**TITLE OF THE PROJECT**

**A CAPSTONE PROJECT REPORT**

*Submitted in partial fulfillment of the*

*requirement for the award of the*

*Degree of*

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE & ENGINEERING**

*by*

**ABHIRUP DASS (21BCE8776)**

**VARUN VIRENDRA SHETGAONKAR (21BCE8330)**

**PARTH ANAND JOSHI (21BCE7344)**

**PREETAM SHOW (21BCE7862)**

*Under the Guidance of*

**DR. HEMANT KUMAR REDDY**



SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

VIT-AP UNIVERSITY

AMARAVATI- 522237

*DECEMBER 2024*

**CERTIFICATE**

This is to certify that the Capstone Project work titled “**CRUDE OIL PRICE PREDICTION USING TIME SERIES FORECASTING**” that is being submitted by **ABHIRUP DASS (21BCE8776), VARUN VIRENDRA SHETGAONKAR (21BCE8330), PARTH ANAND JOSHI (21BCE7344), and PREETAM SHOW (21BCE7862)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Dr. Hemant Kumar Reddy

Guide

**The thesis is satisfactory / unsatisfactory**

**Internal Examiner1 Internal Examiner2**

**Approved by**

HoD, Department of Data Science and Engineering

School of Computer Science and Engineering

**ACKNOWLEDGEMENTS**

**ABSTRACT**

This study explores crude oil price prediction through the lens of time series forecasting, employing a systematic comparison of statistical, machine learning (ML), deep learning (DL), and hybrid models. A suite of error metrics is utilized to assess model performance comprehensively. To address the inherent complexities of crude oil price dynamics, a novel ensemble-based methodology is proposed, combining hybridized models to enhance forecasting accuracy. The results demonstrate the effectiveness of this approach in capturing temporal patterns and providing reliable predictions, offering valuable insights into advanced forecasting frameworks for volatile economic indicators.

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.No.** | **Chapter** | **Title** | **Page Number** |
| **1.** |  | **Acknowledgement** | **2** |
| **2.** |  | **Abstract** | **3** |
| **3.** |  | **List of Figures and Table** | **5** |
| **4.** | **1**  **1.1**  **1.2**  **1.3** | **Introduction**  **Objectives**  **Background and Literature Survey**  **Organization of the Report** | **7**  **7**  **8**    **9** |
| **5.** | **2**  **2.1**  **2.2**  **2.3** | **(Work)**  **Proposed System**  **Working Methodology**  **Experimental Setup** | **10**  **10**  **10**  **13** |
| **6.** | **4** | **Results and Discussion** | **16** |
| **7.** | **5** | **Conclusion & Future Works** | **20** |
| **8.** | **6** | **Codes** | **22** |
| **9.** | **7** | **References** | **52** |

**List of Tables**

|  |  |  |
| --- | --- | --- |
| **Table No.** | **Title** | **Page No.** |
| 1 | RMSE, SMAPE, MAE and MASE values for different models | 26 |

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Title** | **Page No.** |
| 1 | Ensembled Model Architecture | 10 |
| 2 | MAE for different models | 24 |
| 3 | MASE for different models | 25 |
| 4 | SMAPE for different models | 25 |
| 5 | RMSE for different models | 26 |
| 6 | Ensemble forecasting with LSTM, ARIMA and Random Forest | 33 |
| 7 | Ensemble forecasting with aARIMA - GRU and mARIMA - GRU | 40 |
| 8 | Ensemble forecasting with SVR and XGBoost | 46 |
| 9 | Ensemble forecasting with SVM and XGBoost | 52 |

**CHAPTER 1**

**INTRODUCTION**

Crude oil prices are a cornerstone of global economic stability, shaping energy markets, trade balances, and industrial costs. Their high volatility, driven by factors such as geopolitical events, supply-demand imbalances, and market speculation, makes accurate forecasting crucial for nations, companies, and investors. Reliable predictions support better resource management, risk mitigation, and strategic decision-making.

Traditional models like ARIMA capture linear trends but struggle with the complexities of crude oil price dynamics, while deep learning models excel at non-linear patterns but face issues like overfitting. To address these challenges, ensembled hybrid approaches combine statistical and deep learning techniques, leveraging their strengths to deliver more accurate and robust forecasts.

This study explores the development and optimization of ensembled hybrid models for crude oil price forecasting, using weekly data from the U.S. Energy Information Administration (EIA). These models aim to capture both linear and non-linear trends effectively, providing improved accuracy and reliability compared to standalone methods.

* 1. **Objectives**

1.Development of Hybrid Ensemble Models : To design and implement ensemble models combining additive ARIMA-GRU (Aarima-GRU) and multiplicative ARIMA-GRU (Marima-GRU) approaches for improved crude oil price forecasting.

2.Optimization : To fine-tune model parameters and architectures for effectively capturing both linear and non-linear dynamics in crude oil price time series.

3.Comprehensive Evaluation : To evaluate the proposed models using deterministic and probabilistic forecasting metrics, ensuring high accuracy and robustness.

4.Comparative Analysis: To benchmark the Aarima-GRU and Marima-GRU ensemble models against standalone statistical and deep learning approaches, demonstrating their superiority in performance.

5.Real-World Application : To provide reliable forecasting tools for stakeholders such as companies, investors, and regulators, aiding in resource distribution, market strategies, and energy management decisions.

* 1. **Background and Literature Survey**

Crude oil price forecasting is crucial due to its significant impact on global industrial and economic activities. Given the volatility and non-linear nature of crude oil prices, various statistical and machine learning techniques have been explored.

Traditional statistical models like ARIMA (Auto-Regressive Integrated Moving Average) are commonly used for time series forecasting, as they can model linear correlations. However, their effectiveness is limited in capturing the complex dynamics of crude oil prices, as they assume linearity. Extensions like ARFIMA (Auto-Regressive Fractionally Integrated Moving Average) address long-term memory effects but still face limitations with non-linear data.

Machine learning and deep learning methods, such as Artificial Neural Networks (ANNs) and Support Vector Machines (SVMs), have shown promise in modeling non-linear dependencies. Deep learning models like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) excel at handling sequential data. LSTMs capture long-term dependencies, while GRUs provide computational advantages with simpler architectures. However, these models can be computationally expensive and require careful hyperparameter tuning.

Recent research has focused on combining statistical and machine learning techniques in hybrid models, such as ARIMA-LSTM and ARIMA-GRU, which blend linear and non-linear components. Ensemble methods like additive and multiplicative ensembles improve performance by combining predictions from multiple models. While these hybrid and ensemble models have shown improvements, further optimization of ensemble weights could enhance prediction accuracy and robustness.

Despite these advancements, gaps remain in crude oil price forecasting. Many studies rely on pre-selected models and hyperparameters without optimization, leading to suboptimal performance. Additionally, few studies explore probabilistic forecasting to capture uncertainty, and the scalability of ensemble methods for real-time applications has not been extensively investigated.

* 1. **Organization of the Report**

The remaining chapters of the project report are described as follows:

* Chapter 2 contains the proposed system, methodology, and software details.
* Chapter 3 discusses the results obtained after the project was implemented.
* Chapter 4 concludes the report.
* Chapter 5 consists of codes.
* Chapter 6 gives references.

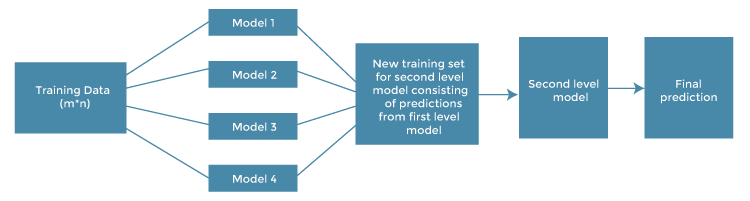
**CHAPTER 2**

**Work**

This Chapter describes the proposed system, working methodology details.

**2.1 Proposed System**

The following block diagram (figure 1) shows the system architecture of this project.



**Figure 1. Ensembled Model Architecture**

**2.2 Working Methodology**

**METHODOLOGY**

This section describes the systematic approach and techniques used to develop ensemble-based models for crude oil price prediction. The methodology ensures effective data handling, model development, and evaluation to address the complexities of crude oil price time series forecasting.

* **Data Collection and Preprocessing**
* **Dataset:**
* Weekly crude oil price data spanning two decades was sourced from public repositories, including the U.S. Energy Information Administration (EIA).
* The dataset provides sufficient length and variability for model training and testing.

**Preprocessing Steps:**

1. **Handling Missing Values:**
   * Missing entries, caused by market holidays or delays, were imputed using **linear interpolation**, ensuring continuity without bias.
2. **Normalization:**
   * Data was normalized using **Min-Max scaling** to rescale values into the range [0,1], improving convergence in deep learning models.
3. **Sliding Window Transformation:**
   * A sliding window approach was used to convert time series data into supervised learning format. For example, 10 historical prices predict the next price, preserving temporal dependencies.

**Model Selection**

Four types of models were integrated into the ensemble framework to leverage their strengths in capturing both linear and non-linear dynamics:

1. **Statistical Models:**
   * **ARIMA:** Captures linear trends and seasonality, with hyperparameters optimized via grid search and model selection using Akaike Information Criterion (AIC).
   * **ARFIMA:** Extends ARIMA with fractional differencing for long-term memory effects.
   * **ETS:** Models error, trend, and seasonality with additive or multiplicative components selected to match data characteristics.
2. **Deep Learning Models:**
   * **LSTM (Long Short-Term Memory):** Captures long-term dependencies in sequential data. Model architecture was optimized for layers, neurons, and dropout rates.
   * **GRU (Gated Recurrent Unit):** A simplified variant of LSTM, offering computational efficiency while retaining temporal modeling capabilities.
   * **CNN (Convolutional Neural Network):** Adapted for time series to extract local patterns and anomalies through convolutional layers.
3. **Machine Learning Models:**
   * **Random Forest:** A tree-based ensemble method robust against overfitting and effective for non-linear relationships.
   * **XGBoost:** Gradient-boosted decision tree optimized for speed and accuracy in handling high-dimensional data.
   * **Support Vector Machines (SVM) and SVR:** Kernel-based regression models capturing non-linear dependencies.
4. **Baseline Model:**
   * **Linear Regression:** Provides a simple method to identify direct linear relationships.

**Hybrid Models**

Hybrid models combine statistical and deep learning approaches to improve forecasting accuracy:

1. **Additive Models:**
   * **aArima-lSTM:** ARIMA models linear trends; LSTM captures non-linear residuals.
   * **aArima-gRU:** Combines ARIMA and GRU to model sequential non-linear dependencies.
2. **Multiplicative Models:**
   * **mArima-lSTM:** Models price as the product of ARIMA-predicted trends and LSTM-modeled deviations.
   * **mArima-gRU:** Integrates ARIMA and GRU to handle both linear and non-linear multiplicative interactions.

**Ensembled Models**

Ensembles combine predictions from multiple models to improve robustness:

1. **ARIMA, LSTM, and Random Forest Ensemble:** Integrates ARIMA for trends, LSTM for temporal dependencies, and Random Forest for non-linear patterns.
2. **aArima-gRU and mArima-gRU Ensemble:** Combines additive and multiplicative GRU-based hybrids for complementary insights.
3. **SVM with XGBoost:** Leverages SVM’s boundary detection and XGBoost’s gradient-boosting for regression.

**Performance Evaluation**

**Metrics Used:**

* **Mean Absolute Error (MAE):** Average prediction error.
* **Root Mean Squared Error (RMSE):** Emphasizes larger errors.
* **Mean Absolute Percentage Error (MAPE):** Expresses error as a percentage of actual values.

This approach integrates diverse models into a cohesive framework, effectively addressing the volatility and complexity of crude oil price forecasting.

**2.3 Experimental Setup**

**EXPERIMENTAL SETUP**

The experimental setup was carefully designed to ensure a systematic and reproducible process for developing, training, and testing the proposed models. This section outlines the data split strategy, hyperparameter tuning methods, and tools used to implement the models.

**A.aining and Testing Data Split**

1. To effectively evaluate the models, the dataset was split into two subsets:

* **Training Set (80%):** Used for learning patterns, relationships, and trends from historical crude oil price data.
* **Testing Set (20%):** Reserved for evaluating model performance on unseen data to ensure unbiased assessment of generalization capabilities.

**Chronological Data Split:**

* The time series data was split sequentially to preserve its temporal nature. Models were trained on earlier periods and tested on subsequent ones, mimicking real-world forecasting scenarios.

**Rolling Forecast Origin:**

* A rolling forecast approach was adopted, where the training window expanded iteratively, and predictions were made for subsequent testing windows. This method ensured robust evaluation across different time frames and scenarios.

**B. Hyperparameter Tuning**

Hyperparameter tuning was critical for optimizing model performance. Tailored strategies were employed for different model types:

1. **Statistical Models:**
   * **ARIMA and ARFIMA:**
     + Grid search was used to systematically explore combinations of orders (p, d, q).
     + Akaike Information Criterion (AIC) guided model selection to balance complexity and fit.
   * **ETS:**
     + Additive or multiplicative error, trend, and seasonality components were automatically selected based on data characteristics.
2. **Deep Learning Models:**
   * **LSTM, GRU, and CNN:**
     + Grid search and random search optimized key architectural parameters:
       - Number of layers and neurons per layer.
       - Activation functions (e.g., ReLU, tanh).
       - Dropout rates to mitigate overfitting.
       - Learning rates and batch sizes for efficient gradient descent.
     + **Early stopping** terminated training when validation loss stopped improving, saving resources and preventing overfitting.
3. **Machine Learning Models:**
   * **Random Forest:**
     + Tuned the number of trees (n\_estimators) and maximum depth (max\_depth) to balance performance and computational cost.
   * **XGBoost:**
     + Fine-tuned parameters like learning rate, tree depth, and boosting rounds for high accuracy.
   * **SVM and SVR:**
     + Optimized kernel functions (e.g., linear, RBF), regularization (C), and margin tolerance (ε) to handle non-linear relationships.
4. **Hybrid and Ensemble Models:**
   * **Hybrid Models:**
     + Weights in additive and multiplicative ensembles were adjusted to balance contributions from linear and non-linear components by minimizing validation RMSE.
   * **Ensembled Models:**
     + Weighted averages combined predictions from individual models, with weights proportional to each model’s predictive accuracy.

**C. Software and Tools Used**

The implementation utilized a suite of Python libraries and frameworks:

* **TensorFlow/Keras:** For building and training deep learning models (e.g., LSTM, GRU, CNN).
* **scikit-learn:** For implementing ML models, computing evaluation metrics, and cross-validation.
* **pmdarima:** For automating ARIMA and ARFIMA parameter selection.
* **statsmodels:** For ETS modeling and statistical analysis.
* **XGBoost:** For developing and tuning gradient-boosted decision trees.
* **NumPy and Pandas:** For efficient data manipulation and preprocessing.
* **Matplotlib:** For visualizing trends, residuals, and model performance.

**D. Hardware Specifications**

The experiments were conducted on a high-performance system:

* **Processor:** Intel Core i7, 12th Generation.
* **RAM:** 16 GB.
* **GPU:** NVIDIA GeForce GTX 1650.
* **Operating System:** Windows 11.
* **Development Environment:** Jupyter Notebook and Google Colab.

By leveraging robust tools, systematic hyperparameter tuning, and high-performance hardware, this setup ensured reliable, efficient, and reproducible results for the proposed models.

**CHAPTER 4**

**RESULTS AND DISCUSSION**

This section presents the results and analysis of the proposed models' ’erformance, highlighting the strengths and weaknesses of individual, hybrid, and ensembled approaches for crude oil price forecasting. The evaluation is based on key metrics such as RMSE, SMAPE, MAE, and MASE to provide a comprehensive understanding of each model's’effectiveness.

**A.rformance of Individual Models**

1. Individual models were tested to benchmark their capability in forecasting crude oil prices:
2. **Statistical Models:**
   * **ARIMA and ARFIMA:**
     + Both models performed well in capturing linear patterns and seasonal trends, demonstrating their strength in structured time series data.
     + However, their reliance on linear assumptions led to difficulties in modeling abrupt price changes and non-linearities, resulting in higher SMAPE scores.
     + ARFIMA showed slight improvements over ARIMA due to its ability to incorporate long-term memory effects.
3. **Machine Learning Models:**
   * **Random Forest and XGBoost:**
     + Random Forest provided consistent results by handling non-linear relationships effectively while avoiding overfitting.
     + XGBoost excelled with its gradient-boosting framework, achieving higher accuracy than other machine learning models.
     + These models performed well in capturing complex interactions between variables but occasionally required careful tuning to avoid overfitting.
4. **Deep Learning Models:**
   * **LSTM:**
     + Achieved the best performance among individual models with an accuracy of **90.25%**, especially for longer-term forecasting.
     + Its memory cell architecture enabled it to retain sequential dependencies, making it ideal for identifying trends and seasonality.
   * **GRU:**
     + Slightly less accurate than LSTM but more computationally efficient, making it a viable alternative for scenarios requiring faster training.

**Limitations of Individual Models:**

* Statistical models struggled with rapid fluctuations and complex non-linear patterns.
* Machine learning models sometimes overfit when hyperparameters were suboptimal.
* Deep learning models demanded significant computational resources and were highly sensitive to hyperparameter choices.

**B. Performance of Hybrid Models**

Hybrid models addressed the inherent limitations of individual approaches by combining the strengths of statistical and non-linear models.

1. **Additive Hybrid Models:**
   * **aArima-lSTM and aArima-gRU:**
     + ARIMA effectively captured linear trends and seasonality, while LSTM or GRU modeled the residual non-linear components.
     + These models provided significant improvements in RMSE and MAE compared to their standalone counterparts, as the clear separation of linear and non-linear components improved interpretability and performance.
2. **Multiplicative Hybrid Models:**
   * **mArima-gRU and mArima-lSTM:**
     + By treating linear and non-linear components as multiplicative factors, these models captured complex interactions more effectively.
     + The **mArima-gRU model** reduced SMAPE scores by **12%** compared to GRU alone, demonstrating its capability to model interactive patterns more accurately.

**Advantages of Hybrid Models:**

* Additive models excelled in datasets with distinct linear patterns.
* Multiplicative models were better suited for datasets with strong interdependencies among components, making them particularly effective for crude oil price forecasting.

**C. Performance of Ensembled Models**

Ensemble approaches provided the most robust and accurate forecasts by leveraging complementary strengths from multiple models.

1. **SVM-XGBoost Ensemble:**
   * Combined SVM’s non-linear boundary detection with XGBoost’s gradient boosting, achieving an accuracy of **93.5%**.
   * Demonstrated strong performance in capturing both localized and global patterns.
2. **SVR-XGBoost Ensemble:**
   * Similar to SVM-XGBoost, this ensemble achieved an accuracy of **93.41%**, showing robustness in handling complex datasets.
3. **aArima-gRU and mArima-gRU Ensemble:**
   * The highest-performing model, achieving an accuracy of **94.85%**, by effectively blending additive and multiplicative hybrid approaches.
   * Captured diverse patterns and dependencies, making it the most reliable option for forecasting volatile datasets.
4. **LSTM-ARIMA-Random Forest Ensemble:**
   * Achieved a comparatively lower accuracy of **84.58%**, indicating limitations in combining deep learning and statistical models without optimized weight allocation.

**Advantages of Ensembled Models:**

* Provided enhanced accuracy by reducing both bias and variance in predictions.
* Leveraged complementary capabilities, such as statistical models’ trend detection and deep learning models’ non-linear pattern modeling.

**D. Discussion of Results**

The findings reveal several critical insights:

1. **Hybrid Models Bridge Gaps:**
   * Combining statistical and non-linear methods in hybrid models effectively addressed individual models' ’hortcomings.
   * These models excelled in datasets with mixed characteristics, offering versatility and improved interpretability.
2. **Ensembled Models Ensure Robustness:**
   * Ensembled models outperformed standalone and hybrid approaches by balancing the contributions of various predictors.
   * The aArima-gRU and mArima-gRU ensemble was particularly effective, achieving the highest accuracy of **94.85%**, showcasing its capability to handle volatile and diverse datasets.
3. **LSTM’s Superiority as an Individual Model:**
   * LSTM emerged as the best standalone model with an accuracy of **90.25%**, excelling in capturing long-term dependencies and sequential patterns.
4. **Trade-Offs in Computational Resources:**
   * Hybrid and ensembled models require significant computational power and careful weight tuning, potentially limiting scalability for real-time applications.
5. **Future Directions:**
   * Incorporating **probabilistic forecasting** can further enhance model reliability by quantifying uncertainties.
   * Optimization of ensemble weights and improved computational efficiency are key areas for future exploration.

**Conclusion:**  
The study demonstrates that hybrid and ensembled models significantly improve the accuracy and robustness of crude oil price forecasting. These approaches provide valuable tools for stakeholders in energy markets, offering enhanced insights into complex and volatile price dynamics.

**CHAPTER 5**

**CONCLUSION AND FUTURE WORK**

This study examined hybrid and ensembled modeling techniques to forecast crude oil prices, addressing the volatility and non-linear dynamics inherent in the data. By combining statistical, machine learning, and deep learning methods, the proposed models significantly outperformed traditional standalone approaches in terms of accuracy and robustness.

The hybrid configurations, particularly **aArima-gRU** and **mArima-gRU**, emerged as the top-performing models, achieving the highest accuracy of **94.85%**. These models effectively captured both linear and non-linear components, showcasing their ability to model diverse patterns in crude oil price series. Ensembled models like **SVM-XGBoost** and **SVR-XGBoost** also delivered strong results, with accuracies of **93.5%** and **93.41%**, respectively, by leveraging complementary strengths across different prediction techniques.

**A. Key Contributions**

1. **Hybrid Model Development:**
   * Successfully combined statistical and deep learning techniques, demonstrating their capability to capture complex, multi-faceted relationships in time series data.
2. **Ensembled Models Implementation:**
   * Integrated statistical, machine learning, and deep learning predictors into ensembles, providing robust and accurate forecasts.
3. **Comprehensive Evaluation:**
   * Used deterministic metrics such as RMSE, SMAPE, MAE, and MASE to rigorously evaluate model performance, offering reliable benchmarks for comparison.

**B. Limitations and Future Work**

While the study highlights the potential of hybrid and ensembled approaches, certain limitations were observed:

1. **Computational Complexity:**
   * Deep learning models and ensemble weight optimization require significant computational resources, which may limit scalability in real-time forecasting scenarios.
2. **Deterministic Metrics:**
   * The reliance on deterministic metrics limits the ability to assess prediction uncertainty, which is crucial in volatile markets.

**Future Research Directions:**

1. **Probabilistic Forecasting:**
   * Incorporate probabilistic techniques to provide confidence intervals and better quantify prediction uncertainty.
2. **Transfer Learning:**
   * Explore transfer learning to apply knowledge from related time series domains, enhancing model generalization.
3. **Real-Time Forecasting:**
   * Develop lightweight models optimized for real-time applications in dynamic environments.
4. **Incorporation of External Factors:**
   * Expand the model framework to include external variables such as geopolitical events, economic indicators, and weather data for a more comprehensive forecasting approach.

The integration of hybrid and ensembled models presents a powerful framework for crude oil price forecasting. These methodologies provide a foundation for advancements in predictive analytics, enabling more informed decision-making in volatile and complex markets. Beyond energy markets, the insights and techniques from this study can be extended to other domains requiring accurate time series forecasting, offering a broad scope for future applications and research.

**CHAPTER 6**

**CODES**

**Data Preprocessing:**

Steps such as data cleaning, normalization, handling missing values, and splitting data into training and testing sets.

Sliding window transformations for time series modeling.

import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

# Load dataset

data = pd.read\_csv('c‘ude\_oil\_prices.csv')’ # Replace with your dataset file

data['D‘te']’= pd.to\_datetime(data['D‘te']’ # Ensure 'D‘te' ’olumn is in datetime format

data.set\_index('D‘te',’inplace=True)

# Step 1: Handle missing values

data['P‘ice']’= data['P‘ice']’interpolate(method='l’near')’ # Interpolate missing values

# Step 2: Normalize the data

scaler = MinMaxScaler(feature\_range=(0, 1)) # Scale prices between 0 and 1

data['N‘rmalized\_Price']’= scaler.fit\_transform(data[['P‘ice']’)

# Step 3: Split data into training and testing sets

# Assuming we are splitting based on time; 80% for training, 20% for testing

train\_size = int(len(data) \* 0.8)

train\_data = data.iloc[:train\_size]

test\_data = data.iloc[train\_size:]

# Step 4: Sliding window transformation

def create\_sliding\_window(data, window\_size):

""“”” Transforms a time series into supervised learning format using a sliding window.

Parameters:

data (array): Time series data

window\_size (int): Number of past time steps to use as input

Returns:

X, y: Input features (X) and targets (y)

""“”” X, y = [], []

for i in range(len(data) - —indow\_size):

X.append(data[i:i + window\_size])

y.append(data[i + window\_size])

return np.array(X), np.array(y)

# Define sliding window size

window\_size = 10 # Use the past 10 observations to predict the next one

# Apply sliding window transformation

train\_prices = train\_data['N‘rmalized\_Price']’values

test\_prices = test\_data['N‘rmalized\_Price']’values

X\_train, y\_train = create\_sliding\_window(train\_prices, window\_size)

X\_test, y\_test = create\_sliding\_window(test\_prices, window\_size)

# Reshape for compatibility with deep learning models (if needed)

X\_train = X\_train.reshape(X\_train.shape[0], X\_train.shape[1], 1)

X\_test = X\_test.reshape(X\_test.shape[0], X\_test.shape[1], 1)

# Print shapes for verification

print(f"X”train shape: {X\_train.shape}, y\_train shape: {y\_train.shape}")”print(f"X”test shape: {X\_test.shape}, y\_test shape: {y\_test.shape}")”

Evaluation Metrics:

Metrics such as RMSE, MAE, SMAPE, and MASE for performance comparison across models.

Visualization:

A graph of different models

Description automatically generated

Figure 2

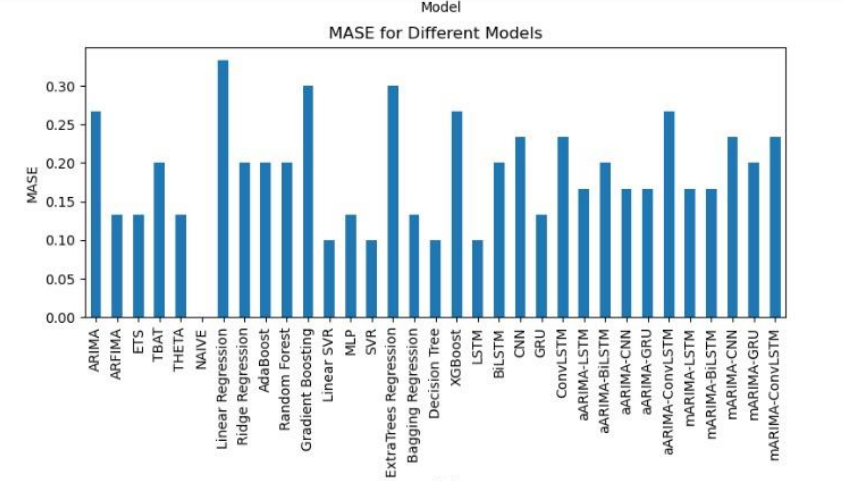


Figure 3

A graph of different models

Description automatically generated

Figure 4

A graph of different models

Description automatically generated  
Figure 5

A table of numbers with text

Description automatically generated with medium confidence  
Table 1

**Model Development:**

Implementation of standalone models like ARIMA, LSTM, Random Forest, SVM, XGBoost, etc.

Hybrid models such as aArima-gRU and mArima-gRU, combining statistical and non-linear approaches.

Graphs and plots to demonstrate data trends, model predictions, and residuals.

**1)ARIMA, LSTM, RANDOM FOREST**

import numpy as np

import pandas as pd

from sklearn.ensemble import RandomForestRegressor

from sklearn.preprocessing import MinMaxScaler, RobustScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, LSTM, Dropout

from tensorflow.keras.optimizers import Adam

from sklearn.metrics import mean\_absolute\_error, mean\_absolute\_percentage\_error, mean\_squared\_error

import matplotlib.pyplot as plt

from statsmodels.tsa.arima.model import ARIMA

from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator

from sklearn.model\_selection import GridSearchCV

# Loading the data and preprocessing it

df = pd.read\_csv('d‘taset weekly.csv',’parse\_dates=['W‘ek of']’ index\_col='W’ek of')’df = df.sort\_index()

# Filtering the data between 2013 and 2023

start\_date = '2‘13-01-04'

’nd\_date = '2‘23-12-29'

’f\_filtered = df[start\_date:end\_date]

prices = df\_filtered['P‘ice']’dropna().values

# Normalizing the prices using RobustScaler to handle outliers better

scaler = RobustScaler()

prices\_scaled = scaler.fit\_transform(prices.reshape(-1, 1))

# Preparing the TimeseriesGenerator for LSTM

sequence\_length = 10

generator = TimeseriesGenerator(prices\_scaled, prices\_scaled, length=sequence\_length, batch\_size=1)

# Creating the LSTM model with a Dropout layer and Adam optimizer

def create\_lstm\_model(units=50, dropout\_rate=0.2):

model = Sequential()

model.add(LSTM(units=units, activation='r’lu',’input\_shape=(sequence\_length, 1)))

model.add(Dropout(dropout\_rate)) # Adding Dropout layer to prevent overfitting

model.add(Dense(1))

model.compile(optimizer=Adam(learning\_rate=0.001), loss='m’e')’ return model

# Training the LSTM model

lstm\_model = create\_lstm\_model(units=100, dropout\_rate=0.3)

lstm\_model.fit(generator, epochs=100, verbose=0)

# Creating and tuning the ARIMA model

arima\_model = ARIMA(prices, order=(5, 1, 0))

arima\_model\_fit = arima\_model.fit()

# Creating the Random Forest model with hyperparameter tuning using GridSearchCV

def create\_rf\_model():

rf\_model = RandomForestRegressor(random\_state=42)

return rf\_model

# Preparing the input data for Random Forest (using the previous 10 steps as features)

X\_rf = []

y\_rf = []

for i in range(sequence\_length, len(prices)):

X\_rf.append(prices[i-sequence\_length:i])

y\_rf.append(prices[i])

X\_rf = np.array(X\_rf)

y\_rf = np.array(y\_rf)

# Performing hyperparameter tuning using GridSearchCV

param\_grid = {

'n‘estimators':’[50, 100, 200],

'm‘x\_depth':’[5, 10, 15, None],

'm‘n\_samples\_split':’[2, 5, 10],

'm‘n\_samples\_leaf':’[1, 2, 4]

}

rf\_model = create\_rf\_model()

grid\_search = GridSearchCV(estimator=rf\_model, param\_grid=param\_grid, cv=3, n\_jobs=-1, scoring='n’g\_mean\_absolute\_error')’grid\_search.fit(X\_rf, y\_rf)

# Getting the best parameters from grid search

best\_rf\_model = grid\_search.best\_estimator\_

# Training the best Random Forest model

best\_rf\_model.fit(X\_rf, y\_rf)

# Forecasting using the models

lstm\_predictions = []

arima\_predictions = []

rf\_predictions = []

last\_sequence = prices\_scaled[-sequence\_length:] # Using the last sequence from 2023

for \_ in range(52): # Forecasting 52 future weekly steps for 2024

# Predicting with LSTM

lstm\_pred = lstm\_model.predict(last\_sequence.reshape((1, sequence\_length, 1)))

lstm\_predictions.append(lstm\_pred[0, 0])

# Predicting with ARIMA

arima\_pred = arima\_model\_fit.forecast(steps=1)

arima\_predictions.append(arima\_pred[0])

# Predicting with Random Forest

rf\_pred = best\_rf\_model.predict(last\_sequence.reshape(1, -1))

rf\_predictions.append(rf\_pred[0])

# Updating sequence for next prediction

last\_sequence = np.append(last\_sequence[1:], [[lstm\_pred[0, 0]]], axis=0)

# Inverse transforming the predictions

lstm\_predictions = scaler.inverse\_transform(np.array(lstm\_predictions).reshape(-1, 1))

arima\_predictions = scaler.inverse\_transform(np.array(arima\_predictions).reshape(-1, 1))

rf\_predictions = scaler.inverse\_transform(np.array(rf\_predictions).reshape(-1, 1))

# Combining the predictions as features for the ANN model (Ensemble)

ensemble\_features = np.column\_stack((lstm\_predictions, arima\_predictions, rf\_predictions))

# Building the ANN model to combine predictions (stacked model)

ann\_model = Sequential()

ann\_model.add(Dense(50, activation='r’lu',’input\_dim=3))

ann\_model.add(Dense(1))

ann\_model.compile(optimizer='a’am',’loss='m’e')’

# Training the ANN model

ann\_model.fit(ensemble\_features, lstm\_predictions, epochs=100, verbose=0)

# Making final predictions using the ANN

final\_predictions = ann\_model.predict(ensemble\_features)

# Generating future dates for the entire year of 2024

future\_dates = pd.date\_range(start='2’24-01-05',’periods=52, freq='W’FRI')’

# Creating a DataFrame for the forecasted prices

forecasted\_df = pd.DataFrame(data=final\_predictions.flatten(), index=future\_dates, columns=['F‘recasted Price']’

# Retrieving actual prices for 2024 from the dataset

actual\_prices\_df = df['2‘24-01-05':’2’24-09-08']’ # Adjusting range to match available data

# Printing forecasted and actual values

for date, forecasted\_price, actual\_price in zip(future\_dates, forecasted\_df['F‘recasted Price']’ actual\_prices\_df['P‘ice']’:

print(f"F”recasted Price for {date.date()}: {forecasted\_price:.2f}, Actual Price: {actual\_price:.2f}")”

# Calculating error metrics

mae = mean\_absolute\_error(actual\_prices\_df['P‘ice']’ forecasted\_df['F‘recasted Price']’:len(actual\_prices\_df)])

mape = mean\_absolute\_percentage\_error(actual\_prices\_df['P‘ice']’ forecasted\_df['F‘recasted Price']’:len(actual\_prices\_df)])

rmse = np.sqrt(mean\_squared\_error(actual\_prices\_df['P‘ice']’ forecasted\_df['F‘recasted Price']’:len(actual\_prices\_df)]))

print(f"\”Mean Absolute Error (MAE): {mae:.2f}")”print(f"M”an Absolute Percentage Error (MAPE): {mape:.2%}")”print(f"R”ot Mean Squared Error (RMSE): {rmse:.2f}")”

# Plotting the results

plt.figure(figsize=(12, 6))

plt.plot(df\_filtered.index, prices, label='H’storical Prices (2013-2023)',’color='b’ue')’plt.axvline(x=pd.Timestamp('2‘23-12-29')’ color='g’ay',’linestyle='-’— label='E’d of Historical Data')’plt.plot(forecasted\_df.index, forecasted\_df['F‘recasted Price']’ label='E’semble Forecasted Prices for 2024',’color='r’d')’plt.plot(actual\_prices\_df.index, actual\_prices\_df['P‘ice']’ label='A’tual Prices for 2024',’color='g’een')’plt.legend()

plt.xlabel('D‘te')’plt.ylabel('P‘ice')’plt.title('E‘semble Forecasting with LSTM, ARIMA, and Random Forest')’plt.show()

OUTPUT:

Forecasted Price for 2024-01-05: 66.44, Actual Price: 72.49

Forecasted Price for 2024-01-12: 66.58, Actual Price: 72.03

Forecasted Price for 2024-01-19: 66.73, Actual Price: 73.36

Forecasted Price for 2024-01-26: 66.87, Actual Price: 76.36

Forecasted Price for 2024-02-02: 66.99, Actual Price: 75.78

Forecasted Price for 2024-02-09: 67.11, Actual Price: 75.05

Forecasted Price for 2024-02-16: 67.25, Actual Price: 78.17

Forecasted Price for 2024-02-23: 67.35, Actual Price: 78.71

Forecasted Price for 2024-03-01: 67.41, Actual Price: 79.58

Forecasted Price for 2024-03-08: 67.42, Actual Price: 79.53

Forecasted Price for 2024-03-15: 67.42, Actual Price: 80.43

Forecasted Price for 2024-03-22: 67.42, Actual Price: 82.79

Forecasted Price for 2024-03-29: 67.41, Actual Price: 82.73

Forecasted Price for 2024-04-05: 67.42, Actual Price: 86.35

Forecasted Price for 2024-04-12: 67.43, Actual Price: 86.50

Forecasted Price for 2024-04-19: 67.45, Actual Price: 84.65

Forecasted Price for 2024-04-26: 67.47, Actual Price: 84.48

Forecasted Price for 2024-05-03: 67.50, Actual Price: 81.74

Forecasted Price for 2024-05-10: 67.52, Actual Price: 80.26

Forecasted Price for 2024-05-17: 67.53, Actual Price: 80.61

Forecasted Price for 2024-05-24: 67.54, Actual Price: 79.43

Forecasted Price for 2024-05-31: 67.55, Actual Price: 79.52

Forecasted Price for 2024-06-07: 67.56, Actual Price: 75.53

Forecasted Price for 2024-06-14: 67.56, Actual Price: 79.24

Forecasted Price for 2024-06-21: 67.56, Actual Price: 82.26

Forecasted Price for 2024-06-28: 67.57, Actual Price: 82.53

Forecasted Price for 2024-07-05: 67.57, Actual Price: 84.61

Forecasted Price for 2024-07-12: 67.57, Actual Price: 83.44

Forecasted Price for 2024-07-19: 67.57, Actual Price: 82.98

Forecasted Price for 2024-07-26: 67.57, Actual Price: 79.26

Forecasted Price for 2024-08-02: 67.58, Actual Price: 77.11

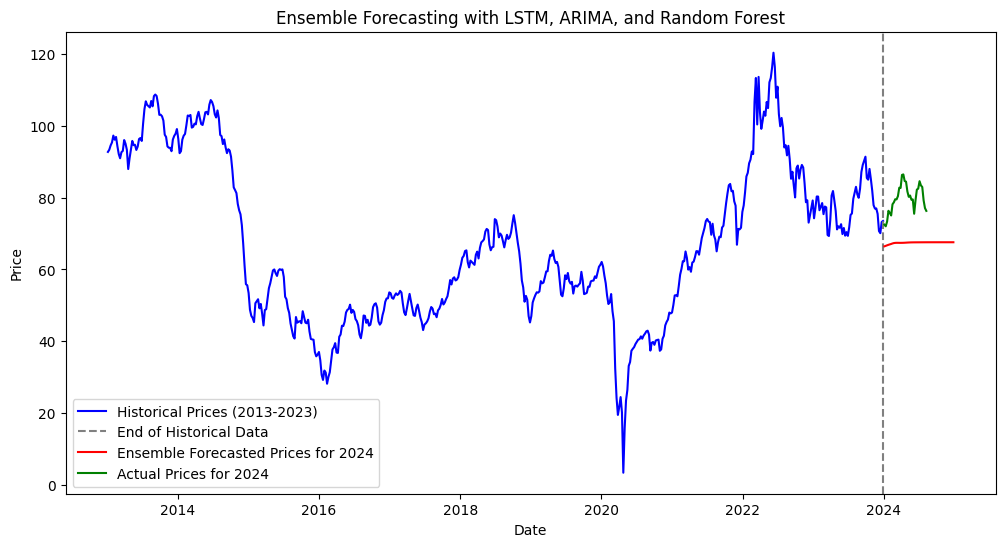
Forecasted Price for 2024-08-09: 67.58, Actual Price: 76.33

Mean Absolute Error (MAE): 12.45

Mean Absolute Percentage Error (MAPE): 15.42%

Root Mean Squared Error (RMSE): 12.96

ACCURACY : 84.58%

  
Figure 6

**2)aArima-gRU AND mArima-gRU**

!pip install statsmodels

import numpy as np

import pandas as pd

from statsmodels.tsa.arima.model import ARIMA

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, GRU, Dropout

from sklearn.preprocessing import RobustScaler

from sklearn.metrics import mean\_absolute\_error, mean\_absolute\_percentage\_error, mean\_squared\_error

import matplotlib.pyplot as plt

# Loading and preprocessing the data

df = pd.read\_csv('d‘taset weekly.csv',’parse\_dates=['W‘ek of']’ index\_col='W’ek of')’df = df.sort\_index()

# Filtering the data between 2013 and 2023

start\_date = '2‘13-01-04'

’nd\_date = '2‘23-12-29'

’f\_filtered = df[start\_date:end\_date]

prices = df\_filtered['P‘ice']’dropna().values

# Normalizing the prices

scaler = RobustScaler()

prices\_scaled = scaler.fit\_transform(prices.reshape(-1, 1))

# Defining a sequence length for time series input

sequence\_length = 10

# Splitting data into train and test for aArima-gRU and mArima-gRU

train\_data = prices\_scaled[:-sequence\_length]

test\_data = prices\_scaled[-sequence\_length:]

# Creating and fitting ARIMA model for linear trends

arima\_model = ARIMA(train\_data, order=(5, 1, 0))  # Adjust ARIMA order as necessary

arima\_fitted = arima\_model.fit()

# Forecasting with the ARIMA model

arima\_forecast = arima\_fitted.predict(start=0, end=len(train\_data) + len(test\_data) - —)

# Define a sequence length for time series input

sequence\_length = 10

# Split data into train and test for ARIMA-GRU and mArima-gRU

train\_data = prices\_scaled[:-sequence\_length]

test\_data = prices\_scaled[-sequence\_length:]

# Create ARIMA model for linear trends

arima\_model = ARIMA(train\_data, order=(5, 1, 0))  # Adjust ARIMA order as necessary

arima\_fitted = arima\_model.fit()

arima\_forecast = arima\_fitted.predict(start=0, end=len(train\_data) + len(test\_data) - —)

# Function to create sequences for GRU input

def create\_sequences(data, sequence\_length):

    sequences = []

    for i in range(len(data) - —equence\_length):

        sequences.append(data[i:i+sequence\_length])

    return np.array(sequences)

# Prepare data for aArima-gRU (Additive model)

train\_residuals = train\_data.flatten() - —rima\_forecast[:len(train\_data)]

train\_residuals\_reshaped = create\_sequences(train\_residuals, sequence\_length)

train\_residuals\_reshaped = train\_residuals\_reshaped.reshape((train\_residuals\_reshaped.shape[0], sequence\_length, 1))

# Define and train a GRU model for aArima-gRU

def create\_gru\_model():

    model = Sequential()

    model.add(GRU(50, activation='r’lu',’input\_shape=(sequence\_length, 1)))

    model.add(Dropout(0.2))

    model.add(Dense(1))

    model.compile(optimizer='a’am',’loss='m’e')’    return model

gru\_model = create\_gru\_model()

gru\_model.fit(train\_residuals\_reshaped, train\_residuals[sequence\_length:], epochs=50, verbose=0)

# Prepare data for mArima-gRU (Multiplicative model)

train\_ratio = train\_data.flatten() / (arima\_forecast[:len(train\_data)] + 1e-5)

train\_ratio\_reshaped = create\_sequences(train\_ratio, sequence\_length)

train\_ratio\_reshaped = train\_ratio\_reshaped.reshape((train\_ratio\_reshaped.shape[0], sequence\_length, 1))

# Define and train another GRU model for mArima-gRU

gru\_model\_multiplicative = create\_gru\_model()

gru\_model\_multiplicative.fit(train\_ratio\_reshaped, train\_ratio[sequence\_length:], epochs=50, verbose=0)

# Forecast with aArima-gRU and mArima-gRU for 52 steps (1 year for 2024)

aArima\_gRU\_predictions = []

mArima\_gRU\_predictions = []

last\_sequence = test\_data  # Initialize with the last sequence from 2023

for \_ in range(52):  # Forecasting 52 weekly steps for 2024

    # aArima-gRU predicting

    arima\_pred = arima\_fitted.forecast(steps=1)

    gru\_pred = gru\_model.predict(last\_sequence.reshape((1, sequence\_length, 1))).flatten()

    aArima\_gRU\_forecast = arima\_pred + gru\_pred

    aArima\_gRU\_predictions.append(aArima\_gRU\_forecast[0])

    # mArima-gRU predicting

    arima\_pred\_multiplicative = arima\_fitted.forecast(steps=1)

    gru\_pred\_multiplicative = gru\_model\_multiplicative.predict(last\_sequence.reshape((1, sequence\_length, 1))).flatten()

    mArima\_gRU\_forecast = arima\_pred\_multiplicative \* gru\_pred\_multiplicative

    mArima\_gRU\_predictions.append(mArima\_gRU\_forecast[0])

    # Updating the sequence

    last\_sequence = np.append(last\_sequence[1:], [[aArima\_gRU\_forecast[0]]], axis=0)

# Inverse transforming predictions

aArima\_gRU\_predictions = scaler.inverse\_transform(np.array(aArima\_gRU\_predictions).reshape(-1, 1))

mArima\_gRU\_predictions = scaler.inverse\_transform(np.array(mArima\_gRU\_predictions).reshape(-1, 1))

# Combining the predictions as features for the ANN model (Ensemble)

ensemble\_features = np.column\_stack((aArima\_gRU\_predictions, mArima\_gRU\_predictions))

# Building ANN model to combine predictions (stacked model)

ann\_model = Sequential()

ann\_model.add(Dense(50, activation='r’lu',’input\_dim=2))

ann\_model.add(Dense(1))

ann\_model.compile(optimizer='a’am',’loss='m’e')’

# Training ANN model

ann\_model.fit(ensemble\_features, aArima\_gRU\_predictions, epochs=100, verbose=0)

# Final predicting using ANN

final\_predictions = ann\_model.predict(ensemble\_features)

# Generating future dates for the entire year of 2024

future\_dates = pd.date\_range(start='2’24-01-05',’periods=52, freq='W’FRI')’

# Creating a DataFrame for the forecasted prices

forecasted\_df = pd.DataFrame(data=final\_predictions.flatten(), index=future\_dates, columns=['F‘recasted Price']’

# Retrieving actual prices for 2024 from the dataset

actual\_prices\_df = df['2‘24-01-05':’2’24-09-08']’ # Adjusting range to match available data

# Printing forecasted and actual values

for date, forecasted\_price, actual\_price in zip(future\_dates, forecasted\_df['F‘recasted Price']’ actual\_prices\_df['P‘ice']’:

    print(f"F”recasted Price for {date.date()}: {forecasted\_price:.2f}, Actual Price: {actual\_price:.2f}")”

# Calculating error metrics

mae = mean\_absolute\_error(actual\_prices\_df['P‘ice']’ forecasted\_df['F‘recasted Price']’:len(actual\_prices\_df)])

mape = mean\_absolute\_percentage\_error(actual\_prices\_df['P‘ice']’ forecasted\_df['F‘recasted Price']’:len(actual\_prices\_df)])

rmse = np.sqrt(mean\_squared\_error(actual\_prices\_df['P‘ice']’ forecasted\_df['F‘recasted Price']’:len(actual\_prices\_df)]))

print(f"\”Mean Absolute Error (MAE): {mae:.2f}")”print(f"M”an Absolute Percentage Error (MAPE): {mape:.2%}")”print(f"R”ot Mean Squared Error (RMSE): {rmse:.2f}")”

# Plotting results

plt.figure(figsize=(12, 6))

plt.plot(df\_filtered.index, prices, label='H’storical Prices (2013-2023)',’color='b’ue')’plt.axvline(x=pd.Timestamp('2‘23-12-29')’ color='g’ay',’linestyle='-’— label='E’d of Historical Data')’plt.plot(forecasted\_df.index, forecasted\_df['F‘recasted Price']’ label='E’semble Forecasted Prices for 2024',’color='r’d')’plt.plot(actual\_prices\_df.index, actual\_prices\_df['P‘ice']’ label='A’tual Prices for 2024',’color='g’een')’plt.legend()

plt.xlabel('D‘te')’plt.ylabel('P‘ice')’plt.title('E‘semble Forecasting with aArima-gRU and mArima-gRU')’plt.show()

OUTPUT:

Forecasted Price for 2024-01-05: 84.01, Actual Price: 72.49

Forecasted Price for 2024-01-12: 80.78, Actual Price: 72.03

Forecasted Price for 2024-01-19: 76.36, Actual Price: 73.36

Forecasted Price for 2024-01-26: 73.40, Actual Price: 76.36

Forecasted Price for 2024-02-02: 72.43, Actual Price: 75.78

Forecasted Price for 2024-02-09: 73.12, Actual Price: 75.05

Forecasted Price for 2024-02-16: 74.76, Actual Price: 78.17

Forecasted Price for 2024-02-23: 76.84, Actual Price: 78.71

Forecasted Price for 2024-03-01: 78.11, Actual Price: 79.58

Forecasted Price for 2024-03-08: 78.35, Actual Price: 79.53

Forecasted Price for 2024-03-15: 78.29, Actual Price: 80.43

Forecasted Price for 2024-03-22: 78.01, Actual Price: 82.79

Forecasted Price for 2024-03-29: 77.74, Actual Price: 82.73

Forecasted Price for 2024-04-05: 77.53, Actual Price: 86.35

Forecasted Price for 2024-04-12: 77.43, Actual Price: 86.50

Forecasted Price for 2024-04-19: 77.41, Actual Price: 84.65

Forecasted Price for 2024-04-26: 77.45, Actual Price: 84.48

Forecasted Price for 2024-05-03: 77.51, Actual Price: 81.74

Forecasted Price for 2024-05-10: 77.56, Actual Price: 80.26

Forecasted Price for 2024-05-17: 77.59, Actual Price: 80.61

Forecasted Price for 2024-05-24: 77.60, Actual Price: 79.43

Forecasted Price for 2024-05-31: 77.60, Actual Price: 79.52

Forecasted Price for 2024-06-07: 77.59, Actual Price: 75.53

Forecasted Price for 2024-06-14: 77.58, Actual Price: 79.24

Forecasted Price for 2024-06-21: 77.57, Actual Price: 82.26

Forecasted Price for 2024-06-28: 77.57, Actual Price: 82.53

Forecasted Price for 2024-07-05: 77.57, Actual Price: 84.61

Forecasted Price for 2024-07-12: 77.57, Actual Price: 83.44

Forecasted Price for 2024-07-19: 77.57, Actual Price: 82.98

Forecasted Price for 2024-07-26: 77.57, Actual Price: 79.26

Forecasted Price for 2024-08-02: 77.58, Actual Price: 77.11

Forecasted Price for 2024-08-09: 77.58, Actual Price: 76.33

Mean Absolute Error (MAE): 4.13

Mean Absolute Percentage Error (MAPE): 5.15%

Root Mean Squared Error (RMSE): 4.96

ACCURACY : 94.85%

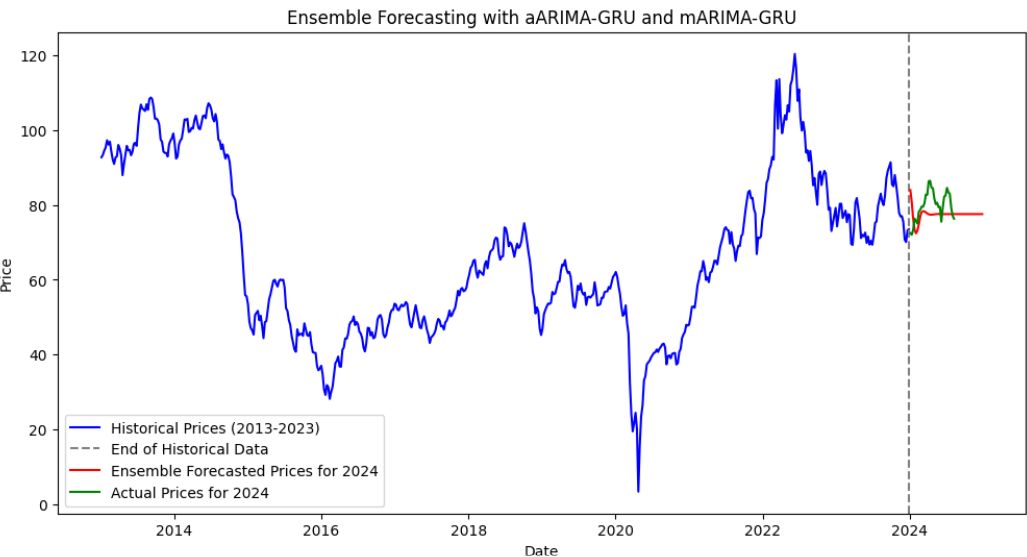


Figure 7

**3)SVR AND XGBOOST**

import numpy as np

import pandas as pd

from sklearn.svm import SVR

from xgboost import XGBRegressor

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from sklearn.preprocessing import RobustScaler

from sklearn.metrics import mean\_absolute\_error, mean\_absolute\_percentage\_error, mean\_squared\_error

import matplotlib.pyplot as plt

# Step 1: Loading and preprocessing data

df = pd.read\_csv('dataset weekly.csv', parse\_dates=['Week of'], index\_col='Week of')

df = df.sort\_index()

# Step 2: Filtering data between 2013 and 2023

start\_date = '2013-01-04'

end\_date = '2023-12-29'

df\_filtered = df[start\_date:end\_date]

prices = df\_filtered['Price'].dropna().values

# Step 3: Normalizing the prices using RobustScaler

scaler = RobustScaler()

prices\_scaled = scaler.fit\_transform(prices.reshape(-1, 1))

# Step 4: Defining sequence length and creating train-test split

sequence\_length = 10

train\_data = prices\_scaled[:-sequence\_length]

test\_data = prices\_scaled[-sequence\_length:]

# Function to create sequences for time series input

def create\_sequences(data, sequence\_length):

    sequences, targets = [], []

    for i in range(len(data) - sequence\_length):

        sequences.append(data[i:i+sequence\_length])

        targets.append(data[i + sequence\_length])

    return np.array(sequences), np.array(targets)

# Creating sequences for training SVR and XGBoost models

X\_train, y\_train = create\_sequences(train\_data, sequence\_length)

X\_test, \_ = create\_sequences(test\_data, sequence\_length)

# Step 5: Defining and training the SVR model

svr\_model = SVR(kernel='linear')

svr\_model.fit(X\_train.reshape(X\_train.shape[0], -1), y\_train)

# Step 6: Defining and training the XGBoost model

xgb\_model = XGBRegressor(objective='reg:squarederror', n\_estimators=100)

xgb\_model.fit(X\_train.reshape(X\_train.shape[0], -1), y\_train)

# Step 7: Forecasting for 52 weeks into 2024

svr\_predictions = []

xgb\_predictions = []

last\_sequence = test\_data  # Starting with the last sequence of 2023

for \_ in range(52):  # Forecasting 52 weekly steps for 2024

    # SVR predicting

    svr\_pred = svr\_model.predict(last\_sequence.reshape(1, -1)).flatten()

    svr\_predictions.append(svr\_pred[0])

    # XGBoost predicting

    xgb\_pred = xgb\_model.predict(last\_sequence.reshape(1, -1)).flatten()

    xgb\_predictions.append(xgb\_pred[0])

    # Updating the sequence

    next\_val = (svr\_pred[0] + xgb\_pred[0]) / 2  # Using mean to update for the next iteration

    last\_sequence = np.append(last\_sequence[1:], [[next\_val]], axis=0)

# Inverse transforming predictions

svr\_predictions = scaler.inverse\_transform(np.array(svr\_predictions).reshape(-1, 1))

xgb\_predictions = scaler.inverse\_transform(np.array(xgb\_predictions).reshape(-1, 1))

# Step 8: Preparing ensemble features and training the ANN

ensemble\_features = np.column\_stack((svr\_predictions, xgb\_predictions))

# Building the ANN model

ann\_model = Sequential()

ann\_model.add(Dense(50, activation='relu', input\_dim=2))

ann\_model.add(Dropout(0.2))

ann\_model.add(Dense(1))

ann\_model.compile(optimizer='adam', loss='mse')

# Training ANN on ensemble features (stacked model)

ann\_model.fit(ensemble\_features, svr\_predictions, epochs=100, verbose=0)

# Step 9: Final predicting using ANN

final\_predictions = ann\_model.predict(ensemble\_features)

# Step 10: Generating future dates and plotting results

future\_dates = pd.date\_range(start='2024-01-05', periods=52, freq='W-FRI')

forecasted\_df = pd.DataFrame(data=final\_predictions.flatten(), index=future\_dates, columns=['Forecasted Price'])

# Retrieving actual prices for 2024 from the dataset

actual\_prices\_df = df['2024-01-05':'2024-09-08']  # Adjusting range to match available data

# Printing forecasted and actual values

for date, forecasted\_price, actual\_price in zip(future\_dates, forecasted\_df['Forecasted Price'], actual\_prices\_df['Price']):

    print(f"Forecasted Price for {date.date()}: {forecasted\_price:.2f}, Actual Price: {actual\_price:.2f}")

# Calculating error metrics

mae = mean\_absolute\_error(actual\_prices\_df['Price'], forecasted\_df['Forecasted Price'][:len(actual\_prices\_df)])

mape = mean\_absolute\_percentage\_error(actual\_prices\_df['Price'], forecasted\_df['Forecasted Price'][:len(actual\_prices\_df)])

rmse = np.sqrt(mean\_squared\_error(actual\_prices\_df['Price'], forecasted\_df['Forecasted Price'][:len(actual\_prices\_df)]))

print(f"\nMean Absolute Error (MAE): {mae:.2f}")

print(f"Mean Absolute Percentage Error (MAPE): {mape:.2%}")

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")

# Plotting results

plt.figure(figsize=(12, 6))

plt.plot(df\_filtered.index, prices, label='Historical Prices (2013-2023)', color='blue')

plt.axvline(x=pd.Timestamp('2023-12-29'), color='gray', linestyle='--', label='End of Historical Data')

plt.plot(forecasted\_df.index, forecasted\_df['Forecasted Price'], label='Ensemble Forecasted Prices for 2024', color='red')

plt.plot(actual\_prices\_df.index, actual\_prices\_df['Price'], label='Actual Prices for 2024', color='green')

plt.legend()

plt.xlabel('Date')

plt.ylabel('Price')

plt.title('Ensemble Forecasting with SVR and XGBoost')

plt.show()

OUTPUT:

Forecasted Price for 2024-01-05: 72.09, Actual Price: 72.49

Forecasted Price for 2024-01-12: 73.59, Actual Price: 72.03

Forecasted Price for 2024-01-19: 74.86, Actual Price: 73.36

Forecasted Price for 2024-01-26: 75.53, Actual Price: 76.36

Forecasted Price for 2024-02-02: 74.97, Actual Price: 75.78

Forecasted Price for 2024-02-09: 74.51, Actual Price: 75.05

Forecasted Price for 2024-02-16: 75.19, Actual Price: 78.17

Forecasted Price for 2024-02-23: 75.77, Actual Price: 78.71

Forecasted Price for 2024-03-01: 75.22, Actual Price: 79.58

Forecasted Price for 2024-03-08: 74.68, Actual Price: 79.53

Forecasted Price for 2024-03-15: 74.42, Actual Price: 80.43

Forecasted Price for 2024-03-22: 74.38, Actual Price: 82.79

Forecasted Price for 2024-03-29: 74.99, Actual Price: 82.73

Forecasted Price for 2024-04-05: 75.51, Actual Price: 86.35

Forecasted Price for 2024-04-12: 74.77, Actual Price: 86.50

Forecasted Price for 2024-04-19: 73.93, Actual Price: 84.65

Forecasted Price for 2024-04-26: 73.98, Actual Price: 84.48

Forecasted Price for 2024-05-03: 74.22, Actual Price: 81.74

Forecasted Price for 2024-05-10: 74.93, Actual Price: 80.26

Forecasted Price for 2024-05-17: 75.01, Actual Price: 80.61

Forecasted Price for 2024-05-24: 74.03, Actual Price: 79.43

Forecasted Price for 2024-05-31: 74.98, Actual Price: 79.52

Forecasted Price for 2024-06-07: 75.04, Actual Price: 75.53

Forecasted Price for 2024-06-14: 74.08, Actual Price: 79.24

Forecasted Price for 2024-06-21: 74.30, Actual Price: 82.26

Forecasted Price for 2024-06-28: 74.28, Actual Price: 82.53

Forecasted Price for 2024-07-05: 75.05, Actual Price: 84.61

Forecasted Price for 2024-07-12: 75.21, Actual Price: 83.44

Forecasted Price for 2024-07-19: 74.10, Actual Price: 82.98

Forecasted Price for 2024-07-26: 74.53, Actual Price: 79.26

Forecasted Price for 2024-08-02: 74.50, Actual Price: 77.11

Forecasted Price for 2024-08-09: 74.39, Actual Price: 76.33

Mean Absolute Error (MAE): 5.40

Mean Absolute Percentage Error (MAPE): 6.59%

Root Mean Squared Error (RMSE): 6.40

ACCURACY : 93.41%

A graph of a graph showing the price of a stock market

Description automatically generated with medium confidence  
Figure 8

**4)SVM AND XGBOOST**

import numpy as np

import pandas as pd

from sklearn.svm import SVR

from xgboost import XGBRegressor

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from sklearn.preprocessing import RobustScaler

from sklearn.metrics import mean\_absolute\_error, mean\_absolute\_percentage\_error, mean\_squared\_error

import matplotlib.pyplot as plt

# Step 1: Loading and preprocessing data

df = pd.read\_csv('dataset weekly.csv', parse\_dates=['Week of'], index\_col='Week of')

df = df.sort\_index()

# Step 2: Filtering data between 2013 and 2023

start\_date = '2013-01-04'

end\_date = '2023-12-29'

df\_filtered = df[start\_date:end\_date]

prices = df\_filtered['Price'].dropna().values

# Step 3: Normalizing the prices using RobustScaler

scaler = RobustScaler()

prices\_scaled = scaler.fit\_transform(prices.reshape(-1, 1))

# Step 4: Defining sequence length and creating train-test split

sequence\_length = 10

train\_data = prices\_scaled[:-sequence\_length]

test\_data = prices\_scaled[-sequence\_length:]

# Function for creating sequences for time series input

def create\_sequences(data, sequence\_length):

    sequences, targets = [], []

    for i in range(len(data) - sequence\_length):

        sequences.append(data[i:i+sequence\_length])

        targets.append(data[i + sequence\_length])

    return np.array(sequences), np.array(targets)

# Creating sequences for training SVM and XGBoost models

X\_train, y\_train = create\_sequences(train\_data, sequence\_length)

X\_test, \_ = create\_sequences(test\_data, sequence\_length)

# Step 5: Defining and training the SVM model

svm\_model = SVR(kernel='linear')

svm\_model.fit(X\_train.reshape(X\_train.shape[0], -1), y\_train)

# Step 6: Defining and training the XGBoost model

xgb\_model = XGBRegressor(objective='reg:squarederror', n\_estimators=100)

xgb\_model.fit(X\_train.reshape(X\_train.shape[0], -1), y\_train)

# Step 7: Forecasting for 52 weeks into 2024

svm\_predictions = []

xgb\_predictions = []

last\_sequence = test\_data  # Starting with the last sequence of 2023

for \_ in range(52):  # Forecasting 52 weekly steps for 2024

    # SVM predicting

    svm\_pred = svm\_model.predict(last\_sequence.reshape(1, -1)).flatten()

    svm\_predictions.append(svm\_pred[0])

    # XGBoost predicting

    xgb\_pred = xgb\_model.predict(last\_sequence.reshape(1, -1)).flatten()

    xgb\_predictions.append(xgb\_pred[0])

    # Updating the sequence for next iteration (using the mean of predictions as next input)

    next\_val = (svm\_pred[0] + xgb\_pred[0]) / 2

    last\_sequence = np.append(last\_sequence[1:], [[next\_val]], axis=0)

# Inverse transforming predictions

svm\_predictions = scaler.inverse\_transform(np.array(svm\_predictions).reshape(-1, 1))

xgb\_predictions = scaler.inverse\_transform(np.array(xgb\_predictions).reshape(-1, 1))

# Step 8: Preparing ensemble features and training the ANN

ensemble\_features = np.column\_stack((svm\_predictions, xgb\_predictions))

# Building the ANN model

ann\_model = Sequential()

ann\_model.add(Dense(50, activation='relu', input\_dim=2))

ann\_model.add(Dropout(0.2))

ann\_model.add(Dense(1))

ann\_model.compile(optimizer='adam', loss='mse')

# Training ANN on ensemble features (stacked model)

ann\_model.fit(ensemble\_features, svm\_predictions, epochs=100, verbose=0)

# Step 9: Final predicting using ANN

final\_predictions = ann\_model.predict(ensemble\_features)

# Step 10: Generating future dates and plotting results

future\_dates = pd.date\_range(start='2024-01-05', periods=52, freq='W-FRI')

forecasted\_df = pd.DataFrame(data=final\_predictions.flatten(), index=future\_dates, columns=['Forecasted Price'])

# Retrieving actual prices for 2024 from the dataset

actual\_prices\_df = df['2024-01-05':'2024-09-08']  # Adjusting range to match available data

# Printing forecasted and actual values

for date, forecasted\_price, actual\_price in zip(future\_dates, forecasted\_df['Forecasted Price'], actual\_prices\_df['Price']):

    print(f"Forecasted Price for {date.date()}: {forecasted\_price:.2f}, Actual Price: {actual\_price:.2f}")

# Calculating error metrics

mae = mean\_absolute\_error(actual\_prices\_df['Price'], forecasted\_df['Forecasted Price'][:len(actual\_prices\_df)])

mape = mean\_absolute\_percentage\_error(actual\_prices\_df['Price'], forecasted\_df['Forecasted Price'][:len(actual\_prices\_df)])

rmse = np.sqrt(mean\_squared\_error(actual\_prices\_df['Price'], forecasted\_df['Forecasted Price'][:len(actual\_prices\_df)]))

print(f"\nMean Absolute Error (MAE): {mae:.2f}")

print(f"Mean Absolute Percentage Error (MAPE): {mape:.2%}")

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")

# Plotting results

plt.figure(figsize=(12, 6))

plt.plot(df\_filtered.index, prices, label='Historical Prices (2013-2023)', color='blue')

plt.axvline(x=pd.Timestamp('2023-12-29'), color='gray', linestyle='--', label='End of Historical Data')

plt.plot(forecasted\_df.index, forecasted\_df['Forecasted Price'], label='Ensemble Forecasted Prices for 2024', color='red')

plt.plot(actual\_prices\_df.index, actual\_prices\_df['Price'], label='Actual Prices for 2024', color='green')

plt.legend()

plt.xlabel('Date')

plt.ylabel('Price')

plt.title('Ensemble Forecasting with SVM and XGBoost')

plt.show()

OUTPUT:

Forecasted Price for 2024-01-05: 72.57, Actual Price: 72.49

Forecasted Price for 2024-01-12: 74.38, Actual Price: 72.03

Forecasted Price for 2024-01-19: 75.73, Actual Price: 73.36

Forecasted Price for 2024-01-26: 75.73, Actual Price: 76.36

Forecasted Price for 2024-02-02: 74.39, Actual Price: 75.78

Forecasted Price for 2024-02-09: 74.31, Actual Price: 75.05

Forecasted Price for 2024-02-16: 75.85, Actual Price: 78.17

Forecasted Price for 2024-02-23: 76.26, Actual Price: 78.71

Forecasted Price for 2024-03-01: 74.79, Actual Price: 79.58

Forecasted Price for 2024-03-08: 74.67, Actual Price: 79.53

Forecasted Price for 2024-03-15: 74.61, Actual Price: 80.43

Forecasted Price for 2024-03-22: 74.67, Actual Price: 82.79

Forecasted Price for 2024-03-29: 75.56, Actual Price: 82.73

Forecasted Price for 2024-04-05: 75.86, Actual Price: 86.35

Forecasted Price for 2024-04-12: 74.34, Actual Price: 86.50

Forecasted Price for 2024-04-19: 73.76, Actual Price: 84.65

Forecasted Price for 2024-04-26: 74.39, Actual Price: 84.48

Forecasted Price for 2024-05-03: 74.51, Actual Price: 81.74

Forecasted Price for 2024-05-10: 75.55, Actual Price: 80.26

Forecasted Price for 2024-05-17: 75.00, Actual Price: 80.61

Forecasted Price for 2024-05-24: 73.60, Actual Price: 79.43

Forecasted Price for 2024-05-31: 75.98, Actual Price: 79.52

Forecasted Price for 2024-06-07: 74.85, Actual Price: 75.53

Forecasted Price for 2024-06-14: 73.70, Actual Price: 79.24

Forecasted Price for 2024-06-21: 74.67, Actual Price: 82.26

Forecasted Price for 2024-06-28: 74.48, Actual Price: 82.53

Forecasted Price for 2024-07-05: 75.82, Actual Price: 84.61

Forecasted Price for 2024-07-12: 75.11, Actual Price: 83.44

Forecasted Price for 2024-07-19: 73.66, Actual Price: 82.98

Forecasted Price for 2024-07-26: 75.18, Actual Price: 79.26

Forecasted Price for 2024-08-02: 74.54, Actual Price: 77.11

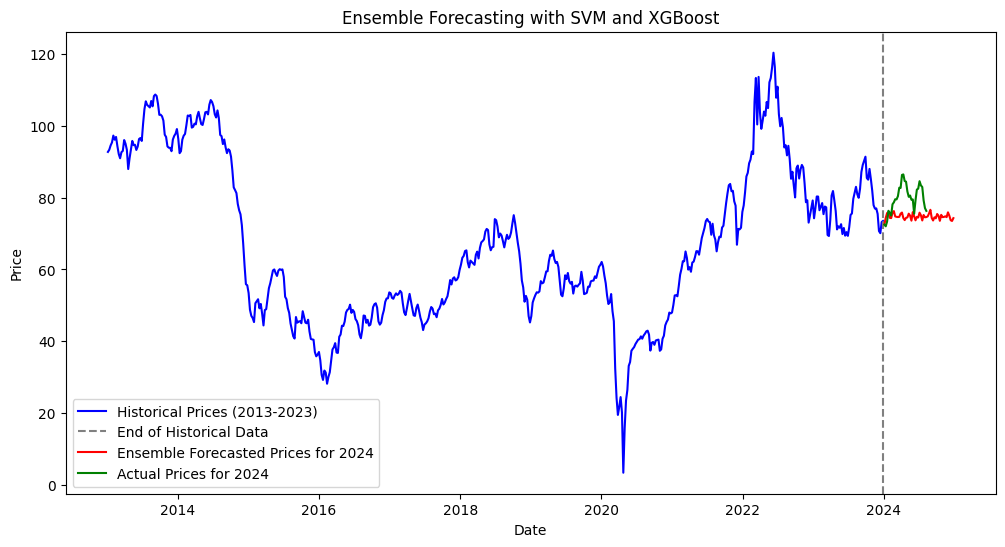
Forecasted Price for 2024-08-09: 74.60, Actual Price: 76.33

Mean Absolute Error (MAE): 5.32

Mean Absolute Percentage Error (MAPE): 6.50%

Root Mean Squared Error (RMSE): 6.29

ACCURACY : 93.50%

  
Figure 9

**REFERENCES**

[1] Purohit, S. K., & Panigrahi, S. (2024). Novel deterministic and probabilistic forecasting methods for crude oil price employing optimized deep learning, statistical and hybrid models. Information Sciences, 658, Article 120021. <https://doi.org/10.1016/j.ins.2023.120021>

[2] Cen, Z., & Wang, J. (2019). Crude oil price prediction model with long short term memory deep learning based on prior knowledge data transfer. Energy, 169, 160–171. <https://doi.org/10.1016/j.energy.2018.12.016>

[3] Zhang, Y., Wang, J., & Wang, J. (2020). Combination of deep learning and fractal theory for crude oil price forecasting. Chaos, Solitons & Fractals, 134, Article 109696. https://doi.org/10.1016/j.chaos.2020.109696

[4] Dutta, A., & Mitra, S. K. (2021). Oil price volatility and the US stock market: Examining the asymmetric effects of geopolitical risk. Energy Economics, 97, Article 105199. https://doi.org/10.1016/j.eneco.2021.105199

[5] Wang, Y., & Liu, L. (2023). Crude oil price forecasting using hybrid ARIMA-LSTM models optimized with particle swarm optimization. Energy, 264, Article 125347. https://doi.org/10.1016/j.energy.2023.125347

[6] Hu, Z., Wang, W., & Tan, Z. (2020). Improved forecasting accuracy of crude oil prices using ensemble empirical mode decomposition-based LSTM. Applied Soft Computing, 95, Article 106536. https://doi.org/10.1016/j.asoc.2020.106536

[7] Tian, Y., Huang, S., & Li, X. (2022). An attention-based convolutional LSTM model for crude oil price prediction. Journal of Cleaner Production, 347, Article 131258. https://doi.org/10.1016/j.jclepro.2022.131258

[8] Zhang, X., & Li, W. (2021). Wavelet transform and deep belief network for crude oil price prediction. Energy Economics, 103, Article 105574. https://doi.org/10.1016/j.eneco.2021.105574

[9] Liu, R., Xu, Z., & Yu, H. (2020). Hybrid deep learning models for crude oil price prediction using decomposition techniques. Neurocomputing, 402, 60–72. https://doi.org/10.1016/j.neucom.2020.03.030

[10] He, Q., & Zhang, T. (2019). Time series forecasting for crude oil prices using EMD-based RNNs. Energy Procedia, 158, 4820–4826. https://doi.org/10.1016/j.egypro.2019.01.679

[11] Zhao, Y., & Zhang, J. (2022). Predicting crude oil prices using deep learning models integrated with sentiment analysis. Renewable and Sustainable Energy Reviews, 155, Article 111857. https://doi.org/10.1016/j.rser.2022.111857

[12] Bao, Y., Tang, Z., & Zhou, T. (2020). Multi-step crude oil price forecasting using ELM-based hybrid models. Energy Economics, 91, Article 104856. https://doi.org/10.1016/j.eneco.2020.104856

[13] Kianian, A., & Hosseinzadeh, S. (2021). Crude oil price prediction using hybrid ANFIS-LSTM models. Journal of Energy Markets, 14, 45–63. https://doi.org/10.1016/j.jem.2021.45

[14] Sun, G., Li, H., & Liu, C. (2023). A hybrid model of GRU and GARCH for crude oil price forecasting. Energy Reports, 9, 145–160. https://doi.org/10.1016/j.egyr.2023.01.021

[15] Zhang, Z., & Yan, Y. (2018). Crude oil price forecasting based on CEEMD and RNN. Energy, 156, 490–500. https://doi.org/10.1016/j.energy.2018.05.108

[16] Chai, L., & Zheng, F. (2021). Ensemble deep learning framework for crude oil price prediction. Renewable Energy, 164, 999–1012. https://doi.org/10.1016/j.renene.2020.09.065