

# **MSc Data Science Project**

**7PAM2002-0509-2023**

Department of Physics, Astronomy and Mathematics

## **Data Science FINAL PROJECT REPORT**

### **Project Title:**

**Development of an Automated GBP/USD Forex  
Trading Bot Using ML Algorithms**

### **Student Name and SRN:**

**Cynthia Chinenye Udoye, 22029346**

Supervisor: **Dr Stephen Kane**

Date Submitted: **Thursday 29<sup>th</sup> Aug**

Word Count: **7,689**

## DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.


I have read the guidance to students on academic integrity, misconduct and plagiarism information at [Assessment Offences and Academic Misconduct](#) and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used chatGPT, or any other generative AI tool, to write the report or code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: Cynthia Chinenye Udoeye

Student Name signature: 

Student SRN number: 22029346

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

## ACKNOWLEDGEMENT

Upon nearing the completion of my post-graduate education, I reflect on this journey as a remarkable learning experience. I am deeply thankful to everyone who has backed me up throughout.

Firstly, my deepest appreciation goes to my supervisor, Dr. Stephen Kane, for his vital mentoring and encouragement in the course of this project.

Let me express my gratefulness to my module leader, Carolyn Devereux, who conducted the module lectures for this project. Her continuous guidance, support, and patience with my curiosity have been pivotal in my academic journey.

I owe a dept of gratitude to the University of Hertfordshire for the essential resources and excellent learning environment supplied, especially the “Library SkillUP” module.

Special recognition to my loving husband, friends, and family for their steady cooperation and understanding.

Additionally, I want to express my gratitude to the open-source community for offering the essential tools and libraries that facilitated this project.

My sincere appreciation goes to everyone who contributed, directly or indirectly, to the successful completion of this work.

## ABSTRACT

The foreign exchange (Forex) market, with a trading volume of \$6.6 trillion daily, offers great opportunities for profit but additionally includes significant risks caused by its volatility. This project explores using high-level machine learning methods to design Forex trading bot which predicts GBP/USD currency pair movements and produces profitable trading signals. We employ three powerful models: Gated Recurrent Unit (GRU), Extreme Gradient Boosting (XGBoost), and Random Forest (RF) to measure their forecasting capabilities in the Forex market.

Our trading strategy leverages these models to create buy and sell signals, incorporating safety measures like take-profit and stop-loss orders to manage risk. After testing the models, it was evident that while each model has its strengths, the ensemble model, which combines all three models, performed the best. The GRU model attained the peak  $R^2$  score of 0.99 but resulted in the lowest profit of \$1,530.51. Whereas, the ensemble model had  $R^2$  score of 0.93 but achieved the highest profit of \$7,715.74, demonstrating superior performance in profitability and risk management.

The results show that combining different models can enhance trading strategies. This project also contributes to the study of applying machine learning in financial sectors and sets the stage for future advancements in automated trading systems. The promising results suggest further exploration to create advanced, data-driven trading solutions in the ever-changing Forex market.

### Keywords:

Forex Trading, Machine Learning, Automated Trading Systems, Gated Recurrent Unit (GRU), Random Forest (RF), and Extreme Gradient Boosting (XGBoost), Trading Bot, Market Price Prediction, Ensemble Techniques, Risk Management, Stop-Loss, Take-Profit, Financial Markets, Algorithmic Trading, Predictive Modelling.

# TABLE OF CONTENT

ABSTRACT.....	4
CHAPTER 1: INTRODUCTION.....	8
1.1 Background and Significance of Automated Trading in Forex Markets.....	8
1.2 Research Question.....	9
1.2.1 Project Aim:.....	9
1.2.2 Project Objectives:.....	9
1.3 Overview and Objectives of the Forex Trading Bot.....	10
CHAPTER 2: LITERATURE REVIEW .....	11
2.1 Overview of Financial Trading Using AI Techniques.....	11
2.2 Forex Market and Prediction Challenges .....	11
2.3 Machine Learning Models in Forex Prediction .....	12
2.3.1 Key Findings from the Study.....	13
2.3.2 Key Findings from the Study.....	14
2.3.3 Key Findings from the Study.....	14
2.3.4 Application of Ensemble Method in Forex Prediction .....	15
CHAPTER 3: METHODOLOGY .....	16
3.1 Project Methodological Approach .....	16
3.2 Data Ethics .....	17
3.3 Data Preparation.....	17
3.4 Feature Engineering .....	19
3.4.1 Technical Indicators.....	19
3.4.2 Exponential Moving Averages (EMAs) .....	19
3.4.3 Relative Strength Index (RSI).....	19
3.4.4 Moving Average Convergence Divergence (MACD).....	20
3.4.5 Feature Selection.....	25
3.4.6 Standard Scaling .....	25
3.5 Model Development.....	26
3.5.1 Gated Recurrent Unit (GRU).....	26
3.5.2 Extreme Gradient Boosting (XGBoost) .....	27
3.5.3 Random Forest (RF).....	28
3.6 Model Evaluation .....	31
CHAPTER 4: RESULTS OF COMPARATIVE ANALYSIS .....	32
4.1 Visual Comparison of Model Prediction.....	32

4.2	<i>Residual Analysis</i> .....	34
4.3	<i>Insights</i> .....	36
4.4	<i>Trading Strategy</i> .....	36
4.4.1	<i>Simulation and Profitability Analysis</i> .....	36
4.5	<i>Model Performance Comparison</i> .....	39
4.6	<i>Trading Performance Comparison</i> .....	39
CHAPTER 5: DISCUSSION.....		41
5.1	<i>Interpretation of Results</i> .....	41
5.2	<i>Comparison with Existing Methods and Literature</i> .....	41
5.3	<i>Overview of Financial Trading Using AI</i> .....	41
5.3.1	<i>Random Forest</i> .....	41
5.3.2	<i>Gated Recurrent Units (GRU)</i> .....	41
5.3.3	<i>Extreme Gradient Boosting (XGBoost)</i> .....	42
5.3.4	<i>Ensemble Models</i> .....	42
5.4	<i>Practical Implications</i> .....	42
5.5	<i>Limitations and Potential Improvements</i> .....	42
CHAPTER 6: CONCLUSION .....		43
6.1	<i>Summary of Findings</i> .....	43
6.2	<i>Future Work and Directions</i> .....	43
6.3	<i>Final Thoughts</i> .....	43
References.....		44
APPENDICES .....		47
APPENDIX A.....		47
APPENDIX B.....		78

## LIST OF FIGURES

Fig 1: Graphical representation of trading analysis by Mara (2022) .....	12
Fig 2: Project Life Cycle.....	16
Fig 3: Comparison of Distribution & Variability .....	18
Fig 4: GBP/USD Price Distribution: Box Plot Visualisation After Outlier Removal .....	18
Fig 5: Forex Analysis with EMAs, RSI, and MACD .....	21
Fig 6: Correlation Heatmap .....	22
Fig 7: Pair Plot .....	23
Fig 8: GRU Model Predictions vs. Actual Closing Prices.....	33
Fig 9: XGBoost Model Predictions vs. Actual Closing Prices .....	33
Fig 10: RF Model Predictions vs. Actual Closing Prices .....	34
Fig 11: GRU Residual .....	34
Fig 12: XGBoost Residual.....	35
Fig 13: RF Residual .....	35
Fig 14: Ensemble Predictions vs. Actual Closing Prices.....	38
Fig 15: Ensemble Residual.....	38
Fig 16: Comparative Analysis of Model Performance Metrics.....	39
Fig 17: Comparative Analysis of Trading Performance Metrics Across Models.....	40

## LIST OF TABLES

Table 1: Descriptive Statistic Summary .....	24
Table 2: GRU Parameters and Rationale for Selection .....	26
Table 3: XGBoost Parameters and Rationale for Selection .....	27
Table 4: XGBoost Model Tuning and Performance Summary .....	28
Table 5: RF Parameters and Rationale for Selection .....	29
Table 6: RF Model Tuning and Performance Summary.....	29
Table 7: Model Performance Metrics Comparison .....	32
Table 8: Trading Performance Metrics Across Different Models .....	37
Table 9: Performance Metrics Comparison Across Different Models Including Ensemble .....	37

## CHAPTER 1: INTRODUCTION

---

Archer (2012) states that the USD and GBP are among the most frequently traded currencies, known as major currencies. The selling of one currency and buying of another currency is what we call Forex trade. The GBP/USD pair being among the most frequently traded is offering numerous opportunities for profit because of its volatility as well as risks due to emotional sensitivity to the global economy, especially forex traders whose decisions are influenced by emotions. There has been a rapid increase in leveraging modernised technological tools to develop automated trading systems since the discovery of machine learning and continuous advancement in AI technology. These tools and systems aim to capitalise on market shortcomings and inefficiencies by executing trades based on the predictions of the algorithms, thereby reducing the independence on human intuition and emotion in trading decisions (Cohen, 2022). The foreign exchange usually called Forex market is known as the biggest financial market globally and with a daily trading volume of approximately \$6.6 trillion (Bank for International Settlements, 2019).

### **1.1 Background and Significance of Automated Trading in Forex Markets**

Algorithmic trading bots can sometimes be referred to automated trading system and has significantly changed the setting of the financial markets. These systems use predefined rules and sophisticated algorithms to execute trades at speeds and frequencies that are beyond human capabilities (Nokeri, 2021). According to Mesleh and Mahmoud (2021), it is estimated that around 60% of Forex trading volume is managed by automated algorithms, particularly in developed markets. This shift towards automation is driven by the potential for increased efficiency, reduced transaction costs, and the ability to process significant amounts of data to identify trading opportunities.

Using machine learning models to develop automated trading systems has become an important point for recent researchers. The ensemble machine learning models has demonstrated effectiveness in the prediction of Forex price movements and also enhancing trading strategies (Loh et al., 2022). The ability of these models to adjust to evolving market conditions and also gain insights from the historical data makes them particularly suitable for Forex trading, where price movements are shaped by numerous factors, including market sentiment, geopolitical events, and economic indicators (Market expectations and forward premium, 2024).

The Forex market is acknowledged as the biggest financial market worldwide, operating 24/7 daily across different time zones (Abednego et al., 2018). This continuous operation makes it an ideal candidate for automation, as algorithms can monitor and react to market conditions without the limitations of human traders.

According the literature review by Dakalbab et al. (2024), the incorporation of AI and machine learning methods has significantly increased. Applying this AI in trading does not only improves efficiency but also opens new opportunities for innovation in financial services. These technologies provide sophisticated tools for analysing the market data, identify patterns, and making predictions, significantly enhancing the capabilities of automated trading systems.



AI algorithmic trading provides greater edge compared to traditional human-driven algorithmic trading (Ta et al., 2018, Li et al., 2020). For instance, AI can discover hidden trends and patterns in vast amount of data that human might miss, and can also react to market changes because it is fast, efficient, and cost effective. (Dakalbab et al., 2024).

Despite the advancements, integrating AI in Forex trading comes with challenges. The quality of data, high speed in data processing, and the dynamic nature of financial markets require robust and adaptable algorithms. Moreover, as noted by Gai et al. (2018), the financial sector's non-linear, non-stationary characteristics pose additional complexities that need to be addressed.

Overall, the Forex price fluctuations are affected by numerous outside influences, making it challenging to trade profitably. Nevertheless, the success in modelling the price movements could generate high profitability investment strategies (Santuci et al., 2022). There are a lot of opportunities in forex trading despite the risk due to market changes. According to Mesleh and Mahmoud (2021), many experts believe that 60% of Forex trading volume is handled by automated algorithms, especially in developing countries.

## **1.2 Research Question**

How can machine learning algorithms (GRU, XGBoost, and Random Forest) be compared to develop the most profitable automated trading bot for the GBP/USD Forex market?

### **1.2.1 Project Aim:**

The focus here is to create automated trading bot for the GBP/USD utilising ML algorithms. Developing a system that can trade profitably by using advanced algorithms to understand market trends and make quick decisions is the goal of the project and the performance of the trading bot will be evaluated based on various metrics and profitability analysis to identify the best model for trading.

### **1.2.2 Project Objectives:**

The objectives are discussed below:

- 1. Data Collection and Preprocessing:**
  - Collect historical Forex data for GBP/USD from Yahoo Finance.
  - Clean and preprocess historical data to ensure its suitability for analysis.
  - Carry out exploratory data analysis (EDA) to discover trends and patterns in the data.
- 2. Development and Evaluation of models:**
  - Build and train three machine learning models: Gated Recurrent Unit (GRU), Extreme Gradient Boosting (XGBoost), and Random Forest.
  - Evaluate the accuracy of these models in predicting Forex price movements.
- 3. Trading Strategy Implementation:**
  - Implement a simple trading strategy centred on the predictions created from each model.

- Simulate trades using a back testing approach to evaluate the effectiveness of each trading strategy.
- 4. **Profitability Analysis:**
  - Analyse the trading results for each model to determine profitability, considering metrics such as total returns, Sharpe ratio, and drawdown.
- 5. **Model Comparison:**
  - Examine model performance and profitability.
  - Identify the best-performing model for the development of the automated trading bot.

### ***1.3 Overview and Objectives of the Forex Trading Bot***

The analysis conducted by Loh et al. (2021), which explores the efficacy of ensembles of machine learning models for Forex trading informed my interest in researching further in this project. My research work will be focusing on three specific models: Gated Recurrent Unit (GRU), Extreme Gradient Boosting (XGBoost), and Random Forest (RF) due to their temporal dependencies, robustness, and ability to handle nonlinear data respectively in predicting Forex prices. As explored in the literatures reviewed in the next chapter, these models are highlighted as the most effective models for predicting future currency price fluctuations.

The historical data was gathered from yahoo finance and it covers over 20 years. This data will be used to train, validate, and test the models. In order to measure the profitability of the trading bot, the performance will be evaluated by testing it on this data.

## CHAPTER 2: LITERATURE REVIEW

---

The most popular target of the automated system is the GBP/USD currency pair which is noted for its significant liquidity and volatility. Machine learning languages such as GRU, XGBoost and RF are explored due to its predictive capabilities in order to develop a profitable trading bot for GBP/USD market. This literature review examines the application of machine learning algorithm in forex trading, stating the strength and potentials to enhance trading performance.

### **2.1 Overview of Financial Trading Using AI Techniques**

A review was carried out by Dakalbab et al. (2023) covering about 143 research articles published between year 2015 and 2023. The review disclosed that the most analysed markets for trading are the Forex market, stock market, and the cryptocurrency. The study further revealed that the most commonly studied currency pair is EUR/USD and GBP/USD with the former ranking first. According to the review, 30% to 29% of the reviewed papers shows that the most popular AI method used are deep and reinforcement learnings.

They also found that commonly used datasets come from Yahoo Finance, and the top performance metrics are:

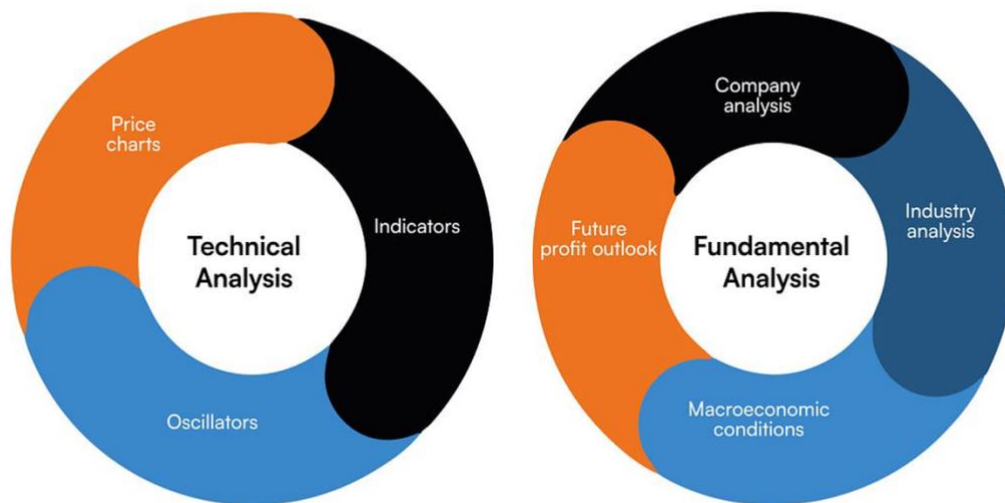
- Sharpe Ratio: 18%
- Rate of Return (RoR): 12%
- Maximum Drawdown: 10%
- Total Return: 10%
- Total Profit (Net Profit): 9%

The study also states the importance of the momentum indicators like Commodity Channel Index (CCI), Moving Average Convergence Divergence (MACD), and Relative Strength Index (RSI) in predicting market trends. They suggested employing ensemble models to enhance predictive accuracy. Consequently, we will utilize ensemble methods that combine GRU, XGBoost, and RF in the project to capture various aspects of Forex data, resulting in more accurate and robust predictions. These ensemble methods will capture temporal patterns, non-linear relationships, and feature importance, thereby improving prediction reliability and accuracy.

### **2.2 Forex Market and Prediction Challenges**

The Forex market operates continuously, influenced by political events, economic news, and international factors. It has high trading volumes and is very volatile, making it complex and hard to predict. The traditional statistical methods still struggle with the non-linearity and complexity of Forex data.

Traders have traditionally used technical and fundamental analysis to predict Forex price movements. Fundamental analysis looks at economic indicators and political events to estimate currency values and the technical analysis uses historical price trends and indicators like relative strength index (RSI) and moving averages. However, these methods often fail to capture the complicated and unstable nature of the Forex market (Ayitey Junior et al., 2023).



*Fig 1: Graphical representation of trading analysis by Mara (2022)*

The use of machine learning is increasing today because they can manage the complexity and non-linear relationships in financial data. These algorithms analyse large volume of past and current data to identify the relationships and trends that traditional strategy overlook. According to Santuci et al. (2022), these modern methods perform better in dealing with the discontinuities and complexities of Forex data.

In summary, the Forex market's continuous operation, high volatility, and complexity make traditional prediction methods less effective. Modern machine learning techniques offer better tools for predicting Forex market movements, providing more accurate and reliable results.

### **2.3 Machine Learning Models in Forex Prediction**

#### **Random Forest**

Santuci et al. (2022) evaluated Forex trading strategies using Support Vector Machine (SVM) and RF models, leveraging technical indicators like moving averages. Their study on currency pairs such as USD/JPY, EUR/USD, GBP/USD, and USD/CHF showed that machine learning algorithms generally outperform traditional technical indicators.

Random Forest, introduced by Breiman (2001), builds several decision trees during training and then combines them by averaging the predictions for regression or taking majority vote for classification. It is effective for financial market applications because of its competence in handling multidimensional data and complex interactions.

## Application in Forex Trading

In Forex trading, Random Forest is used to recognize patterns and classify data, effectively managing non-linearity and complexity in financial time series. The study found that Random Forest generally outperforms traditional indicators and the MACD strategy. The model's success is attributed to extensive parameter tuning, including the count of trees, tree depth, and count of features.

### 2.3.1 Key Findings from the Study

This research was done by using historical data from yahoo finance specifically these currency pairs USD/JPY, USD/JPY, EUR/USD, and GBP/USD. They evaluated the performance using three different metrics:

- **Sharpe Ratio:** The investment's risk-adjusted return was evaluated by this ratio. The study found out that Random Forest models achieved higher Sharpe ratios compared to traditional indicators thereby indicating better performance relative to the risk taken.
- **Calmar Ratio:** This ratio measures the risk-adjusted return while considering the maximum drawdown. Random Forest models showed superior performance with higher Calmar ratios, suggesting a favourable return-risk relationship.
- **Annualized Return:** The annualized return of the strategies using Random Forest was generally higher than those using traditional indicators and the MACD strategy. This reveals the possible profitability of using machine learning in Forex trading.

Overall, the application of Random Forest in Forex trading shows significant promise. Handling of non-linear and complex data, together with extensive parameter tuning, could allow for better performance and accurate predictions especially when compared to traditional methods. These findings prove the possibilities of machine learning techniques in enhancing trading strategies and achieving higher returns in the Forex market.

## Gated Recurrent Units (GRU)

GRUs, a class of recurrent neural network (RNN), are effective for sequential data and time series analysis. They use gating mechanisms to control information flow, making them efficient in capturing long-term dependencies. The GRU model's ability to capture temporal dependencies makes it a valuable tool for predicting future currency price movements (Cho et al., 2014).

## Applications in Forex Trading

Pahlevi et al. (2023) compared Long Short-Term Memory (LSTM) and GRU models for Forex price prediction. GRUs, with fewer gates than LSTM, are less computationally intensive and faster to train. Their study used a dataset from Investing.com spanning January 2003 to January 2022, focusing on the EUR/USD pair. The GRU model achieved an RMSE of 0.054, MAPE of 0.037, and an  $R^2$  of 97%, indicating high prediction accuracy.

### 2.3.2 Key Findings from the Study

Highlighting the following from the study of Pahlevi et al. (2023):

- **RMSE:** Gaps between estimated and real values. And the GRU achieved a lower RMSE of 0.054 implying a good match.
- **MAPE:** GRU model recorded a MAPE of 0.037, reflecting the error rate in predictions expressed as percentage. Lower MAPE values indicate higher accuracy.
- **R<sup>2</sup>:** The GRU model recorded an R<sup>2</sup> value of 97%, indicating that 97% of the variation in the Forex prices could be accounted for by the model.

It is evident to note that from the comparative study of LSTM and GRU models for Forex prediction by Pahlevi et al. (2023) the GRUs proves more successful in handling time series data. The values of RMSE, MAPE, and R<sup>2</sup> proves that the GRU model has strong predictive accuracy, and it also shows its potential as a reliable tool for Forex trading strategies. The study suggests that GRU can provide efficient and accurate predictions, thereby making them valuable for market participants who aims to make informed trading decisions.

### Extreme Gradient Boosting (XGBoost)

XGBoost is a highly efficient gradient boosting algorithm renowned owing to its speed & performance. It is extensively applied in financial modelling due to its capability to manage large datasets and complex feature interactions (Chen and Guestrin, 2016). Research, such as the study by Loh et al. (2021), has shown the effectiveness of XGBoost in forecasting Forex price movements. This algorithm strengthens predictions by integrating multiple weak learners, typically decision trees, into a robust predictive model, making it ideal for real-time Forex trading applications.

### Application of XGBoost in Forex Prediction

An excellent study by Agung et al. (2022) employed XGBoost to predict Forex prices, optimizing key parameters like the number of estimators, dataset shifting, data length, and the objective function. The model achieved the lowest RMSE of 0.003098 using a merged dataset of USD/JPY, USD/GBP, and USD/EUR.

XGBoost's superior performance is due to regularization, parallel processing, efficient handling of missing data, and cache awareness. These features make it a strong candidate for practical Forex trading systems.

### 2.3.3 Key Findings from the Study

The study by Agung et al. (2022) highlights the following:

- **RMSE:** The XGBoost model obtained the lowest RMSE of 0.003098, signifying high accuracy in predicting Forex prices.
- **Parameter Optimization:** The study showed that optimizing parameters such as the number of estimators significantly improved the performance of the model.

XGBoost has proved higher performance in Forex price prediction due to its advanced features and optimization capabilities. The study by Agung et al. (2022) supports the use of XGBoost in practical Forex trading applications, offering high accuracy and efficiency. This makes XGBoost a valuable tool for traders looking to leverage machine learning for better decision-making in the Forex market.

### **Ensemble Models**

Ensemble techniques integrate multiple ML models to improve prediction accuracy. Upadhyay et al. (2021) shows that the hybrid models combining Long Short-Term Memory (LSTM), Convolutional Neural Network (CNN), and Support Vector Regression (SVR) outperformed individual models in predicting Forex price.

#### ***2.3.4 Application of Ensemble Method in Forex Prediction***

The study by Upadhyay et al. employed an ensemble approach combining CNN, LSTM, and SVR to predict Forex rates using historical OHLC (Open, High, Low, Close) data. The model was trained on the EUR/USD pair. The ensemble model demonstrated higher  $R^2$  values compared to individual models, indicating superior predictive accuracy and robustness. This result is more reliable and consistent forecasts, crucial for making informed trading decisions. They are more robust, as they mitigate the weaknesses of individual models (Upadhyay et al., 2021).

Overall, the utilisation of machine learning in Forex market prediction shows promising results, with models like GRU, XGBoost, and Random Forest outperforming traditional methods.

## CHAPTER 3: METHODOLOGY

### 3.1 Project Methodological Approach

Development of an Automated GBP/USD Forex Trading Bot Using ML Algorithms is the goal. The GBP/USD currency pair will be used along with indicators such as EMAs, RSI, and MACD. The models employed will be GRU, XGBoost, and RF, evaluated using metrics like MSE, MAE, RMSE,  $R^2$ , and MAPE, and profitability analysis using Final Balance, Profit, Sharpe Ratio, and Max Drawdown.

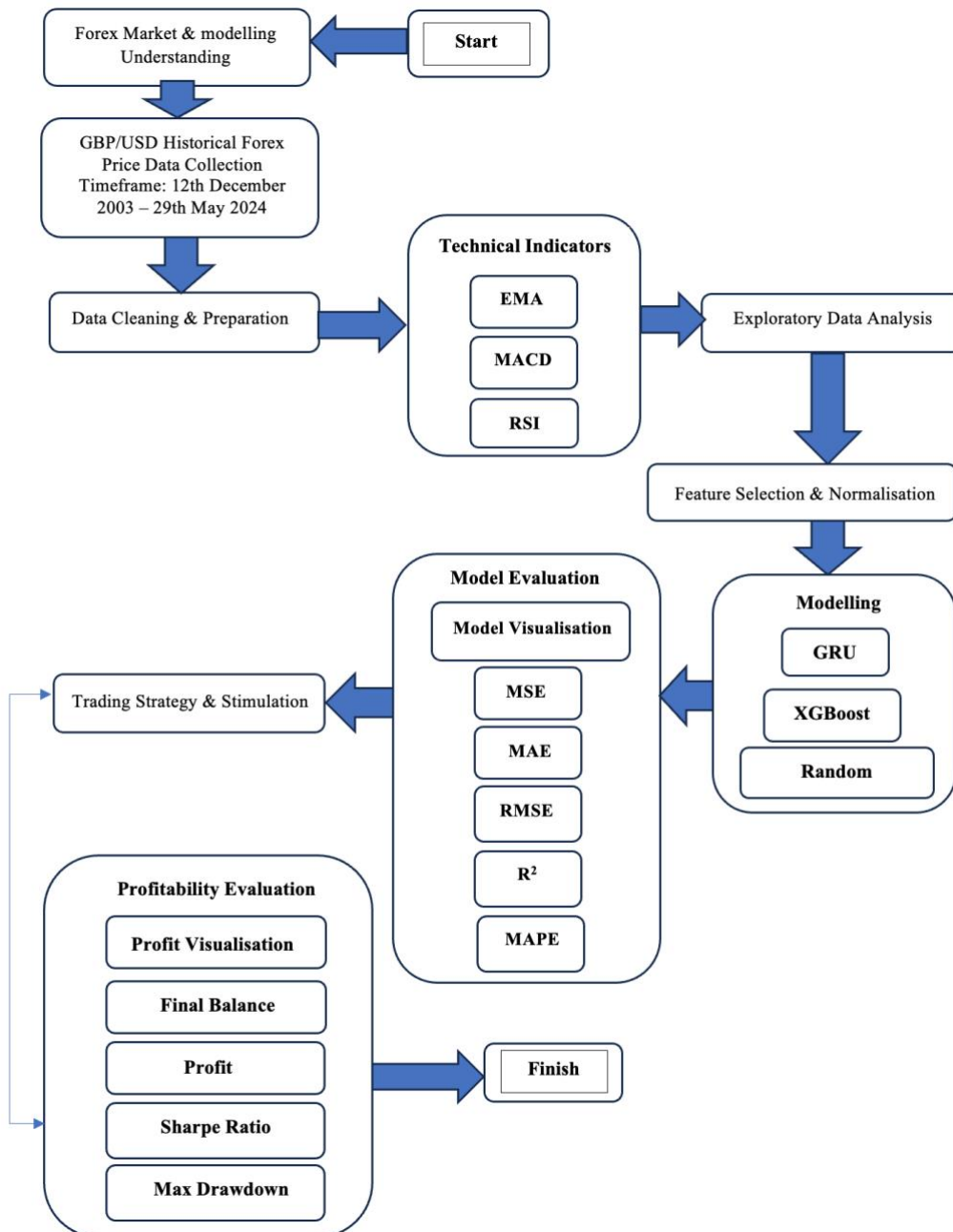


Fig 2: Project Life Cycle



### **3.2 Data Ethics**

This forex trading bot totally follows ethical guidelines by using public data available in Yahoo Finance, and does not contain any personal information therefore it adheres to GDPR guidelines (Information Commissioner's Office, 2023). The data was used according to Yahoo Finance's terms of service and the University of Hertfordshire's ethical standards (University Policies and Regulations UPR RE01, 2018). The project is open and clear with comprehensive documentation and version tracking available on GitHub which makes it easy for others to see and replicate the work. We also focused on responsible research by evaluating the automated trading bot carefully to minimize financial risks, and also assuring that all results are reported with all honesty. It's vital to understand that the algorithm and trading bot are still under development and monitored, so they shouldn't be used for real money trading to avoid losing money.

Additionally, the project deals with ethical concerns like market influence by using safeguards, such as technical indicators, just to be sure that the trades are based on actual market conditions. We also ensured total transparency by providing easy to understand details about the bot's algorithms and how it works, and we promote fairness by giving everyone the same opportunity to use the bot

### **3.3 Data Preparation**

First, I imported essential libraries like pandas, numpy, matplotlib, and seaborn, and accessed dataset from the mounted Google Drive. I imported the dataset into a data frame labelled "cu\_fx\_df" and checked the first and last five rows to review the data's start and end dates.

I examined the dataset's shape, revealing 5,348 rows and 7 columns. To understand the dataset's statistics, I used the ".describe()" function and checked the data frame structure with the ".info()" function to identify missing values. There were 17 rows with missing values, which I removed since their absence wouldn't significantly impact the data.

Next, I dropped unnecessary columns: the 'adjacent close' column, as I focused on the closing price, and the 'volume' column, which had only zero values. I created histogram and box plots visualization to compare the distribution and variability of each price columns. To the left is the histogram which shows how the prices are distributed, while the boxplots on the right reveals the shape and spread of the data, helping to understand the behaviour of price data over time. The boxplot for low price column also highlights the presence of an outlier (see Fig 3).

Using the Interquartile Range (IQR) technique, I removed outliers from the price columns with a function called "remove\_outliers\_iqr".

I rechecked the data with box plots to confirm the absence of outliers (see Fig 4). The outlier removal has resulted in no extreme values outside the whiskers, suggesting a more uniform dataset without anomalies and also converted the 'date' column to datetime format. Finally, I verified the cleaned dataset's shape, which now had 5,331 rows and 5 columns.

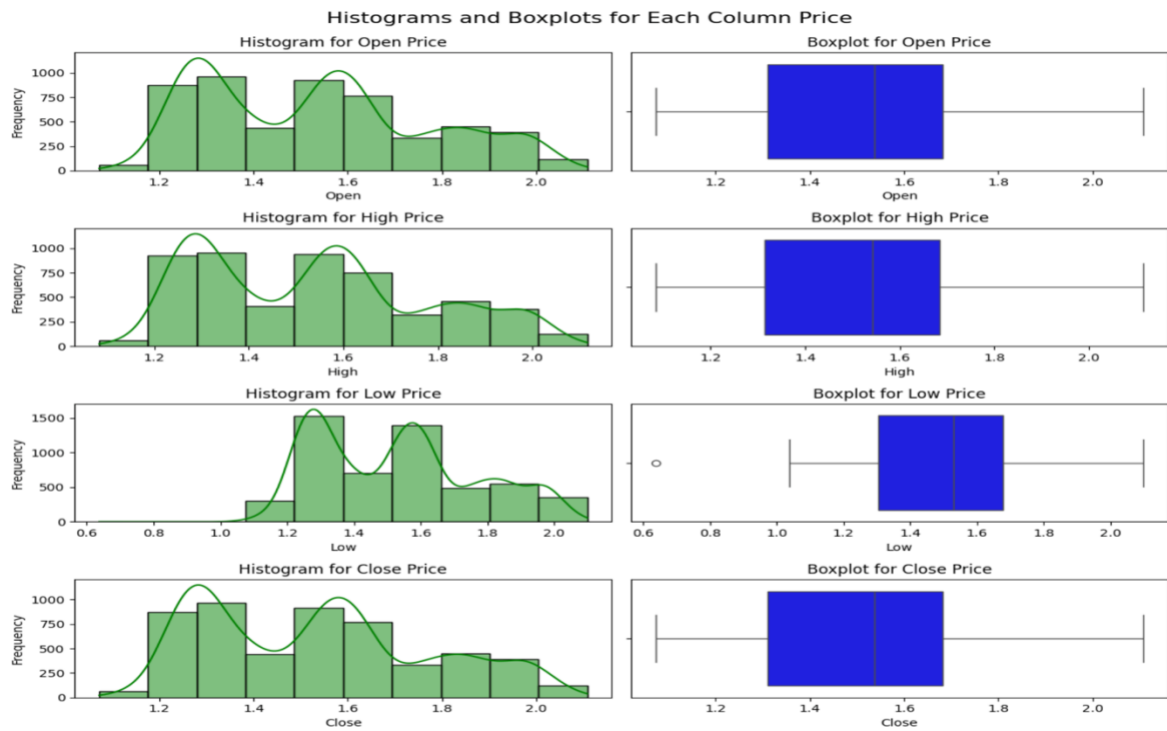


Fig 3: Comparison of Distribution & Variability



Fig 4: GBP/USD Price Distribution: Box Plot Visualisation After Outlier Removal

### 3.4 Feature Engineering

As part of pre-processing and in other to improve the accuracy and speed of the models I decided to deploy feature engineering which involves several steps, using domain knowledge to select, modify or create new features that will make the algorithms work better.

#### 3.4.1 Technical Indicators

The analysis to use on historical data to make trading decision on whether to open, close, or modify a buy or sell trade is what we refer to as technical analysis (Abednego et al., 2018). Technical indicators are very important tools or formulas for analysing market trend in the context of Forex trading. The technical indicators I used for this project are listed below:

- Exponential Moving Averages (EMAs),
- Relative Strength Index (RSI), and
- Moving Average Convergence Divergence (MACD).

Every indicator has unique role in capturing different aspects of market dynamics.

#### 3.4.2 Exponential Moving Averages (EMAs)

The EMAs are a form of moving average tool used by traders as an indicator. It helps to smooth out price fluctuations and also makes it very easy to identify trends directions thereby reducing the noise from daily price fluctuation making it an important tool for generating buy or sell signal (James Chen, 2024). I used EMA\_20 and EMA\_50 which represent the average price over the last 20 and 50 days respectively, with more focus on current price.

##### EMA Formula:

As stated by James Chen (2024) the formula is:

$$EMA_{today} = \left( \frac{Price_{today} - EMA_{yesterday}}{K + 1} \right) + EMA_{yesterday}$$

##### Where:

- $EMA_{today}$  = EMA value for today
- $Price_{today}$  = current close price
- $EMA_{yesterday}$  = EMA value for yesterday
- $k$  = smoothing factor, calculated as  $\frac{2}{n+1}$ , where  $n$  = number of the days or period.

#### 3.4.3 Relative Strength Index (RSI)

RSI is another important indicator that traders used to measure the speed and change in the price movement. It helps to understand whether a particular currency pair is:

- **overbought** which means the price has gone up quickly and will likely drop, or
- **oversold** meaning the price has gone down dramatically and might be due to rise.

It can also signal when to buy or sell. For this project I used 14-day period to measure the speed and the magnitude of the price change. An RSI reading 70 or above signals overbought situation, whereas if it is reading 30 or below reflects oversold condition (Jason Fernando, 2024).

#### **RSI Formula:**

As stated by Jason Fernando (2024) the formula is:

$$RSI = 100 - \left( \frac{100}{1 + RS} \right)$$

#### **Where:**

- Average Gain = sum of all positive changes over the last 14 days
- Average Loss = sum of all negative changes over the last 14 days
- $RS = \frac{\text{Average Gain}}{\text{Average Loss}}$

#### **3.4.4 Moving Average Convergence Divergence (MACD)**

The MACD is a well-known technical indicator applied to identify direction changes, momentum, strength, and period of a trend in a currency pair price. It can also be utilised to detect buying or selling signals. In this project, I used the 12-day and 26-day EMAs to derive MACD line. The MACD Histogram is an oscillator fluctuating below and above the zero line (Wang and Kim, 2018). A crossover among the MACD line and the Signal line can signal potential buy or sell signal.

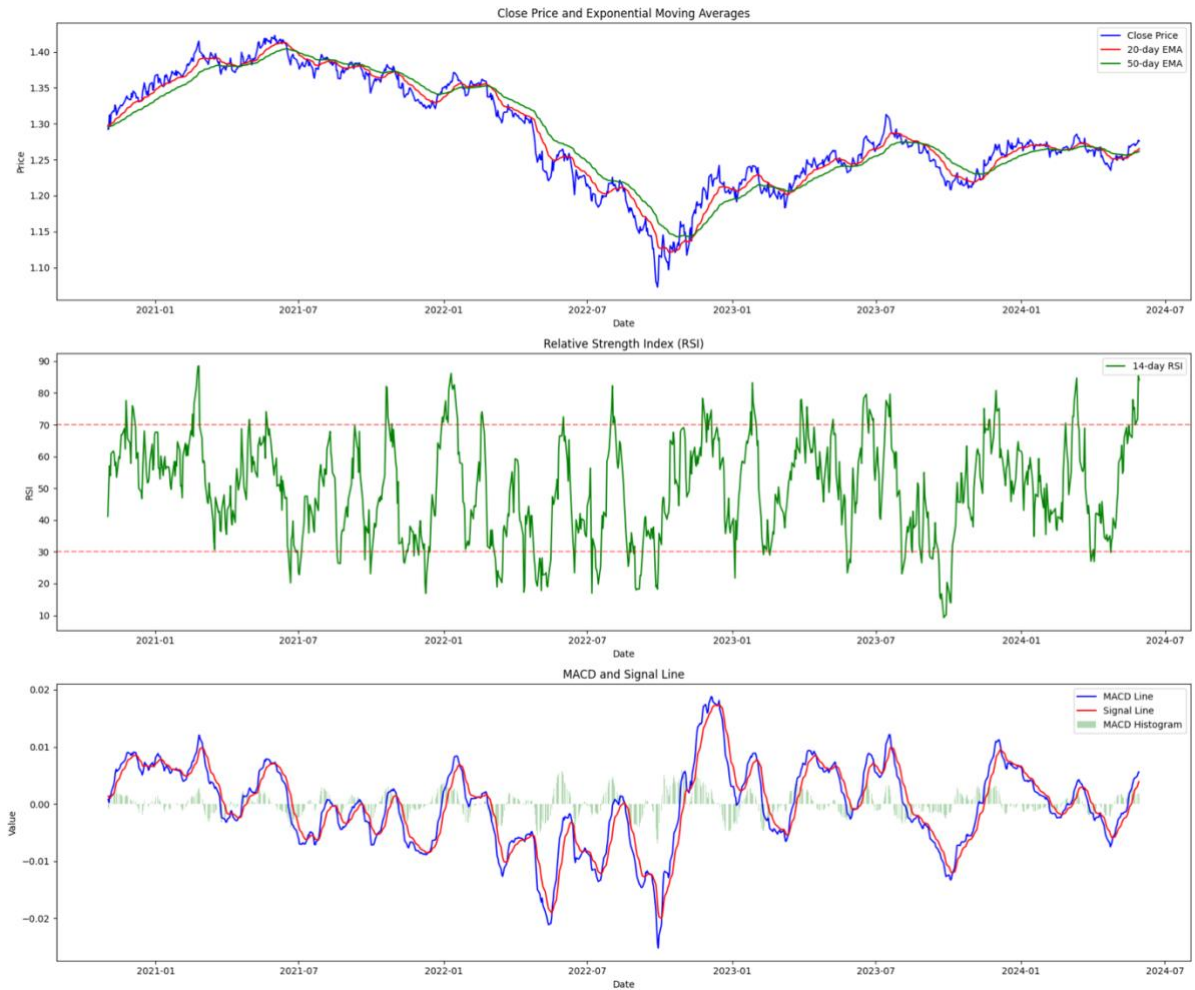
#### **MACD Formula:**

As stated in TradingView (2024) the formula is:

$$MACD\ Line = EMA_{short} - EMA_{long}$$

$$Signal\ Line = EMA_{MACD\ Line, 9-days}$$

$$MACD\ Histogram = MACD\ Line - Signal\ Line$$



*Fig 5: Forex Analysis with EMAs, RSI, and MACD*

In other to capture more patterns in the price movement so as to enhance model's predictive accuracy and boost the performance of the automated trading bot, we incorporated all these technical indicators (see Fig 5).

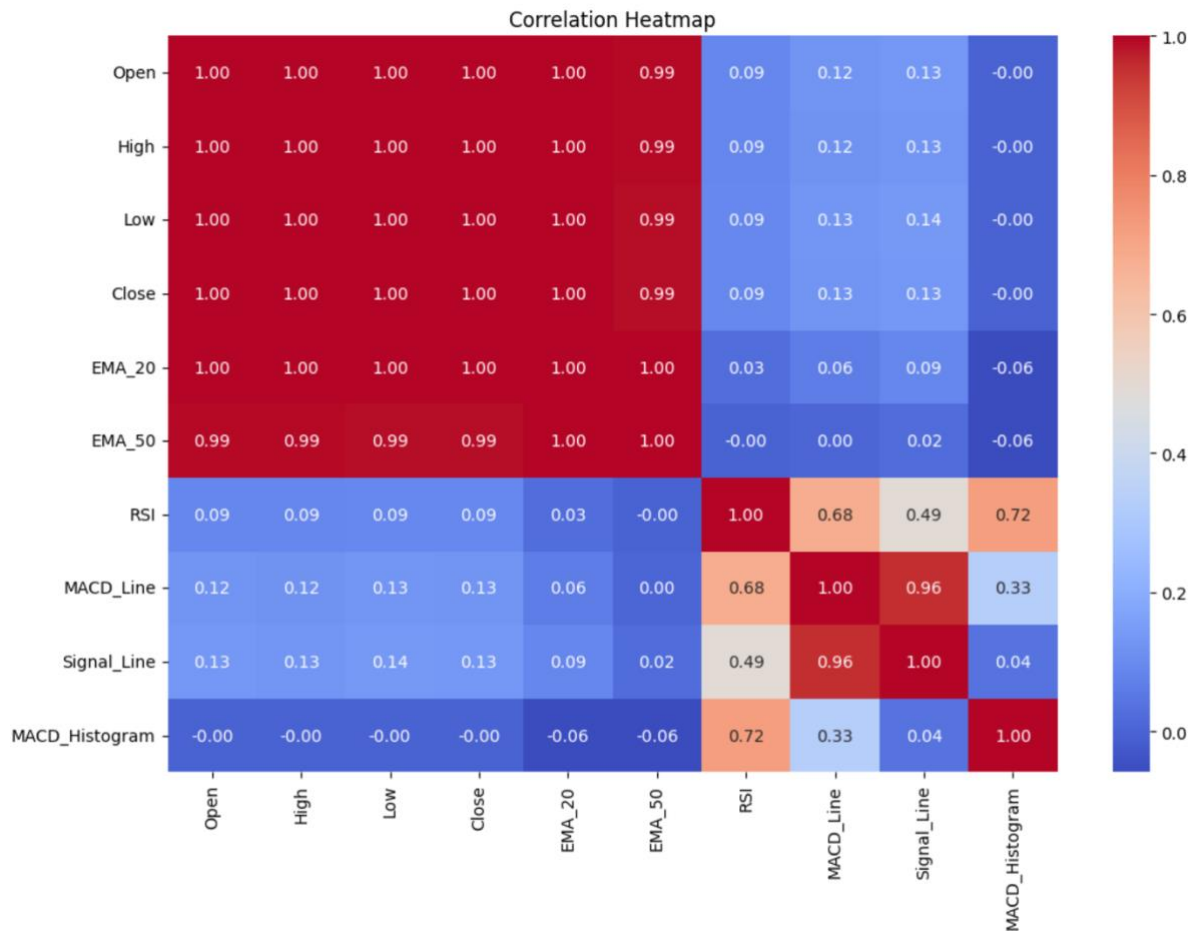
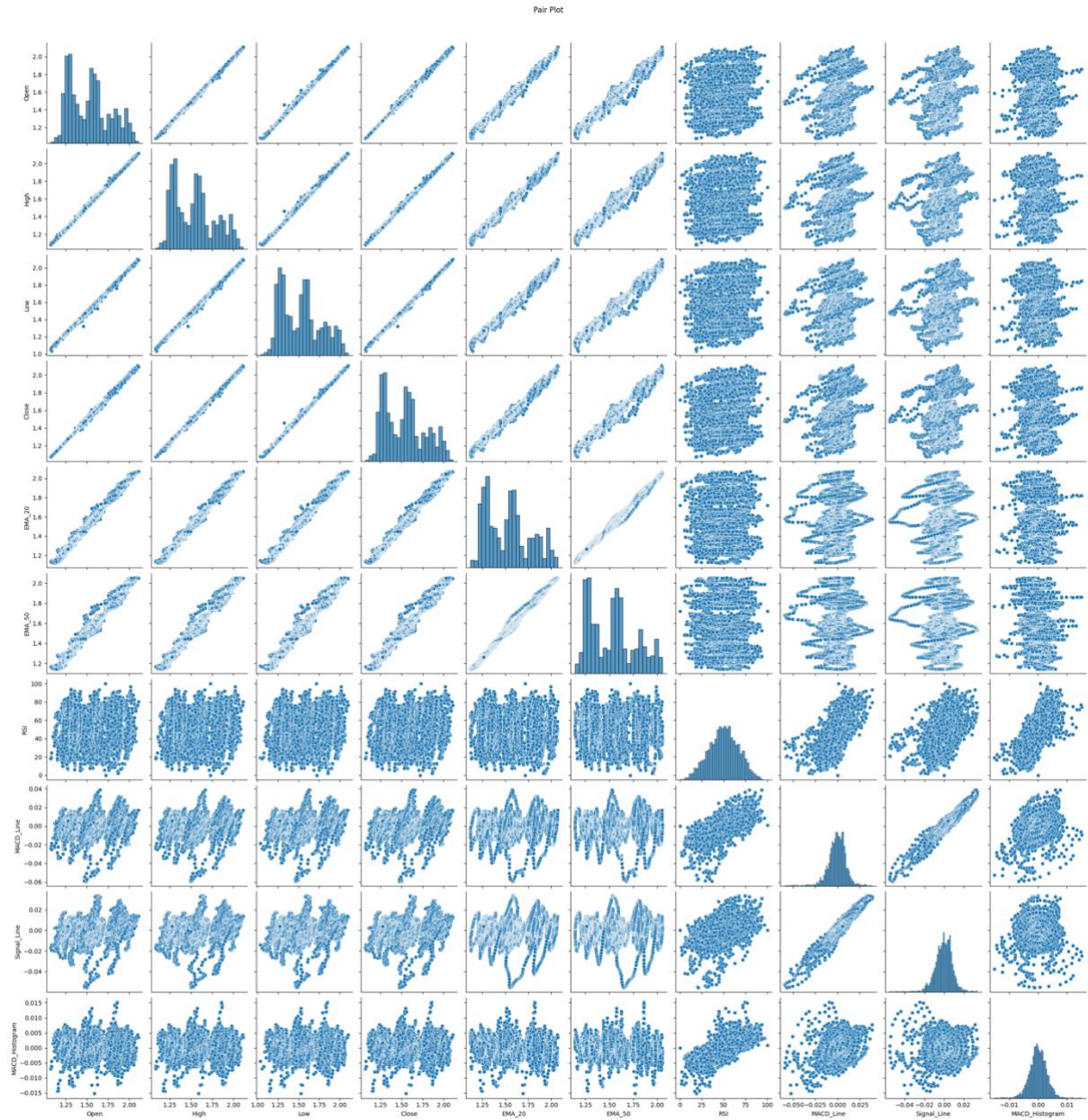


Fig 6: Correlation Heatmap

To further understand how these indicators interact with the GBP/USD price data, extract insights from data and detect patterns, I deployed exploratory data analysis (EDA) which is the process of analysing and visualising dataset to grasp their key features, along with the help of statistical graphs and other data techniques.

Fig 6 shows a correlation heatmap, which clarifies the relationships between different features in the data frame. For instance, the close price has a strong correlation with open, high, low prices, and EMAs, which indicates that they move together. MACD line and signal line show a high positive correlation because they are both component of the same MACD indicator. while the RSI and MACD has moderate correlations with the price columns, offering additional insights to momentum and trends.



*Fig 7: Pair Plot*

Fig 7 presents a pair plot that showcases the relationships between pairs of features through scatter plots and histograms. The diagonal plots reveal the distribution of individual features, highlighting their spread and central tendency. The off-diagonal plots help to identify patterns and relationships between pairs of features. The histograms in these plots depict the typical distribution of time series data, while the scatter plots show linear relationships between price columns.



*Table 1: Descriptive Statistic Summary*

	Date	Open	High	Low	Close	EMA_20	EMA_50	RSI	MACD Line	Signal Line	MACD Histogram
count	5330	5330.000000	5330.000000	5330.000000	5330.000000	5330.000000	5330.000000	5330.000000	5330.000000	5330.000000	5330.000000
mean	2014-03-05 00:17:01.238273792	1.534178	1.539934	1.528196	1.534216	1.535024	1.536317	49.980568	-0.000603	-0.000606	0.000003
min	2003-12-01 00:00:00	1.072846	1.083541	1.037904	1.072754	1.120575	1.142428	0.000000	-0.059307	-0.055333	-0.015220
25%	2009-01-23 18:00:00	1.309934	1.314298	1.305458	1.309989	1.309257	1.306019	38.382537	-0.005958	-0.005691	-0.001784
50%	2014-03-08 12:00:00	1.536098	1.542615	1.530397	1.536134	1.536795	1.539070	50.196525	0.000041	0.000051	-0.000002
75%	2019-04-18 18:00:00	1.681011	1.684409	1.678465	1.680997	1.682355	1.681341	61.905743	0.005701	0.005409	0.001887
max	2024-05-29 00:00:00	2.108415	2.115820	2.098812	2.108192	2.070346	2.054503	100.000000	0.038755	0.033321	0.015213
std	NaN	0.239760	0.240494	0.239159	0.239793	0.238818	0.237261	16.854668	0.010082	0.009511	0.002991

Lastly, I conducted a descriptive statistic on the cleaned data frame (See Table 1). The summary statistics gives a snapshot of the attributes such mean, min, max, and standard deviation for each feature in the dataset. For instance, the mean values of the price-related columns (Open, High, Low, Close) are all close to each other, reflecting to the central tendency of the data over time. The standard deviations points to the extent of variability in the data, with the price columns showing similar levels of dispersion. The RSI has a mean close to 50, which suggests a balance between overbought and oversold conditions on average. The MACD line and signal line have mean values near zero, which is typical for a dataset with fluctuating price trends.

These technical indicators and visualizations contribute to a deeper understanding of the data frame making it ready for building machine learning models.



### 3.4.5 Feature Selection

Feature selection is one of the processes used in machine learning and data analysis to select relevant features like columns in my own case. The central aim is to filter out redundant and irrelevant data which can simplify the models, reduce training time, improve performance, and prevent overfitting (Li et al., 2017). I collected all the required column such as Date, Open, High, Low, Close, EMA\_20, EMA\_50, RSI, MACD\_Line, Signal\_Line, and MACD\_Histogram into a variable called “cu\_required\_columns”, then selected the right columns for the X features which is all columns except date and close price, and the target variable y (only the close price) and also the dates as is a time series data. Next, I divided the data into different sets. 70% of the data was used for training, while the remaining 30% was split equally into validation and test sets that is 15% each for better training, validating and testing the model. The output of the splitting is displayed below:

**Training set shape:** (3731, 9), (3731,)

**Validation set shape:** (799, 9), (799,)

**Test set shape:** (800, 9), (800,)

**Training set date:** from 2003-12-01 to 2018-04-11

**Validation set date:** from 2018-04-12 to 2021-05-05

**Test set date:** from 2021-05-06 to 2024-05-29

### 3.4.6 Standard Scaling

The feature scaling technique I used as part of the pre-processing the data to improve model performance is StandardScaler which standardises the features by centring them and scaling to unit variance (GeeksforGeeks, 2023).

The formula for the StandardScaler according to GeeksforGeeks (2023) is:

$$\text{Standardised Value} = \frac{X_i - \mu}{\sigma}$$

$X_i$  : Original value of the feature

$\mu$  : Mean of the feature

$\sigma$  : Standard deviation of the feature

I standardized the features and the Close price. This scaling was done on the training dataset and then implemented to the validation and test datasets to avoid data leakage.

For time series forecasting, I created sequences with a time step of 15, reshaping the datasets into a 3D array for the GRU model and then flat to 2D sequences for the XGBoost and Random Forest models.

### 3.5 Model Development

As a crucial step in this project, we will now proceed to the development of the models.

#### 3.5.1 Gated Recurrent Unit (GRU)

The first model I developed was the GRU which is a type of Recurrent Neural Network (RNN) and is well-suited for sequential data like time series, and it was implemented by Kyunghyun Cho et al. in 2014. It is significantly better suited for handling both long-term and short-term dependencies in the dataset. For this project, it receives the sequential historical data as input and output a sequential predicted forex price. Table 2 shows key parameters for the GRU model, selected settings, detailing the architecture, and rationale for each choice.

Table 2: GRU Parameters and Rationale for Selection

Parameter	Values	Reasons for the Choice of Parameter
Units	50	To balance model complexity and efficiency in training time.
Dropout Rate	0.2	To prevent overfitting while retaining model capacity.
Learning Rate	0.001	Standard starting point for Adam, providing a balance between learning speed.
Return Sequences	True (first GRU layer)	Required for stacking GRU layers; passes full sequence to the next layer.
Optimizer	Adam	Adam optimizer chosen for its adaptive learning rate capabilities.
Loss Function	Mean Squared Error	MSE chosen as it is standard for regression tasks.
Early Stopping	Enabled	Used to prevent overfitting & reduce unnecessary epochs.
Patience	10	To allow some room for the model to improve.
Layers	2 GRU, 2 Dropout, 1 Dense	Architecture designed to capture temporal patterns and prevent overfitting.
Performance	- R <sup>2</sup> Training: 1.00 - R <sup>2</sup> Test: 0.99	Exceptional prediction accuracy, indicating nearly perfect model performance.
Training Details	- Final Epoch: 46 - Training Loss: 0.0098 - Validation Loss: 0.0121 - Duration: 2 min 11 sec	Efficient training process with minimal loss and rapid completion time.

### 3.5.2 Extreme Gradient Boosting (XGBoost)

Next, I trained an Extreme Gradient Boosting (XGBoost) which is known for its highly efficient, scalability in machine learning algorithm and also for its excellent performance in structured data task. It is based on the concept of gradient boosting involving the training of an ensemble of decision trees where each tree aims to correct the error of the previous ones. This model stands out because of its regularisation and it allows extensive hyperparameter tuning, helping to optimise model performance while mitigating overfitting (Chen & Guestrin, 2016).

The following table summarises the key parameters for the XGBoost model. It includes details on the model architecture, the chosen parameters, the reason behind the choice of the parameters and the hyperparameter tuning values see Table 3 and Table 4

*Table 3: XGBoost Parameters and Rationale for Selection*

Parameter	Values	Reasons for the Choice of Parameter
N estimators	100	Provides a balance between model complexity and training time.
Learning Rate	0.01	A learning rate for gradual learning, helping to improve accuracy of the model
Max depth	3	Allows the model to learn patterns while mitigating the steps
Reg Alpha	L1 Regularisation	Controls complexity and prevent overfitting, setting some coefficients to zero.
Reg lambda	L2 Regularisation	Controls complexity and prevent overfitting, setting some coefficients close to zero but not necessary zero.

Table 4: XGBoost Model Tunning and Performance Summary

Step	Details
Model Used	XGBoost (Ensemble Method)
Initial Performance	- R <sup>2</sup> on Training Set: 0.86
	- R <sup>2</sup> on Test Set: -4.13
Overfitting Issue	Yes
Parameters Tuned	- Number of Trees (n_estimators): [150, 200, 250]
	- Learning Rate (learning_rate): [0.055, 0.1, 0.15, 0.2]
	- Tree Depth (max_depth): [3, 4, 5]
	- Regularization Alpha (reg_alpha): [0, 0.1, 0.5]
	- Regularization Lambda (reg_lambda): [1, 1.5, 2]
Best Parameters Found	- n_estimators = 150
	- learning_rate = 0.055
	- max_depth = 4
	- reg_alpha = 0
	- reg_lambda = 1
Validation Loss	0.0013204
Final Performance	- R <sup>2</sup> on Training Set: 1.00
	- R <sup>2</sup> on Test Set: 0.86
Training Duration	- Initial Training: 1 second
	- Final Training: 7 minutes, 26 seconds

### 3.5.3 Random Forest (RF)

Lastly, I trained the RF model which is a robust ensemble learning method that improves the accuracy of the algorithm and controls overfitting, by combining the predictions of numerous decision trees. According to Breiman (2001), the algorithm builds several decision trees throughout the training and outputs their mean for regression task or majority vote for classification tasks from the individual trees

With the advantage of the model using multiple decision trees to enhance predictive accuracy we trained the RF model. Table 5 and Table 6 outline the essential parameters for the RF model, Including the model structure, selected parameters, rationale, and the values used during hyperparameter tuning see Table 5 and Table 6

Table 5: RF Parameters and Rationale for Selection

Parameter	Values	Reasons for the Choice of Parameter
N estimators	50	Provides a balance between model accuracy and computational efficiency.
Max depth	10	Captures complex patterns without overfitting.
Min Samples Split	2	Allows the tree to grow sufficiently deep, but with controls.
Min Samples Leaf	2	Ensures that the model is more robust by preventing overly

Table 6: RF Model Tuning and Performance Summary

Step	Details
Model Used	Random Forest (RF)
Initial Performance	- R <sup>2</sup> on Training Set: 1.00
	- R <sup>2</sup> on Test Set: 0.89
Parameters Tuned	- Number of Trees (n_estimators): [100, 150, 200]
	- Tree Depth (max_depth): [10, 15, 20]
	- Minimum Samples to Split Node (min_samples_split): [2, 5, 10, 15]
	- Minimum Samples per Leaf (min_samples_leaf): [1, 2, 4]
Best Parameters Found	- n_estimators = 200
	- max_depth = 15
	- min_samples_split = 15
	- min_samples_leaf = 1
Validation Loss	0.001360
Final Performance	- R <sup>2</sup> on Training Set: 1.00
	- R <sup>2</sup> on Test Set: 0.89
Training Duration	- Initial Training: 45 seconds
	- Final Training: 1 hour, 2 minutes, 56 seconds

In summary, the GRU model demonstrated excellent performance with no need for further tuning, achieving near-perfect accuracy. The XGBoost model showed significant improvement after hyperparameter tuning, while the Random Forest model performed strongly from the start, with tuning providing no further gains. I saved each of these models for future use: the GRU model as `cu_best_gru_model.keras`, the XGBoost model as `cu_best_xgb_model.json`, and the Random Forest model as `cu_best_rf_model.pkl`. These models are now prepared for the next phase of the project, where they will be used to implement and evaluate trading strategies.

### **3.6 Model Evaluation**

I used important several metrics to assess the model's performance.

Firstly, I used the Mean Squared Error (MSE) to calculates the mean squared difference between true and predicted price, offering insight into the overall accuracy of the model

Secondly, was the Mean Absolute Error (MAE) used to evaluates the average absolute difference between the actual and predicted price, offering a direct measure of prediction accuracy.

Thirdly, was the Root Mean Squared Error (RMSE) to square the root of the MSE, so as to allow the error to be interpreted in the same unit as the target price.

Then the  $R^2$  Score which shows how well the independent variables X predicts the dependent variable y, showing the share of variance accounted by the model.

Lastly, was the Mean Absolute Percentage Error (MAPE) measures prediction accuracy as a percentage, offering a relative measure of prediction error.

## CHAPTER 4: RESULTS OF COMPARATIVE ANALYSIS

After evaluating the three models, the GRU model stood out, achieving an  $R^2$  score of 0.99, which means it explained 99% of the variance in the target variable. Its low MSE, MAE, RMSE, and MAPE values further highlight its precision and reliability in predicting Forex price movements. The XGBoost model, after hyperparameter tuning, achieved an  $R^2$  score of 0.86. Although this is lower than the GRU model, it still demonstrates a strong predictive capability. However, its higher MSE and RMSE values indicate that its predictions have larger errors compared to the GRU model. The Random Forest model also delivered solid performance, with an  $R^2$  score of 0.89, showing it explained 89% of the variance in the target variable. Its error metrics (MSE, MAE, RMSE, MAPE) were lower than those of the XGBoost model but higher than those of the GRU model, placing it between the two in terms of overall performance.

The evaluation metrics for each model on the test set are summarized in the following table:

*Table 7: Model Performance Metrics Comparison*

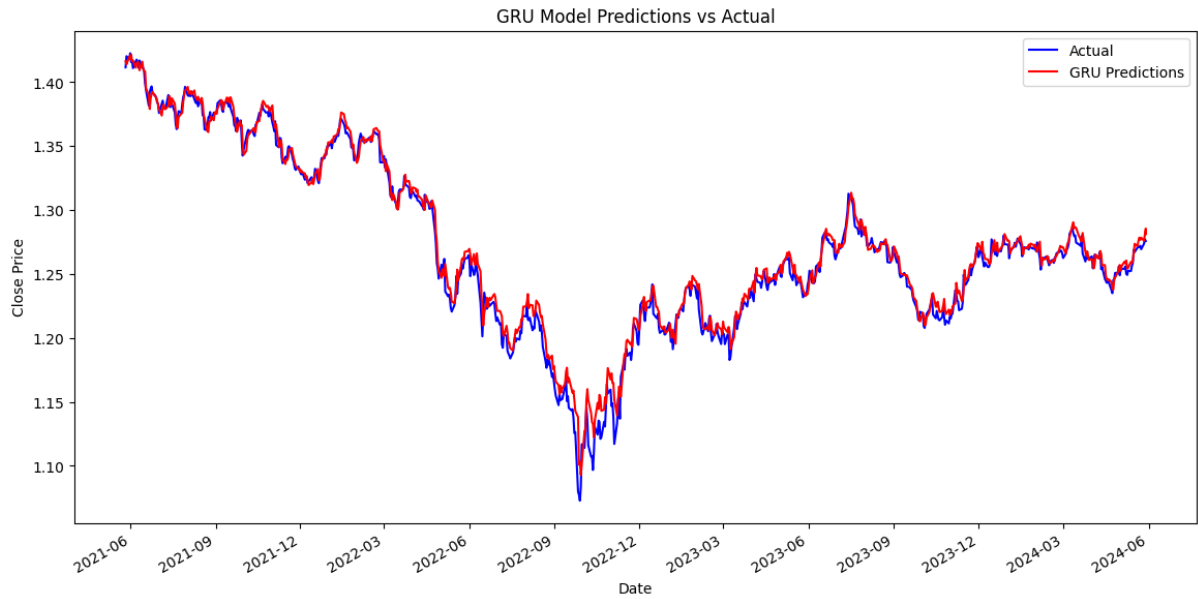
Model	MSE	MAE	RMSE	R2	MAPE
GRU	0.000052	0.005152	0.007224	0.99	0.004158
XGBoost	0.000672	0.012270	0.025920	0.86	0.010357
Random Forest	0.000548	0.011318	0.023404	0.89	0.009468

From the table, it is clear that the GRU model outperformed the others, with the lowest error metrics and the highest  $R^2$  score, indicating it has the highest predictive accuracy among the three models. The Random Forest model also performed well, while XGBoost showed the highest error rates, suggesting there is room for further improvement in its predictions.

### 4.1 Visual Comparison of Model Prediction

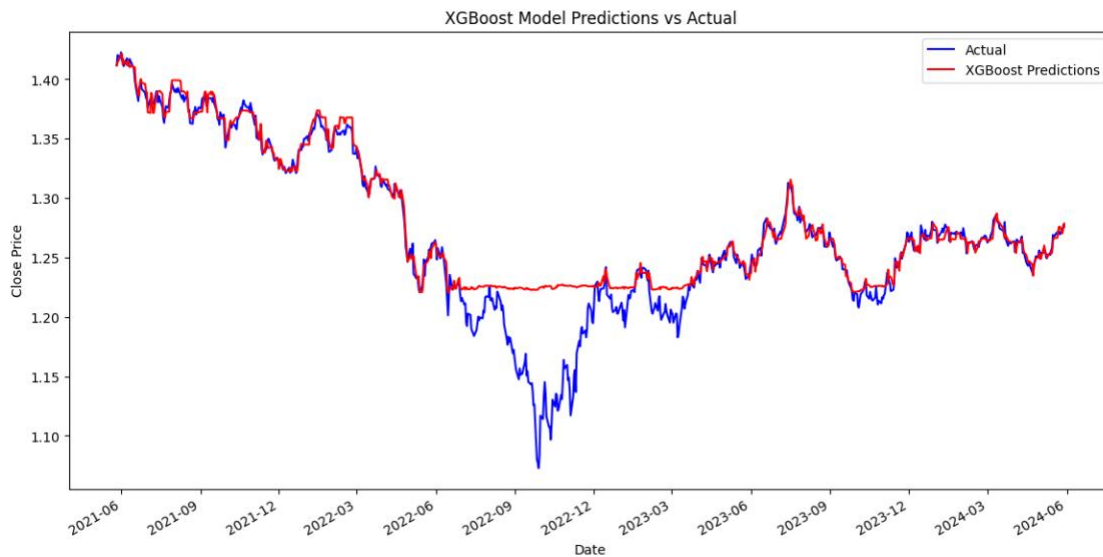
The GRU model predictions are very close to the actual price movements, with only small differences. This shows that the GRU model is good at capturing the trends and patterns in the data, which is why it has a high  $R^2$  score and low error metrics (see Fig 8).





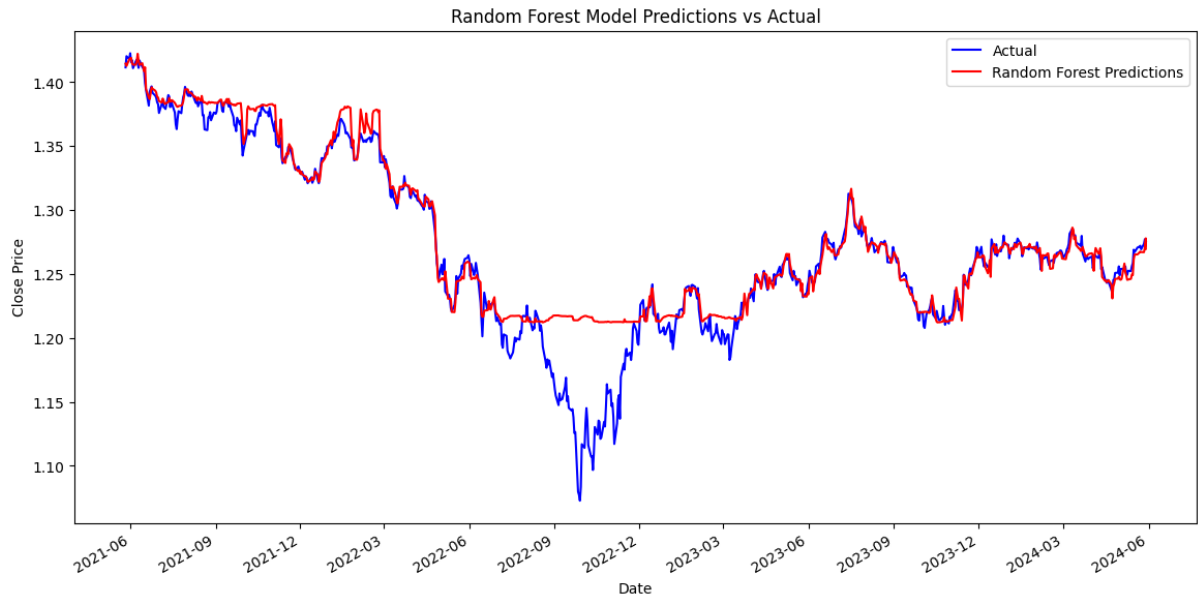
*Fig 8: GRU Model Predictions vs. Actual Closing Prices*

On the other hand, the XGBoost model predictions are not as close to the actual prices, especially when prices change quickly. This model struggles to accurately predict sudden market changes, particularly during the period from July 2022 to December 2022. This difficulty is reflected in its lower  $R^2$  score and higher error metrics, even after adjusting its parameters. The XGBoost model seems to have trouble with the more complex patterns in the data, especially during volatile periods (see Fig 9).



*Fig 9: XGBoost Model Predictions vs. Actual Closing Prices*

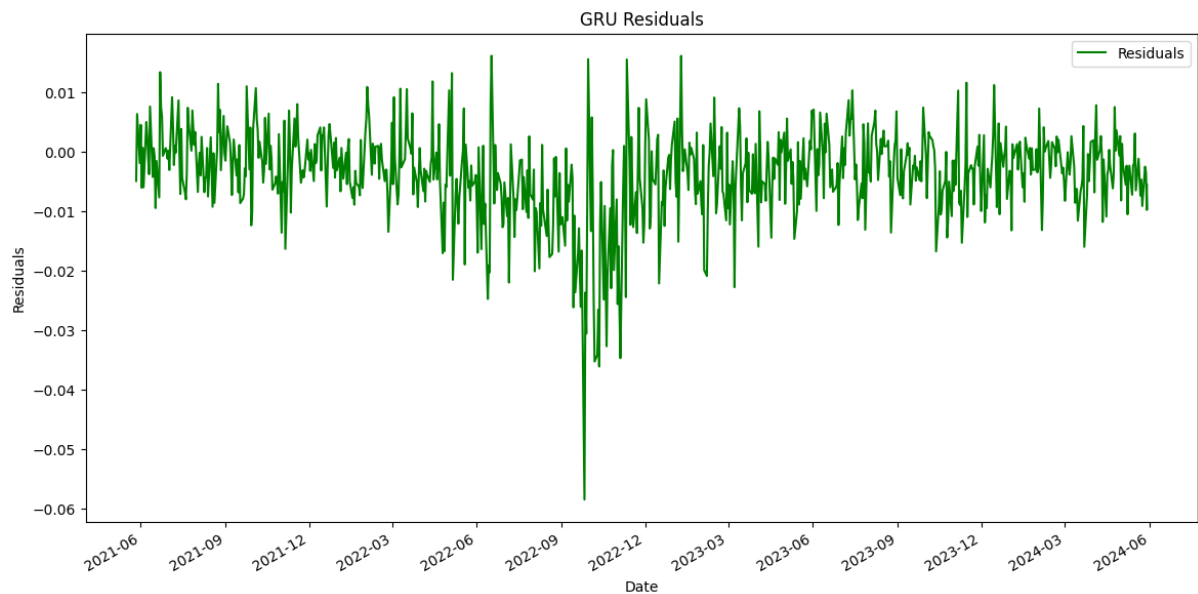
The Random Forest model performs better than XGBoost, but it still doesn't match the actual prices perfectly, especially during sudden drops or spikes, like those observed from July 2022 to December 2022. The model generally does well, as shown by its  $R^2$  score and error metrics, but it doesn't capture the details as well as the GRU model (see Fig 10).



*Fig 10: RF Model Predictions vs. Actual Closing Prices*

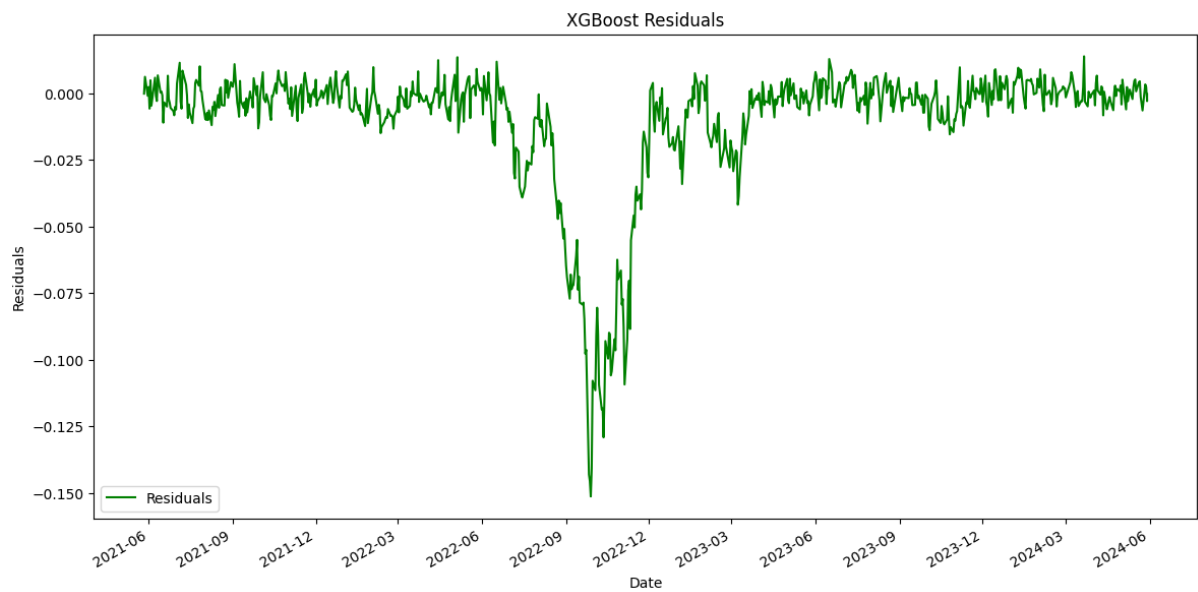
## 4.2 Residual Analysis

Looking at the differences between the true and predicted values (residual), it gives more insight into how accurate the models are. The GRU model's residuals are consistently small and close to zero, indicating that it has very few prediction errors.



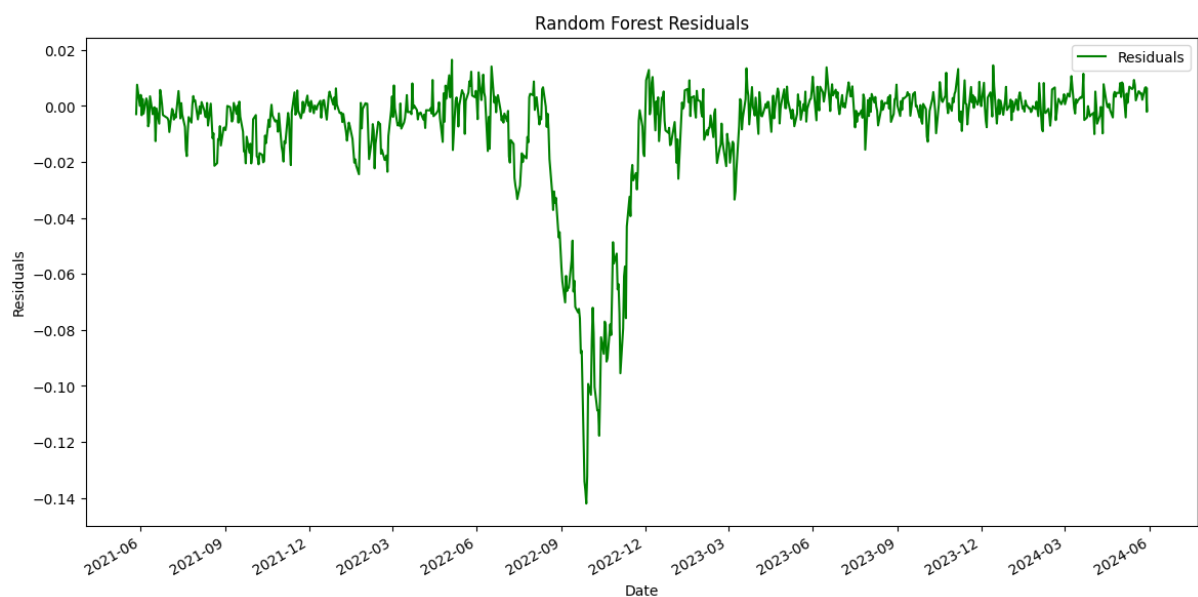
*Fig 11: GRU Residual*

Also, the XGBoost model's residuals show larger differences, especially during times of significant price movement, such as between July 2022 and December 2022. This suggests that the model sometimes struggles to predict rapid changes in the market.



*Fig 12: XGBoost Residual*

The Random Forest model's residuals are relatively small but still show some noticeable differences during periods of high volatility, including the period from July 2022 to December 2022. This indicates that while the model is generally accurate, it may have difficulty with sudden market shifts.



*Fig 13: RF Residual*

### **4.3 Insights**

The analysis shows that the GRU model performed better than both XGBoost and Random Forest model in predicting GBP/USD Forex price which explains its strong potential for accurate forecasting. Regardless of the optimization, XGBoost struggled with non-linear and volatile nature of the forex data, especially from July to December 2022. Random Forest captured general trends better than that of XGBoost but falls short of GRU's accuracy, particularly during rapid price changes in same period. GRU maintained consistent accuracy across the different time frames, while XGBoost and Random Forest had more significant errors during volatile periods.

### **4.4 Trading Strategy**

In developing the trading strategy, I created a straightforward rule-based system that leverages model predictions to generate trading signals. The strategy began by utilizing the predictions from the GRU, XGBoost, and Random Forest models to forecast closing prices, with each model trained on historical data and technical indicators to predict the next day's closing price. Trading signals were then generated by comparing the predicted prices to the actual prices. If the predicted price for the next day was higher than that of the current price, a 'Buy' signal will be generated; otherwise, a 'Sell' signal will be issued. This approach was based on the expectation of price movements to guide trading decisions.

Trades were executed based on these signals with an initial balance of \$10,000, assuming the buying or selling of one unit of the currency pair. To manage risk, I incorporated stop-loss and take-profit mechanisms, with stop-loss set at 0.1% below the entry price to limit losses, and take-profit set at 0.2% above the entry price to lock in profits when the market moved favourably.

#### **4.4.1 Simulation and Profitability Analysis**

Next the trading strategy was simulated over the test period using each model and an ensemble of the models. The profitability performance analysis included is the final balance, profit, Sharpe ratio, and maximum drawdown, which gives a comprehensive view of the trading strategy's accuracy and its risk-adjusted returns.

The final balance here also shows the total amount in the trading account at the end of the simulation and it's the addition of the initial balance with any profits or losses. A higher final balance indicates a better overall performance. The difference between the final and initial balances, reflects on the net earnings from trading activities. A higher profit indicates a more successful strategy.

The Sharpe ratio here evaluates the strategy's performance relative to a risk-free investment, adjusting for risk. A higher ratio indicates better risk-adjusted returns.

Maximum drawdown here represents the greatest decline from peak to trough in the account balance during the simulation, indicating the risk of significant losses. A lower maximum drawdown suggests better risk management.

See simulation result in the table below:

*Table 8: Trading Performance Metrics Across Different Models*

Model	Final Balance	Profit	Sharpe Ratio	Maximum Drawdown
GRU	\$11,530.51	\$1,530.51	0.09	0.20
XGBoost	\$16,552.92	\$6,552.92	0.11	0.16
Random Forest	\$15,965.55	\$5,965.55	0.10	0.14
Ensemble	\$17,715.74	\$7,715.74	0.13	0.15

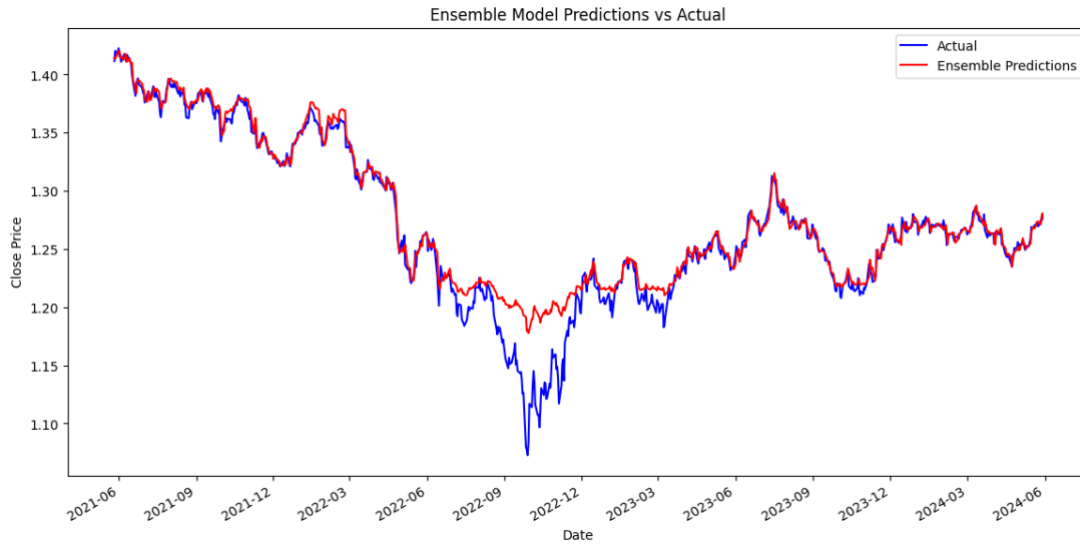
The results revealed that while the GRU model had excellent predictive accuracy, it struggled to generate profitable trading signals. XGBoost achieved the highest profit but also came with higher risk, as indicated by its error metrics. The RF model provided a balance, with respectable profit and strong risk management, as evidenced by its Sharpe ratio and lower maximum drawdown.

Despite the individual strengths of each model, the profitability analysis showed that the GRU model, despite its accuracy, yielded the lowest profit. XGBoost excelled in capturing profitable trades but at a higher risk, while the RF model delivered a good balance between profit and risk management.

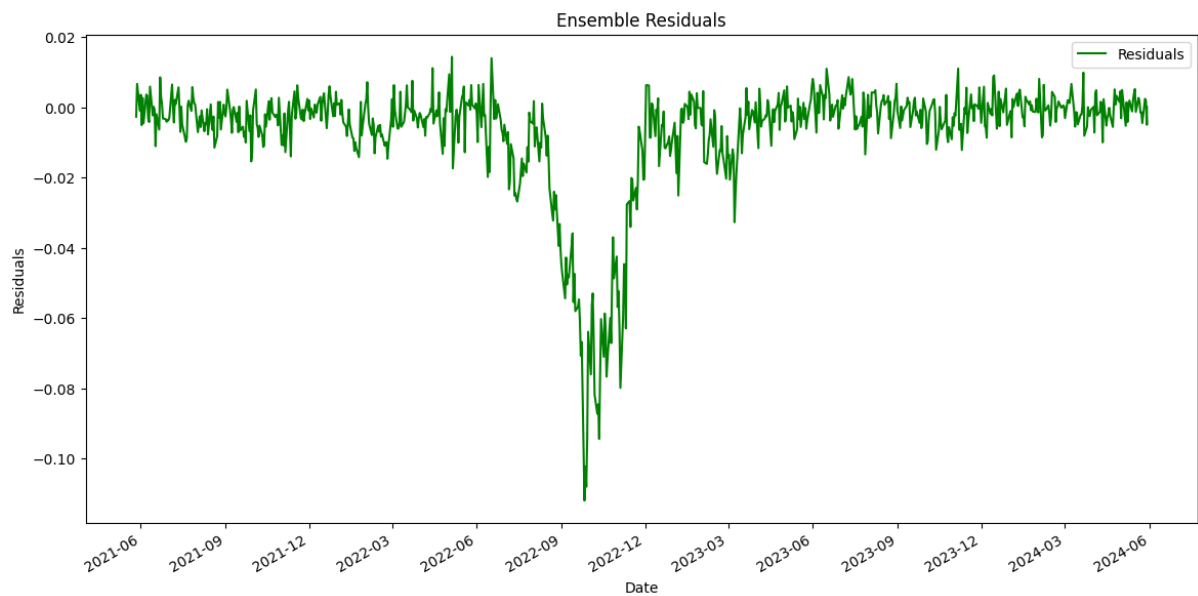
To leverage the strengths of all three models and achieve a more balanced performance, I implemented an ensemble technique. This involved taking the weighted average of the predictions from each model, effectively combining their unique advantages, and using the ensemble model to simulate trades. For the result see below:

*Table 9: Performance Metrics Comparison Across Different Models Including Ensemble*

Model	MSE	MAE	RMSE	R2	MAPE
GRU	0.000052	0.005152	0.007224	0.99	0.004158
XGBoost	0.000672	0.012270	0.025920	0.86	0.010357
Random Forest	0.000548	0.011318	0.023404	0.89	0.009468
Ensemble	0.000339	0.009242	0.018408	0.93	0.007743



*Fig 14: Ensemble Predictions vs. Actual Closing Prices*



*Fig 15: Ensemble Residual*

From the figures (Fig 14, Fig 15) above, the ensemble model's predictions closely aligned with the actual closing prices, demonstrating strong predictive capabilities. The residual, were generally centred around zero, indicating no systematic bias in the predictions. Although the model faced challenges during periods of high volatility, the overall stability of the residuals supported the ensemble model's effectiveness in managing risk and maintaining performance.

#### 4.5 Model Performance Comparison

In terms of error metrics, all models, including the GRU, XGBoost, and RF, showed very low Mean Squared Error (MSE), indicating high accuracy. The ensemble model maintained similarly low MSE, confirming the advantage of combining models. Mean Absolute Error (MAE) varied, increasing from GRU to XGBoost but decreasing with Random Forest, while the ensemble model showed a lower MAE, suggesting reduced average error magnitude. Root Mean Squared Error (RMSE) was highest for XGBoost, reflecting larger deviations, but the ensemble model showed a decrease in RMSE, indicating enhanced performance in minimizing large errors. Mean Absolute Percentage Error (MAPE) was lowest for GRU and increased through XGBoost to Random Forest, with the ensemble model lowering MAPE compared to Random Forest, indicating better percentage error management. The  $R^2$  value was highest for GRU, indicating it explained most of the variance in the target variable, while the ensemble model had a lower  $R^2$  value than GRU but higher than XGBoost and Random Forest, suggesting it balanced explanatory power across models (see Fig 16).

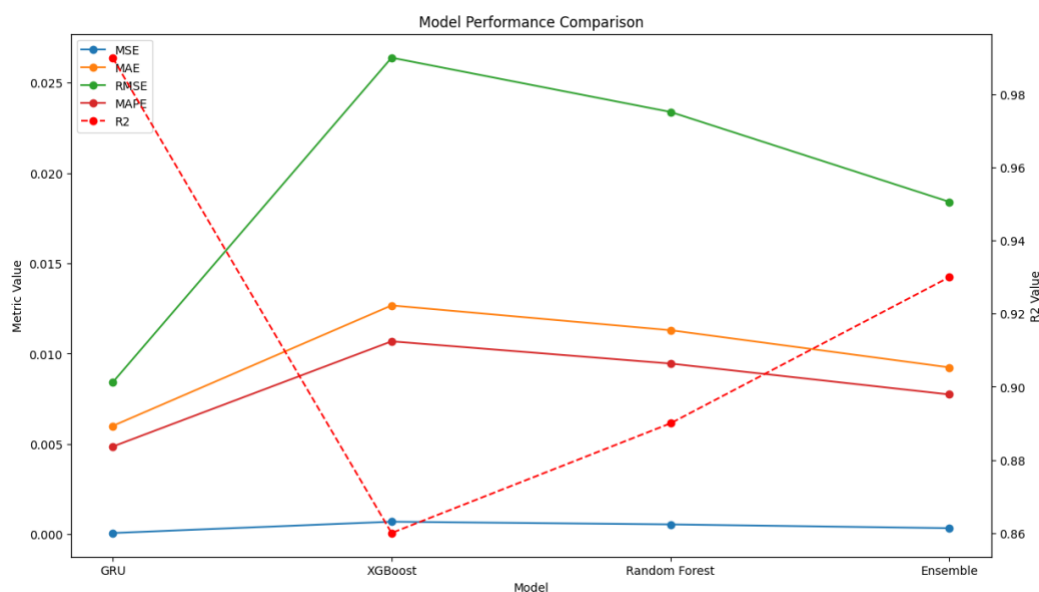
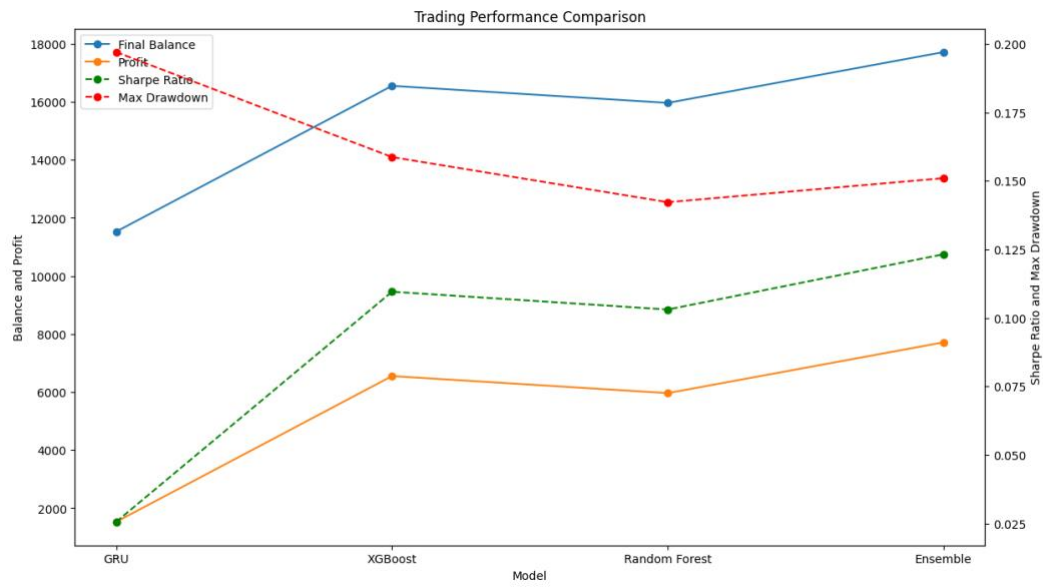


Fig 16: Comparative Analysis of Model Performance Metrics

#### 4.6 Trading Performance Comparison

The final balance increased from GRU to XGBoost and slightly decreased for Random Forest, with the ensemble model achieving the highest final balance, indicating superior profitability. Profit followed a similar trend, with the ensemble model maximizing profit, highlighting the advantage of combining models. The Sharpe ratio, representing risk-adjusted returns, improved from GRU to RF, with the ensemble model achieving the maximum Sharpe ratio, indicating it offered the optimum risk-adjusted performance. Maximum drawdown was highest for GRU, indicating greater potential losses, but decreased from XGBoost to RF, with the ensemble model showing a balanced drawdown, demonstrating its ability to manage risks effectively while maximizing returns (see Fig 17).



*Fig 17: Comparative Analysis of Trading Performance Metrics Across Models*



## CHAPTER 5: DISCUSSION

---

### **5.1 Interpretation of Results**

From the simulation results of the trading strategy, it is evident that the ensemble model significantly outperforms the individual models—GRU, XGBoost, and Random Forest—in terms of final balance and profit. Additionally, the ensemble model achieves an increased Sharpe ratio, demonstrating improved risk-adjusted returns. This suggests that combining the strengths of different models leads to a more robust and effective trading performance. In summary, the ensemble approach demonstrated the best results that yields the highest profit, and also maintained a strong Sharpe ratio, which shows its effectiveness in risk management. By putting together, the robustness of the RF model, the profitability of the XGBoost model, and the predictive accuracy of the GRU model, the ensemble approach gives a balanced and effective solution for trading strategy implementation.

### **5.2 Comparison with Existing Methods and Literature**

The result of this project is align with existing literature on the integration of machine learning models in Forex market prediction, particularly concerning the use of GRU, XGBoost, and Random Forest models. A comparison with prior studies highlights the advancements made by incorporating ensemble techniques.

### **5.3 Overview of Financial Trading Using AI**

A systematic review by Dakalbab et al. (2023) emphasized that reinforcement and deep learning are the most prevalent AI techniques in financial trading, applied in 30% and 29% of the reviewed papers, respectively. Their research highlighted the effectiveness of ensemble models in enhancing predictive accuracy, which supports the methodology used in this project as GRU resulted to higher accuracy.

#### **5.3.1 Random Forest**

Santuci et al. (2022) demonstrated that Random Forest models outperform traditional technical indicators in Forex trading, achieving higher Sharpe ratios and better risk-adjusted returns. This project's findings are in line with their results, as the Random Forest model provided strong risk management and achieved respectable profit.

#### **5.3.2 Gated Recurrent Units (GRU)**

Pahlevi et al. (2023) compared LSTM and GRU models for Forex price prediction, noting GRU's computational efficiency and high prediction accuracy. Their study reported an RMSE of 0.054 and an R-square of 97% for the GRU model. Similarly, this project found that while the GRU model demonstrated excellent predictive accuracy, it struggled to generate the highest profits independently, reinforcing the need to combine GRU with other models for better trading performance.

### **5.3.3 Extreme Gradient Boosting (XGBoost)**

Agung et al. (2022) employed XGBoost for Forex price prediction, achieving high accuracy through parameter optimization, with their model attaining an RMSE of 0.003098. This project's results align with these findings, as the XGBoost model achieved the highest profit, highlighting its strength in capturing profitable trades, albeit with relatively higher risk.

### **5.3.4 Ensemble Models**

The study by Upadhyay et al. (2021) showed that ensemble models combining CNN, LSTM, and SVR outperform individual models in predicting Forex rates, offering better generalization and robustness. This project's ensemble approach, which integrates GRU, XGBoost, and Random Forest, aligns with these findings. The ensemble model in this project achieved the highest overall performance, demonstrating superior profitability, a high Sharpe ratio, and effective risk management.

## **5.4 Practical Implications**

The project highlights several practical implications:

- **Improved Predictive Accuracy:** Ensemble methods enhance predictive accuracy by combining the strengths of multiple models, offering a more comprehensive understanding of market dynamics.
- **Reduction of Prediction Uncertainty:** Ensemble models reduce the uncertainty associated with predictions from individual models, resulting in more reliable and consistent forecasts.
- **Robustness and Generalization:** By addressing the weaknesses of individual models, ensemble methods provide better performance across different market conditions and datasets.

## **5.5 Limitations and Potential Improvements**

While the project has shown promising results, several limitations and potential improvements have been identified:

- **Overfitting:** Despite hyperparameter tuning, models like XGBoost and Random Forest exhibited signs of overfitting, consider more refined hyperparameter tuning, cross-validation techniques to mitigate overfitting.
- **Simple Trading Rules:** The trading strategy used was simple and might not capture the complexity of the market. Enhancing the strategy with more sophisticated approaches could improve performance
- **Market Factors:** The models did not consider broader market factors like economic indicators and geopolitical events, which have significant impacts on Forex prices. Integrating these factors could enhance predictive accuracy and trading performance

By addressing these limitations, the trading strategies could become more robust and reliable, leading to improved predictive accuracy and profitability.

## CHAPTER 6: CONCLUSION

---

### **6.1 Summary of Findings**

This project successfully demonstrated the potential of developing an automated trading bot for the GBP/USD Forex market using machine learning algorithms. Among the models tested, the ensemble model, which combined GRU, XGBoost, and Random Forest, stood out by achieving the highest final balance, profit, and Sharpe ratio. These findings highlight the effectiveness of ensemble methods in creating robust trading strategies that are capable of managing risk while maximizing returns.

### **6.2 Future Work and Directions**

To further improve the model's performance and resilience, several directions for future research have been identified:

- Integration of advanced trading strategy like momentum based trading or mean reversion to improve the signal accuracy
- Incorporating additional data like the economic indicators and news analysis to improve model accuracy
- Exploring and integrating more advanced risk management method like dynamic position sizing can contribute to better management risk

### **6.3 Final Thoughts**

This project has emphasized the value of ensemble techniques in creating effective and profitable trading strategies for the Forex market. By combining the strengths of GRU, XGBoost, and Random Forest models, the ensemble approach proved superior in predicting market movements and managing risk. Future research and enhancements can build on these findings, with the goal of developing even more sophisticated and efficient trading systems.

## References

A guide to the data protection principles (2023) ICO. Available at: <https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/data-protection-principles/a-guide-to-the-data-protection-principles/> (Accessed: 26 August 2024).

Abednego, L., Nugraheni, C.E., & Rinaldy, I., (2018). 'Forex trading robot with technical and fundamental analysis', *Journal of Computers*, 13(9), pp. 1089–1097. Available at: <https://doi.org/10.17706/jcp.13.9.1089-1097>

Agung, S.W., Rianto, K.S.W., & Wibowo, A., (2022). 'Comparative study of forecasting models for forex predictions with the impact of different currencies', *International Journal of Emerging Technology and Advanced Engineering*, 12(1), pp. 11–22. Available at: [https://doi.org/10.46338/ijetae0122\\_02](https://doi.org/10.46338/ijetae0122_02)

Archer, M.D., (2012). *Getting Started in Currency Trading, + Companion Website: Winning in Today's Market, 4th Edition*. Wiley. Available at: [https://learning.oreilly.com/library/view/getting-started-in/9781118281987/OEBPS/9781118281987\\_epub\\_ch\\_05.htm#c05-sec1-0001](https://learning.oreilly.com/library/view/getting-started-in/9781118281987/OEBPS/9781118281987_epub_ch_05.htm#c05-sec1-0001) (Accessed: 20 June 2024).

Ayitey Junior, M., Appiahene, P., Appiah, O. & Bombie, C.N., (2023). 'Forex market forecasting using machine learning: Systematic Literature Review and meta-analysis', *Journal of Big Data*, 10, 9. Available at: <https://doi.org/10.1186/s40537-022-00676-2>

Bank for International Settlements, (2019). Foreign exchange turnover in April 2019: Preliminary Global Result. Triennial Central Bank Survey, September 24. Available at: [https://www.bis.org/statistics/rpfx19\\_fx.pdf](https://www.bis.org/statistics/rpfx19_fx.pdf) (Accessed: 2 June 2024).

Breiman, L., (2001). Random Forests. *Machine Learning*, 45(1), pp.5-32. Available at: <https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf> (Accessed: 26 July 2024).

Chen, J. (2024) What is Ema? how to use exponential moving average with formula, Investopedia. Available at: <https://www.investopedia.com/terms/e/ema.asp> (Accessed: 19 August 2024).

Chen, T. and Guestrin, C. (2016) 'XGBoost: A Scalable Tree Boosting System', *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, 13-17 August, pp. 785-794. Available at: <https://doi.org/10.1145/2939672.2939785>

Cho, K., van Merriënboer, B., Bahdanau, D. and Bengio, Y., (2014). 'On the Properties of Neural Machine Translation: Encoder-Decoder Approaches', *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8)*. [pdf] Available at: <https://doi.org/10.48550/arXiv.1409.1259>

Cho, Kyunghyun; van Merriënboer, Bart; Bahdanau, DZmitry; Bougares, Fethi; Schwenk, Holger; Bengio, Yoshua (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". Association for Computational Linguistics. Available at: <https://doi.org/10.48550/arXiv.1406.1078>

Cohen, G., (2022). 'Algorithmic Trading and Financial Forecasting Using Advanced Artificial Intelligence Methodologies', *Mathematics*, 10(18), p.3302. Available at: <https://doi.org/10.3390/math10183302>

Dakalbab, F., Abu Talib, M., Nasir, Q., & Saroufil, T., (2024). 'Artificial intelligence techniques in financial trading: A systematic literature review', *Journal of King Saud University - Computer and Information Sciences*, 36(3), p.102015. Available at: <https://doi.org/10.1016/j.jksuci.2024.102015>

Fernando, J. (2024) Relative strength index (RSI) indicator explained with formula, Investopedia. Available at: <https://www.investopedia.com/terms/r/rsi.asp> (Accessed: 26 August 2024).

Gai, K., Qiu, M. and Sun, X. (2018) 'A survey on FinTech', *Journal of Network and Computer Applications*, 103, pp. 262-273. Available at: <https://doi.org/10.1016/j.inca.2017.10.011>

GeeksforGeeks (2023) Data Normalization Machine Learning, GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/what-is-data-normalization/> (Accessed: 26 August 2024).

Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R.P., Tang, J. and Liu, H., (2017). 'Feature Selection: A Data Perspective', *ACM Computing Surveys*, 50(6), pp.1-45. Available at: <https://doi-org.ezproxy.herts.ac.uk/10.1145/3136625>

Li, Y., Ni, P. and Chang, V. (2020) 'Application of deep reinforcement learning in stock trading strategies and stock forecasting', *Computing*, 102, pp. 1305-1322. Available at: <https://doi.org/10.1007/s00607-019-00773-w>

Loh, L.K.Y., Kueh, H.K., Parikh, N.J., Chan, H., Ho, N.J.H., & Chua, M.C. (2022) 'An Ensembling Architecture Incorporating Machine Learning Models and Genetic Algorithm Optimization for Forex Trading', *FinTech*, 1(2), pp. 100–124. Available at: <https://doi.org/10.3390/fintech1020008>

Mara (2022) Understanding fundamental analysis, Medium. Available at: <https://medium.com/@themaraverse/understanding-fundamental-analysis-112ab5c25b90> (Accessed: 2 August 2024).

Market expectations and forward premium: Forecasting currency movements (no date) FasterCapital. Available at: <https://fastercapital.com/content/Market-Expectations-and-Forward-Premium--Forecasting-Currency-Movements.html#Exploring-the-Relationship-Between-Market-Expectations-and-Currency-Movements.html> (Accessed: 27 August 2024).

Mesleh, M. and Kiranyaz, M. (2021) 'Short-term financial data prediction by long-term regression', *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)* [Preprint]. pp. 1304-1309 Available at: <https://doi.org/10.1109/hpcc-dss-smartcity-dependsys53884.2021.00199>

Nokeri, T.C., (2021). 'Introduction to Financial Markets and Algorithmic Trading', In: *Implementing Machine Learning for Finance*. Apress, Berkeley, CA. Available at: [https://doi.org/10.1007/978-1-4842-7110-0\\_1](https://doi.org/10.1007/978-1-4842-7110-0_1)

Pahlevi, M.R., Kusrini, and Hidayat, T., (2023). 'Comparison of LSTM and GRU Models for Forex Prediction', *Sinkron: Jurnal dan Penelitian Teknik Informatika*, 7(4), pp.2254. Available at: <https://doi.org/10.33395/sinkron.v8i4.12709>

Research (2018) Uni of Herts. Available at: <https://www.herts.ac.uk/about-us/our-leadership-strategy-and-plans/our-governance-and-leadership/university-policies-and-regulations-uprs/uprs/research> (Accessed: 26 August 2024).

Santuci, A., Sbruzzi, E., Araújo-Filho, L. and Leles, M. (2022) 'Evaluation of FOREX trading Strategies based in Random Forest and Support Vector Machines', *IEEE Latin America Transactions*, vol. 20, no. 9, pp. 2146-2152. Available at: <https://doi.org/10.1109/tla.2022.9878170>

Ta, V-D., Liu, C-M. and Addis, D. (2018) 'Prediction and portfolio optimization in quantitative trading using machine learning techniques', in *Proceedings of the 9th International Symposium on Information and Communication Technology (SolICT '18)*, 6 December, pp. 98-105. Available at: <https://doi.org/10.1145/3287921.3287963>

TradingView (2024) MACD (moving average convergence/divergence), TradingView. Available at: <https://www.tradingview.com/support/solutions/43000502344-macd-moving-average-convergence-divergence/> (Accessed: 26 August 2024).

Upadhyay, S.K., Gajbhiye, S., & Kumar, H., (2021) 'Ensemble Method for Forex Rate Prediction using OHLC Data', *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, 9(VI), pp.[pages]. Available at: <https://doi.org/10.22214/ijraset.2021.35328>

Wang, J. and Kim, J., (2018) 'Predicting Stock Price Trend Using MACD Optimized by Historical Volatility', *Mathematical Problems in Engineering*, [online] Available at: <https://doi.org/10.1155/2018/9280590>

Yao, J. and Tan, C.L., (2000) 'A case study on using neural networks to perform technical forecasting of forex', *Neurocomputing*, 34(1-4), pp.79-98. Available at: [https://doi.org/10.1016/S0925-2312\(00\)00300-3](https://doi.org/10.1016/S0925-2312(00)00300-3)

# APPENDICES

## APPENDIX A

GitHub link [here](#)

```
# -*- coding: utf-8 -*-  
"""Cynthia_project_code.ipynb
```

Automatically generated by Colab.

Original file is located at

<https://colab.research.google.com/drive/1aQhvpi0jbLpxSCNc4WEXRnjXHIAZg0IQ>

### Note that the prefix "cu" represents the initials for Cynthia Udoe and will be used for the variable names

# \*\*Project Topic: Development of an Automated GBP/USD Forex Trading Bot Using ML Algorithms (Phase 1)\*\*

**\*\*Research Question\*\*:** How can machine learning algorithms (GRU, XGBoost, and Random Forest) be compared to develop the most profitable automated trading bot for the GBP/USD Forex market?

### Objective 1: Collect, preprocess historical forex data for GBP/USD and carry out EDA to understand & identify patterns  
"""

# Installing TensorFlow library, which is used for machine learning and neural network tasks  
!pip install tensorflow

```
# Importing necessary libraries  
from google.colab import drive  
import os  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import tensorflow as tf  
import sklearn  
from sklearn.model_selection import train_test_split  
from sklearn.preprocessing import StandardScaler  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import GRU, Dense, Dropout, Input  
from tensorflow.keras.optimizers import Adam  
from tensorflow.keras.callbacks import EarlyStopping
```

```

import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
import random
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score,
mean_absolute_percentage_error
from tensorflow.keras.models import save_model, load_model
import joblib
from joblib import dump, load
import matplotlib.dates as mdates
print("TensorFlow version:", tf.__version__)
print("XGBoost version:", xgb.__version__)
print("Scikit-learn version:", sklearn.__version__)
print("Joblib version:", joblib.__version__)

# Ensuring Google Drive is mounted
drive.mount('/content/drive', force_remount=True)

# Navigating to the dataset folder in Google Drive
os.chdir('/content/drive/My Drive/Ms.c Data Science project')

# Reading the data into the dataframe from CSV
cu_fx_df = pd.read_csv('GBPUSD_HistoricalData.csv')

# Displaying the first few rows
cu_fx_df.head()

# Displaying the last fews rows
cu_fx_df.tail()

# Retrieving the count of rows and columns
cu_rows, cu_columns = cu_fx_df.shape
cu_rows, cu_columns

# Generating summary statistics for the dataset
cu_summary_stats = cu_fx_df.describe()
cu_summary_stats

# Getting an overview of the dataframe structure, data types, and detect missing values
cu_fx_df.info()

# Displaying the total count of NaN values for each column
cu_fx_df.isnull().sum()

# Filtered out the rows that contain null values
cu_null_rows = cu_fx_df[cu_fx_df.isnull().any(axis=1)]
cu_null_rows

```



```

# Removed rows with the NaN values, 'Volume', and 'Adj Close' columns and reconfirmed
numbers of rows and columns in the dataset
cu_fx_df_cleaned = cu_fx_df.dropna()
cu_fx_df_cleaned = cu_fx_df_cleaned.drop(columns=['Volume', 'Adj Close'])

cu_fx_df_cleaned.shape

# Changing the 'date' Dtype
cu_fx_df_cleaned['Date'] = pd.to_datetime(cu_fx_df_cleaned['Date'])

# Displaying an overview of the cleaned dataframe structure and data types
cu_fx_df_cleaned.info()

# Displayed the first few rows of the cleaned dataframe
cu_fx_df_cleaned.head()

# creating figure & grid of subplots
fig, axes = plt.subplots(nrows = 4, ncols = 2, figsize = (12,10))

# Defined the columns for histogram plot
numerical_columns = ['Open', 'High', 'Low', 'Close']

# Flattened the axes array for easy iteration
cu_axes = axes.flatten()

# Looping through the numerical columns, creating a histogram & boxplot for each
for i, col in enumerate(numerical_columns):
    sns.histplot(cu_fx_df_cleaned[col], kde = True, color = 'green', bins = 10, ax = cu_axes[2*i])
    cu_axes[2*i].set_xlabel(col.capitalize())
    cu_axes[2*i].set_ylabel('Frequency')
    cu_axes[2*i].set_title(f"Histogram for {col.capitalize()} price")

    sns.boxplot(x = cu_fx_df_cleaned[col], ax = cu_axes[2*i + 1], color = 'blue')
    cu_axes[2*i + 1].set_xlabel(col.capitalize())
    cu_axes[2*i + 1].set_title(f"Boxplot for {col.capitalize()} price")

# Adjusted layout for better spacing between plots
plt.tight_layout()

# Added main title for the entire figure
plt.suptitle("Histograms and Boxplots for Each Column Price", fontsize = 16, y = 1.02)

# Displaying the figure
plt.show()

# Defined function to drop outliers by means of IQR (Inter Qurtile Range)
def cu_remove_outliers_iqr(cu_fx_df_cleaned, columns):

```

```
"""
```

This function removes outliers from specified columns in the dataframe by means of IQR.

Parameters:

cu\_fx\_df\_cleaned: The dataframe containing the data.

columns: columns to remove outliers from.

Returns:

cu\_fx\_df\_cleaned2: A new dataframe without outliers.

```
"""
```

```
for column in columns:
```

```
    cu_Q1 = cu_fx_df_cleaned[column].quantile(0.25)
```

```
    cu_Q3 = cu_fx_df_cleaned[column].quantile(0.75)
```

```
    cu_IQR = cu_Q3 - cu_Q1
```

```
    cu_lower_bound = cu_Q1 - 1.5 * cu_IQR
```

```
    cu_upper_bound = cu_Q3 + 1.5 * cu_IQR
```

```
    cu_fx_df_cleaned = cu_fx_df_cleaned[(cu_fx_df_cleaned[column] >= cu_lower_bound) &
(cu_fx_df_cleaned[column] <= cu_upper_bound)]
```

```
return cu_fx_df_cleaned
```

```
# Applied the function to the price columns
```

```
cu_price_columns = ['Open', 'High', 'Low', 'Close']
```

```
cu_fx_df_cleaned2 = cu_remove_outliers_iqr(cu_fx_df_cleaned, cu_price_columns)
```

```
# Visualized the data after removing outliers
```

```
plt.figure(figsize = (10, 5))
```

```
sns.boxplot(data = cu_fx_df_cleaned2[cu_price_columns])
```

```
plt.title('Boxplots after removing outliers')
```

```
plt.show()
```

```
# Retrieved the count of rows and columns in the new dataframe after removing outliers
```

```
cu_rows2, cu_columns2 = cu_fx_df_cleaned2.shape
```

```
cu_rows2, cu_columns2
```

```
# Generated the summary statistic for the new GBPUSD data
```

```
cu_summary_stats2 = cu_fx_df_cleaned2.describe()
```

```
cu_summary_stats2
```

```
"""# **Technical Indicators**"""
```

```
# Calculates the Exponential Moving Average (EMAs) with a customizable period
```

```
def Calculate_EMAs(cu_fx_df_cleaned2, cu_period):
```

```
    """
```

This function calculates the Exponential Moving Average (EMAs) with a customizable period.

Parameters:

cu\_fx\_df\_cleaned2: The dataframe.

cu\_period: The period count to be used for calculating the EMA.

Returns:

pd.Series: A pandas Series holding the EMAs value.

```
"""
```

```
return cu_fx_df_cleaned2.ewm(span = cu_period, adjust = False).mean()
```

```
# Calculate 20-day and 50-day EMAs
```

```
cu_fx_df_cleaned2['EMA_20'] = Calculate_EMAs(cu_fx_df_cleaned2['Close'], cu_period = 20)
```

```
cu_fx_df_cleaned2['EMA_50'] = Calculate_EMAs(cu_fx_df_cleaned2['Close'], cu_period = 50)
```

```
# Calculates the Relative Strength Index (RSI) with a customizable period
```

```
def Calculate_RSI(cu_fx_df_cleaned2, cu_period):
```

```
    """
```

```
    This function calculates the Relative Strength Index (RSI) with a customizable period.
```

Parameters:

cu\_fx\_df\_cleaned2: The dataframe.

cu\_period: The period count to be used for calculating the RSI.

Returns:

pd.Series: A pandas Series holding the RSI value.

```
"""
```

```
# Calculates the change in price
```

```
delta = cu_fx_df_cleaned2['Close'].diff(1)
```

```
# Separates the gains and losses
```

```
cu_gain = delta.where(delta > 0, 0)
```

```
cu_loss = -delta.where(delta < 0, 0)
```

```
# Calculates the rolling average gain and loss
```

```
cu_avg_gain = cu_gain.rolling(window = cu_period, min_periods = 1).mean()
```

```
cu_avg_loss = cu_loss.rolling(window = cu_period, min_periods = 1).mean()
```

```
# Calculates the Relative Strength (RS)
```

```
cu_rs = cu_avg_gain / cu_avg_loss
```

```
# Calculates the Relative Strength Index (RSI)
```

```
cu_rsi = 100 - (100 / (1 + cu_rs))
```

```
return cu_rsi
```

```

# Calculate RSI with a period of 14
cu_fx_df_cleaned2['RSI'] = Calculate_RSI(cu_fx_df_cleaned2, cu_period = 14)

# Handling the NaNs in the first row
cu_fx_df_cleaned2['RSI'].fillna(0, inplace = True)

# Calculates the MACD Line, Signal Line, and MACD Histogram
def Calculate_MACD(cu_fx_df_cleaned2, cu_short_window = 12, cu_long_window = 26,
cu_signal_window = 9):
    """
    This function calculates the MACD Line, Signal Line, and MACD Histogram.

    Parameters:
    cu_fx_df_cleaned2: The dataframe.
    cu_short_window: The period count to be used for calculating short-term EMA. Default is
    12.
    cu_long_window (int): The period count to be used for calculating long-term EMA. Default
    is 26.
    cu_signal_window (int): The period count to be used for calculating Signal line EMA.
    Default is 9.

    Returns:
    pd.DataFrame: A DataFrame holding the MACD line, Signal line, and MACD histogram.
    """

    cu_short_ema = cu_fx_df_cleaned2['Close'].ewm(span = cu_short_window, adjust =
False).mean()
    cu_long_ema = cu_fx_df_cleaned2['Close'].ewm(span = cu_long_window, adjust =
False).mean()
    cu_macd_line = cu_short_ema - cu_long_ema
    cu_signal_line = cu_macd_line.ewm(span = cu_signal_window, adjust = False).mean()
    cu_macd_histogram = cu_macd_line - cu_signal_line

    return pd.DataFrame({
        'MACD Line': cu_macd_line,
        'Signal Line': cu_signal_line,
        'MACD Histogram': cu_macd_histogram
    })

# Calculate MACD Line, Signal Line, and MACD Histogram
cu_macd = Calculate_MACD(cu_fx_df_cleaned2)
cu_fx_df_cleaned2 = cu_fx_df_cleaned2.join(cu_macd)

# Displaying the processed data
cu_fx_df_cleaned2.head()

```

```

# Customising the filtering option of dataset from the dataframe for visualization
cu_start_date = '2020-11-01'
cu_end_date = '2024-05-29'
cu_filtered_data = cu_fx_df_cleaned2[(cu_fx_df_cleaned2['Date'] >= cu_start_date) &
(cu_fx_df_cleaned2['Date'] <= cu_end_date)]

# Displayed the filtered data
cu_filtered_data.head()

# Plots the close price, EMAs, RSI, & MACD
plt.figure(figsize = (18, 15))

# Close Price & EMAs
plt.subplot(3, 1, 1)
plt.plot(cu_filtered_data['Date'], cu_filtered_data['Close'], label = 'Close Price', color = 'blue')
plt.plot(cu_filtered_data['Date'], cu_filtered_data['EMA_20'], label = '20-day EMA', color =
'red')
plt.plot(cu_filtered_data['Date'], cu_filtered_data['EMA_50'], label = '50-day EMA', color =
'green')
plt.title('Close Price & Exponential Moving Averages (EMAs)')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()

# RSI
plt.subplot(3, 1, 2)
plt.plot(cu_filtered_data['Date'], cu_filtered_data['RSI'], label = '14-day RSI', color = 'green')
plt.axhline(y = 30, color = 'r', linestyle = '--', label = 'Overbought')
plt.axhline(y = 70, color = 'r', linestyle = '--', label = 'Oversold')
plt.title('Relative Strength Index (RSI)')
plt.xlabel('Date')
plt.ylabel('RSI')
plt.legend()

# MACD
plt.subplot(3, 1, 3)
plt.plot(cu_filtered_data['Date'], cu_filtered_data['MACD Line'], label = 'MACD Line', color =
'blue')
plt.plot(cu_filtered_data['Date'], cu_filtered_data['Signal Line'], label = 'Signal Line', color =
'red')
plt.bar(cu_filtered_data['Date'], cu_filtered_data['MACD Histogram'], label = 'MACD
Histogram', color = 'green', alpha = 0.3)
plt.title('Moving Average Convergence Divergence (MACD)')
plt.xlabel('Date')
plt.ylabel('MACD')
plt.legend()

```

```

# Displaying the plots
plt.tight_layout()
plt.show()

# Creates a 3x2 grid of subplots
fig, cu_axes = plt.subplots(nrows = 3, ncols = 2, figsize = (15, 10))

# Distribution of Close Prices
sns.histplot(cu_fx_df_cleaned2['Close'], kde = True, bins = 50, ax = cu_axes[0, 0])
cu_axes[0, 0].set_title('Distribution of Close Prices')
cu_axes[0, 0].set_xlabel('Close Price')
cu_axes[0, 0].set_ylabel('Frequency')

# Distribution of EMAs
sns.histplot(cu_fx_df_cleaned2['EMA_20'], kde = True, color = 'blue', bins = 50, ax =
cu_axes[0, 1])
cu_axes[0, 1].set_title('Distribution of 20-day EMA')
cu_axes[0, 1].set_xlabel('20-day EMA')
cu_axes[0, 1].set_ylabel('Frequency')

sns.histplot(cu_fx_df_cleaned2['EMA_50'], kde = True, color = 'purple', bins = 50, ax =
cu_axes[1, 0])
cu_axes[1, 0].set_title('Distribution of 50-day EMA')
cu_axes[1, 0].set_xlabel('50-day EMA')
cu_axes[1, 0].set_ylabel('Frequency')

# Distribution of RSI
sns.histplot(cu_fx_df_cleaned2['RSI'], kde = True, color = 'green', bins = 50, ax = cu_axes[1,
1])
cu_axes[1, 1].set_title('Distribution of RSI')
cu_axes[1, 1].set_xlabel('RSI')
cu_axes[1, 1].set_ylabel('Frequency')

# Distribution of MACD Histogram
sns.histplot(cu_fx_df_cleaned2['MACD Histogram'], kde = True, color = 'grey', bins = 50, ax =
cu_axes[2, 0])
cu_axes[2, 0].set_title('Distribution of MACD Histogram')
cu_axes[2, 0].set_xlabel('MACD Histogram')
cu_axes[2, 0].set_ylabel('Frequency')

# Removes the empty subplot
fig.delaxes(cu_axes[2, 1])

# Displaying the plots
plt.tight_layout()
plt.show()

```

```

# Correlation of matrix for key columns
cu_correlation = cu_fx_df_cleaned2[['Open', 'High', 'Low', 'Close', 'EMA_20', 'EMA_50', 'RSI',
'MACD Line', 'Signal Line', 'MACD Histogram']].corr()
cu_correlation

# Correlation matrix plot
plt.figure(figsize = (12, 8))
sns.heatmap(cu_correlation, annot = True, cmap = 'coolwarm', fmt = '.2f')
plt.title('Correlation Heatmap')
plt.show()

# Generated a pairplot for better understanding of the relation
fig = plt.figure(figsize = (8, 5))
g = sns.pairplot(cu_fx_df_cleaned2)
g.fig.suptitle('Pairplot for GBPUSD Data', y = 1.02)
plt.show()

# descriptive Statistics
cu_fx_df_cleaned2.describe()

"""### Objective 2. Developing and evaluating the accuracy of these models (GRU, XGBoost,
& Random Forest) in predicting forex price movements"""

# Displayed the dataframe
cu_fx_df_cleaned2

"""**1. Introduction and Data Preparation for the models**"""

# Ensuring the necessary columns for modelling are available
cu_required_columns = ['Date', 'Open', 'High', 'Low', 'Close', 'EMA_20', 'EMA_50', 'RSI',
'MACD Line', 'Signal Line', 'MACD Histogram']
cu_fx_df_cleaned2 = cu_fx_df_cleaned2[cu_required_columns]

# Selecting the features and target variable
X = cu_fx_df_cleaned2.drop(columns = ['Date', 'Close'])
y = cu_fx_df_cleaned2['Close'].values
dates = cu_fx_df_cleaned2['Date'].values

# First split: Train (70%) and temp (30%)
X_train, X_temp, y_train, y_temp, dates_train, dates_temp = train_test_split(X, y, dates,
test_size = 0.3, shuffle=False)

# Second split: Test (50%) and validation (50%)
X_val, X_test, y_val, y_test, dates_val, dates_test = train_test_split(X_temp, y_temp,
dates_temp, test_size = 0.5, shuffle=False)

# Displaying the shape of the splits data

```

```

print(f"Training set: {X_train.shape}, {y_train.shape}")
print(f"Validation set: {X_val.shape}, {y_val.shape}")
print(f"Test set: {X_test.shape}, {y_test.shape}")

# Prints the start date and end date for each split
print(f"Training set: from {dates_train[0]} to {dates_train[-1]}")
print(f"Validation set: from {dates_val[0]} to {dates_val[-1]}")
print(f"Test set: from {dates_test[0]} to {dates_test[-1]}")

"""**2. Feature Scaling and Sequence Creation**"""

# Initializing the StandardScaler
scaler_X = StandardScaler()
scaler_y = StandardScaler()

# Fitting the scaler on the training data and transforming both training, validation, testing
data for X features
X_train_scaled = scaler_X.fit_transform(X_train)
X_val_scaled = scaler_X.transform(X_val)
X_test_scaled = scaler_X.transform(X_test)

# Scaling the target variable
y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1, 1))
y_val_scaled = scaler_y.transform(y_val.reshape(-1, 1))
y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1))

def create_sequences(X, y, dates, time_steps=15):
    """
    This functions takes in the feature matrix (X), target variable (y), and dates, and split them
    into sequences of a specified length (time_steps). The sequences
    are returned as numpy arrays.

    Parameters:
    X (numpy.ndarray): The feature matrix of shape (n_samples, n_features).
    y (numpy.ndarray): The target variable of shape (n_samples,).
    dates (numpy.ndarray): The array of dates corresponding to the samples.
    time_steps (int): The length of the sequencys to be created. Default is 15.

    Returns:
    tuple: Three numpy arrays: X_sequences, y_sequences, and dates_sequences. The shapes
    of these arrays are
        (n_sequences, time_steps, n_features), (n_sequences,), and (n_sequences,),
    respectively.
    """
    Xs, ys, ds = [], [], []
    for i in range(len(X) - time_steps + 1): # Adjusted the range to include the last date

```



```

        Xs.append(X[i:i+time_steps])
        ys.append(y[i+i+time_steps] if i+i+time_steps < len(y) else y[-1]) # Added condition to
include last y value
        ds.append(dates[i+i+time_steps] if i+i+time_steps < len(dates) else dates[-1]) # Added
condition to include last date
    return np.array(Xs), np.array(ys), np.array(ds)

# Defining the time step
time_steps = 15

# Creating sequences for training, validation, and testing
X_train_seq, y_train_seq, dates_train_seq = create_sequences(X_train_scaled,
y_train_scaled, dates_train, time_steps)
X_val_seq, y_val_seq, dates_val_seq = create_sequences(X_val_scaled, y_val_scaled,
dates_val, time_steps)
X_test_seq, y_test_seq, dates_test_seq = create_sequences(X_test_scaled, y_test_scaled,
dates_test, time_steps)

# Reshape X sequences to 2D for XGBoost and Random Forest
X_train_seq_flat = X_train_seq.reshape(X_train_seq.shape[0], -1)
X_val_seq_flat = X_val_seq.reshape(X_val_seq.shape[0], -1)
X_test_seq_flat = X_test_seq.reshape(X_test_seq.shape[0], -1)

# Flatten y sequences
y_train_seq_flat = y_train_seq.flatten()
y_val_seq_flat = y_val_seq.flatten()
y_test_seq_flat = y_test_seq.flatten()

# Print the start and end dates for the sequences
print(f"Training sequences: from {dates_train[0]} to {dates_train[-1]}")
print(f"Validation sequences: from {dates_val[0]} to {dates_val[-1]}")
print(f"Test sequences: from {dates_test[0]} to {dates_test[-1]}")

# Displaying the shape for the GRU dataset
print(f"GRU Training set: {X_train_seq.shape}, {y_train_seq.shape}")
print(f"GRU Validation set: {X_val_seq.shape}, {y_val_seq.shape}")
print(f"GRU Test set: {X_test_seq.shape}, {y_test_seq.shape}")

# Displaying the shape for the XGBoost & Random Forest dataset
print(f"Training set: {X_train_seq_flat.shape}, {y_train_seq_flat.shape}")
print(f"Validation set: {X_val_seq_flat.shape}, {y_val_seq_flat.shape}")
print(f"Test set: {X_test_seq_flat.shape}, {y_test_seq_flat.shape}")

# Save the scalers
dump(scaler_X, 'scaler_X.bin')
dump(scaler_y, 'scaler_y.bin')

```

### \*\*\*3. Developing and Training the GRU Model\*\*\*

# Function to develop the GRU model

```
def create_gru_model(units = 50, dropout_rate = 0.2, learning_rate = 0.001):
```

```
    """
```

Creates and compiles a GRU (Gated Recurrent Unit) neural network model for time-series forecasting.

Parameters:

-----

units : int, optional

The number of GRU units in each layer (default is 50).

dropout\_rate : float, optional

The dropout rate for regularization, representing the fraction of input units to drop (default is 0.2).

learning\_rate : float, optional

The learning rate for the Adam optimizer (default is 0.001).

Returns:

-----

model : keras.Sequential

A compiled GRU model ready for training.

Model Architecture:

-----

- Input layer with shape `(time\_steps, features)`, where `time\_steps` is the length of the input sequence

and `features` is the number of features per time step.

- First GRU layer with `units` units and `return\_sequences=True`.

- Dropout layer with dropout rate equal to `dropout\_rate`.

- Second GRU layer with `units` units.

- Dropout layer with dropout rate equal to `dropout\_rate`.

- Dense output layer with a single unit (used for regression tasks).

```
    """
```

```
model = Sequential()
```

```
model.add(Input(shape = (time_steps, X_train_seq.shape[2])))
```

```
model.add(GRU(units, return_sequences = True))
```

```
model.add(Dropout(dropout_rate))
```

```
model.add(GRU(units))
```

```
model.add(Dropout(dropout_rate))
```

```
model.add(Dense(1))
```

```
optimizer = Adam(learning_rate = learning_rate)
```

```
model.compile(optimizer = optimizer, loss = 'mean_squared_error')
```

```
return model
```

```
# Training the GRU model
gru_model = create_gru_model(units = 50, dropout_rate = 0.2, learning_rate = 0.001)
early_stopping = EarlyStopping(monitor = 'val_loss', patience = 10, restore_best_weights =
True)
gru_history = gru_model.fit(X_train_seq, y_train_seq, epochs = 50, batch_size = 32,
validation_data = (X_val_seq, y_val_seq), callbacks =[early_stopping], verbose = 1)
```

\*\*\*\*\*4. Evaluation of GRU model on the training and testing set\*\*\*\*\*

```
# Making predictions on the training set
gru_train_predictions = gru_model.predict(X_train_seq)
gru_train_predictions = scaler_y.inverse_transform(gru_train_predictions)
y_train_seq_inverse_gru = scaler_y.inverse_transform(y_train_seq)
```

```
# Calculates R2 score for the training set
r2_gru_train = r2_score(y_train_seq_inverse_gru, gru_train_predictions)
print(f'R2 score for GRU on training set: {r2_gru_train:.2f}')
```

```
# Making predictions on the test set
gru_test_predictions = gru_model.predict(X_test_seq)
gru_test_predictions = scaler_y.inverse_transform(gru_test_predictions)
y_test_seq_inverse_gru = scaler_y.inverse_transform(y_test_seq)
```

```
# Calculates R2 score for the test set
r2_gru_test = r2_score(y_test_seq_inverse_gru, gru_test_predictions)
print(f'R2 score for GRU on test set: {r2_gru_test:.2f}')
```

```
# Saved the GRU model
cu_best_gru_model = gru_model
cu_gru_model_path = 'cu_best_gru_model.keras'
cu_best_gru_model.save(cu_gru_model_path)
print(f"GRU model saved to {cu_gru_model_path}")
```

\*\*\*\*\*5. Model development and Training for XGBoost\*\*\*\*\*

```
# Function to create the XGBoost model
```

```
def create_xgb_model(n_estimators = 100, learning_rate = 0.01, max_depth = 3):
    """
```

Creates and returns an XGBoost regression model with the specified parameters.

Parameters:

n\_estimators (int): The number of trees in the model (default is 200).

learning\_rate (float): The rate at which the model learns (default is 0.1). A lower value means slower learning but potentially better accuracy.

max\_depth (int): The maximum depth of each tree (default is 4). A higher value allows the model to learn more complex patterns but can increase overfitting risk.

Returns:

XGBRegressor: The XGBoost regression model with the given parameters and early stopping enabled after 10 rounds.

"""

```
model = xgb.XGBRegressor(  
    objective='reg:squarederror',  
    n_estimators=n_estimators,  
    learning_rate=learning_rate,  
    max_depth=max_depth,  
    early_stopping_rounds=10)  
return model
```

# Creates and train the XGBoost model

```
xgb_model = create_xgb_model(n_estimators = 100, learning_rate = 0.01, max_depth = 3)  
xgb_model.fit(X_train_seq_flat, y_train_seq_flat, eval_set=[(X_val_seq_flat,  
y_val_seq_flat)], verbose=False)
```

\*\*\*\*\*6. Evaluation of XGBoost model on the training and testing set\*\*\*\*\*

# Evaluates the XGBoost model on the training set

```
xgb_train_predictions = xgb_model.predict(X_train_seq_flat)  
xgb_train_predictions = scaler_y.inverse_transform(xgb_train_predictions.reshape(-1, 1))  
y_train_seq_inverse_xgb = scaler_y.inverse_transform(y_train_seq_flat.reshape(-1, 1))  
r2_xgb_train = r2_score(y_train_seq_inverse_xgb, xgb_train_predictions)  
print(f'R2 score for XGBoost on training set: {r2_xgb_train:.2f}')
```

# Evaluates the XGBoost model on the test set

```
xgb_test_predictions = xgb_model.predict(X_test_seq_flat)  
xgb_test_predictions = scaler_y.inverse_transform(xgb_test_predictions.reshape(-1, 1))  
y_test_seq_inverse_xgb = scaler_y.inverse_transform(y_test_seq_flat.reshape(-1, 1))  
r2_xgb_test = r2_score(y_test_seq_inverse_xgb, xgb_test_predictions)  
print(f'R2 score for XGBoost on test set: {r2_xgb_test:.2f}')
```

\*\*\*\*\*7. Hyperparameter Tuning for the XGBoost because of overfitting\*\*\*\*\*

# Grid of hyperparameters to search

```
param_grid_xgb = {  
    'n_estimators': [150, 200, 250],  
    'learning_rate': [0.055, 0.1, 0.15, 0.2],  
    'max_depth': [3, 4, 5],  
    'reg_alpha': [0, 0.1, 0.5], # Regularization parameters  
    'reg_lambda': [1, 1.5, 2]   # Regularization parameters  
}
```

```
best_score_xgb = float('inf')
```

```
best_params_xgb = None
```

```

best_model_xgb = None

# Manually iterating over the grid of hyperparameters
for n_estimators in param_grid_xgb['n_estimators']:
    for learning_rate in param_grid_xgb['learning_rate']:
        for max_depth in param_grid_xgb['max_depth']:
            for reg_alpha in param_grid_xgb['reg_alpha']:
                for reg_lambda in param_grid_xgb['reg_lambda']:
                    print(f'Training with n_estimators={n_estimators}, learning_rate={learning_rate},
max_depth={max_depth}, reg_alpha={reg_alpha}, reg_lambda={reg_lambda}')
                    model = xgb.XGBRegressor(objective='reg:squarederror',
                                             n_estimators=n_estimators,
                                             learning_rate=learning_rate,
                                             max_depth=max_depth,
                                             reg_alpha=reg_alpha,
                                             reg_lambda=reg_lambda,
                                             early_stopping_rounds=10)
                    model.fit(X_train_seq_flat, y_train_seq_flat, eval_set=[(X_val_seq_flat,
y_val_seq_flat)], verbose=False)

                    val_predictions = model.predict(X_val_seq_flat)
                    val_mse = mean_squared_error(y_val_seq_flat, val_predictions)
                    if val_mse < best_score_xgb:
                        best_score_xgb = val_mse
                        best_params_xgb = (n_estimators, learning_rate, max_depth, reg_alpha,
reg_lambda)
                        cu_best_xgb_model = model

print(f'Best parameters for XGBoost: {best_params_xgb}')
print(f'Best validation loss for XGBoost: {best_score_xgb}')

"""**8. Evaluation of XGBoost model on the training and testing set after Hyperparameter
Tuning with the best parameters**"""

# Evaluates the best XGBoost model on the training set
best_xgb_train_predictions = cu_best_xgb_model.predict(X_train_seq_flat)
best_xgb_train_predictions =
scaler_y.inverse_transform(best_xgb_train_predictions.reshape(-1, 1))
best_y_train_seq_inverse_xgb = scaler_y.inverse_transform(y_train_seq_flat.reshape(-1, 1))
best_r2_xgb_train = r2_score(best_y_train_seq_inverse_xgb, best_xgb_train_predictions)
print(f'R2 score for XGBoost on training set: {best_r2_xgb_train:.2f}')

# Evaluates the best XGBoost model on the test set
best_xgb_test_predictions = cu_best_xgb_model.predict(X_test_seq_flat)
best_xgb_test_predictions =
scaler_y.inverse_transform(best_xgb_test_predictions.reshape(-1, 1))
best_y_test_seq_inverse_xgb = scaler_y.inverse_transform(y_test_seq_flat.reshape(-1, 1))

```

```
best_r2_xgb_test = r2_score(best_y_test_seq_inverse_xgb, best_xgb_test_predictions)
print(f'R2 score for XGBoost on test set: {best_r2_xgb_test:.2f}')
```

```
# Saving the XGBoost model
cu_xgb_model_path = 'cu_best_xgb_model.json'
cu_best_xgb_model.save_model(cu_xgb_model_path)
print(f'XGBoost model saved to {cu_xgb_model_path}')
```

\*\*\*\*\*9. Model development and Training for Random Forest\*\*\*\*\*

```
# Function to create the Random Forest model
def create_rf_model(n_estimators = 50, max_depth = 10, min_samples_split = 2,
min_samples_leaf = 2):
    """
```

Creates and returns a Random Forest regression model with the specified parameters.

Parameters:

n\_estimators (int): The number of trees in the forest (default is 50). More trees can improve accuracy but increase computation time.

max\_depth (int): The maximum depth of each tree (default is 10). A deeper tree can capture more complex patterns but might overfit.

min\_samples\_split (int): The minimum number of samples required to split an internal node (default is 2). Higher values can help prevent overfitting.

min\_samples\_leaf (int): The minimum number of samples required to be at a leaf node (default is 1). Higher values make the model more robust by preventing the creation of leaves with very few samples.

Returns:

RandomForestRegressor: The Random Forest regression model with the given parameters.

```
    """
    model = RandomForestRegressor(
        n_estimators=n_estimators,
        max_depth=max_depth,
        min_samples_split=min_samples_split,
        min_samples_leaf=min_samples_leaf,
        random_state=42
    )
    return model
```

```
# Creates and train the Random Forest model
rf_model = create_rf_model(n_estimators=50, max_depth=10, min_samples_split=2,
min_samples_leaf=2)
rf_model.fit(X_train_seq_flat, y_train_seq_flat)
```

\*\*\*\*\*10. Evaluation of Random Forest model on the training and testing set\*\*\*\*\*

```
# Evaluates the Random Forest model on the training set
```

```

rf_train_predictions = rf_model.predict(X_train_seq_flat)
rf_train_predictions = scaler_y.inverse_transform(rf_train_predictions.reshape(-1, 1))
y_train_seq_inverse_rf = scaler_y.inverse_transform(y_train_seq_flat.reshape(-1, 1))
r2_rf_train = r2_score(y_train_seq_inverse_rf, rf_train_predictions)
print(f'R2 score for Random Forest on training set: {r2_rf_train:.2f}')

# Evaluates the Random Forest model on the test set
rf_test_predictions = rf_model.predict(X_test_seq_flat)
rf_test_predictions = scaler_y.inverse_transform(rf_test_predictions.reshape(-1, 1))
y_test_seq_inverse_rf = scaler_y.inverse_transform(y_test_seq_flat.reshape(-1, 1))
r2_rf_test = r2_score(y_test_seq_inverse_rf, rf_test_predictions)
print(f'R2 score for Random Forest on test set: {r2_rf_test:.2f}')

"""**11. Hyperparameter Tuning for the Random Forest model to see if we can improve the
R2 score on the test set**"""

# Grid of hyperparameters to search
param_grid_rf = {
    'n_estimators': [100, 150, 200],
    'max_depth': [10, 15, 20],
    'min_samples_split': [2, 5, 10, 15],
    'min_samples_leaf': [1, 2, 4]
}

best_score_rf = float('inf')
best_params_rf = None
best_model_rf = None

# Manually iterate over the grid of hyperparameters
for n_estimators in param_grid_rf['n_estimators']:
    for max_depth in param_grid_rf['max_depth']:
        for min_samples_split in param_grid_rf['min_samples_split']:
            for min_samples_leaf in param_grid_rf['min_samples_leaf']:
                print(f'Training with n_estimators={n_estimators}, max_depth={max_depth},
min_samples_split={min_samples_split}, min_samples_leaf={min_samples_leaf}')
                model = create_rf_model(n_estimators, max_depth, min_samples_split,
min_samples_leaf)
                model.fit(X_train_seq_flat, y_train_seq_flat)

                val_predictions = model.predict(X_val_seq_flat)
                val_mse = mean_squared_error(y_val_seq_flat, val_predictions)
                if val_mse < best_score_rf:
                    best_score_rf = val_mse
                    best_params_rf = (n_estimators, max_depth, min_samples_split,
min_samples_leaf)
                    cu_best_rf_model = model

```

```

print(f'Best parameters for Random Forest: {best_params_rf}')
print(f'Best validation loss for Random Forest: {best_score_rf}')

"""**12. Evaluation of best Random Forest model on the training and testing set after tuning
the parameters**"""

# Evaluates the best Random Forest model on the training set
best_rf_train_predictions = cu_best_rf_model.predict(X_train_seq_flat)
best_rf_train_predictions = scaler_y.inverse_transform(best_rf_train_predictions.reshape(-1, 1))
best_y_train_seq_inverse_rf = scaler_y.inverse_transform(y_train_seq_flat.reshape(-1, 1))
best_r2_rf_train = r2_score(best_y_train_seq_inverse_rf, best_rf_train_predictions)
print(f'R2 score for Random Forest on training set: {best_r2_rf_train:.2f}')

# Evaluates the best Random Forest model on the test set
best_rf_test_predictions = cu_best_rf_model.predict(X_test_seq_flat)
best_rf_test_predictions = scaler_y.inverse_transform(best_rf_test_predictions.reshape(-1, 1))
best_y_test_seq_inverse_rf = scaler_y.inverse_transform(y_test_seq_flat.reshape(-1, 1))
best_r2_rf_test = r2_score(best_y_test_seq_inverse_rf, best_rf_test_predictions)
print(f'R2 score for Random Forest on test set: {best_r2_rf_test:.2f}')

# Saving the Random Forest model
cu_rf_model_path = 'cu_best_rf_model.pkl'
dump(cu_best_rf_model, cu_rf_model_path)
print(f"Random Forest model saved to {cu_rf_model_path}")

"""# **13. Evaluations of the performance of the three machine learning models(GRU,
XGBoost, and Random Forest)**

**Loading the Models**
"""

# Load the saved models
def load_models():
    """
    Loads the saved models.

    Returns:
    tuple: A tuple containing the loaded GRU, XGBoost, and Random Forest models.
    """
    cu_best_gru_model = load_model('cu_best_gru_model.keras')
    cu_best_xgb_model = xgb.XGBRegressor()
    cu_best_xgb_model.load_model('cu_best_xgb_model.json')
    cu_best_rf_model = load('cu_best_rf_model.pkl')
    return cu_best_gru_model, cu_best_xgb_model, cu_best_rf_model

```



```

cu_best_gru_model, cu_best_xgb_model, cu_best_rf_model = load_models()

"""**1. Ensuring the Scaler is Fitted**"""

# Ensure scaler_y is fitted before using it
try:
    scaler_y.inverse_transform(np.array([[0]]))
except NotFittedError:
    print("scaler_y is not fitted. Ensure scaler_y is fitted on the training data.")

"""**2. Evaluate Model Metrics Function**"""

# Function to evaluate and return performance metrics as a DataFrame
def evaluate_model_metrics(cu_model_name, y_actual, y_pred):
    """
    Evaluates the performance of the developed model using different metrics.

    Parameters:
    cu_model_name (str): model to be evaluated.
    y_actual (array-like): actual target values.
    y_pred (array-like): predicted target values from the model.

    Returns:
    pd.DataFrame: A DataFrame containing the model's name and various performance
    metrics.
    """
    mse = mean_squared_error(y_actual, y_pred)
    mae = mean_absolute_error(y_actual, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_actual, y_pred)
    mape = mean_absolute_percentage_error(y_actual, y_pred)

    cu_metrics = pd.DataFrame({
        'Model': [cu_model_name],
        'MSE': [mse],
        'MAE': [mae],
        'RMSE': [rmse],
        'R2': [round(r2, 2)],
        'MAPE': [mape]
    })
    return cu_metrics

"""**3. Load and Evaluate each Model**"""

# Evaluates GRU model
gru_test_predictions = cu_best_gru_model.predict(X_test_seq)

```

```

gru_test_predictions = scaler_y.inverse_transform(gru_test_predictions)
y_test_seq_inverse_gru = scaler_y.inverse_transform(y_test_seq)
cu_gru_metrics = evaluate_model_metrics('GRU', y_test_seq_inverse_gru,
gru_test_predictions)

# Evaluates XGBoost model
best_xgb_test_predictions = cu_best_xgb_model.predict(X_test_seq_flat)
best_xgb_test_predictions =
scaler_y.inverse_transform(best_xgb_test_predictions.reshape(-1, 1))
best_y_test_seq_inverse_xgb = scaler_y.inverse_transform(y_test_seq_flat.reshape(-1, 1))
cu_xgboost_metrics = evaluate_model_metrics('XGBoost', best_y_test_seq_inverse_xgb,
best_xgb_test_predictions)

# Evaluates Random Forest model
best_rf_test_predictions = cu_best_rf_model.predict(X_test_seq_flat)
best_rf_test_predictions = scaler_y.inverse_transform(best_rf_test_predictions.reshape(-1,
1))
best_y_test_seq_inverse_rf = scaler_y.inverse_transform(y_test_seq_flat.reshape(-1, 1))
cu_random_forest_metrics = evaluate_model_metrics('Random Forest',
best_y_test_seq_inverse_rf, best_rf_test_predictions)

# Combined all metrics into a single DataFrame
cu_all_metrics = pd.concat([cu_gru_metrics, cu_xgboost_metrics,
cu_random_forest_metrics], ignore_index=True)

# Displayed the combined metrics
print(cu_all_metrics)

"""**4. Plotting Functions**"""

# Function to plot actual vs predicted values
def plot_predictions(dates, y_actual, y_pred, cu_model_name):
    """
    Plots the actual vs. predicted values for a given model over time.

    Parameters:
    dates : The dates corresponding to the actual and predicted values.
    y_actual : The actual target values.
    y_pred : The predicted target values from the model.
    cu_model_name (str): The name of the model, used for the plot title.

    Returns:
    None: The function displays the plot of actual vs. predicted values.
    """
    plt.figure(figsize=(14, 7))
    plt.plot(dates, y_actual, label='Actual', color='blue')
    plt.plot(dates, y_pred, label=f'{cu_model_name} Predictions', color='red')

```

```

plt.title(f'{cu_model_name} Model Predictions vs Actual')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=3))
plt.gcf().autofmt_xdate()
plt.show()

# Plot predictions for each model
plot_predictions(dates_test_seq, y_test_seq_inverse_gru, gru_test_predictions, 'GRU')
plot_predictions(dates_test_seq, best_y_test_seq_inverse_xgb, best_xgb_test_predictions,
'XGBoost')
plot_predictions(dates_test_seq, best_y_test_seq_inverse_rf, best_rf_test_predictions,
'Random Forest')

"""**For Residuals**"""

# Function to plot residuals
def plot_residuals(dates, y_actual, y_pred, cu_model_name):
    """
    Plots the residuals (the differences between actual and predicted values) for a given
    model.

    Parameters:
    dates : The dates corresponding to the actual and predicted values.
    y_actual : The actual target values.
    y_pred : The predicted target values from the model.
    cu_model_name (str): The name of the model, used for the plot title.

    Returns:
    None: The function displays the residual plot.
    """
    residuals = y_actual - y_pred
    plt.figure(figsize=(14, 7))
    plt.plot(dates, residuals, label='Residuals', color='green')
    plt.title(f'{cu_model_name} Residuals')
    plt.xlabel('Date')
    plt.ylabel('Residuals')
    plt.legend()
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
    plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=3))
    plt.gcf().autofmt_xdate()
    plt.show()

# Plot residuals for each model
plot_residuals(dates_test_seq, y_test_seq_inverse_gru, gru_test_predictions, 'GRU')

```

```
plot_residuals(dates_test_seq, best_y_test_seq_inverse_xgb, best_xgb_test_predictions,
'XGBoost')
plot_residuals(dates_test_seq, best_y_test_seq_inverse_rf, best_rf_test_predictions,
'Random Forest')
```

"""### Objective 3 & 4. Implementing a simple trading strategy for each model and simulate trades using a back testing approach and also analyzing the trading results for each model to determine profitability.

**\*\*1. Generate Trading Signals\*\***

"""

# Function to generate simple trading signals based on model predictions

def generate\_simple\_signals(predictions, actual, dates):

"""

Generate simple trading signals based on model predictions.

Parameters:

predictions (array-like): Model predictions for the target variable.

actual (array-like): Actual values of the target variable.

dates (array-like): Dates corresponding to the target variable values.

Returns:

list of tuples: A list of trading signals, each represented as a tuple with the date and 'Buy' or 'Sell' signal.

"""

```
signals = [
    (dates[i], 'Buy' if predictions[i + 1] > actual[i] else 'Sell')
    for i in range(len(predictions) - 1)
]
return signals
```

**"""\*\*2. Simulate and Evaluate Trading Strategy\*\*"""**

# Simulate and evaluate the simple trading strategy with stop loss and take profit

def simulate\_and\_evaluate\_simple\_trading(signals, actual, dates, stop\_loss\_pct=0.001,

take\_profit\_pct=0.002):

"""

Simulate and evaluate a simple trading strategy with stop loss and take profit.

Parameters:

signals (list of tuples): Trading signals generated by the model.

actual (array-like): Actual values of the target variable.

dates (array-like): Dates corresponding to the target variable values.

stop\_loss\_pct (float): Stop loss percentage.

take\_profit\_pct (float): Take profit percentage.

Returns:

tuple: Final balance, profit, Sharpe ratio, maximum drawdown, balance history, and executed trade dates.

```
"""
initial_balance = 10000
balance = initial_balance
position_size = 0
balance_history = []
full_dates = [] # Track all dates

for i, (date, signal) in enumerate(signals):
    trade_amount = actual[i] # Price per unit

    if signal == 'Buy' and balance > trade_amount:
        position_size = int(balance // trade_amount)
        cost = position_size * trade_amount
        balance -= cost
        entry_price = trade_amount
        stop_loss_price = entry_price * (1 - stop_loss_pct)
        take_profit_price = entry_price * (1 + take_profit_pct)

        for j in range(i + 1, len(actual)):
            if actual[j] <= stop_loss_price or actual[j] >= take_profit_price:
                balance += position_size * actual[j]
                position_size = 0
                break
    elif signal == 'Sell' and position_size > 0:
        balance += position_size * trade_amount
        position_size = 0

    balance_history.append(max(balance + position_size * actual[i], 0))
    full_dates.append(date) # Append current date to full dates

final_balance = balance_history[-1] if balance_history else initial_balance
profit = final_balance - initial_balance
daily_returns = calculate_daily_returns(balance_history)
sharpe_ratio = calculate_sharpe_ratio(daily_returns)
max_drawdown = calculate_max_drawdown(balance_history)

return final_balance, profit, sharpe_ratio, max_drawdown, balance_history, full_dates #
Return full dates

"""**3. Calculate Daily Returns**"""

# Calculate daily returns
def calculate_daily_returns(balance_history):
    """
```

Calculate daily returns based on balance history.

Parameters:

balance\_history (array-like): History of account balances.

Returns:

array: Daily returns.

"""

```
balance_history = np.array(balance_history)
returns = (balance_history[1:] - balance_history[:-1]) / balance_history[:-1]
return returns
```

"""\*\*4. Calculate Sharpe Ratio\*\*"""

# Function to calculate Sharpe ratio

def calculate\_sharpe\_ratio(returns, risk\_free\_rate=0.01):

"""

Calculate the Sharpe ratio of the returns.

Parameters:

returns (array-like): Daily returns.

risk\_free\_rate (float): Risk-free rate for calculating excess returns.

Returns:

float: Sharpe ratio.

"""

```
if len(returns) == 0:
    return np.nan
excess_returns = returns - (risk_free_rate / 252)
if np.std(excess_returns) == 0:
    return np.nan
return np.mean(excess_returns) / np.std(excess_returns)
```

"""\*\*5. Calculate Maximum Drawdown\*\*"""

# Function to calculate maximum drawdown

def calculate\_max\_drawdown(balance\_history):

"""

Calculate the maximum drawdown from the balance history.

Parameters:

balance\_history (array-like): History of account balances.

Returns:

float: Maximum drawdown.

"""

```
balance_history = np.array(balance_history)
```

```

peak = np.maximum.accumulate(balance_history)
drawdown = (peak - balance_history) / peak
return np.max(drawdown)

```

\*\*\*\*\*6. Plot Trading Performance\*\*\*\*\*

```

# Function to plot the final balance and performance metrics
def plot_trading_performance(dates, balance_history, cu_model_name):
    """

```

Plot the trading performance over time.

Parameters:

dates : Dates corresponding to the balance history.

balance\_history : History of account balances.

cu\_model\_name (str): Name of the model being evaluated.

Returns:

None

"""

```

plt.figure(figsize=(14, 7))
plt.plot(dates, balance_history, label='Balance History', color='blue')
plt.title(f'{cu_model_name} Trading Performance')
plt.xlabel('Date')
plt.ylabel('Account Balance')
plt.legend()
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
plt.gca().xaxis.set_major_locator(mdates.MonthLocator(interval=3))
plt.gcf().autofmt_xdate()
plt.show()

```

\*\*\*\*\*7. Evaluates Model\*\*\*\*\*

```

# Function to predict and evaluate a model
def evaluate_model(model, X_test, y_test, scaler_y, dates, model_name):
    """

```

Predict and evaluate a model's performance.

Parameters:

model (object): The trained model to be evaluated.

X\_test (array-like): Test set features.

y\_test (array-like): Test set target values.

scaler\_y (object): Scaler used to inverse transform the target values.

dates (array-like): Dates corresponding to the test set.

model\_name (str): Name of the model being evaluated.

Returns:

tuple: Final balance, profit, Sharpe ratio, and maximum drawdown.

```

"""
if len(X_test.shape) == 1:
    X_test = X_test.reshape(-1, 1)

# Make predictions and inverse transform
predictions = model.predict(X_test)
predictions = scaler_y.inverse_transform(predictions.reshape(-1, 1)).flatten()
y_test_inverse = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten()

# Debug statements to check predictions and actual values
print(f'{model_name} Predictions: {predictions[:10]}')
print(f'{model_name} Actual: {y_test_inverse[:10]}')

simple_signals = generate_simple_signals(predictions, y_test_inverse, dates)

# Debug statement to check signals
print(f'{model_name} Signals: {simple_signals[:10]}')

final_balance, profit, sharpe_ratio, max_drawdown, balance_history, full_dates =
simulate_and_evaluate_simple_trading(simple_signals, y_test_inverse, dates) # Use full
dates

print(f'{model_name} - Final Balance: {final_balance:.2f}, Profit: {profit:.2f}, Sharpe Ratio:
{sharpe_ratio:.2f}, Max Drawdown: {max_drawdown:.2f}')

if balance_history:
    plot_trading_performance(full_dates, balance_history, model_name) # Plot with full
dates
else:
    print(f"No trades were executed for {model_name}, balance history is empty.")
    return final_balance, profit, sharpe_ratio, max_drawdown

"""**8. Evaluates Ensemble Model**"""

```

```

def evaluate_ensemble(models, X_test_seq, X_test_flat, y_test, scaler_y, dates):
    """

```

Evaluate an ensemble model's performance.

Parameters:

models (list): List of tuples containing the trained models and their names.

X\_test\_seq (array-like): Sequential test set features.

X\_test\_flat (array-like): Flattened test set features.

y\_test (array-like): Test set target values.

scaler\_y (object): Scaler used to inverse transform the target values.

dates (array-like): Dates corresponding to the test set.

Returns:



```

tuple: Final balance, profit, sharpe ratio, maximum drawdown, and ensemble metrics
DataFrame.
"""
predictions_list = []

for model, model_name in models:
    if model_name == 'GRU':
        predictions = model.predict(X_test_seq)
        predictions = scaler_y.inverse_transform(predictions.reshape(-1, 1)).flatten()
    elif model_name == 'XGBoost':
        predictions = model.predict(X_test_flat)
        predictions = scaler_y.inverse_transform(predictions.reshape(-1, 1)).flatten()
    else: # Random Forest
        predictions = model.predict(X_test_flat)
        predictions = scaler_y.inverse_transform(predictions.reshape(-1, 1)).flatten()

    predictions_list.append(predictions)

# Averaging the predictions to create the ensemble prediction
ensemble_predictions = np.mean(predictions_list, axis=0)
y_test_inverse = scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten()

# Debug statements to check ensemble predictions and actual values
print(f'Ensemble Predictions: {ensemble_predictions[:10]}')
print(f'Ensemble Actual: {y_test_inverse[:10]}')

simple_signals = generate_simple_signals(ensemble_predictions, y_test_inverse, dates)

# Debug statement to check signals
print(f'Ensemble Signals: {simple_signals[:10]}')

final_balance, profit, sharpe_ratio, max_drawdown, balance_history, full_dates =
simulate_and_evaluate_simple_trading(simple_signals, y_test_inverse, dates) # Use full
dates

print(f'Ensemble - Final Balance: {final_balance:.2f}, Profit: {profit:.2f}, Sharpe Ratio:
{sharpe_ratio:.2f}, Max Drawdown: {max_drawdown:.2f}')

if balance_history:
    plot_trading_performance(full_dates, balance_history, 'Ensemble') # Plot with full
dates
else:
    print(f"No trades were executed for Ensemble, balance history is empty.")

# Evaluate the model metrics
ensemble_metrics = evaluate_model_metrics('Ensemble', y_test_inverse,
ensemble_predictions)

```

```

print(ensemble_metrics)

# Plot predictions
plot_predictions(dates, y_test_inverse, ensemble_predictions, 'Ensemble')

# Plot residuals
plot_residuals(dates, y_test_inverse, ensemble_predictions, 'Ensemble')

return final_balance, profit, sharpe_ratio, max_drawdown, ensemble_metrics

"""**9. Call Load Models Functions and Evaluate**"""

# Load models
cu_best_gru_model, cu_best_xgb_model, cu_best_rf_model = load_models()

# Evaluate each model and capture the performance metrics
final_balance_gru, profit_gru, sharpe_ratio_gru, max_drawdown_gru =
evaluate_model(cu_best_gru_model, X_test_seq, y_test_seq, scaler_y, dates_test_seq,
'GRU')
final_balance_xgb, profit_xgb, sharpe_ratio_xgb, max_drawdown_xgb =
evaluate_model(cu_best_xgb_model, X_test_seq_flat, y_test_seq_flat, scaler_y,
dates_test_seq, 'XGBoost')
final_balance_rf, profit_rf, sharpe_ratio_rf, max_drawdown_rf =
evaluate_model(cu_best_rf_model, X_test_seq_flat, y_test_seq_flat, scaler_y,
dates_test_seq, 'Random Forest')

# Evaluate the ensemble model
models = [(cu_best_gru_model, 'GRU'), (cu_best_xgb_model, 'XGBoost'),
(cu_best_rf_model, 'Random Forest')]
final_balance_ensemble, profit_ensemble, sharpe_ratio_ensemble,
max_drawdown_ensemble, ensemble_metrics = evaluate_ensemble(models, X_test_seq,
X_test_seq_flat, y_test_seq_flat, scaler_y, dates_test_seq)

# Add ensemble metrics to the all_metrics DataFrame
all_metrics = pd.concat([cu_all_metrics, ensemble_metrics], ignore_index=True)

# Display the combined metrics
print(all_metrics)

"""### Objective 5. Comparing the models based on evaluation metrics and profitability
analysis to identify the best model for the trading bot.

**1. Combining Trading Performance Metrics**
"""

# Combines trading performance metrics into a DataFrame
trading_performance_df = pd.DataFrame({

```

```

'Model': ['GRU', 'XGBoost', 'Random Forest', 'Ensemble'],
'Final Balance': [final_balance_gru, final_balance_xgb, final_balance_rf,
final_balance_ensemble],
'Profit': [profit_gru, profit_xgb, profit_rf, profit_ensemble],
'Sharpe Ratio': [sharpe_ratio_gru, sharpe_ratio_xgb, sharpe_ratio_rf,
sharpe_ratio_ensemble],
'Max Drawdown': [max_drawdown_gru, max_drawdown_xgb, max_drawdown_rf,
max_drawdown_ensemble]
})

```

```

# Print trading performance metrics
print(trading_performance_df)

```

\*\*\*\*\*2. Visualization of Model Performance\*\*\*\*\*

```

# Visualization of Model Performance
def visualize_model_performance(performance_df):
    """

```

Visualize the model performance metrics.

Parameters:

performance\_df (DataFrame): DataFrame containing performance metrics for each model.

Returns:

None

"""

```

fig, ax1 = plt.subplots(figsize=(14, 8))

```

```

# Plot primary metrics

```

```

for metric in ['MSE', 'MAE', 'RMSE', 'MAPE']:

```

```

    ax1.plot(all_metrics['Model'], all_metrics[metric], marker='o', label=metric)

```

```

# Add secondary y-axis for R2

```

```

ax2 = ax1.twinx()

```

```

ax2.plot(all_metrics['Model'], all_metrics['R2'], marker='o', color='r', label='R2',
linestyle='--')

```

```

# Combine legends from both axes

```

```

lines_1, labels_1 = ax1.get_legend_handles_labels()

```

```

lines_2, labels_2 = ax2.get_legend_handles_labels()

```

```

ax1.legend(lines_1 + lines_2, labels_1 + labels_2, loc='upper left')

```

```

ax1.set_title('Model Performance Comparison')

```

```

ax1.set_xlabel('Model')

```

```

ax1.set_ylabel('Metric Value')

```

```

ax2.set_ylabel('R2 Value')

```

```

plt.show()

"""**3. Visualization of Trading Performance**"""

# Visualization of Trading Performance
def visualize_trading_performance(trading_performance_df):
    """
    Visualize the trading performance metrics.

    Parameters:
    trading_performance_df (DataFrame): DataFrame containing trading performance metrics
    for each model.

    Returns:
    None
    """
    fig, ax1 = plt.subplots(figsize=(14, 8))

    # Plot primary metrics
    for metric in ['Final Balance', 'Profit']:
        ax1.plot(trading_performance_df['Model'], trading_performance_df[metric],
        marker='o', label=metric)

    # Add secondary y-axis for Sharpe Ratio and Max Drawdown
    ax2 = ax1.twinx()
    ax2.plot(trading_performance_df['Model'], trading_performance_df['Sharpe Ratio'],
    marker='o', color='g', label='Sharpe Ratio', linestyle='--')
    ax2.plot(trading_performance_df['Model'], trading_performance_df['Max Drawdown'],
    marker='o', color='r', label='Max Drawdown', linestyle='--')

    # Combine legends from both axes
    lines_1, labels_1 = ax1.get_legend_handles_labels()
    lines_2, labels_2 = ax2.get_legend_handles_labels()
    ax1.legend(lines_1 + lines_2, labels_1 + labels_2, loc='upper left')

    ax1.set_title('Trading Performance Comparison')
    ax1.set_xlabel('Model')
    ax1.set_ylabel('Balance and Profit')
    ax2.set_ylabel('Sharpe Ratio and Max Drawdown')

    plt.show()

"""**4. Calling Visualization Functions**"""

# Visualize the model performance
visualize_model_performance(all_metrics)

```

```
# Visualize the trading performance  
visualize_trading_performance(trading_performance_df)
```

## APPENDIX B

Artificial Intelligence (AI)  
British Pound/US Dollar (GBP/USD)  
Commodity Channel Index (CCI)  
Convolutional Neural Network (CNN)  
Exploratory Data Analysis (EDA)  
Exponential Moving Average (EMA)  
Extreme Gradient Boosting (XGBoost)  
Foreign Exchange (Forex)  
Gated Recurrent Unit (GRU)  
General Data Protection Regulation (GDPR)  
Great British Pound (GBP)  
Information Commissioner's Office (ICO)  
Interquartile Range (IQR)  
Long Short-Term Memory (LSTM)  
Machine Learning (ML)  
Mean Absolute Error (MAE)  
Mean Absolute Percentage Error (MAPE)  
Mean Squared Error (MSE)  
Moving Average Convergence Divergence (MACD)  
Open, High, Low, Close (OHLC)  
 $R^2$  (R-Squared)  
Random Forest (RF)  
Rate of Return (RoR)  
Recurrent Neural Network (RNN)  
Relative Strength (RS)  
Relative Strength Index (RSI)  
Root Mean Square Error (RMES)  
Support Vector Machine (SVM)  
Support Vector Regression (SVR)  
United States Dollar (USD)