outperformed recursive methods, particularly for Decision Trees. Recursive forecasting suffered from cascading errors, where inaccuracies in one step propagated into subsequent predictions. This was evident in the Decision Tree models, where the recursive approach (MSE = 0.4560) outperformed the direct approach (MSE = 3.3744). Direct Decision Trees likely struggled due to overfitting and fragmentation of training data for each independent step.

However, for KNN, recursive forecasting surprisingly performed better than direct forecasting. The recursive KNN approach (MSE = 0.3259) outperformed the direct KNN method (MSE = 2.6581), demonstrating that cascading errors were less detrimental in this case. This suggests that KNN is better suited to sequential forecasting when each step leverages previous predictions.

While direct forecasting methods are preferred for robustness and avoiding cascading errors in most models (e.g., Decision Trees), recursive forecasting can still yield strong results for specific models like KNN.

Ensemble Methods

Ensemble methods significantly enhanced model performance, particularly for Decision Trees. Bagging was the most effective technique, as seen in the Bagged Decision Tree model, which achieved the lowest MSE (0.1902). By combining predictions from multiple

bootstrapped datasets, bagging reduced variance and stabilized predictions. Boosting was also highly effective (MSE = 0.2396) by sequentially correcting errors and capturing complex, non-linear relationships. However, boosting came with higher computational demands and a sensitivity to overfitting, which requires careful tuning.
For KNN, bagging slightly improved performance (MSE = 0.3120) compared to the recursive approach (MSE = 0.3259). However, it did not fully address KNN's limitations in handling extreme variations. Overall, ensemble techniques, especially bagging, proved essential for improving accuracy and robustness in crude oil price forecasting.

Model comparisons reveal that recursive KNN outperformed direct KNN, with bagging providing minor stability improvements, though KNN's computational demands and sensitivity to scaling made it less favorable than Decision Trees. For Decision Trees, bagging delivered the best results (MSE = 0.1902), followed by boosting (MSE = 0.2396), while recursive Decision Trees outperformed direct ones due to error propagation issues in the latter. Direct forecasting methods are recommended for models prone to cascading errors, such as Decision Trees, whereas recursive forecasting remains viable for KNN. Bagging emerged as the most effective ensemble method, especially for Decision Trees, offering a strong balance of accuracy and robustness. Boosting, though effective in capturing complex patterns, is computationally intensive and susceptible to overfitting. Overall, Decision Trees, particularly with bagging, are the preferred choice for crude oil price forecasting, while KNN is best suited for simpler scenarios with manageable complexity.

Improving standard solution

This stage of the study focused on enhancing the accuracy and reliability of machine learning models for time series prediction by addressing key challenges such as data stationarity, feature selection, model initialization, and hyperparameter tuning. By carefully examining the trade-offs between information loss and improved model performance, we employed differenced data to achieve higher accuracy. Additionally, advanced feature engineering and selection methods were applied to construct a robust dataset, while systematic hyperparameter tuning ensured optimal performance across various models. Finally, we evaluated the models for both short-term and long-term forecasting horizons, identifying their strengths and weaknesses under different scenarios.

Data Stationarity and Information Loss
A crucial aspect of our study was determining whether to use stationarized data, as stationarization through differencing can lead to information loss. Despite this concern, transforming the Adjusted Close Prices into a stationary series significantly improved regression model performance. Metrics like Mean Squared Error (MSE), RMSE, MAE and Mean Absolute Percentage Error (MAPE) for Decision Tree and KNN models showed marked improvement with stationarized data compared to raw prices. Initially, we evaluated the potential information loss due to differencing, but subsequent results confirmed its positive impact on accuracy.

Model	RMSE original	MAE original	RMSE differenced	MAE differenced
Decision Tree	1.8235	0.6847	2.6537	1.6597
KNN	0.4799	16.1050	1.8511	1.2576

Differenced data became the standard throughout our study, with forecasts reversed to original values for practical application. To ensure flexibility, our code was designed to seamlessly switch between original and differenced data, enabling robust validation of the approach.

Enhanced Feature Engineering and Selection

Feature engineering and selection were integral parts of our study, enabling us to create a refined dataset that improved model accuracy and efficiency. In the first part we simply selected them using correlation but here we implement several other selection methods. To facilitate this process, we developed a flexible pipeline that automates feature creation, selection, and dataset preparation. This pipeline empowers us to easily switch between features or adjust target variables as needed.

We implemented several feature selection methods to ensure that only the most relevant features were retained:

Correlation-Based Selection: Features were ranked by their Spearman correlation with the target variable, with the top n_features selected.

Mutual Information Scores: Mutual information regression identified features with the highest dependency on the target, ensuring selection of those with strong nonlinear relationships.

Lasso Regression: Features with non-zero coefficients in a Lasso regression model were selected, effectively filtering out less impactful variables.

Ensemble Selection: An ensemble approach combined the results from all three methods. Features selected by at least two methods were prioritized, and a feature importance plot visualized their significance.

To support this feature selection process, we also created a function for generating new features. These included:

Technical Indicators: Moving averages (e.g., 5-day and 20-day) to capture trends.

Lagged Features: Historical price data (e.g., 1-day, 5-day, and 10-day lags) to incorporate temporal dependencies.

Return Metrics: Percentage returns over various time windows (e.g., 1-day, 5-day, and 10-day).

Once features were created and selected, the dataset was prepared for modeling. Missing and infinite values were handled through forward/backward filling or replacement with zeros. The ensemble feature selection ensured that the most relevant features were retained, enabling efficient model training and better predictive performance.

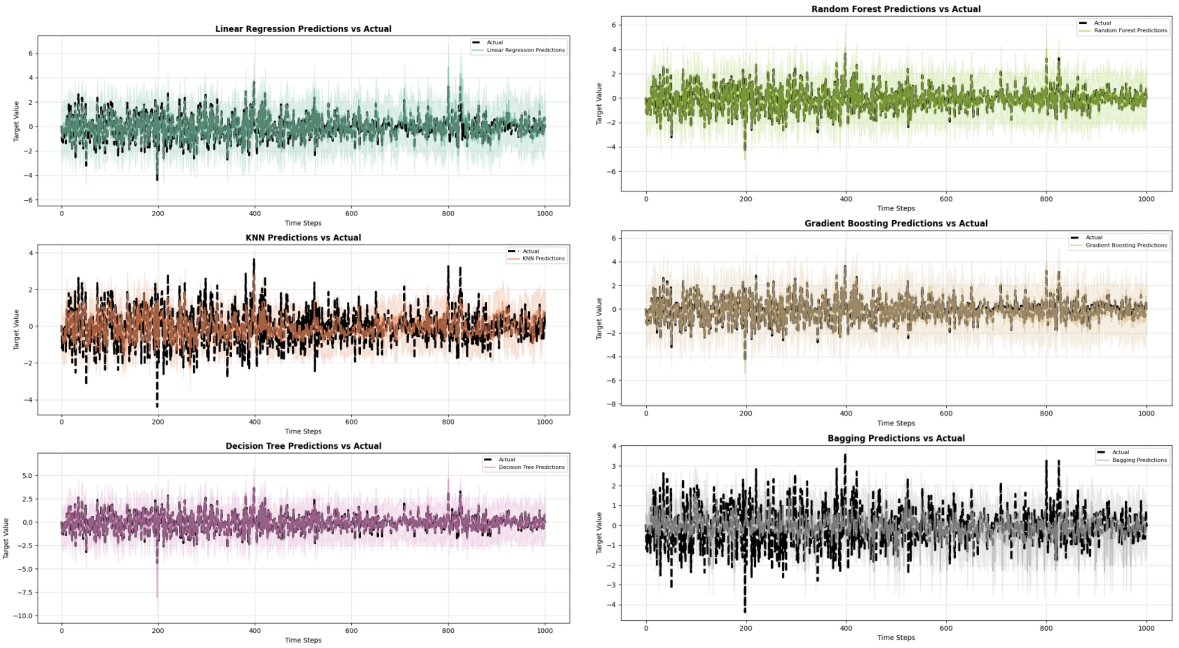
['Close*', 'return_1d', 'return_5d', 'return_10d', 'price_lag_1', 'price_lag_5', 'price_lag_10', 'Open', 'High', 'Low', 'Volume', 'ma_5', 'ma_20']

This streamlined framework allowed for easy experimentation with different target variables, such as switching between raw Adjusted Close Prices and their differenced versions, offering flexibility and robustness in time series modeling.

Streamlined Model Initialization and Training

To facilitate experimentation and optimization, we developed a robust pipeline for model initialization and training. This pipeline allowed us to compare methods quickly and evaluate the impacts of adjustments. A dedicated *initialize_models* function was created to set up a collection of machine learning models within preprocessing pipelines. StandardScaler was applied uniformly to standardize features, essential for models like KNN and Linear Regression. The models included Linear Regression, KNN, Decision Tree, Random Forest, Gradient Boosting, and Bagging, each configured with specific settings and controlled by a *random_state* parameter for reproducibility.

This flexible framework supports the integration of new models and configurations, ensuring consistent preprocessing, reproducibility, and reliable model comparisons.



Performance Summary	RMSE	MAE	MAPE (%)	R2	Skewness	Kurtosis
Linear Regression	0.3271	0.2426	1.889105e+14	0.8834	-0.9156	3.7912
KNN	0.6239	0.4888	5.240716e+14	0.5759	-0.0706	0.8276
Decision Tree	0.1534	0.0616	1.164260e+01	0.9744	12.5359	329.5299
Random Forest	0.0617	0.0355	1.797844e+11	0.9959	-1.1046	39.1401
Gradient Boosting	0.1935	0.1399	1.101333e+14	0.9592	0.2668	1.8793
Bagging	1.1686	0.9063	1.246625e+15	0.4878	0.0837	0.3978

Hyperparameter Tuning for Optimal Performance

A significant focus was placed on optimizing model performance through hyperparameter tuning. A hyperparameter_tuning function leveraging GridSearchCV was developed to identify the best parameter combinations for models like KNN, Decision Tree, Gradient Boosting, Random Forest, and Bagging. The tuning process involved cross-validation to ensure robust evaluation and avoid overfitting.

Explored Hyperparameter Ranges:

- KNN: Number of neighbors (n_neighbors) ranging from 1 to 30.
- Decision Tree: Maximum depth (max_depth from 1 to 20) and minimum samples per split (min_samples_split from 2 to 10).
- Random Forest: Number of estimators (n_estimators from 50 to 200) and maximum depth (max_depth from 1 to 20).
- Gradient Boosting: Number of estimators (n_estimators from 50 to 200) and learning rate (learning_rate from 0.01 to 0.3).
- Bagging: Number of base estimators (n_estimators from 10 to 100).

Impact of Hyperparameter Tuning:

- For KNN, the optimal n_neighbors minimized overfitting while capturing local patterns.
- Decision Tree and Random Forest models achieved improved generalization with optimized depth and split parameters.
- Gradient Boosting benefitted from fine-tuned learning rates and estimators, balancing bias and variance.

Model	Best Parameters	Time Taken	Folds x Candidates
KNN	{'model__n_neighbors': 3, 'model__p': 2, 'model__weights': 'distance'}	8.67 seconds	5 folds x 12 candidates
Decision Tree	{'model__max_depth': None, 'model__min_samples_leaf': 2, 'model__min_samples_split': 5}	2.67 seconds	5 folds x 12 candidates
Random Forest	{'model__bootstrap': True, 'model__max_depth': None, 'model__min_samples_leaf': 1, 'model__min_samples_split': 2, 'model__n_estimators': 50}	31.60 seconds	5 folds x 2 candidates
Gradient Boosting	{'model__learning_rate': 0.1, 'model__max_depth': 5, 'model__n_estimators': 100}	34.94 seconds	5 folds x 2 candidates
Bagging	{'model__max_samples': 0.5, 'model__n_estimators': 50}	8.68 seconds	5 folds x 4 candidates

Post-tuning, models were evaluated on an independent test set, confirming their ability to generalize to unseen data. This systematic approach improved performance, reduced overfitting, and ensured consistency across evaluations.

Evaluation of Short-Term vs. Long-Term Predictions
In our study, we aimed to evaluate the performance of machine learning models for both short-term and long-term prediction horizons. The short-term horizon was defined as 7 days, while the long-term horizon covered 30 days. This approach allowed us to analyze model behavior across varying timeframes, ensuring robust and adaptable forecasting methods. Each model was trained and tested on the dataset, and predictions were evaluated separately for the two horizons. Metrics such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE) were calculated to compare performance. The results revealed distinct strengths of each model, with some excelling in short-term accuracy and others performing better over longer durations.

To make informed decisions on model selection, we developed a ranking function that prioritizes models based on RMSE for both horizons. The ranking provided insights into the optimal models depending on the forecasting timeframe. For instance, Random Forest emerged as the top performer for both short-term and long-term predictions, while Gradient Boosting demonstrated competitive accuracy, particularly in the long term. This structured evaluation enabled us to identify and recommend the most suitable models for different prediction needs, ensuring both precision and reliability in our forecasts**.**

Model	Short-Term RMSE	Long-Term RMSE	Short-Term RMSE Rank	Long-Term RMSE Rank
Random Forest	0.008164	0.016819	1	1
Gradient Boosting	0.038842	0.067909	2	3
Decision Tree	0.046599	0.045387	3	2
Linear Regression	0.334399	0.291406	4	4
KNN	0.397173	0.64471	5	5
Bagging	1.016363	1.39925	6	6

The systematic approach adopted in this stage led to significant improvements in predictive accuracy and model robustness. Differenced data emerged as a critical transformation, reducing noise and enhancing model performance, while the ensemble feature selection process ensured the inclusion of highly relevant variables. Hyperparameter tuning further optimized model configurations, resulting in reduced overfitting and improved generalization. The evaluation of short-term and long-term horizons highlighted the varying strengths of the models, with Random Forest and Gradient Boosting consistently ranking among the top performers. These comprehensive steps have laid a strong foundation for reliable and adaptable forecasting in diverse time series applications.

implementation of advanced solutions

In this stage, the focus was on advancing crude oil price forecasting by introducing Long Short-Term Memory (LSTM) networks, a type of deep learning model specifically designed to capture temporal dependencies in sequential data. Unlike traditional machine learning models such as Decision Trees, Random Forest, and Gradient Boosting, LSTMs excel in handling time-series data, making them ideal for financial forecasting tasks.

To ensure a robust comparison, LSTM performance was evaluated alongside the top models from the previous stage (Decision Tree, random forest, and gradient boosting) using metrics like RMSE, MAE, MAPE, and R^2. The targeted value, when implementing LSTM, is the adjusted closed price and not the differenced (stationnarized one used for others ML methods).

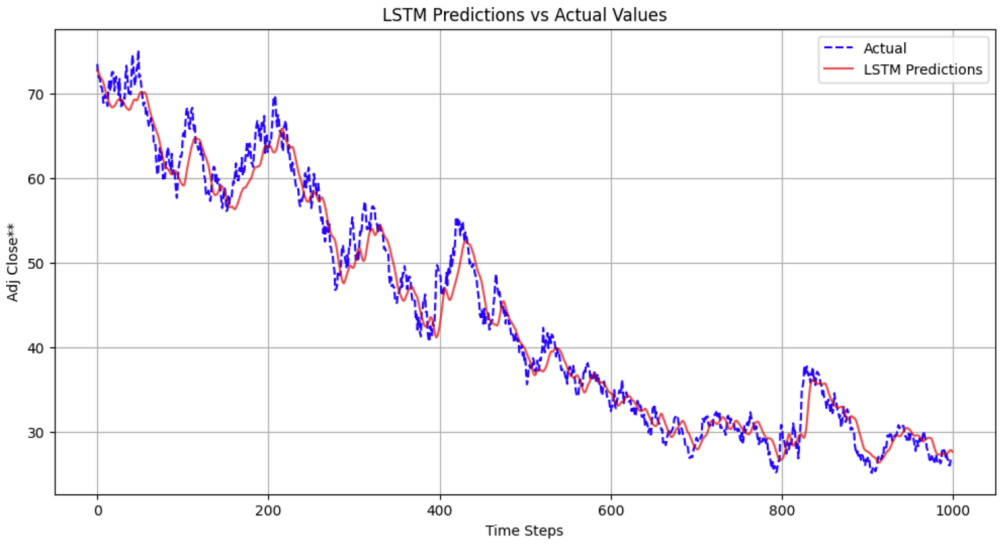
Methodology & features

The methodology included retaining the feature set ['return_1d', 'return_5d', 'return_10d', 'price_lag_1', 'price_lag_5', 'price_lag_10', 'Open', 'High', 'Low', 'Volume', 'ma_5', 'ma_20'] from previous stage, stationarizing the target variable for traditional models, and normalizing all features using MinMaxScaler, crucial for LSTM training.
LSTMs were trained on sequences of past data, using a two-layer architecture with 50 units per layer, optimized through grid search and early stopping.

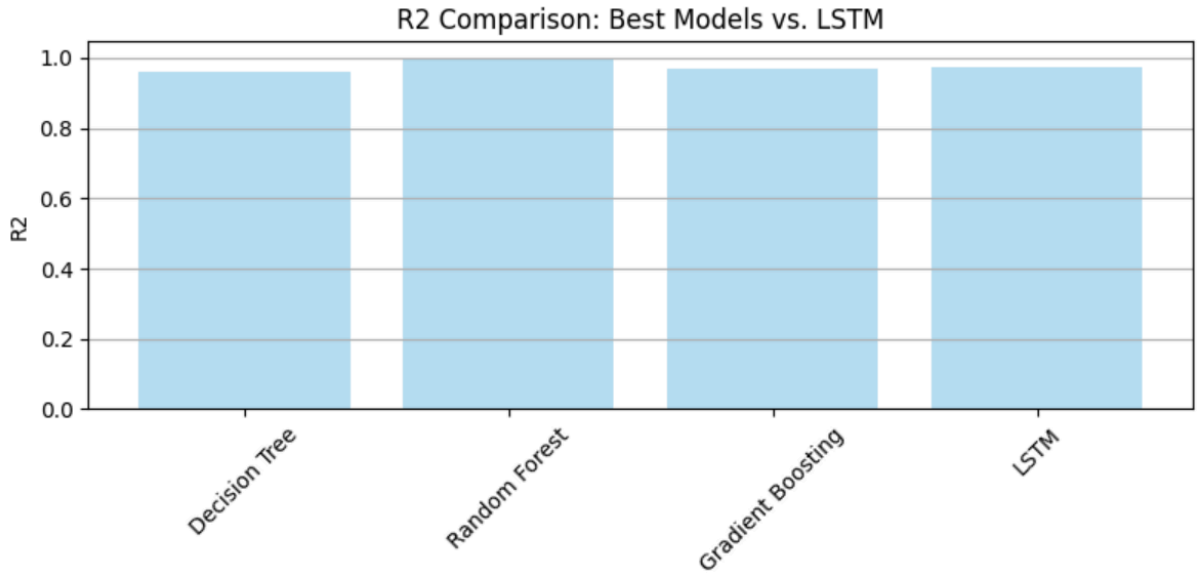
```
lstm_model = Sequential([
    LSTM(50, activation='relu', return_sequences=True, input_shape=(X_train_seq.shape[1], X_train_seq.shape[2])),
    Dropout(0.2),
    LSTM(50, activation='relu', return_sequences=False),
    Dropout(0.2),
    Dense(1)
])

lstm_model.compile(optimizer='adam', loss='mse')
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
history = lstm_model.fit(X_train_seq, y_train_seq, validation_split=0.2, epochs=50, batch_size=32, callbacks=[early_stopping])

y_pred_lstm = lstm_model.predict(X_test_seq)
lstm_metrics = {
    'RMSE': np.sqrt(mean_squared_error(y_test_seq, y_pred_lstm)),
    'MAE': mean_absolute_error(y_test_seq, y_pred_lstm),
    'R2': r2_score(y_test_seq, y_pred_lstm),
    'MAPE': mean_absolute_percentage_error(y_test_seq, y_pred_lstm)
}
```

Implementation

While LSTMs demonstrated their potential for capturing sequential trends and modeling non-stationary data, their performance highlighted challenges like the need for extensive hyperparameter tuning and larger datasets to achieve superior results. This stage underscored LSTM's promise for complex forecasting tasks while emphasizing the strengths of traditional models in current applications.



Evaluation and Performance Comparison
After evaluating the models, key performance metrics— R^2, RMSE, MAE, and MAPE—highlighted distinct strengths and weaknesses among the approaches.



Model	RMSE	MAE	R2	MAPE
Decision Tree	0.190101	0.065379	0.960632	1.206665e-01
Random Forest	0.059518	0.036449	0.996141	1.057340e-01
Gradient Boosting	0.174895	0.130764	0.966678	3.658083e+11
LSTM	2.293658	1.758964	0.974208	4.031034e-02

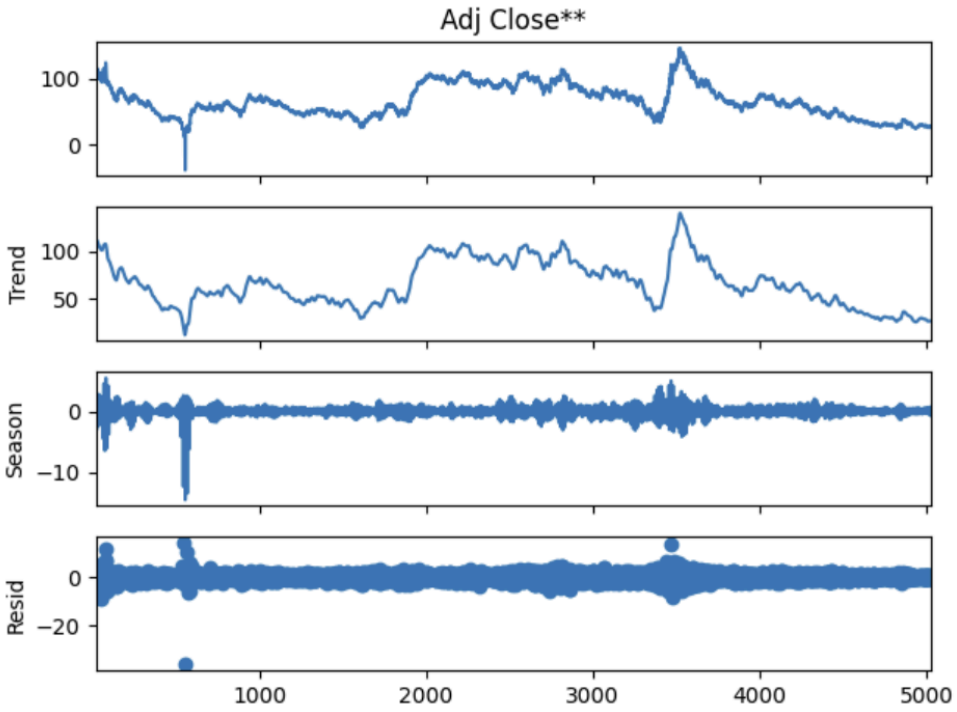
Random Forest emerged as the best-performing model, achieving the lowest errors and the highest R^2 value (0.996), indicating superior accuracy and fit. In contrast, the LSTM model underperformed, with significantly higher RMSE (2.378) and MAE (1.942), suggesting its inability to generalize effectively on the dataset used. Interestingly, Gradient Boosting showed competitive performance in RMSE and R^2 but struggled with MAPE due to potential outliers or instability during training. The observed shortcomings in LSTM performance can be attributed to insufficient hyperparameter tuning, limited data size, and possible overfitting despite regularization measures like early stopping.

While LSTM offers advantages such as capturing sequential patterns and flexibility with non-stationary data, its success depends heavily on optimized hyperparameters and large datasets, both of which were constrained in this study. Traditional models, particularly Random Forest, demonstrated their robustness and practicality for this forecasting task. Future efforts could focus on enhancing LSTM performance through advanced tuning, larger datasets, and hybrid approaches that combine its strengths with those of traditional models.

Review of Time-Series decomposition

In the earlier stages of this work, we attempted a standard time series decomposition into trend, seasonal, and residual components, using an additive model defined as $Y(t)=T(t)+S(t)+R(t)$. However, this approach failed to detect meaningful patterns in the data. The residuals were not white noise, and we could not draw any useful conclusions. We only identified that the data lacked stationarity, which led us to create stationarized returns and compare accuracy metrics with and without stationarization.

Now, we aim to implement a more advanced decomposition method: STL Decomposition (Seasonal and Trend Decomposition using Loess). This robust and flexible tool is specifically designed for handling non-linear trends and changing seasonality over time. STL supports both additive and multiplicative models, can manage irregular or evolving seasonality, and automatically handles missing data. A better understanding of time series decomposition is crucial for generating relevant engineered features linked to these components, which can ultimately improve the performance of forecasting models.



This figure provides a detailed decomposition of the "Adj Close**" time series into its main components: observed, trend, seasonal, and residual. The observed series reveals significant fluctuations, including downward trends and notable spikes. The trend captures the overall direction of crude oil prices, showing a general decline with occasional recovery periods, likely influenced by economic or geopolitical events. The seasonal component highlights recurring patterns, though its small magnitude suggests that seasonality might not be a dominant factor or needs further validation. The residuals, representing the noise or irregularities after removing the trend and seasonal components, exhibit extreme outliers at specific points, indicating unexplained anomalies. These results align with the earlier review of time-series decomposition, where an additive model failed to detect meaningful patterns, leaving residuals that were not white noise and highlighting a lack of stationarity in the data.

Conclusion

In this project, we explored a variety of machine learning techniques for crude oil price forecasting, ranging from traditional models such as K-Nearest Neighbors (KNN) and Decision Trees to advanced approaches like ensemble methods and Long Short-Term Memory (LSTM) networks. Through systematic experimentation, we identified Random Forest as the most effective model, offering superior accuracy and robustness for both short-term and long-term predictions. Our work also highlighted the importance of data preprocessing, feature engineering, and hyperparameter tuning in enhancing model performance. While LSTM networks showed potential for capturing temporal dependencies, their performance was limited by data constraints and hyperparameter challenges. Overall, this study underscores the value of combining robust traditional methods with emerging advanced techniques for time series forecasting.

To enhance crude oil price forecasting, future research could incorporate market-specific factors such as geopolitical events, macroeconomic indicators, and weather-related variables. Geopolitical events like wars, trade embargoes, and OPEC decisions often influence trends and residuals, while macroeconomic indicators such as global GDP growth, inflation rates, and currency exchange trends can explain anomalies in the data. Additionally, weather variables like heating degree days (HDDs) and cooling degree days (CDDs) can capture seasonal demand fluctuations, and extreme weather events can account for supply chain disruptions. Integrating sentiment analysis from news and social media could provide real-time insights into market dynamics, offering improved forecast precision by capturing short-term fluctuations driven by public and market perceptions. By including these factors alongside advanced economic indicators, researchers can develop more comprehensive and adaptable forecasting systems for crude oil price prediction.

Bibliographie

- **Dataset** Crude Oil price data
- **Deep learning systems for forecasting the prices of crude oil and precious metals** Published in: Financial Innovation, 2024 JFin
- **Forecasting Crude Oil Price Using Multiple Factors** Published in: Journal of Risk and Financial Management, 2024 MDPI
- **Crude oil price forecasting using K-means clustering and LSTM deep learning models** Published in: Journal of Big Data, 2024 Journal of Big Data
- **Enhancing Multistep Brent Oil Price Forecasting with a Multi-Aspect Metaheuristic Optimization Approach and Ensemble Deep Learning Models** Published in: arXiv preprint, 2024 Arxiv