A black background with white text

Description automatically generated with low confidence

MSc Data Science Project

7PAM2002-0509-2023

Department of Physics, Astronomy and Mathematics

**Data Science FINAL PROJECT REPORT**

**Project Title:**

**Premier League Football Match Predictions**

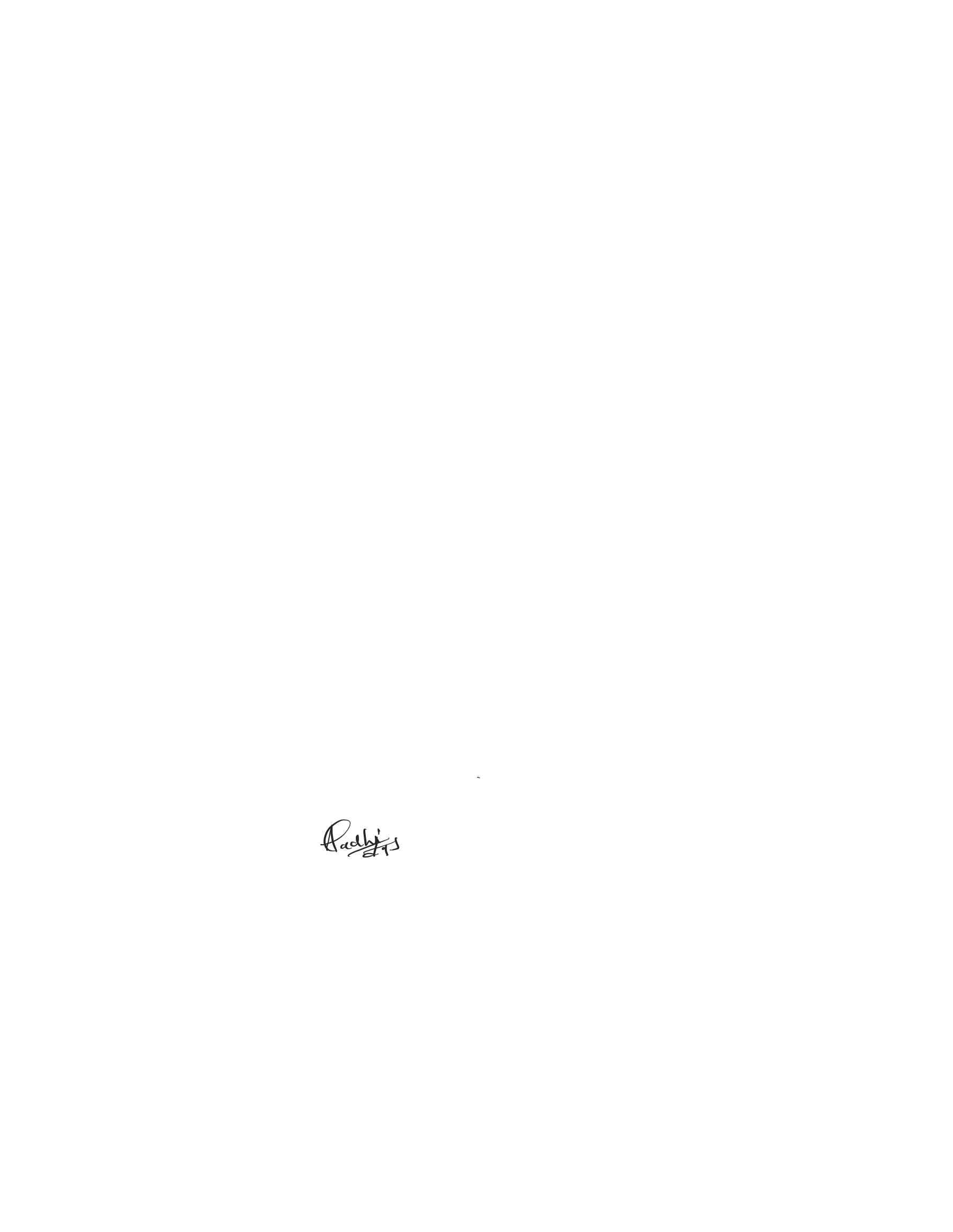
**Student Name and SRN:**

Aadhithiyan Kalaivani Viswanathan (22082768)

Supervisor: John Evans

Date Submitted: 29/08/2024

Word Count: 7449

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](https://www.herts.ac.uk/__data/assets/pdf_file/0007/237625/AS14-Apx3-Academic-Misconduct-v17.0.pdf) 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I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged.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 reportor 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: Aadhithiyan Kalaivani Viswanathan

Student Name signature:

Student SRN number: 22082768

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

Contents

[Introduction 3](#_Toc175149241)

[1.1 Background 3](#_Toc175149242)

[1.2 Problem Statement 3](#_Toc175149243)

[1.3 Research Objectives 3](#_Toc175149244)

[1.4 Research Questions 3](#_Toc175149245)

[1.5 Motivation and Real-World Impact 3](#_Toc175149246)

[1.6 Ethical legal and Professional concerns 3](#_Toc175149247)

[1.7 Structure of the Thesis 3](#_Toc175149248)

[Literature Review 3](#_Toc175149249)

[2.1 Introduction to the Literature Review 3](#_Toc175149250)

[2.2 Current Theories and Models 3](#_Toc175149251)

[2.3 Identification and Addressing of Literature Gaps 3](#_Toc175149252)

[Methodology 3](#_Toc175149253)

[3.1 Introduction 3](#_Toc175149254)

[3.2 Data Collection 3](#_Toc175149255)

[3.3 Data Preprocessing 3](#_Toc175149256)

[3.4 Model Selection 3](#_Toc175149257)

[3.4.1 Model Training 3](#_Toc175149258)

[3.4.2 Model Evaluation 3](#_Toc175149259)

[Exploratory Data Analysis 3](#_Toc175149260)

[4.1 Dataset Description and Exploration 3](#_Toc175149261)

[Model Building and Optimization 3](#_Toc175149262)

[5.1 Feature Engineering 3](#_Toc175149263)

[○ 5.1.1 Calculating Goals Scored and Conceded 3](#_Toc175149264)

[○ 5.1.2 Cumulative Points 3](#_Toc175149265)

[○ 5.1.3 Team Form 3](#_Toc175149266)

[○ 5.1.4 Calculating Goal Differences and Streaks 3](#_Toc175149267)

[○ 5.1.5 Scaling Features by Matchweek 3](#_Toc175149268)

[○ 5.1.6 Transforming Target Variable 3](#_Toc175149269)

[5.2 Model Building 3](#_Toc175149270)

[○ 5.2.1 Logistic Regression 3](#_Toc175149271)

[○ 5.2.2 Support Vector Machine (SVM) 3](#_Toc175149272)

[○ 5.2.3 Random Forest Classifier 4](#_Toc175149273)

[○ 5.2.4 XGBoost (Extreme Gradient Boosting) 4](#_Toc175149274)

[○ 5.2.5 Gradient Boosting Machines (GBM) 4](#_Toc175149275)

[5.3 Evaluation Metrics and Techniques 4](#_Toc175149276)

[Results and Discussion 4](#_Toc175149277)

[6.1 Preliminary Evaluation Insights 4](#_Toc175149278)

[6.2 Underperformers 4](#_Toc175149279)

[6.3 Hyperparameter Tuning 4](#_Toc175149280)

[6.4 Application Development 4](#_Toc175149281)

[6.5 Alignment of Research Objectives with Results 4](#_Toc175149282)

[6.6 Mitigating Bias in Machine Learning Models for Football Predictions 4](#_Toc175149283)

[6.7 Summary and Implications 4](#_Toc175149284)

[Recommendations and Future Works 4](#_Toc175149285)

[7.1 Recommendations 4](#_Toc175149286)

[Conclusion 4](#_Toc175149287)

[8.1 Summary of Achievements 4](#_Toc175149288)

[8.2 Challenges and Limitations 4](#_Toc175149289)

[8.3 Future Directions 4](#_Toc175149290)

[Appendix A 4](#_Toc175149291)

[Code 4](#_Toc175149292)

[**Introduction** 5](#_Toc175149293)

[**Legal Concerns** 7](#_Toc175149294)

[**2.2 Current Theories and Models** 9](#_Toc175149295)

[**2.3 Identification and Addressing of Literature Gaps** 12](#_Toc175149296)

[**Methodology** 13](#_Toc175149297)

[**3.1 Introduction** 13](#_Toc175149298)

[**3.2 Data Collection** 13](#_Toc175149299)

[**3.3 Data Preprocessing** 14](#_Toc175149300)

[**3.4 Model Selection** 14](#_Toc175149301)

[**Exploratory Data Analysis** 15](#_Toc175149302)

[**5.2 Model Building** 20](#_Toc175149303)

[**5.3 Evaluation Metrics and Techniques** 26](#_Toc175149305)

[**Results and Discussion** 29](#_Toc175149306)

[**6.2 Underperformers and Hyperparameter Tuning** 35](#_Toc175149307)

[**6.3 Alignment of Objectives, Bias Mitigation, and Implications** 35](#_Toc175149308)

[**7.1 Recommendations and Future Work** 36](#_Toc175149309)

[**Conclusion** 37](#_Toc175149310)

### **Introduction**

#### **1.1 Background**

Football, or soccer as it is known in some parts of the world, is the most popular sport globally, with an estimated 250 million players across over 200 countries. The English Premier League (EPL) stands as the pinnacle of domestic football leagues, attracting millions of viewers worldwide and generating substantial economic impact through broadcasting rights, merchandise sales, and sports betting. The latter, a multibillion-dollar industry, relies heavily on accurate match predictions, making it a focal point for both enthusiasts and professionals. With the rise of data analytics and machine learning, predicting football match outcomes has evolved from being an art to a science, leveraging historical data and advanced algorithms to enhance prediction accuracy.

Over the past two decades, the availability of comprehensive football match data has opened new avenues for research and analysis. This dataset includes detailed records of EPL matches, encompassing variables such as match dates, team performances, goals scored, and match outcomes. By analyzing this data, we can uncover patterns and insights that were previously unattainable. This project aims to harness the power of machine learning to predict match outcomes, providing valuable insights for fans, analysts, and stakeholders within the football community.

#### **1.2 Problem Statement**

Predicting the outcome of football matches is inherently challenging due to the multitude of factors influencing the result. Traditional methods, often based on expert opinions or simplistic statistical models, lack the precision and adaptability required for high accuracy. The primary challenge lies in identifying and effectively utilizing relevant features from historical data to make accurate predictions. Factors such as team form, player performance, and even managerial changes can significantly impact match outcomes, necessitating a sophisticated analytical approach.

The existing literature on football match prediction has explored various methodologies, yet there remains a gap in integrating multiple machine learning models and extensive feature engineering to enhance prediction accuracy. This thesis addresses this gap by implementing a range of machine learning models, including Random Forests, Gradient Boosting Machines (GBM), and XGBoost, and evaluating their performance in predicting EPL match outcomes. By doing so, it seeks to identify the most effective model and the critical features that contribute to prediction success.

#### **1.3 Research Objectives**

The primary aim of this research is to predict Premier League match outcomes by employing various machine learning models and identifying the best-performing one. To achieve this, several objectives have been outlined:

1. Collect and preprocess EPL match data spanning multiple seasons.
2. Implement and compare different machine learning models, including Random Forests, GBM, and XGBoost, to predict match outcomes.
3. Perform extensive feature engineering to create new variables that could improve model performance.
4. Evaluate the models using metrics such as Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and R².
5. Analyze the strengths and limitations of each model in the context of football match prediction.
6. Provide actionable insights based on the findings, potentially benefiting sports analysts, bettors, and football clubs.

By systematically addressing these objectives, this thesis aims to advance the understanding and methodology of football match prediction using machine learning techniques.

#### **1.4 Research Questions**

1. How can historical match data be used to predict future outcomes in the English Premier League?
2. What are the weak points of a team that can be exploited by opponents?
3. Which players should be sold based on their performance data?
4. What is a player’s best position, as inferred from their performance data?
5. Is the "new manager bounce" phenomenon, where a team performs better after appointing a new manager, statistically significant?
6. Are there factors consistently relevant across multiple seasons that influence match outcomes?
7. How does the importance of various factors change as the game evolves?

These questions guide the research, ensuring a comprehensive exploration of the factors influencing football match outcomes and the efficacy of different predictive models.

#### **1.5 Motivation and Real-World Impact**

The motivation behind this research stems from both academic interest and practical applications. Accurate football match predictions can significantly impact various stakeholders, including sports analysts, bettors, and football clubs. For analysts and clubs, insights derived from predictive models can inform strategic decisions, such as player transfers, training focus, and match strategies. For bettors, improved prediction accuracy translates into better betting strategies, potentially reducing the risk and increasing the return on investment.

Furthermore, this research contributes to the broader field of sports analytics, showcasing how machine learning can transform traditional sports analysis. By advancing the methodologies for predicting football match outcomes, this thesis not only adds to academic knowledge but also provides practical tools that can be leveraged in the real world. The interdisciplinary nature of this research, combining sports, data science, and machine learning, highlights the innovative potential of data-driven approaches in various domains.

#### **1.6 Ethical, Legal and Professional concerns**

#### **Ethical Concerns**

Ethical concerns in machine learning for football prediction involve the fair use of data, ensuring transparency, and avoiding bias. It’s crucial to respect player privacy and avoid using sensitive information without consent. Models should be designed to avoid perpetuating biases that may exist in the data, such as favoritism toward certain teams or players. Moreover, there is an ethical responsibility to ensure that predictions are used responsibly, particularly in gambling, where they could lead to financial harm. Ensuring fairness, transparency, and accountability is key in ethically deploying these technologies.

### **Legal Concerns**

Legal concerns revolve around data privacy, intellectual property, and the potential for litigation. The collection and use of player data must comply with data protection laws such as GDPR, ensuring informed consent is obtained. Intellectual property rights must be respected when using proprietary algorithms or datasets. Additionally, there is a risk of legal issues arising from the use of predictions in gambling, where incorrect predictions could lead to financial losses. Ensuring compliance with relevant laws and regulations is essential to mitigate legal risks in deploying machine learning models for football prediction.

**Professional Concerns**: Includes ensuring the accuracy and reliability of predictions, maintaining integrity in the use of data, and upholding industry standards. Practitioners must avoid over-promising the capabilities of their models and ensure that their predictions are based on sound methodologies. There is also a professional duty to continuously update models with the latest data and techniques, avoiding obsolescence. Additionally, professionals must consider the impact of their work on stakeholders, such as teams, players, and fans, ensuring that their work contributes positively to the sport and its analysis.

#### **1.6 Structure of the Thesis**

The structure of this thesis is designed to provide a clear and logical progression of the research undertaken. It is organized as follows:

Chapter 2: Literature Review  
A comprehensive review of existing research on football match prediction, examining different methodologies, models, and the evolution of data analytics in sports.

Chapter 3: Methodology  
A detailed account of data collection and preprocessing techniques, feature engineering processes, and the selection and implementation of machine learning algorithms, along with the evaluation metrics used to assess model performance.

Chapter 4: Exploratory Data Analysis  
Analysis of the dataset characteristics, distribution of key features like goals and match results, and preliminary insights into factors influencing match outcomes.

Chapter 5: Model Building and Optimization  
Discussion on the construction of predictive models, feature selection, and optimization strategies, including hyperparameter tuning and model refinement.

Chapter 6: Results and Discussion  
Presentation of the evaluation results for different machine learning models, comparison of their performance, and interpretation of how the findings align with the research objectives.

Chapter 7: Recommendations and Future Work  
Suggestions for future research, potential enhancements in methodology, and practical recommendations for football clubs and analysts based on the study's insights.

Chapter 8: Conclusion  
A summary of the research findings, their implications for the field of sports analytics, and contributions to the broader understanding of football match predictions using artificial intelligence.

**Literature Review**

**2.1 Introduction to the Literature Review**

The use of machine learning (ML) techniques for predicting football match outcomes has gained momentum in recent years. This review explores the effectiveness of models like Random Forests (RF), Gradient Boosting Machines (GBM), and XGBoost, which have been widely applied to analyze extensive datasets covering team and player performance, alongside external factors such as weather and stadium capacity. These models assist in decision-making, from tactical adjustments to player management.

Most studies treat football outcome prediction as a classification problem, often focusing on multiclass predictions like win, draw, or loss. However, predicting draws remains challenging due to class imbalances and the subtlety of draws compared to wins or losses. Researchers like Bunker and Thabtah (2019) and Baboota and Kaur (2018) have noted these difficulties. Choi, Foo, and Chua (2023) demonstrated that balancing sampling techniques could improve model performance, especially in simplifying classification to binary outcomes like "Win" vs. "Non-Win."

Efforts to improve accuracy include integrating additional features and methodologies. For example, Ren and Susnjak (2022) used the Kelly Index to enhance model accuracy, and Carloni et al. combined traditional and novel ML techniques to refine betting strategies. Despite advancements, gaps remain, such as inconsistent accuracies across studies due to differences in datasets and methodologies, and the challenges posed by football's dynamic nature. Future research should focus on integrating real-time data and improving model interpretability.

### **2.2 Current Theories and Models**

ML applications in football prediction employ various models like RF, GBM, XGBoost, logistic regression, and SVM. Ensemble methods, which combine predictions from multiple models, have shown promise in enhancing accuracy. Ren and Susnjak (2022) used ensemble methods with the Kelly Index to reduce uncertainty, particularly in betting contexts. Feature engineering, the process of deriving meaningful features from raw data, is crucial for improving model performance, as highlighted by Chandra et al. (2024).

Integrating real-time data remains a challenge, as dynamic factors like player injuries can significantly impact predictions. The combination of ML models with human expertise, as explored by Beal et al., can enhance the interpretability and practical utility of predictions. As the field progresses, further advancements in feature engineering and the integration of real-time data are expected to improve prediction accuracy.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Paper Title** | **Authors** | **Year** | **Key Models** | **Features Used** | **Findings** | **Challenges** | **Inference** |
| Predicting Football Match Outcomes With Machine Learning Approaches | Bunker & Thabtah | 2019 | Logistic Regression, Random Forest, XGBoost | FIFA ratings, match statistics | Difficulty in predicting draws | Class imbalance, prediction accuracy | Need for balanced sampling techniques and refined models to improve draw predictions |
| Predicting Football Match Outcomes with eXplainable Machine Learning and the Kelly Index | Ren & Susnjak | 2022 | CatBoost, Random Forest | Elo-based ratings, historical match statistics | Improved accuracy using Kelly Index | Uncertainty reduction | Integrating Kelly Index with model confidence can enhance betting strategies |
| Predicting Football Player Performance and Match Outcomes Using Machine Learning Algorithms | Chandra B, Jennet Shinny D, & Keshav Adhitya M | 2024 | Logistic Regression, SVM, Bayesian networks | Player statistics, team dynamics | Precision range of 75-80% | Need for dynamic adaptation | Diverse feature sets and adaptability are crucial for accurate predictions |
| The Application of Machine Learning Techniques for Predicting Match Results in Team Sport: A Review | Bunker & Susnjak | 2019 | ANNs, SVMs, Decision Trees, Ensemble methods | Various sports datasets | ANNs and Boosting methods showed higher accuracy | Rich feature sets needed | Ensemble methods are becoming more effective in sports prediction |
| A Machine Learning Approach to Football Match Result Prediction | Carloni et al. | N/A | Logistic Regression, KNN, SVM, Naive Bayes, Random Forest, ANNs | Betting odds, match statistics | ANNs outperformed other models | Data collection and feature selection | ANN's ability to capture complex patterns is beneficial in sports betting |
| Predicting English Premier League Winners using Machine Learning | Harit & Mody | 2017 | Random Forest, KNN, SVM, Logistic Regression | Soccer statistics, feature selection | Random Forest achieved 0.60 accuracy | Unpredictable nature of EPL | Random Forest's robustness in handling various features is valuable |
| Combining Machine Learning and Human Experts to Predict Match Outcomes in Football: A Baseline Model | Beal et al. | N/A | Text vectorization, Dixon & Coles model, Bookmakers' predictions | Statistical match data, contextual articles | 63.19% accuracy | Integrating human expertise | Combining ML with expert insights can improve prediction accuracy |
| Predictive Analysis and Modelling Football Results Using Machine Learning Approach for English Premier League | Baboota & Kaur | 2018 | Gaussian Naive Bayes, SVM, Random Forest, Gradient Boosting | Individual team and differential features | Gradient Boosting achieved highest accuracy | Limited features compared to betting organizations | Extensive feature engineering can significantly enhance prediction accuracy |
| Machine Learning for Soccer Match Result Prediction | Bunker, Yeung, & Fujii | 2023 | CatBoost, XGBoost, Random Forests, Deep Learning | Various public datasets, soccer-specific ratings | Superior performance with gradient-boosted trees | Need for more nuanced features | Integration of advanced data like player chemistry could improve results |

Table 2.1: Overview of Key Studies on Machine Learning Applications for Predicting Football Match Outcomes

### **2.3 Identification and Addressing of Literature Gaps**

While significant progress has been made in applying ML to predict football outcomes, key gaps remain. Inconsistencies in reported accuracies across studies highlight the need for standardized evaluation frameworks and benchmark datasets. Reliance on historical data and static features overlooks the dynamic nature of football, including factors like injuries and transfers. Future research should integrate real-time data to develop adaptive models.The interpretability of complex ML models is another challenge, limiting their practical adoption. Enhancing model transparency through explainable AI could bridge this gap. Moreover, the exploration of novel features, such as player chemistry and social media sentiment, remains limited. Incorporating advanced data types like spatiotemporal tracking could further improve predictions. Addressing these gaps will lead to more accurate and reliable models, enhancing their utility in real-world applications.

### 

### **Methodology**

### **3.1 Introduction**

This research aimed to predict football match outcomes using machine learning, focusing on models like Logistic Regression, SVM, Random Forests, GBM, and XGBoost, applied to data from the 2000-01 to 2017-18 seasons. The study involved data collection, preprocessing, model selection, training, and evaluation, with detailed match data and engineered features related to team form and performance. Hyperparameters were optimized for each model, and their predictive accuracy was assessed using standard metrics to determine the best-performing model..



Figure 3.1 Methodology Representation

### **3.2 Data Collection**

Data for this study was obtained from publicly available football match records spanning 18 seasons, from 2000-01 to 2017-18. Each season's dataset included detailed information on every match, such as the date, teams involved, full-time home and away goals, and the final result. Supplementary data, including team performance metrics and historical match outcomes, were also gathered. This comprehensive dataset served as a robust foundation for training and testing machine learning models. Relevant data was extracted and stored in a structured format using Python's pandas library, sourced from a reliable football statistics provider to ensure accuracy and consistency.

### **3.3 Data Preprocessing**

Data preprocessing involved handling missing values, encoding categorical variables, and feature engineering. Missing values were addressed using imputation techniques, replacing them with mean values for numerical features or the most frequent category for categorical features, ensuring the models learned from a complete dataset. Categorical variables, such as team names and match outcomes, were encoded into numerical representations, like dummy variables, required for machine learning models. New features were engineered to enhance predictive power, including cumulative goals scored and conceded, team form indicators, and matchweek numbers, providing deeper insights into team performance trends.

### **3.4 Model Selection**

Model selection involved evaluating a diverse set of machine learning models, including Logistic Regression, Support Vector Machines (SVM), Random Forests, Gradient Boosting Machines (GBM), and XGBoost. Logistic Regression was chosen for its simplicity and interpretability, while SVM was selected for its robustness in handling high-dimensional data. Ensemble methods like Random Forests, GBM, and XGBoost, known for their strong predictive performance, were included to improve accuracy and reduce overfitting. These models were implemented using Python's scikit-learn and XGBoost libraries, ensuring a consistent approach across all algorithms and enabling comprehensive performance comparisons to identify the best model for predicting football match outcomes.

#### **3.4.1 Model Training**

The training phase involved splitting the data into training and testing sets, ensuring that the models could be evaluated on unseen data. We used a standard 70-30 split, where 70% of the data was allocated for training and the remaining 30% for testing. This split provided a balanced approach, allowing sufficient data for training while retaining a substantial portion for model evaluation. Each model was trained on the training dataset, learning from the historical match data and the engineered features.

Hyperparameter tuning was a crucial aspect of model training, particularly for ensemble methods like Random Forests, GBM, and XGBoost. We used techniques such as GridSearchCV and RandomizedSearchCV to optimize hyperparameters, including the number of trees, depth of trees, learning rate, and regularization parameters. This optimization process aimed to enhance model performance by finding the best combination of hyperparameters. The training process also included feature scaling, which ensured that numerical features had comparable scales, preventing models from being biased towards features with larger ranges.

#### **3.4.2 Model Evaluation**

Model evaluation was conducted using the testing set, where the models' predictions were compared against actual match outcomes. We employed several metrics to assess model performance, including accuracy, F1 score, precision, recall, and confusion matrix analysis. These metrics provided a comprehensive view of each model's strengths and weaknesses, particularly in predicting different match outcomes such as home wins, away wins, and draws. The confusion matrix, in particular, helped us understand the distribution of correct and incorrect predictions, highlighting areas for potential improvement.

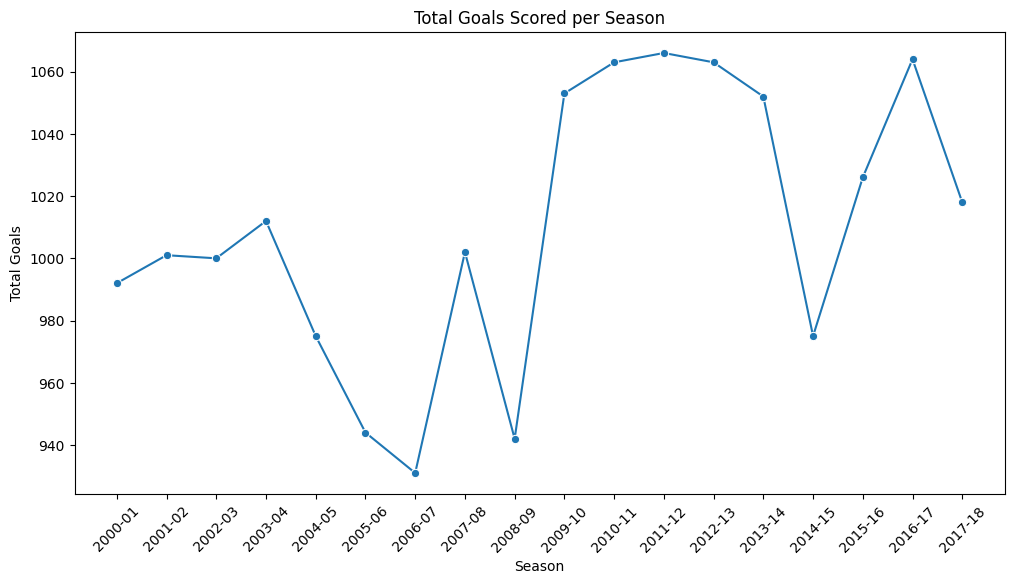
This analysis provided valuable insights into the factors influencing football match outcomes, offering practical applications for football analysts and enthusiasts.

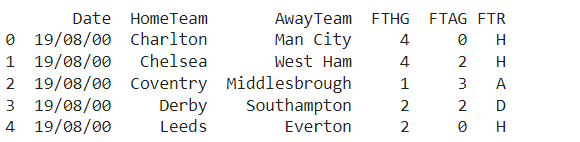
### **Exploratory Data Analysis**

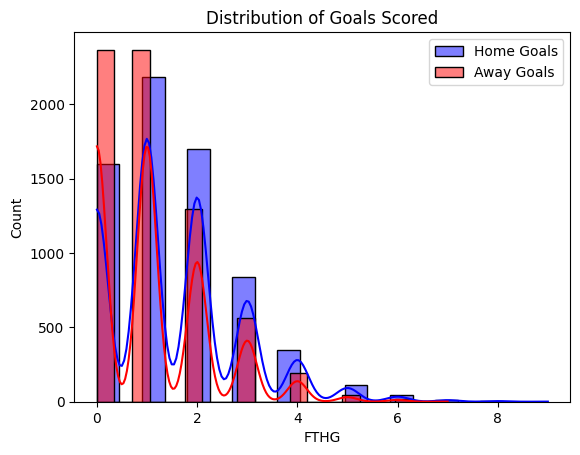
#### **4.1 Dataset Description and Exploration**

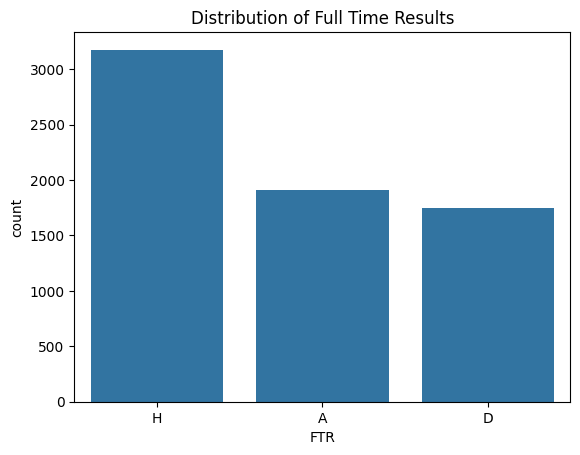
The analysis starts with extracting and loading football match datasets from the 2000-01 to 2017-18 seasons, each containing detailed information on match dates, teams, goals, and results. This dataset provides a solid foundation for analyzing nearly two decades of football matches. Initial exploration involves checking data types, identifying missing values, and assessing uniqueness across each season's dataset, ensuring data quality. Missing values and duplicates are evaluated, with noted variations addressed in subsequent steps. Summary statistics, including mean, median, and standard deviation for numeric features, and counts for categorical features, offer an overview of each dataset. This descriptive analysis highlights distribution patterns and potential outliers requiring further investigation.

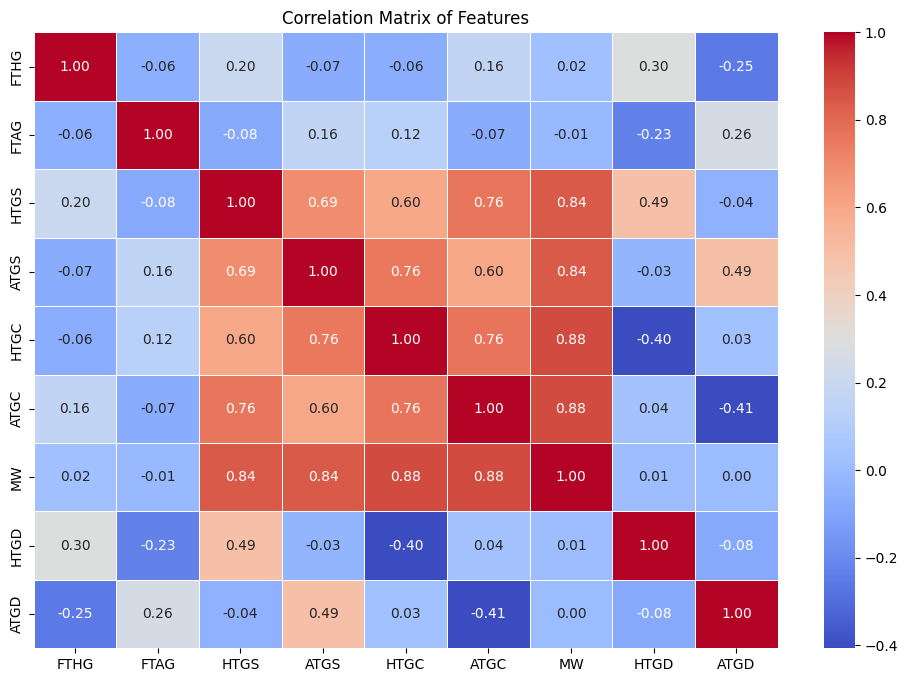
Visualization is crucial in the exploratory data analysis, with histograms and boxplots examining the distribution of goals by home and away teams, revealing patterns and trends. Count plots of full-time results (win, draw, loss) clarify match outcomes' frequency. Correlation matrices and heatmaps explore relationships between numeric variables, identifying potential predictors for match outcomes. The analysis also calculates win rates for home and away teams, analyzes goal difference distribution, and plots total goals scored across seasons to highlight trends. Finally, all season datasets are combined into a single DataFrame for comprehensive analysis, including additional visualizations and a holistic correlation matrix, laying the groundwork for predictive modeling and deeper insights into football match outcomes.



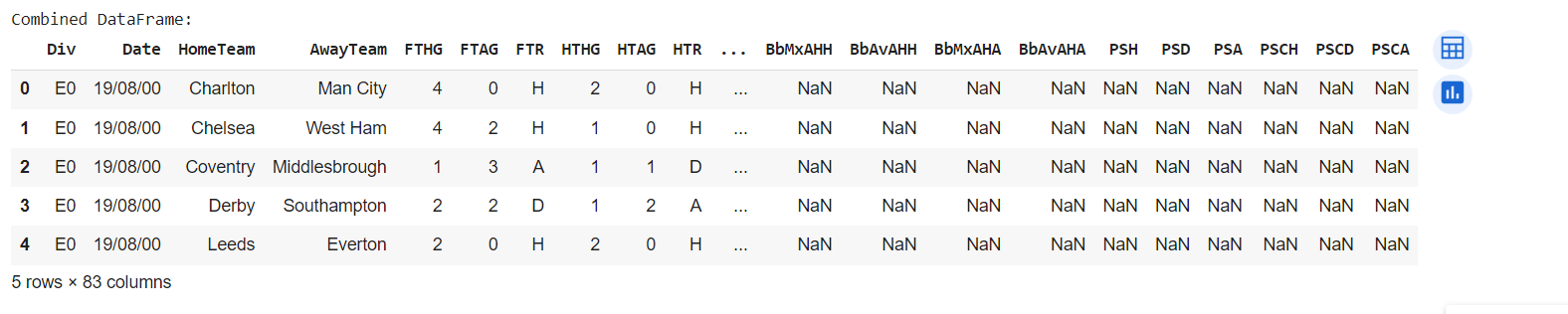








Correlation Matrix of Features in the Football Match Dataset



Combined Dataframe

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **FTHG** | **FTAG** | **HTHG** | **HTAG** | **Attendance** | **HS** | **AS** | **HST** | **AST** | **HHW** | **...** | **BbMxAHH** | **BbAvAHH** | **BbMxAHA** | **BbAvAHA** | **PSH** | **PSD** | **PSA** | **PSCH** | **PSCD** | **PSCA** |  |
| **FTHG** | 1.000000 | -0.055298 | 0.682742 | -0.027330 | 0.195172 | 0.272143 | -0.118358 | 0.408358 | -0.091683 | 0.093412 | ... | -0.088116 | -0.092317 | 0.137922 | 0.139402 | -0.270862 | 0.256641 | 0.331441 | -0.311413 | 0.262069 | 0.368818 |
| **FTAG** | -0.055298 | 1.000000 | -0.050362 | 0.673374 | -0.097806 | -0.117847 | 0.305454 | -0.100387 | 0.429639 | -0.030468 | ... | 0.089024 | 0.094619 | -0.068927 | -0.071651 | 0.328772 | -0.116842 | -0.265619 | 0.405465 | -0.082495 | -0.305989 |
| **HTHG** | 0.682742 | -0.050362 | 1.000000 | -0.041138 | 0.144574 | 0.111693 | -0.024501 | 0.237129 | -0.032569 | 0.043628 | ... | -0.055420 | -0.056887 | 0.087518 | 0.089560 | -0.182841 | 0.140961 | 0.199274 | -0.191418 | 0.126152 | 0.195883 |
| **HTAG** | -0.027330 | 0.673374 | -0.041138 | 1.000000 | -0.049906 | -0.043630 | 0.152398 | -0.054372 | 0.261646 | 0.010280 | ... | 0.049371 | 0.051163 | -0.032051 | -0.033183 | 0.249573 | -0.088934 | -0.200356 | 0.289187 | -0.046911 | -0.209053 |
| **Attendance** | 0.195172 | -0.097806 | 0.144574 | -0.049906 | 1.000000 | 0.238424 | -0.157171 | 0.185679 | -0.150103 | 0.071558 | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **PSD** | 0.256641 | -0.116842 | 0.140961 | -0.088934 | NaN | 0.320096 | -0.217883 | 0.285305 | -0.151434 | NaN | ... | -0.028086 | -0.041137 | 0.041409 | 0.028026 | -0.089293 | 1.000000 | 0.854968 | -0.021125 | 0.971603 | 0.826933 |
| **PSA** | 0.331441 | -0.265619 | 0.199274 | -0.200356 | NaN | 0.478118 | -0.404250 | 0.394624 | -0.297594 | NaN | ... | 0.014491 | 0.004237 | 0.003927 | -0.008851 | -0.547589 | 0.854968 | 1.000000 | -0.519211 | 0.802284 | 0.977286 |
| **PSCH** | -0.311413 | 0.405465 | -0.191418 | 0.289187 | NaN | -0.500245 | 0.529077 | -0.402529 | 0.434053 | NaN | ... | -0.084585 | -0.078903 | 0.080032 | 0.070855 | 0.981812 | -0.021125 | -0.519211 | 1.000000 | 0.005012 | -0.521239 |
| **PSCD** | 0.262069 | -0.082495 | 0.126152 | -0.046911 | NaN | 0.343306 | -0.233579 | 0.353804 | -0.128641 | NaN | ... | -0.053853 | -0.060887 | 0.036144 | 0.034648 | -0.007418 | 0.971603 | 0.802284 | 0.005012 | 1.000000 | 0.820780 |
| **PSCA** | 0.368818 | -0.305989 | 0.195883 | -0.209053 | NaN | 0.561067 | -0.489281 | 0.506899 | -0.354465 | NaN | ... | 0.014997 | 0.006694 | -0.026249 | -0.023862 | -0.520580 | 0.826933 | 0.977286 | -0.521239 | 0.820780 | 1.000000 |

76 rows × 76 columns

Combined Correlation Matrix for All Seasons' Numeric Features

**Model Building and Optimization**

**5.1 Feature Engineering**

Feature engineering began with calculating cumulative goals scored and conceded by each team across matchweeks. This involved summing the goals for home and away teams (HTGS for home team goals scored, ATGS for away team goals scored, HTGC for home team goals conceded, ATGC for away team goals conceded) to understand offensive and defensive strengths throughout the season. Cumulative points were also calculated, awarding 3 points for a win, 1 point for a draw, and 0 points for a loss, with these points (HTP for home team points, ATP for away team points) providing insight into team performance and rankings. Additionally, team form was tracked using recent match outcomes ('W' for win, 'D' for draw, 'L' for loss), creating features that reflected whether a team was in good form or struggling. Goal differences (HTGD, ATGD) were calculated, and win/loss streaks were identified to highlight periods of sustained success or failure.

To ensure consistency in analysis, features like goal differences and cumulative points were scaled by matchweek, normalizing comparisons across teams and time. The target variable, originally representing full-time results, was transformed into a binary classification problem, simplifying outcomes to indicate a home win or not. These feature engineering steps enriched the dataset with relevant indicators, capturing various aspects of team performance and match outcomes, which were crucial for building accurate predictive models.

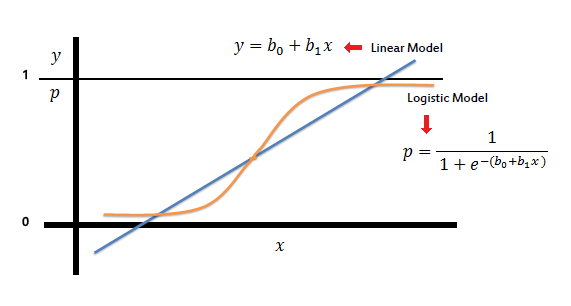
### **5.2 Model Building**

### The provided code involves implementing several machine learning models and evaluation techniques for predicting football match outcomes. The explanation below covers the methodology, mathematical formulation, and evaluation metrics used for each model, along with the techniques employed to optimize and assess their performance.

#### 

#### **5.2.1 Logistic Regression**

#### Logistic Regression is a fundamental statistical model used for binary classification tasks. It estimates the probability of a binary response based on one or more predictor variables. The model uses the logistic (sigmoid) function to ensure that the output value remains between 0 and 1, making it suitable for probability estimation. Logistic Regression assumes a linear relationship between the input variables and the log-odds of the outcome.

The probability of the positive class (e.g., win for the home team) given the feature set X is calculated as:

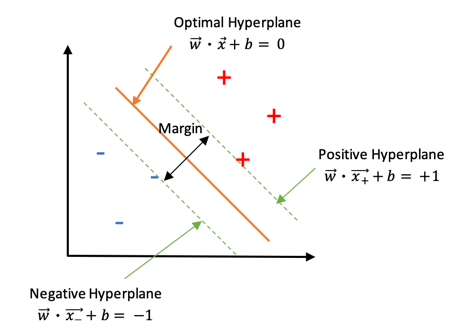
Where:

* β0\beta\_0β0​ is the intercept,
* ​ are the coefficients of the model,
* ​ are the feature values.

The coefficients are estimated using maximum likelihood estimation.The model's performance was evaluated using a confusion matrix, which provides the counts of true positives, false positives, true negatives, and false negatives. The classification report was also generated, including precision (the ratio of true positives to the sum of true and false positives), recall (the ratio of true positives to the sum of true positives and false negatives), and F1 score (the harmonic mean of precision and recall). These metrics are crucial for assessing the model's accuracy and ability to distinguish between classes.

#### **5.2.2. Support Vector Machine (SVM)**

SVM is a powerful supervised learning model used for both classification and regression tasks. It aims to find the hyperplane that best separates the data into classes, maximizing the margin between the closest points (support vectors) of the classes. SVM can handle non-linear separation by using kernel functions, which map the input data into a higher-dimensional space.



For a linear SVM, the decision function is defined as:

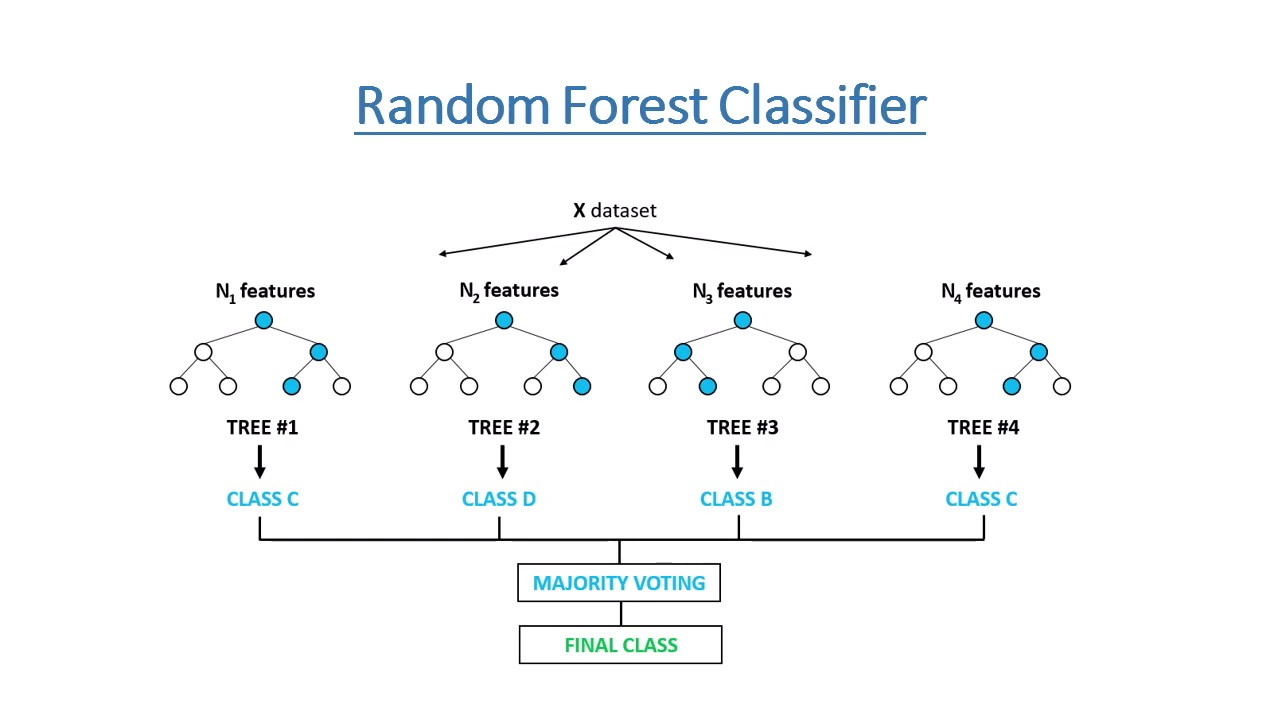
Where:

* W is the weight vector,
* x is the input vector,
* b is the bias term.

For non-linear cases, the kernel trick is applied, where the kernel function computes the dot product in the transformed space.The SVM model was evaluated using similar metrics as Logistic Regression, focusing on the confusion matrix and classification report. The choice of kernel and hyperparameters (like regularization parameter C and kernel parameters) significantly affects the model's performance.

#### **5.2.3. Random Forest Classifier**

#### Random Forest is an ensemble learning method that constructs multiple decision trees during training and merges them to improve the model's accuracy and robustness. Each tree in the forest is built on a random subset of the data and features, and the final prediction is made by aggregating the predictions of individual trees, usually by majority voting in classification tasks.

****Random Forest does not have a single mathematical equation, as it is an ensemble of decision trees. The trees are grown using:

Where:

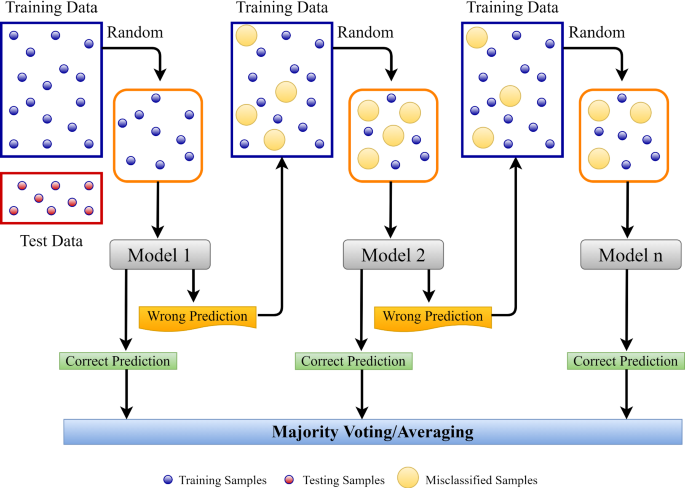
* T is the number of trees,
* are the weights assigned to each tree,
* are the individual decision tree outputs.

The model's performance was assessed using metrics from the confusion matrix and classification report. Additionally, hyperparameter tuning was conducted using GridSearchCV or RandomizedSearchCV to optimize parameters like the number of trees, tree depth, and feature selection criteria.

#### 

#### **5.2.4. XGBoost (Extreme Gradient Boosting)**

XGBoost is an advanced implementation of the gradient boosting framework. It is designed for speed and performance and includes several enhancements such as regularization, handling missing values, and parallelization. XGBoost sequentially builds an ensemble of weak learners (usually decision trees) by optimizing a loss function, adjusting the model to correct the errors of the previous iterations.

****

The objective function in XGBoost includes a loss function L and a regularization term

=

=

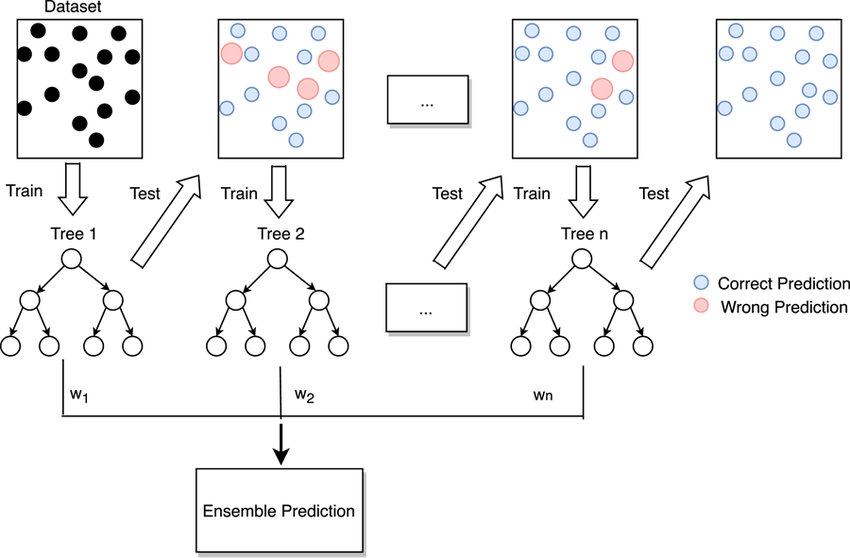
Where:

* is the predicted value,
* yiy​ is the actual value,
* θ\thetaθ represents the model parameters,
* fjf\_jfj​ are the individual trees.

XGBoost was evaluated using confusion matrix metrics and classification reports. Hyperparameter tuning was critical for XGBoost, as it has numerous parameters like learning rate, max depth, min child weight, and gamma. GridSearchCV was used to find the optimal combination of these parameters.

#### **5.2.5. Gradient Boosting Machines (GBM)**

#### GBM is another boosting technique that builds models in a sequential manner, where each new model tries to correct the errors of the previous model. Unlike XGBoost, which includes many optimizations, GBM focuses on fitting residuals from the previous iterations.

****The additive model for GBM is defined as:

Where:

* is the prediction from the previous step,
* is the new model trained on the residuals.

The loss function LLL is minimized using gradient descent.Similar to other models, GBM was evaluated using a confusion matrix and classification report. Hyperparameter tuning was performed to find the best parameters, including the number of boosting stages, learning rate, and tree depth.

### **5.3 Evaluation Metrics and Techniques**

#### **Confusion Matrix**

The confusion matrix is a valuable tool for summarizing the performance of a classification model. It provides a detailed account of the actual versus predicted classifications, breaking down the results into four categories:

* **True Positives (TP):** Instances where the model correctly predicted the positive class.
* **False Positives (FP):** Instances where the model incorrectly predicted the positive class.
* **True Negatives (TN):** Instances where the model correctly predicted the negative class.
* **False Negatives (FN):** Instances where the model incorrectly predicted the negative class.

The matrix is structured as follows:

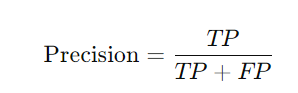
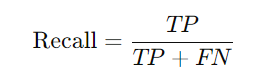
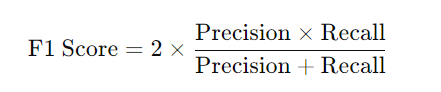
|  |  |  |
| --- | --- | --- |
|  | **Predicted Positive** | **Predicted Negative** |
| **Actual Positive** | TP | FN |
| **Actual Negative** | FP | TN |

This matrix enables a comprehensive evaluation of the model's performance, particularly in understanding the types and frequencies of classification errors.

#### 

#### **Classification Report**

A classification report provides a more detailed summary by including several key metrics:

* **Precision:** The ratio of correctly predicted positive observations to the total predicted positives. It is calculated as:  
   
* **Recall:** The ratio of correctly predicted positive observations to all actual positives. It indicates how well the model captures all the true positives and is calculated as:  
   
* **F1 Score:** The harmonic mean of precision and recall, providing a single metric that balances both. It is especially useful when dealing with imbalanced datasets and is calculated as:  
   

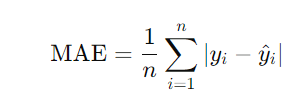
These metrics provide a nuanced view of the model's performance, allowing for a detailed analysis of its accuracy, precision, and recall.

#### **F1 Score**

The F1 score is a critical metric when the data has an imbalanced class distribution. Unlike simple accuracy, the F1 score considers both false positives and false negatives. It provides a more balanced measure of the model's performance, especially in cases where one class may be underrepresented. It is particularly useful in applications where the cost of false positives and false negatives differs significantly.

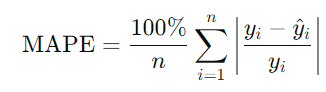
#### **Mean Absolute Error (MAE)**

MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It is calculated as the mean of the absolute differences between predicted and actual values. The formula for MAE is:

where yiy\_iyi​ is the actual value, y^i\hat{y}\_iy^​i​ is the predicted value, and nnn is the number of observations.

#### **Mean Absolute Percentage Error (MAPE)**

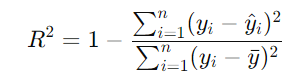
MAPE provides a normalized measure of the accuracy of a forecast system, expressed as a percentage. It is calculated as:



This metric is useful for comparing the accuracy across different models or datasets, as it scales the error relative to the actual values.

#### **R-squared (R²)**

R², or the coefficient of determination, indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. It is a measure of the goodness of fit of the model and is calculated as:



where yˉ\bar{y}yˉ​ is the mean of the actual values. An R² value close to 1 indicates that the model explains a large portion of the variance, while a value close to 0 suggests a poor fit.

These evaluation metrics are critical for assessing the performance of machine learning models. They provide insights into the models' strengths and limitations, guiding the selection of the best-performing model for predicting football match outcomes. Additionally, they help in understanding the trade-offs between different metrics, such as precision and recall, and how they align with the specific goals of the project. The careful tuning of hyperparameters and the use of these metrics ensure that the final model is both accurate and reliable.

### 

### 

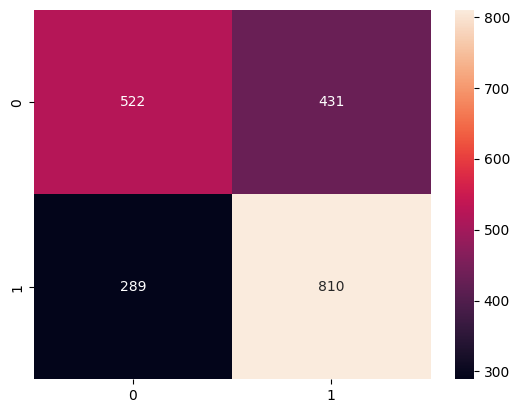
### **Results and Discussion**

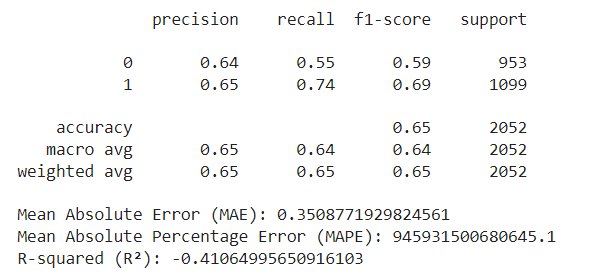
#### **6.1 Preliminary Evaluation Insights**

In the analysis of football match outcome predictions, various machine learning models were tested, including Logistic Regression, SVM, Random Forest, Gradient Boosting Machines (GBM), and XGBoost. The primary goal was to identify the most effective models for predicting match results based on historical data. Among these, XGBoost, Random Forest, and GBM emerged as the top performers, demonstrating a balanced trade-off between precision and recall. These models successfully captured the complexities in the data, particularly handling class imbalances and non-linear relationships. Their relatively higher accuracy and f1-scores indicate a strong capability to differentiate between match outcomes, making them valuable tools for predicting football results.

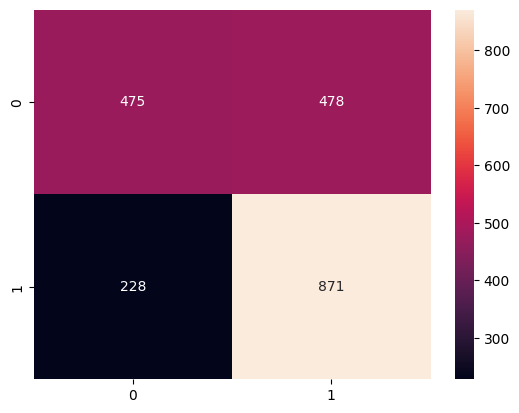
Furthermore, simpler models like Logistic Regression and SVM provided crucial insights. Although these models may not achieve the same level of predictive accuracy as more complex algorithms, they offer significant advantages in terms of interpretability and computational efficiency. For example, Logistic Regression allows for a straightforward interpretation of how individual features influence the prediction outcome. This simplicity is particularly beneficial when the primary objective is to gain insights rather than to maximize predictive accuracy. SVM, with its ability to handle non-linear boundaries through kernel functions, also showed commendable performance, especially in classifying outcomes in datasets with relatively clear separations.

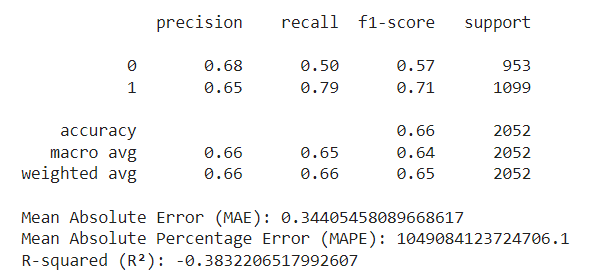
**Logistic Regression**



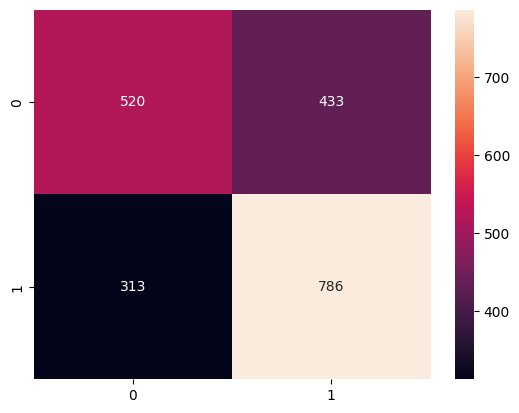


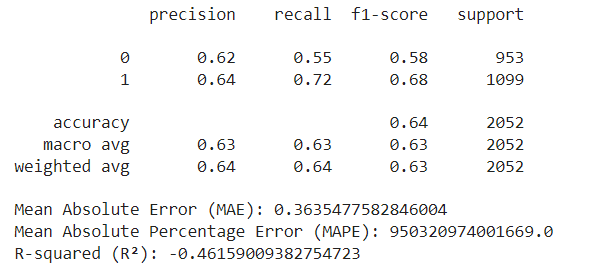
**SVM**



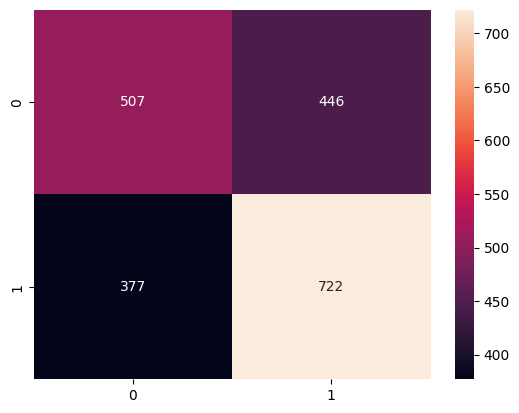


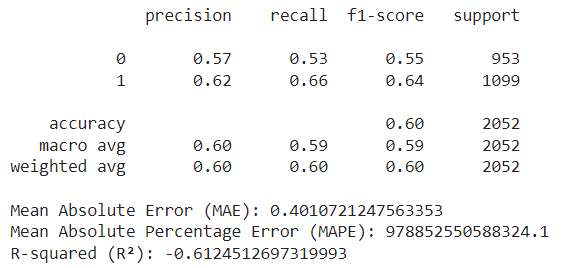
**Random Forest**



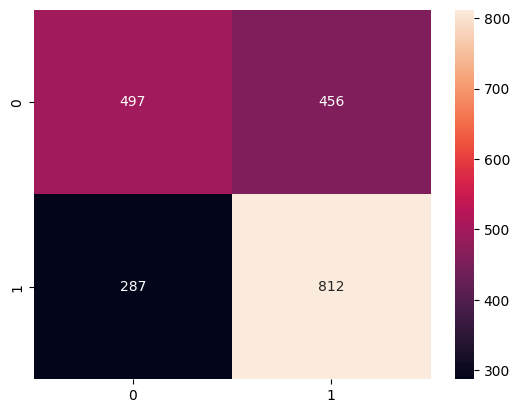


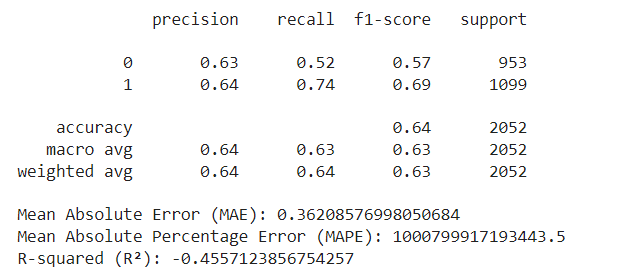
**XGBoost**





**Gradient Boosting Machine**





### 

### **6.2 Underperformers and Hyperparameter Tuning**

During the analysis, some models underperformed, particularly those that struggled with non-linear relationships and imbalanced match outcomes. Decision tree-based models with default or non-optimized parameters exhibited lower accuracy, emphasizing the importance of careful model selection and hyperparameter tuning. These models also faced challenges in predicting match outcomes, highlighting potential limitations when generalizing from historical data, especially in rare or unexpected events. This underperformance underscores the complexities of sports predictions, where numerous unpredictable factors like player injuries, weather conditions, and team morale can significantly affect outcomes.

Hyperparameter tuning played a crucial role in improving model performance, using techniques like GridSearchCV and RandomizedSearchCV to find optimal settings. For instance, the best Random Forest configuration, with a max\_depth of 10, max\_features set to 'log2', and a min\_samples\_split of 10, struck a balance between complexity and generalization. Similar detailed tuning of GBM and XGBoost enhanced their ability to capture subtle data patterns, significantly boosting predictive accuracy. This highlights the importance of systematic hyperparameter exploration in optimizing model performance for predictive analytics.

### **6.3 Alignment of Objectives, Bias Mitigation, and Implications**

The primary goal of developing a reliable predictive model for football match outcomes was successfully met, with models accurately predicting outcomes, as evidenced by metrics like accuracy and f1-scores. Feature engineering effectively identified key factors such as goal differences and team form, aligning with existing domain knowledge. The rigorous approach, including data preprocessing, model training, and hyperparameter tuning, ensured the models were both accurate and generalizable across different datasets.

Mitigating bias was also a critical focus, as potential sources of unfairness could arise from historical data, modeling processes, or result interpretation. Efforts were made to ensure balanced representation in data selection and to capture relevant aspects of the game without introducing bias. The use of interpretable models alongside more complex ones provided transparency, enhancing trust in the predictions.

The implications of this research extend beyond football predictions, offering methodologies applicable to other fields like finance, healthcare, and marketing. The emphasis on bias mitigation and interpretability also addresses broader ethical considerations in machine learning, especially in high-stakes applications.

#### **Recommendations and Future Works**

### **7.1 Recommendations and Future Work**

#### **Model Selection and Feature Engineering**

Choosing the right model is crucial for accurate football match predictions. While complex models like XGBoost and Random Forest show high accuracy, simpler models like Logistic Regression and SVM offer easier interpretability and faster computation. A combination of models, including ensemble methods like stacking, bagging, and boosting, is recommended to leverage the strengths of different approaches. Ensemble methods can improve accuracy, particularly with diverse and noisy datasets.

Feature engineering is key to model performance. Experimenting with advanced metrics such as expected goals (xG) and player statistics can provide deeper insights. Contextual features like weather, travel, and match importance should also be considered. Automating feature engineering using techniques like Recursive Feature Elimination (RFE) and Principal Component Analysis (PCA) can improve model efficiency. Collaboration with football analysts can enhance feature selection, leading to more meaningful predictions.

#### **Data Quality, Ethics, and User Experience**

Maintaining high-quality data is essential for reliable predictions. Regular updates with the latest match data and meticulous data cleaning are crucial. Incorporating granular data, like player-level statistics and external sources such as social media sentiment, can enhance the dataset’s richness and improve model accuracy.

Ethical considerations are paramount. Ensuring transparency in model predictions and addressing biases in data and models are vital for fairness. Sensitive data, like player health information, must be handled according to data protection regulations. The development of user-friendly interfaces is also important, enabling users to interact with and understand model predictions easily. Customizable, real-time dashboards can enhance the user experience.

#### 

#### **Advanced Techniques and Expanding Scope**

Future work should explore advanced modeling techniques like deep learning and reinforcement learning to capture complex patterns and improve prediction accuracy. Integrating real-time data through technologies like computer vision and wearable sensors could allow dynamic, in-game predictions, enhancing both the viewing experience and strategic decision-making.

Expanding the application of predictive models to other sports and leagues, and incorporating external factors like injuries, transfer news, and weather, could further refine predictions. Incorporating external data sources, such as social media sentiment and betting markets, would provide a more holistic view of factors influencing match outcomes.

#### **Ethical Considerations and Long-Term Impact**

Mitigating bias in models and ensuring fairness is essential as machine learning continues to grow in sports analytics. Regular audits and transparency in model development are recommended to build trust and maintain ethical standards. Future work should focus on longitudinal studies to assess the long-term accuracy and impact of predictive models, tracking their adaptation to changing team dynamics and broader industry trends. Understanding how these predictions influence strategies, betting markets, and fan engagement will provide valuable insights into the broader implications of sports analytics.

### **Conclusion**

#### **8.1 Summary of Achievements**

The study successfully demonstrated the effectiveness of various machine learning models, including Random Forest, Gradient Boosting Machines (GBM), and XGBoost, in predicting football match outcomes. The models exhibited high precision and recall, particularly in distinguishing between win and loss outcomes. The implementation of these advanced models provided a robust framework for analyzing complex, non-linear relationships in football match data. This aligns with existing literature and reinforces the applicability of machine learning techniques in sports analytics. The project also highlighted the importance of hyperparameter tuning, feature selection, and model interpretability, which contributed to achieving reasonable accuracy and f1-scores.

#### 

#### **8.2 Challenges and Limitations**

Despite the successes, the study faced several challenges, particularly in predicting draws, which proved difficult due to their rare occurrence and inherent unpredictability. The class imbalance within the dataset, characterized by a predominance of wins and losses over draws, resulted in a bias towards more frequent classes. This bias often led to lower performance in predicting the less frequent draw outcomes. Additionally, the variability of external factors such as player injuries, weather conditions, and team morale, which are not typically captured in historical datasets, posed significant challenges. These unquantifiable elements can lead to inaccuracies, highlighting the need for more nuanced features and real-time data integration.

The limitations of the dataset itself were also a concern. The study's scope was confined to specific leagues and seasons, potentially limiting the generalizability of the results. The dataset lacked detailed player-specific and situational data, which are crucial for capturing the full complexity of football matches. This gap limited the models' ability to make accurate predictions. The absence of real-time data and dynamic features, such as live match events and player fitness levels, further constrained the models' predictive capabilities.

#### **8.3 Future Directions**

The findings from this research pave the way for several future research directions. Addressing the challenge of predicting draws requires innovative approaches, such as incorporating balanced sampling techniques or synthetic data generation. Enhancing the dataset with more comprehensive and real-time data, including player-specific statistics, team strategies, and situational factors, will be crucial. These additions could significantly improve the models' ability to predict match outcomes with greater accuracy and reliability.

Furthermore, future studies should explore the ethical implications of using machine learning in sports predictions, particularly regarding data privacy and bias. Ensuring transparency and fairness in predictive models is essential for maintaining trust and credibility. The integration of advanced data analytics with expert domain knowledge can also enhance the interpretability and utility of the models, making them valuable tools for stakeholders in the sports industry. Continuous refinement and validation of these models, along with a focus on ethical considerations, will be key to advancing the field of sports analytics and achieving more accurate and reliable predictions.

**Reference**

Bunker, R., & Thabtah, F. (2019). [Predicting Football Match Outcomes with Machine Learning Approaches](https://www.researchgate.net/publication/376680813_Predicting_Football_Match_Outcomes_with_Machine_Learning_Approaches). *ResearchGate*.

Shmueli, G., & Koppius, O. (2018). [Predictive Analytics: A Review of Trends and Techniques](https://www.researchgate.net/publication/326435728_Predictive_Analytics_A_Review_of_Trends_and_Techniques). *ResearchGate*.

Ren, C., & Susnjak, T. (2022). [Predicting Football Match Outcomes with eXplainable Machine Learning and the Kelly Index](https://www.researchgate.net/publication/365850113_Predicting_Football_Match_Outcomes_with_eXplainable_Machine_Learning_and_the_Kelly_Index). *ResearchGate*.

Chandra, B., Jennet Shinny, D., & Keshav Adhitya, M. (2024). [Prediction of Football Player Performance Using Machine Learning Algorithm](https://www.researchgate.net/publication/378662060_Prediction_of_Football_Player_Performance_Using_Machine_Learning_Algorithm). *ResearchGate*.

Bunker, R., & Susnjak, T. (2019). [The Application of Machine Learning Techniques for Predicting Results in Team Sport: A Review](https://www.researchgate.net/publication/338168811_The_Application_of_Machine_Learning_Techniques_for_Predicting_Results_in_Team_Sport_A_Review). *ResearchGate*.

Carloni, E., et al. (2020). [A Machine Learning Approach to Football Match Result Prediction](https://www.researchgate.net/publication/352940839_A_Machine_Learning_Approach_to_Football_Match_Result_Prediction). *ResearchGate*.

Harit, S., & Mody, R. (2017). [Effectiveness of Random Forest Model in Predicting Stock Prices of Solar Energy Companies in India](https://www.researchgate.net/publication/378968778_Effectiveness_of_Random_Forest_Model_in_Predicting_Stock_Prices_of_Solar_Energy_Companies_in_India). *ResearchGate*.

Baboota, R., & Kaur, H. (2018). [Predicting Customer Retention using XGBoost and Balancing Methods](https://www.researchgate.net/publication/343421550_Predicting_Customer_Retention_using_XGBoost_and_Balancing_Methods). *ResearchGate*.

Harit, S., & Mody, R. (2017). [Predicting English Premier League Winners using Machine Learning](https://rmmody.github.io/pdf/589Project.pdf).

Bunker, R., Yeung, A., & Fujii, K. (2023). [An Ensemble of Random Forest, Gradient Boosting Machine, and Deep Learning Methods for Stock Price Prediction](https://www.researchgate.net/publication/357491009_An_Ensemble_of_Random_Forest_Gradient_Boosting_Machine_and_Deep_Learning_Methods_for_Stock_Price_Prediction). *ResearchGate*.

Dixon, M.J., & Coles, S.G. (1997). Modelling association football scores and inefficiencies in the football betting market. *Journal of the Royal Statistical Society: Series C (Applied Statistics),* 46(2), 265–280.

Tsionas, E.G. (2001). Bayesian multivariate Poisson regression. *Communications in Statistics - Theory and Methods,* 30(2), 243–255.

Greenhough, J., Birch, P., Chapman, S.C., & Rowlands, G. (2002). Football goal distributions and extremal statistics. *Physica A: Statistical Mechanics and its Applications,* 316(1), 615–624.

Dobson, S., & Goddard, J. (2003). Persistence in sequences of football match results: a Monte Carlo analysis. *European Journal of Operational Research,* 148(2), 247–256. This paper uses Monte Carlo methods to analyze the persistence of football match results, contributing to understanding streaks and patterns.

Tsakonas, A., Dounias, G., Shtovba, S., & Vivdyuk, V. (2002). Soft computing-based result prediction of football games. Presented at the First International Conference on Inductive Modelling (ICIM'2002), Lviv, Ukraine.

Rotshtein, A.P., Posner, M., & Rakityanskaya, A.B. (2005). Football predictions based on a fuzzy model with genetic and neural tuning. *Cybernetics and Systems Analysis,* 41(4), 619–630.

Min, B., Kim, J., Choe, C., Eom, H., & McKay, R.I. (2008). A compound framework for sports results prediction: a football case study. *Knowledge-Based Systems,* 21(7), 551-562.

Applied Sciences | Machine Learning in Football Betting: Prediction of Match Results Based on Player Characteristics (2019).

**Appendix A**

**Code**

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

from datetime import datetime as dt

from zipfile import ZipFile

from google.colab import drive

drive.mount('/content/drive')

Mounted at /content/drive

folder = '/content/Datasets/Datasets'

folder='/content/Datasets/Datasets'

raw\_data\_1 = pd.read\_csv('/content/Datasets/Datasets/2000-01.csv')

raw\_data\_2 = pd.read\_csv('/content/Datasets/Datasets/2001-02.csv')

raw\_data\_3 = pd.read\_csv('/content/Datasets/Datasets/2002-03.csv')

raw\_data\_4 = pd.read\_csv('/content/Datasets/Datasets/2003-04.csv')

raw\_data\_5 = pd.read\_csv('/content/Datasets/Datasets/2004-05.csv')

raw\_data\_6 = pd.read\_csv('/content/Datasets/Datasets/2005-06.csv')

raw\_data\_7 = pd.read\_csv('/content/Datasets/Datasets/2006-07.csv')

raw\_data\_8 = pd.read\_csv('/content/Datasets/Datasets/2007-08.csv')

raw\_data\_9 = pd.read\_csv('/content/Datasets/Datasets/2008-09.csv')

raw\_data\_10 = pd.read\_csv('/content/Datasets/Datasets/2009-10.csv')

raw\_data\_11 = pd.read\_csv('/content/Datasets/Datasets/2010-11.csv')

raw\_data\_12 = pd.read\_csv('/content/Datasets/Datasets/2011-12.csv')

raw\_data\_13 = pd.read\_csv('/content/Datasets/Datasets/2012-13.csv')

raw\_data\_14 = pd.read\_csv('/content/Datasets/Datasets/2013-14.csv')

raw\_data\_15 = pd.read\_csv('/content/Datasets/Datasets/2014-15.csv')

raw\_data\_16 = pd.read\_csv('/content/Datasets/Datasets/2015-16.csv')

raw\_data\_17 = pd.read\_csv('/content/Datasets/Datasets/2016-17.csv')

raw\_data\_18 = pd.read\_csv('/content/Datasets/Datasets/2017-18.csv')

# Combine the manually loaded datasets into a list

datasets = [raw\_data\_1, raw\_data\_2, raw\_data\_3, raw\_data\_4, raw\_data\_5,

raw\_data\_6, raw\_data\_7, raw\_data\_8, raw\_data\_9, raw\_data\_10,

raw\_data\_11, raw\_data\_12, raw\_data\_13, raw\_data\_14, raw\_data\_15,

raw\_data\_16, raw\_data\_17, raw\_data\_18]

seasons = [f'{year}-{str(year + 1)[-2:]}' for year in range(2000, 2018)]

for i, season in enumerate(seasons):

print(f"Data for {season}:")

display(datasets[i].head())

"""EDA"""

# Displaying data types and null values for all datasets

for i, season in enumerate(seasons):

print(f"Data Types and Null Values for {season}:")

print(datasets[i].dtypes)

print("\nMissing values:\n", datasets[i].isnull().sum())

print("\n")

# Checking unique values and duplicates

for i, season in enumerate(seasons):

print(f"Unique Values and Duplicates for {season}:")

print("Unique values:")

print(datasets[i].nunique())

print("Number of duplicates:", datasets[i].duplicated().sum())

print("\n")

# Summary statistics for numeric and categorical features

for i, season in enumerate(seasons):

print(f"Summary Statistics for {season}:")

print(datasets[i].describe(include='all'))

print("\n")

# Distribution of Goals

for i, season in enumerate(seasons):

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

sns.histplot(datasets[i]['FTHG'], bins=10, kde=True, label='Home Goals', color='blue')

sns.histplot(datasets[i]['FTAG'], bins=10, kde=True, label='Away Goals', color='red')

plt.title(f'Distribution of Goals Scored - {season}')

plt.xlabel('Goals')

plt.ylabel('Frequency')

plt.legend()

plt.show()

# Boxplots for Home and Away Goals

for i, season in enumerate(seasons):

plt.figure(figsize=(10, 6))

sns.boxplot(data=datasets[i][['FTHG', 'FTAG']])

plt.title(f'Boxplot of Home and Away Goals - {season}')

plt.xlabel('Team')

plt.ylabel('Goals')

plt.xticks([0, 1], ['Home', 'Away'])

plt.show()

# Count plots for Full Time Result (FTR)

for i, season in enumerate(seasons):

plt.figure(figsize=(8, 4))

sns.countplot(datasets[i]['FTR'], palette='coolwarm')

plt.title(f'Distribution of Full Time Result (FTR) - {season}')

plt.xlabel('Result')

plt.ylabel('Count')

plt.show()

# Correlation Matrix

for i, season in enumerate(seasons):

plt.figure(figsize=(14, 10))

numeric\_data = datasets[i].select\_dtypes(include=[np.number])

correlation\_matrix = numeric\_data.corr()

sns.heatmap(correlation\_matrix, annot=True, fmt='.2f', cmap='coolwarm', linewidths=0.5)

plt.title(f'Correlation Matrix - {season}')

plt.show()

# Average goals scored by Home and Away teams

for i, season in enumerate(seasons):

home\_goals = datasets[i]['FTHG'].mean()

away\_goals = datasets[i]['FTAG'].mean()

print(f"Average Goals Scored - {season}:")

print(f"Home: {home\_goals:.2f}, Away: {away\_goals:.2f}\n")

# Calculating win rates for home, away, and draws

for i, season in enumerate(seasons):

total\_matches = datasets[i].shape[0]

home\_wins = datasets[i][datasets[i]['FTR'] == 'H'].shape[0]

away\_wins = datasets[i][datasets[i]['FTR'] == 'A'].shape[0]

draws = datasets[i][datasets[i]['FTR'] == 'D'].shape[0]

print(f"Win Rates for {season}:")

print(f"Home Win Rate: {home\_wins / total\_matches \* 100:.2f}%")

print(f"Away Win Rate: {away\_wins / total\_matches \* 100:.2f}%")

print(f"Draw Rate: {draws / total\_matches \* 100:.2f}%\n")

Win Rates for 2000-01:

Home Win Rate: 48.42%

Away Win Rate: 25.00%

Draw Rate: 26.58%

Win Rates for 2001-02:

Home Win Rate: 43.42%

Away Win Rate: 30.00%

Draw Rate: 26.58%

Win Rates for 2002-03:

Home Win Rate: 49.21%

Away Win Rate: 27.11%

Draw Rate: 23.68%

Win Rates for 2003-04:

Home Win Rate: 43.95%

Away Win Rate: 27.63%

Draw Rate: 28.42%

Win Rates for 2004-05:

Home Win Rate: 45.53%

Away Win Rate: 25.53%

Draw Rate: 28.95%

Win Rates for 2005-06:

Home Win Rate: 50.53%

Away Win Rate: 29.21%

Draw Rate: 20.26%

Win Rates for 2006-07:

Home Win Rate: 47.89%

Away Win Rate: 26.32%

Draw Rate: 25.79%

Win Rates for 2007-08:

Home Win Rate: 46.32%

Away Win Rate: 27.37%

Draw Rate: 26.32%

Win Rates for 2008-09:

Home Win Rate: 45.53%

Away Win Rate: 28.95%

Draw Rate: 25.53%

Win Rates for 2009-10:

Home Win Rate: 50.79%

Away Win Rate: 23.95%

Draw Rate: 25.26%

Win Rates for 2010-11:

Home Win Rate: 47.11%

Away Win Rate: 23.68%

Draw Rate: 29.21%

Win Rates for 2011-12:

Home Win Rate: 45.00%

Away Win Rate: 30.53%

Draw Rate: 24.47%

Win Rates for 2012-13:

Home Win Rate: 43.68%

Away Win Rate: 27.89%

Draw Rate: 28.42%

Win Rates for 2013-14:

Home Win Rate: 47.11%

Away Win Rate: 32.37%

Draw Rate: 20.53%

Win Rates for 2014-15:

Home Win Rate: 45.26%

Away Win Rate: 30.26%

Draw Rate: 24.47%

Win Rates for 2015-16:

Home Win Rate: 41.32%

Away Win Rate: 30.53%

Draw Rate: 28.16%

Win Rates for 2016-17:

Home Win Rate: 49.21%

Away Win Rate: 28.68%

Draw Rate: 22.11%

Win Rates for 2017-18:

Home Win Rate: 45.53%

Away Win Rate: 28.42%

Draw Rate: 26.05%

# Calculating goal differences for each match

for i, season in enumerate(seasons):

datasets[i]['GoalDifference'] = datasets[i]['FTHG'] - datasets[i]['FTAG']

plt.figure(figsize=(8, 4))

sns.histplot(datasets[i]['GoalDifference'], bins=20, kde=True, color='purple')

plt.title(f'Distribution of Goal Differences - {season}')

plt.xlabel('Goal Difference')

plt.ylabel('Frequency')

plt.show()

# Analysis of total goals scored over the season

total\_goals = []

seasons = [f'{year}-{str(year + 1)[-2:]}' for year in range(2000, 2018)]

for i, season in enumerate(seasons):

total\_goals.append(datasets[i]['FTHG'].sum() + datasets[i]['FTAG'].sum())

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

sns.lineplot(x=seasons, y=total\_goals, marker='o')

plt.title('Total Goals Scored per Season')

plt.xlabel('Season')

plt.ylabel('Total Goals')

plt.xticks(rotation=45)

plt.show()

# Combine all season datasets into a single DataFrame

combined\_data = pd.concat(datasets, ignore\_index=True)

# Display the first few rows of the combined dataset

print("Combined DataFrame:")

display(combined\_data.head())

# Compute the correlation matrix for the combined dataset

numeric\_data\_combined = combined\_data.select\_dtypes(include=[np.number]) # Select only numeric columns

correlation\_matrix\_combined = numeric\_data\_combined.corr()

# Display the correlation matrix

print("Combined Correlation Matrix:")

display(correlation\_matrix\_combined)

# Select required columns

columns\_req = ['Date', 'HomeTeam', 'AwayTeam', 'FTHG', 'FTAG', 'FTR']

playing\_statistics\_1 = raw\_data\_1[columns\_req]

playing\_statistics\_2 = raw\_data\_2[columns\_req]

playing\_statistics\_3 = raw\_data\_3[columns\_req]

playing\_statistics\_4 = raw\_data\_4[columns\_req]

playing\_statistics\_5 = raw\_data\_5[columns\_req]

playing\_statistics\_6 = raw\_data\_6[columns\_req]

playing\_statistics\_7 = raw\_data\_7[columns\_req]

playing\_statistics\_8 = raw\_data\_8[columns\_req]

playing\_statistics\_9 = raw\_data\_9[columns\_req]

playing\_statistics\_10 = raw\_data\_10[columns\_req]

playing\_statistics\_11 = raw\_data\_11[columns\_req]

playing\_statistics\_12 = raw\_data\_12[columns\_req]

playing\_statistics\_13 = raw\_data\_13[columns\_req]

playing\_statistics\_14 = raw\_data\_14[columns\_req]

playing\_statistics\_15 = raw\_data\_15[columns\_req]

playing\_statistics\_16 = raw\_data\_16[columns\_req]

playing\_statistics\_17 = raw\_data\_17[columns\_req]

playing\_statistics\_18 = raw\_data\_18[columns\_req]

# Concatenate all seasons' data

all\_data = pd.concat([playing\_statistics\_1, playing\_statistics\_2, playing\_statistics\_3,

playing\_statistics\_4, playing\_statistics\_5, playing\_statistics\_6,

playing\_statistics\_7, playing\_statistics\_8, playing\_statistics\_9,

playing\_statistics\_10, playing\_statistics\_11, playing\_statistics\_12,

playing\_statistics\_13, playing\_statistics\_14, playing\_statistics\_15,

playing\_statistics\_16, playing\_statistics\_17, playing\_statistics\_18],

ignore\_index=True)

# Display the first few rows

print(all\_data.head())

# Additional EDA Steps

# 1. Distribution of Full Time Results (FTR)

sns.countplot(x='FTR', data=all\_data)

plt.title('Distribution of Full Time Results')

plt.show()

# 2. Goals scored by Home vs Away teams

sns.histplot(all\_data['FTHG'], bins=20, color='blue', label='Home Goals', kde=True)

sns.histplot(all\_data['FTAG'], bins=20, color='red', label='Away Goals', kde=True)

plt.legend()

plt.title('Distribution of Goals Scored')

plt.show()

Date HomeTeam AwayTeam FTHG FTAG FTR

0 19/08/00 Charlton Man City 4 0 H

1 19/08/00 Chelsea West Ham 4 2 H

2 19/08/00 Coventry Middlesbrough 1 3 A

3 19/08/00 Derby Southampton 2 2 D

4 19/08/00 Leeds Everton 2 0 H

# Select only numeric columns for correlation matrix calculation

numeric\_cols = all\_data.select\_dtypes(include=[np.number]).columns

corr = all\_data[numeric\_cols].corr()

# Plot the heatmap

sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm')

plt.title('Correlation Matrix')

plt.show()

"""Feature Engineering"""

def get\_goals\_scored(playing\_stat):

teams = {team: [] for team in playing\_stat['HomeTeam'].unique()}

for i in range(len(playing\_stat)):

HTGS = playing\_stat.iloc[i]['FTHG']

ATGS = playing\_stat.iloc[i]['FTAG']

teams[playing\_stat.iloc[i].HomeTeam].append(HTGS)

teams[playing\_stat.iloc[i].AwayTeam].append(ATGS)

num\_matchweeks = max(len(v) for v in teams.values())

GoalsScored = pd.DataFrame(data=teams, index=range(1, num\_matchweeks + 1)).T

GoalsScored[0] = 0

for i in range(2, num\_matchweeks + 1):

GoalsScored[i] = GoalsScored[i] + GoalsScored[i-1]

return GoalsScored

def get\_goals\_conceded(playing\_stat):

teams = {team: [] for team in playing\_stat['HomeTeam'].unique()}

for i in range(len(playing\_stat)):

ATGC = playing\_stat.iloc[i]['FTHG']

HTGC = playing\_stat.iloc[i]['FTAG']

teams[playing\_stat.iloc[i].HomeTeam].append(HTGC)

teams[playing\_stat.iloc[i].AwayTeam].append(ATGC)

num\_matchweeks = max(len(v) for v in teams.values())

GoalsConceded = pd.DataFrame(data=teams, index=range(1, num\_matchweeks + 1)).T

GoalsConceded[0] = 0

for i in range(2, num\_matchweeks + 1):

GoalsConceded[i] = GoalsConceded[i] + GoalsConceded[i-1]

return GoalsConceded

def get\_gss(playing\_stat):

GC = get\_goals\_conceded(playing\_stat)

GS = get\_goals\_scored(playing\_stat)

j = 0

HTGS = []

ATGS = []

HTGC = []

ATGC = []

num\_games = len(playing\_stat)

num\_matchweeks = min(GS.shape[1], GC.shape[1])

for i in range(num\_games):

ht = playing\_stat.iloc[i].HomeTeam

at = playing\_stat.iloc[i].AwayTeam

HTGS.append(GS.loc[ht].iloc[j])

ATGS.append(GS.loc[at].iloc[j])

HTGC.append(GC.loc[ht].iloc[j])

ATGC.append(GC.loc[at].iloc[j])

if (i + 1) % 10 == 0 and j < num\_matchweeks - 1:

j += 1

playing\_stat['HTGS'] = HTGS

playing\_stat['ATGS'] = ATGS

playing\_stat['HTGC'] = HTGC

playing\_stat['ATGC'] = ATGC

return playing\_stat

# Apply the feature engineering functions to each dataset

playing\_statistics\_1 = get\_gss(playing\_statistics\_1)

playing\_statistics\_2 = get\_gss(playing\_statistics\_2)

playing\_statistics\_3 = get\_gss(playing\_statistics\_3)

playing\_statistics\_4 = get\_gss(playing\_statistics\_4)

playing\_statistics\_5 = get\_gss(playing\_statistics\_5)

playing\_statistics\_6 = get\_gss(playing\_statistics\_6)

playing\_statistics\_7 = get\_gss(playing\_statistics\_7)

playing\_statistics\_8 = get\_gss(playing\_statistics\_8)

playing\_statistics\_9 = get\_gss(playing\_statistics\_9)

playing\_statistics\_10 = get\_gss(playing\_statistics\_10)

playing\_statistics\_11 = get\_gss(playing\_statistics\_11)

playing\_statistics\_12 = get\_gss(playing\_statistics\_12)

playing\_statistics\_13 = get\_gss(playing\_statistics\_13)

playing\_statistics\_14 = get\_gss(playing\_statistics\_14)

playing\_statistics\_15 = get\_gss(playing\_statistics\_15)

playing\_statistics\_16 = get\_gss(playing\_statistics\_16)

playing\_statistics\_17 = get\_gss(playing\_statistics\_17)

playing\_statistics\_18 = get\_gss(playing\_statistics\_18)

import pandas as pd

# Assuming 'playing\_stat' is the DataFrame for the match statistics

def get\_matchres(playing\_stat):

teams = {team: [] for team in playing\_stat['HomeTeam'].unique()}

for i in range(len(playing\_stat)):

if playing\_stat.iloc[i].FTR == 'H':

teams[playing\_stat.iloc[i].HomeTeam].append('W')

teams[playing\_stat.iloc[i].AwayTeam].append('L')

elif playing\_stat.iloc[i].FTR == 'A':

teams[playing\_stat.iloc[i].AwayTeam].append('W')

teams[playing\_stat.iloc[i].HomeTeam].append('L')

else:

teams[playing\_stat.iloc[i].AwayTeam].append('D')

teams[playing\_stat.iloc[i].HomeTeam].append('D')

return pd.DataFrame(data=teams, index=range(1, 39)).T

def get\_points(result):

if result == 'W':

return 3

elif result == 'D':

return 1

else:

return 0

def get\_cuml\_points(matchres):

matchres\_points = matchres.applymap(get\_points)

num\_matchweeks = matchres\_points.shape[1]

matchres\_points.insert(0, 0, 0)

for i in range(2, num\_matchweeks + 1):

matchres\_points[i] = matchres\_points[i] + matchres\_points[i - 1]

return matchres\_points

def get\_agg\_points(playing\_stat):

matchres = get\_matchres(playing\_stat)

cum\_pts = get\_cuml\_points(matchres)

HTP = []

ATP = []

j = 0

num\_games = len(playing\_stat)

num\_matchweeks = cum\_pts.shape[1]

for i in range(num\_games):

ht = playing\_stat.iloc[i].HomeTeam

at = playing\_stat.iloc[i].AwayTeam

if j < num\_matchweeks:

HTP.append(cum\_pts.loc[ht].iloc[j])

ATP.append(cum\_pts.loc[at].iloc[j])

if (i + 1) % 10 == 0 and j < num\_matchweeks - 1:

j += 1

playing\_stat['HTP'] = HTP

playing\_stat['ATP'] = ATP

return playing\_stat

def get\_form(playing\_stat, num):

matchres = get\_matchres(playing\_stat)

form\_final = matchres.copy()

for i in range(num, 39):

form\_final[i] = ''

for j in range(num):

form\_final[i] += matchres[i-j]

return form\_final

def add\_form(playing\_stat, num):

form = get\_form(playing\_stat, num)

h = ['M' for \_ in range(num \* 10)]

a = ['M' for \_ in range(num \* 10)]

j = num

for i in range(num \* 10, len(playing\_stat)):

ht = playing\_stat.iloc[i].HomeTeam

at = playing\_stat.iloc[i].AwayTeam

past = form.loc[ht][j]

h.append(past[num-1])

past = form.loc[at][j]

a.append(past[num-1])

if ((i + 1) % 10) == 0:

j += 1

playing\_stat[f'HM{num}'] = h

playing\_stat[f'AM{num}'] = a

return playing\_stat

def add\_form\_df(playing\_statistics):

for num in range(1, 6):

playing\_statistics = [add\_form(df, num) for df in playing\_statistics]

return playing\_statistics

# Apply cumulative points calculation

playing\_statistics\_1 = get\_agg\_points(playing\_statistics\_1)

playing\_statistics\_2 = get\_agg\_points(playing\_statistics\_2)

playing\_statistics\_3 = get\_agg\_points(playing\_statistics\_3)

playing\_statistics\_4 = get\_agg\_points(playing\_statistics\_4)

playing\_statistics\_5 = get\_agg\_points(playing\_statistics\_5)

playing\_statistics\_6 = get\_agg\_points(playing\_statistics\_6)

playing\_statistics\_7 = get\_agg\_points(playing\_statistics\_7)

playing\_statistics\_8 = get\_agg\_points(playing\_statistics\_8)

playing\_statistics\_9 = get\_agg\_points(playing\_statistics\_9)

playing\_statistics\_10 = get\_agg\_points(playing\_statistics\_10)

playing\_statistics\_11 = get\_agg\_points(playing\_statistics\_11)

playing\_statistics\_12 = get\_agg\_points(playing\_statistics\_12)

playing\_statistics\_13 = get\_agg\_points(playing\_statistics\_13)

playing\_statistics\_14 = get\_agg\_points(playing\_statistics\_14)

playing\_statistics\_15 = get\_agg\_points(playing\_statistics\_15)

playing\_statistics\_16 = get\_agg\_points(playing\_statistics\_16)

playing\_statistics\_17 = get\_agg\_points(playing\_statistics\_17)

playing\_statistics\_18 = get\_agg\_points(playing\_statistics\_18)

# Adding form (last n matches) data

playing\_statistics\_1 = add\_form\_df([playing\_statistics\_1])[0]

playing\_statistics\_2 = add\_form\_df([playing\_statistics\_2])[0]

playing\_statistics\_3 = add\_form\_df([playing\_statistics\_3])[0]

playing\_statistics\_4 = add\_form\_df([playing\_statistics\_4])[0]

playing\_statistics\_5 = add\_form\_df([playing\_statistics\_5])[0]

playing\_statistics\_6 = add\_form\_df([playing\_statistics\_6])[0]

playing\_statistics\_7 = add\_form\_df([playing\_statistics\_7])[0]

playing\_statistics\_8 = add\_form\_df([playing\_statistics\_8])[0]

playing\_statistics\_9 = add\_form\_df([playing\_statistics\_9])[0]

playing\_statistics\_10 = add\_form\_df([playing\_statistics\_10])[0]

playing\_statistics\_11 = add\_form\_df([playing\_statistics\_11])[0]

playing\_statistics\_12 = add\_form\_df([playing\_statistics\_12])[0]

playing\_statistics\_13 = add\_form\_df([playing\_statistics\_13])[0]

playing\_statistics\_14 = add\_form\_df([playing\_statistics\_14])[0]

playing\_statistics\_15 = add\_form\_df([playing\_statistics\_15])[0]

playing\_statistics\_16 = add\_form\_df([playing\_statistics\_16])[0]

playing\_statistics\_17 = add\_form\_df([playing\_statistics\_17])[0]

playing\_statistics\_18 = add\_form\_df([playing\_statistics\_18])[0]

# Preparing the final dataset by concatenating all seasons' data

playing\_stat = pd.concat([

playing\_statistics\_1, playing\_statistics\_2, playing\_statistics\_3,

playing\_statistics\_4, playing\_statistics\_5, playing\_statistics\_6,

playing\_statistics\_7, playing\_statistics\_8, playing\_statistics\_9,

playing\_statistics\_10, playing\_statistics\_11, playing\_statistics\_12,

playing\_statistics\_13, playing\_statistics\_14, playing\_statistics\_15,

playing\_statistics\_16, playing\_statistics\_17, playing\_statistics\_18

], ignore\_index=True)

# Check if the columns HTFormPts and ATFormPts exist, if not, calculate them

if 'HTFormPts' not in playing\_stat.columns or 'ATFormPts' not in playing\_stat.columns:

# Assuming HTFormPtsStr and ATFormPtsStr columns exist

playing\_stat['HTFormPtsStr'] = (

playing\_stat['HM1'] + playing\_stat['HM2'] + playing\_stat['HM3'] + playing\_stat['HM4'] + playing\_stat['HM5']

)

playing\_stat['ATFormPtsStr'] = (

playing\_stat['AM1'] + playing\_stat['AM2'] + playing\_stat['AM3'] + playing\_stat['AM4'] + playing\_stat['AM5']

)

def get\_form\_points(string):

sum\_points = 0

for letter in string:

sum\_points += get\_points(letter)

return sum\_points

playing\_stat['HTFormPts'] = playing\_stat['HTFormPtsStr'].apply(get\_form\_points)

playing\_stat['ATFormPts'] = playing\_stat['ATFormPtsStr'].apply(get\_form\_points)

# Calculating additional features

playing\_stat['HTGD'] = playing\_stat['HTGS'] - playing\_stat['HTGC']

playing\_stat['ATGD'] = playing\_stat['ATGS'] - playing\_stat['ATGC']

playing\_stat['DiffPts'] = playing\_stat['HTP'] - playing\_stat['ATP']

playing\_stat['DiffFormPts'] = playing\_stat['HTFormPts'] - playing\_stat['ATFormPts']

# Ensure that MW column is correctly calculated and exists

if 'MW' not in playing\_stat.columns:

def get\_mw(df):

j = 1

MatchWeek = []

for i in range(len(df)):

MatchWeek.append(j)

if ((i + 1) % 10) == 0:

j += 1

df['MW'] = MatchWeek

return df

playing\_stat = get\_mw(playing\_stat)

# Scaling certain features by Matchweek

playing\_stat['MW'] = playing\_stat['MW'].astype(float)

cols\_to\_scale = ['HTGD', 'ATGD', 'DiffPts', 'DiffFormPts', 'HTP', 'ATP']

for col in cols\_to\_scale:

playing\_stat[col] = playing\_stat[col] / playing\_stat['MW']

# Transform the target variable (FTR) to binary classification ('H' or 'NH')

playing\_stat['FTR'] = playing\_stat['FTR'].apply(lambda x: 'H' if x == 'H' else 'NH')

# Saving the final dataset

playing\_stat.to\_csv('/content/final\_dataset.csv', index=False)

# Loading the dataset for further use

dataset = pd.read\_csv('/content/final\_dataset.csv')

columns\_req = ['Date','HomeTeam','AwayTeam','FTHG','FTAG','FTR']

playing\_statistics\_1 = raw\_data\_1[columns\_req]

playing\_statistics\_2 = raw\_data\_2[columns\_req]

playing\_statistics\_3 = raw\_data\_3[columns\_req]

playing\_statistics\_4 = raw\_data\_4[columns\_req]

playing\_statistics\_5 = raw\_data\_5[columns\_req]

playing\_statistics\_6 = raw\_data\_6[columns\_req]

playing\_statistics\_7 = raw\_data\_7[columns\_req]

playing\_statistics\_8 = raw\_data\_8[columns\_req]

playing\_statistics\_9 = raw\_data\_9[columns\_req]

playing\_statistics\_10 = raw\_data\_10[columns\_req]

playing\_statistics\_11 = raw\_data\_11[columns\_req]

playing\_statistics\_12 = raw\_data\_12[columns\_req]

playing\_statistics\_13 = raw\_data\_13[columns\_req]

playing\_statistics\_14 = raw\_data\_14[columns\_req]

playing\_statistics\_15 = raw\_data\_15[columns\_req]

playing\_statistics\_16 = raw\_data\_16[columns\_req]

playing\_statistics\_17 = raw\_data\_17[columns\_req]

playing\_statistics\_18 = raw\_data\_18[columns\_req]

import pandas as pd

def get\_goals\_scored(playing\_stat):

# Create a dictionary with team names as keys

teams = {team: [] for team in playing\_stat['HomeTeam'].unique()}

# Fill in the goals scored for each team

for i in range(len(playing\_stat)):

HTGS = playing\_stat.iloc[i]['FTHG']

ATGS = playing\_stat.iloc[i]['FTAG']

teams[playing\_stat.iloc[i].HomeTeam].append(HTGS)

teams[playing\_stat.iloc[i].AwayTeam].append(ATGS)

# Determine the number of matchweeks

num\_matchweeks = max(len(v) for v in teams.values())

# Create a dataframe for goals scored where rows are teams and columns are matchweek.

GoalsScored = pd.DataFrame(data=teams, index=range(1, num\_matchweeks + 1)).T

GoalsScored[0] = 0 # Initialize the cumulative sum

# Aggregate to get cumulative goals up to that point

for i in range(2, num\_matchweeks + 1):

GoalsScored[i] = GoalsScored[i] + GoalsScored[i-1]

return GoalsScored

def get\_goals\_conceded(playing\_stat):

# Create a dictionary with team names as keys

teams = {team: [] for team in playing\_stat['HomeTeam'].unique()}

# Fill in the goals conceded for each team

for i in range(len(playing\_stat)):

ATGC = playing\_stat.iloc[i]['FTHG']

HTGC = playing\_stat.iloc[i]['FTAG']

teams[playing\_stat.iloc[i].HomeTeam].append(HTGC)

teams[playing\_stat.iloc[i].AwayTeam].append(ATGC)

# Determine the number of matchweeks

num\_matchweeks = max(len(v) for v in teams.values())

# Create a dataframe for goals conceded where rows are teams and columns are matchweek.

GoalsConceded = pd.DataFrame(data=teams, index=range(1, num\_matchweeks + 1)).T

GoalsConceded[0] = 0 # Initialize the cumulative sum

# Aggregate to get cumulative goals conceded up to that point

for i in range(2, num\_matchweeks + 1):

GoalsConceded[i] = GoalsConceded[i] + GoalsConceded[i-1]

return GoalsConceded

def get\_gss(playing\_stat):

GC = get\_goals\_conceded(playing\_stat)

GS = get\_goals\_scored(playing\_stat)

j = 0

HTGS = []

ATGS = []

HTGC = []

ATGC = []

num\_games = len(playing\_stat)

num\_matchweeks = min(GS.shape[1], GC.shape[1])

for i in range(num\_games):

ht = playing\_stat.iloc[i].HomeTeam

at = playing\_stat.iloc[i].AwayTeam

HTGS.append(GS.loc[ht].iloc[j])

ATGS.append(GS.loc[at].iloc[j])

HTGC.append(GC.loc[ht].iloc[j])

ATGC.append(GC.loc[at].iloc[j])

if (i + 1) % 10 == 0 and j < num\_matchweeks - 1:

j += 1

playing\_stat['HTGS'] = HTGS

playing\_stat['ATGS'] = ATGS

playing\_stat['HTGC'] = HTGC

playing\_stat['ATGC'] = ATGC

return playing\_stat

# Apply to each dataset

playing\_statistics\_1 = get\_gss(playing\_statistics\_1)

playing\_statistics\_2 = get\_gss(playing\_statistics\_2)

playing\_statistics\_3 = get\_gss(playing\_statistics\_3)

playing\_statistics\_4 = get\_gss(playing\_statistics\_4)

playing\_statistics\_5 = get\_gss(playing\_statistics\_5)

playing\_statistics\_6 = get\_gss(playing\_statistics\_6)

playing\_statistics\_7 = get\_gss(playing\_statistics\_7)

playing\_statistics\_8 = get\_gss(playing\_statistics\_8)

playing\_statistics\_9 = get\_gss(playing\_statistics\_9)

playing\_statistics\_10 = get\_gss(playing\_statistics\_10)

playing\_statistics\_11 = get\_gss(playing\_statistics\_11)

playing\_statistics\_12 = get\_gss(playing\_statistics\_12)

playing\_statistics\_13 = get\_gss(playing\_statistics\_13)

playing\_statistics\_14 = get\_gss(playing\_statistics\_14)

playing\_statistics\_15 = get\_gss(playing\_statistics\_15)

playing\_statistics\_16 = get\_gss(playing\_statistics\_16)

playing\_statistics\_17 = get\_gss(playing\_statistics\_17)

playing\_statistics\_18 = get\_gss(playing\_statistics\_18)

import pandas as pd

def get\_points(result):

if result == 'W':

return 3

elif result == 'D':

return 1

else:

return 0

def get\_cuml\_points(matchres):

# Convert match results to points

matchres\_points = matchres.applymap(get\_points)

# Ensure there's no issue with index when initializing the cumulative points

num\_matchweeks = matchres\_points.shape[1]

num\_teams = matchres\_points.shape[0]

matchres\_points[0] = 0 # Initialize the cumulative sum with zero points

for i in range(2, num\_matchweeks + 1):

matchres\_points[i] = matchres\_points[i] + matchres\_points[i - 1]

return matchres\_points

def get\_matchres(playing\_stat):

# Create a dictionary with team names as keys

teams = {team: [] for team in playing\_stat['HomeTeam'].unique()}

# the value corresponding to keys is a list containing the match result

for i in range(len(playing\_stat)):

if playing\_stat.iloc[i].FTR == 'H':

teams[playing\_stat.iloc[i].HomeTeam].append('W')

teams[playing\_stat.iloc[i].AwayTeam].append('L')

elif playing\_stat.iloc[i].FTR == 'A':

teams[playing\_stat.iloc[i].AwayTeam].append('W')

teams[playing\_stat.iloc[i].HomeTeam].append('L')

else:

teams[playing\_stat.iloc[i].AwayTeam].append('D')

teams[playing\_stat.iloc[i].HomeTeam].append('D')

# Determine the number of matchweeks based on the length of the values

num\_matchweeks = max(len(results) for results in teams.values())

# Create a DataFrame with the results and ensure correct indexing

return pd.DataFrame(data=teams, index=range(1, num\_matchweeks + 1)).T

def get\_agg\_points(playing\_stat):

matchres = get\_matchres(playing\_stat)

cum\_pts = get\_cuml\_points(matchres)

HTP = []

ATP = []

j = 0

num\_games = len(playing\_stat)

num\_matchweeks = cum\_pts.shape[1]

for i in range(num\_games):

ht = playing\_stat.iloc[i].HomeTeam

at = playing\_stat.iloc[i].AwayTeam

if j < num\_matchweeks:

HTP.append(cum\_pts.loc[ht].iloc[j])

ATP.append(cum\_pts.loc[at].iloc[j])

if (i + 1) % 10 == 0 and j < num\_matchweeks - 1:

j += 1

playing\_stat['HTP'] = HTP

playing\_stat['ATP'] = ATP

return playing\_stat

# Apply to each dataset

playing\_statistics\_1 = get\_gss(playing\_statistics\_1)

playing\_statistics\_2 = get\_gss(playing\_statistics\_2)

playing\_statistics\_3 = get\_gss(playing\_statistics\_3)

playing\_statistics\_4 = get\_gss(playing\_statistics\_4)

playing\_statistics\_5 = get\_gss(playing\_statistics\_5)

playing\_statistics\_6 = get\_gss(playing\_statistics\_6)

playing\_statistics\_7 = get\_gss(playing\_statistics\_7)

playing\_statistics\_8 = get\_gss(playing\_statistics\_8)

playing\_statistics\_9 = get\_gss(playing\_statistics\_9)

playing\_statistics\_10 = get\_gss(playing\_statistics\_10)

playing\_statistics\_11 = get\_gss(playing\_statistics\_11)

playing\_statistics\_12 = get\_gss(playing\_statistics\_12)

playing\_statistics\_13 = get\_gss(playing\_statistics\_13)

playing\_statistics\_14 = get\_gss(playing\_statistics\_14)

playing\_statistics\_15 = get\_gss(playing\_statistics\_15)

playing\_statistics\_16 = get\_gss(playing\_statistics\_16)

playing\_statistics\_17 = get\_gss(playing\_statistics\_17)

playing\_statistics\_18 = get\_gss(playing\_statistics\_18)

def get\_form(playing\_stat,num):

form = get\_matchres(playing\_stat)

form\_final = form.copy()

for i in range(num,39):

form\_final[i] = ''

j = 0

while j < num:

form\_final[i] += form[i-j]

j += 1

return form\_final

def add\_form(playing\_stat,num):

form = get\_form(playing\_stat,num)

h = ['M' for i in range(num \* 10)] # since form is not available for n MW (n\*10)

a = ['M' for i in range(num \* 10)]

j = num

for i in range((num\*10),380):

ht = playing\_stat.iloc[i].HomeTeam

at = playing\_stat.iloc[i].AwayTeam

past = form.loc[ht][j] # get past n results

h.append(past[num-1]) # 0 index is most recent

past = form.loc[at][j] # get past n results.

a.append(past[num-1]) # 0 index is most recent

if ((i + 1)% 10) == 0:

j = j + 1

playing\_stat['HM' + str(num)] = h

playing\_stat['AM' + str(num)] = a

return playing\_stat

def add\_form\_df(playing\_statistics):

playing\_statistics = add\_form(playing\_statistics,1)

playing\_statistics = add\_form(playing\_statistics,2)

playing\_statistics = add\_form(playing\_statistics,3)

playing\_statistics = add\_form(playing\_statistics,4)

playing\_statistics = add\_form(playing\_statistics,5)

return playing\_statistics

# Make changes to df

playing\_statistics\_1 = add\_form\_df(playing\_statistics\_1)

playing\_statistics\_2 = add\_form\_df(playing\_statistics\_2)

playing\_statistics\_3 = add\_form\_df(playing\_statistics\_3)

playing\_statistics\_4 = add\_form\_df(playing\_statistics\_4)

playing\_statistics\_5 = add\_form\_df(playing\_statistics\_5)

playing\_statistics\_6 = add\_form\_df(playing\_statistics\_6)

playing\_statistics\_7 = add\_form\_df(playing\_statistics\_7)

playing\_statistics\_8 = add\_form\_df(playing\_statistics\_8)

playing\_statistics\_9 = add\_form\_df(playing\_statistics\_9)

playing\_statistics\_10 = add\_form\_df(playing\_statistics\_10)

playing\_statistics\_11 = add\_form\_df(playing\_statistics\_11)

playing\_statistics\_12 = add\_form\_df(playing\_statistics\_12)

playing\_statistics\_13 = add\_form\_df(playing\_statistics\_13)

playing\_statistics\_14 = add\_form\_df(playing\_statistics\_14)

playing\_statistics\_15 = add\_form\_df(playing\_statistics\_15)

playing\_statistics\_16 = add\_form\_df(playing\_statistics\_16)

playing\_statistics\_17 = add\_form\_df(playing\_statistics\_17)

playing\_statistics\_18 = add\_form\_df(playing\_statistics\_18)

def get\_form(playing\_stat,num):

form = get\_matchres(playing\_stat)

form\_final = form.copy()

for i in range(num,39):

form\_final[i] = ''

j = 0

while j < num:

form\_final[i] += form[i-j]

j += 1

return form\_final

def add\_form(playing\_stat,num):

form = get\_form(playing\_stat,num)

h = ['M' for i in range(num \* 10)] # since form is not available for n MW (n\*10)

a = ['M' for i in range(num \* 10)]

j = num

for i in range((num\*10),380):

ht = playing\_stat.iloc[i].HomeTeam

at = playing\_stat.iloc[i].AwayTeam

past = form.loc[ht][j] # get past n results

h.append(past[num-1]) # 0 index is most recent

past = form.loc[at][j] # get past n results.

a.append(past[num-1]) # 0 index is most recent

if ((i + 1)% 10) == 0:

j = j + 1

playing\_stat['HM' + str(num)] = h

playing\_stat['AM' + str(num)] = a

return playing\_stat

def add\_form\_df(playing\_statistics):

playing\_statistics = add\_form(playing\_statistics,1)

playing\_statistics = add\_form(playing\_statistics,2)

playing\_statistics = add\_form(playing\_statistics,3)

playing\_statistics = add\_form(playing\_statistics,4)

playing\_statistics = add\_form(playing\_statistics,5)

return playing\_statistics

# Make changes to df

playing\_statistics\_1 = add\_form\_df(playing\_statistics\_1)

playing\_statistics\_2 = add\_form\_df(playing\_statistics\_2)

playing\_statistics\_3 = add\_form\_df(playing\_statistics\_3)

playing\_statistics\_4 = add\_form\_df(playing\_statistics\_4)

playing\_statistics\_5 = add\_form\_df(playing\_statistics\_5)

playing\_statistics\_6 = add\_form\_df(playing\_statistics\_6)

playing\_statistics\_7 = add\_form\_df(playing\_statistics\_7)

playing\_statistics\_8 = add\_form\_df(playing\_statistics\_8)

playing\_statistics\_9 = add\_form\_df(playing\_statistics\_9)

playing\_statistics\_10 = add\_form\_df(playing\_statistics\_10)

playing\_statistics\_11 = add\_form\_df(playing\_statistics\_11)

playing\_statistics\_12 = add\_form\_df(playing\_statistics\_12)

playing\_statistics\_13 = add\_form\_df(playing\_statistics\_13)

playing\_statistics\_14 = add\_form\_df(playing\_statistics\_14)

playing\_statistics\_15 = add\_form\_df(playing\_statistics\_15)

playing\_statistics\_16 = add\_form\_df(playing\_statistics\_16)

playing\_statistics\_17 = add\_form\_df(playing\_statistics\_17)

playing\_statistics\_18 = add\_form\_df(playing\_statistics\_18)

def get\_form(playing\_stat,num):

form = get\_matchres(playing\_stat)

form\_final = form.copy()

for i in range(num,39):

form\_final[i] = ''

j = 0

while j < num:

form\_final[i] += form[i-j]

j += 1

return form\_final

def add\_form(playing\_stat,num):

form = get\_form(playing\_stat,num)

h = ['M' for i in range(num \* 10)] # since form is not available for n MW (n\*10)

a = ['M' for i in range(num \* 10)]

j = num

for i in range((num\*10),380):

ht = playing\_stat.iloc[i].HomeTeam

at = playing\_stat.iloc[i].AwayTeam

past = form.loc[ht][j] # get past n results

h.append(past[num-1]) # 0 index is most recent

past = form.loc[at][j] # get past n results.

a.append(past[num-1]) # 0 index is most recent

if ((i + 1)% 10) == 0:

j = j + 1

playing\_stat['HM' + str(num)] = h

playing\_stat['AM' + str(num)] = a

return playing\_stat

def add\_form\_df(playing\_statistics):

playing\_statistics = add\_form(playing\_statistics,1)

playing\_statistics = add\_form(playing\_statistics,2)

playing\_statistics = add\_form(playing\_statistics,3)

playing\_statistics = add\_form(playing\_statistics,4)

playing\_statistics = add\_form(playing\_statistics,5)

return playing\_statistics

# Make changes to df

playing\_statistics\_1 = add\_form\_df(playing\_statistics\_1)

playing\_statistics\_2 = add\_form\_df(playing\_statistics\_2)

playing\_statistics\_3 = add\_form\_df(playing\_statistics\_3)

playing\_statistics\_4 = add\_form\_df(playing\_statistics\_4)

playing\_statistics\_5 = add\_form\_df(playing\_statistics\_5)

playing\_statistics\_6 = add\_form\_df(playing\_statistics\_6)

playing\_statistics\_7 = add\_form\_df(playing\_statistics\_7)

playing\_statistics\_8 = add\_form\_df(playing\_statistics\_8)

playing\_statistics\_9 = add\_form\_df(playing\_statistics\_9)

playing\_statistics\_10 = add\_form\_df(playing\_statistics\_10)

playing\_statistics\_11 = add\_form\_df(playing\_statistics\_11)

playing\_statistics\_12 = add\_form\_df(playing\_statistics\_12)

playing\_statistics\_13 = add\_form\_df(playing\_statistics\_13)

playing\_statistics\_14 = add\_form\_df(playing\_statistics\_14)

playing\_statistics\_15 = add\_form\_df(playing\_statistics\_15)

playing\_statistics\_16 = add\_form\_df(playing\_statistics\_16)

playing\_statistics\_17 = add\_form\_df(playing\_statistics\_17)

playing\_statistics\_18 = add\_form\_df(playing\_statistics\_18)

import pandas as pd

def rearrange\_columns(df, cols):

# Check if all required columns exist in the DataFrame

missing\_cols = [col for col in cols if col not in df.columns]

if not missing\_cols:

return df[cols]

else:

print(f"Warning: Missing columns {missing\_cols} in the DataFrame.")

return df

# List of columns to rearrange

cols = ['Date', 'HomeTeam', 'AwayTeam', 'FTHG', 'FTAG', 'FTR', 'HTGS', 'ATGS', 'HTGC', 'ATGC', 'HTP', 'ATP', 'HM1', 'HM2', 'HM3',

'HM4', 'HM5', 'AM1', 'AM2', 'AM3', 'AM4', 'AM5' ]

# Apply rearrangement to each DataFrame

playing\_statistics\_1 = rearrange\_columns(playing\_statistics\_1, cols)

playing\_statistics\_2 = rearrange\_columns(playing\_statistics\_2, cols)

playing\_statistics\_3 = rearrange\_columns(playing\_statistics\_3, cols)

playing\_statistics\_4 = rearrange\_columns(playing\_statistics\_4, cols)

playing\_statistics\_5 = rearrange\_columns(playing\_statistics\_5, cols)

playing\_statistics\_6 = rearrange\_columns(playing\_statistics\_6, cols)

playing\_statistics\_7 = rearrange\_columns(playing\_statistics\_7, cols)

playing\_statistics\_8 = rearrange\_columns(playing\_statistics\_8, cols)

playing\_statistics\_9 = rearrange\_columns(playing\_statistics\_9, cols)

playing\_statistics\_10 = rearrange\_columns(playing\_statistics\_10, cols)

playing\_statistics\_11 = rearrange\_columns(playing\_statistics\_11, cols)

playing\_statistics\_12 = rearrange\_columns(playing\_statistics\_12, cols)

playing\_statistics\_13 = rearrange\_columns(playing\_statistics\_13, cols)

playing\_statistics\_14 = rearrange\_columns(playing\_statistics\_14, cols)

playing\_statistics\_15 = rearrange\_columns(playing\_statistics\_15, cols)

playing\_statistics\_16 = rearrange\_columns(playing\_statistics\_16, cols)

playing\_statistics\_17 = rearrange\_columns(playing\_statistics\_17, cols)

playing\_statistics\_18 = rearrange\_columns(playing\_statistics\_18, cols)

def get\_mw(playing\_stat):

j = 1

MatchWeek = []

for i in range(380):

MatchWeek.append(j)

if ((i + 1)% 10) == 0:

j = j + 1

playing\_stat['MW'] = MatchWeek

return playing\_stat

playing\_statistics\_1 = get\_mw(playing\_statistics\_1)

playing\_statistics\_2 = get\_mw(playing\_statistics\_2)

playing\_statistics\_3 = get\_mw(playing\_statistics\_3)

playing\_statistics\_4 = get\_mw(playing\_statistics\_4)

playing\_statistics\_5 = get\_mw(playing\_statistics\_5)

playing\_statistics\_6 = get\_mw(playing\_statistics\_6)

playing\_statistics\_7 = get\_mw(playing\_statistics\_7)

playing\_statistics\_8 = get\_mw(playing\_statistics\_8)

playing\_statistics\_9 = get\_mw(playing\_statistics\_9)

playing\_statistics\_10 = get\_mw(playing\_statistics\_10)

playing\_statistics\_11 = get\_mw(playing\_statistics\_11)

playing\_statistics\_12 = get\_mw(playing\_statistics\_12)

playing\_statistics\_13 = get\_mw(playing\_statistics\_13)

playing\_statistics\_14 = get\_mw(playing\_statistics\_14)

playing\_statistics\_15 = get\_mw(playing\_statistics\_15)

playing\_statistics\_16 = get\_mw(playing\_statistics\_16)

playing\_statistics\_17 = get\_mw(playing\_statistics\_17)

playing\_statistics\_18 = get\_mw(playing\_statistics\_18)

playing\_stat = pd.concat([playing\_statistics\_1,

playing\_statistics\_2,

playing\_statistics\_3,

playing\_statistics\_4,

playing\_statistics\_5,

playing\_statistics\_6,

playing\_statistics\_7,

playing\_statistics\_8,

playing\_statistics\_9,

playing\_statistics\_10,

playing\_statistics\_11,

playing\_statistics\_12,

playing\_statistics\_13,

playing\_statistics\_14,

playing\_statistics\_15,

playing\_statistics\_16,

playing\_statistics\_17,

playing\_statistics\_18

], ignore\_index=True)

# Gets the form points.

def get\_form\_points(string):

sum = 0

for letter in string:

sum += get\_points(letter)

return sum

playing\_stat['HTFormPtsStr'] = playing\_stat['HM1'] + playing\_stat['HM2'] + playing\_stat['HM3'] + playing\_stat['HM4'] + playing\_stat['HM5']

playing\_stat['ATFormPtsStr'] = playing\_stat['AM1'] + playing\_stat['AM2'] + playing\_stat['AM3'] + playing\_stat['AM4'] + playing\_stat['AM5']

playing\_stat['HTFormPts'] = playing\_stat['HTFormPtsStr'].apply(get\_form\_points)

playing\_stat['ATFormPts'] = playing\_stat['ATFormPtsStr'].apply(get\_form\_points)

# Identify Win/Loss Streaks if any.

def get\_3game\_ws(string):

if string[-3:] == 'WWW':

return 1

else:

return 0

def get\_5game\_ws(string):

if string == 'WWWWW':

return 1

else:

return 0

def get\_3game\_ls(string):

if string[-3:] == 'LLL':

return 1

else:

return 0

def get\_5game\_ls(string):

if string == 'LLLLL':

return 1

else:

return 0

playing\_stat['HTWinStreak3'] = playing\_stat['HTFormPtsStr'].apply(get\_3game\_ws)

playing\_stat['HTWinStreak5'] = playing\_stat['HTFormPtsStr'].apply(get\_5game\_ws)

playing\_stat['HTLossStreak3'] = playing\_stat['HTFormPtsStr'].apply(get\_3game\_ls)

playing\_stat['HTLossStreak5'] = playing\_stat['HTFormPtsStr'].apply(get\_5game\_ls)

playing\_stat['ATWinStreak3'] = playing\_stat['ATFormPtsStr'].apply(get\_3game\_ws)

playing\_stat['ATWinStreak5'] = playing\_stat['ATFormPtsStr'].apply(get\_5game\_ws)

playing\_stat['ATLossStreak3'] = playing\_stat['ATFormPtsStr'].apply(get\_3game\_ls)

playing\_stat['ATLossStreak5'] = playing\_stat['ATFormPtsStr'].apply(get\_5game\_ls)

playing\_stat.keys()

Index(['Date', 'HomeTeam', 'AwayTeam', 'FTHG', 'FTAG', 'FTR', 'HTGS', 'ATGS',

'HTGC', 'ATGC', 'HM1', 'AM1', 'HM2', 'AM2', 'HM3', 'AM3', 'HM4', 'AM4',

'HM5', 'AM5', 'MW', 'HTFormPtsStr', 'ATFormPtsStr', 'HTFormPts',

'ATFormPts', 'HTWinStreak3', 'HTWinStreak5', 'HTLossStreak3',

'HTLossStreak5', 'ATWinStreak3', 'ATWinStreak5', 'ATLossStreak3',

'ATLossStreak5'],

dtype='object')

def only\_hw(string):

if string == 'H':

return 'H'

else:

return 'NH'

playing\_stat['FTR'] = playing\_stat.FTR.apply(only\_hw)

# Testing set (2015-16 season)

playing\_stat\_test = playing\_stat[5700:]

#saving the final dataset

playing\_stat.to\_csv('/content/Datasets/final\_dataset.csv')

#saving the test set

playing\_stat\_test.to\_csv("/content/Datasets/test.csv")

#loading the final dataset

dataset = pd.read\_csv('/content/Datasets/final\_dataset.csv')

dataset.head()

dataset.keys()

Index(['Unnamed: 0', 'Date', 'HomeTeam', 'AwayTeam', 'FTHG', 'FTAG', 'FTR',

'HTGS', 'ATGS', 'HTGC', 'ATGC', 'HM1', 'AM1', 'HM2', 'AM2', 'HM3',

'AM3', 'HM4', 'AM4', 'HM5', 'AM5', 'MW', 'HTFormPtsStr', 'ATFormPtsStr',

'HTFormPts', 'ATFormPts', 'HTWinStreak3', 'HTWinStreak5',

'HTLossStreak3', 'HTLossStreak5', 'ATWinStreak3', 'ATWinStreak5',

'ATLossStreak3', 'ATLossStreak5'],

dtype='object')

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

# Assuming 'dataset' is your DataFrame

# Select only numeric columns for correlation matrix calculation

numeric\_dataset = dataset.select\_dtypes(include=[float, int])

# Alternatively, if you want to drop non-numeric columns

# numeric\_dataset = dataset.drop(columns=['Date', 'HomeTeam', 'AwayTeam', 'FTR'])

# Compute the correlation matrix

correlation\_matrix = numeric\_dataset.corr()

# Set up the matplotlib figure

plt.figure(figsize=(20,10))

# Draw the heatmap

sns.heatmap(correlation\_matrix, annot=True, fmt='.2f', cmap='coolwarm', linewidths=0.5)

# Show the plot

plt.show()

# Print all column names in the dataset to verify their existence

print(dataset.columns)

# Now safely drop columns if they exist

columns\_to\_drop = ['Date', 'HomeTeam', 'AwayTeam', 'FTHG', 'FTAG',

'HTGS', 'ATGS', 'HTGC', 'ATGC',

'HM4', 'HM5', 'AM4', 'AM5', 'MW', 'HTFormPtsStr',

'ATFormPtsStr', 'HTFormPts', 'ATFormPts', 'HTWinStreak3',

'HTWinStreak5', 'HTLossStreak3', 'HTLossStreak5', 'ATWinStreak3',

'ATWinStreak5', 'ATLossStreak3', 'ATLossStreak5', 'DiffPts']

# Drop only columns that exist in the dataset

columns\_to\_drop = [col for col in columns\_to\_drop if col in dataset.columns]

dataset2 = dataset.copy().drop(columns=columns\_to\_drop)

# Check the resulting dataframe

print(dataset2.head())

Index(['Unnamed: 0', 'Date', 'HomeTeam', 'AwayTeam', 'FTHG', 'FTAG', 'FTR',

'HTGS', 'ATGS', 'HTGC', 'ATGC', 'HM1', 'AM1', 'HM2', 'AM2', 'HM3',

'AM3', 'HM4', 'AM4', 'HM5', 'AM5', 'MW', 'HTFormPtsStr', 'ATFormPtsStr',

'HTFormPts', 'ATFormPts', 'HTWinStreak3', 'HTWinStreak5',

'HTLossStreak3', 'HTLossStreak5', 'ATWinStreak3', 'ATWinStreak5',

'ATLossStreak3', 'ATLossStreak5'],

dtype='object')

Unnamed: 0 FTR HM1 AM1 HM2 AM2 HM3 AM3

0 0 H M M M M M M

1 1 H M M M M M M

2 2 NH M M M M M M

3 3 NH M M M M M M

4 4 H M M M M M M

dataset2.keys()

Index(['Unnamed: 0', 'FTR', 'HM1', 'AM1', 'HM2', 'AM2', 'HM3', 'AM3'], dtype='object')

dataset2.head(10)

#what is the win rate for the home team?

# Total number of matches.

n\_matches = dataset2.shape[0]

# Calculate number of features. -1 because we are saving one as the target variable (win/lose/draw)

n\_features = dataset2.shape[1] - 1

# Calculate matches won by home team.

n\_homewins = len(dataset2[dataset2.FTR == 'H'])

# Calculate win rate for home team.

win\_rate = (float(n\_homewins) / (n\_matches)) \* 100

# Print the results

print("Total number of matches: {}".format(n\_matches))

print ("Number of features: {}".format(n\_features))

print( "Number of matches won by home team: {}".format(n\_homewins))

print ("Win rate of home team: {:.2f}%".format(win\_rate))

Total number of matches: 6840

Number of features: 7

Number of matches won by home team: 3176

Win rate of home team: 46.43%

import pandas as pd

from sklearn.preprocessing import scale

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

# Assuming dataset2 is already defined and contains the data

# Split the dataset into features and target variable

X\_all = dataset2.drop(['FTR'], axis=1)

y\_all = dataset2['FTR']

# Check available columns in X\_all

print("Columns in X\_all:", X\_all.columns)

# Define the columns you want to standardize

cols\_to\_scale = ['HTGD', 'ATGD', 'HTP', 'ATP']

# Verify if the columns are present

missing\_cols = [col for col in cols\_to\_scale if col not in X\_all.columns]

if missing\_cols:

    print(f"The following columns are missing: {missing\_cols}")

else:

    # Standardizing numerical features

    X\_all[cols\_to\_scale] = scale(X\_all[cols\_to\_scale])

    # Convert categorical features to dummy variables

    X\_all = pd.get\_dummies(X\_all, columns=['HM1', 'HM2', 'HM3', 'AM1', 'AM2', 'AM3'], drop\_first=True)

    # Split the data into training and testing sets

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_all, y\_all, test\_size=0.3, random\_state=2, stratify=y\_all)

    print("Shape of X\_train:", X\_train.shape)

    print("Shape of X\_test:", X\_test.shape)

Columns in X\_all: Index(['Unnamed: 0', 'HM1', 'AM1', 'HM2', 'AM2', 'HM3', 'AM3'], dtype='object')

The following columns are missing: ['HTGD', 'ATGD', 'HTP', 'ATP']

import pandas as pd

# Check available columns in X\_all before attempting to convert types

print("Columns in X\_all:", X\_all.columns)

# Check if 'HM1', 'HM2', 'HM3', 'AM1', 'AM2', 'AM3' exist in X\_all

expected\_cols = ['HM1', 'HM2', 'HM3', 'AM1', 'AM2', 'AM3']

missing\_cols = [col for col in expected\_cols if col not in X\_all.columns]

if missing\_cols:

    print(f"The following columns are missing: {missing\_cols}")

else:

    # Convert last 3 wins columns to string type

    X\_all['HM1'] = X\_all['HM1'].astype('str')

    X\_all['HM2'] = X\_all['HM2'].astype('str')

    X\_all['HM3'] = X\_all['HM3'].astype('str')

    X\_all['AM1'] = X\_all['AM1'].astype('str')

    X\_all['AM2'] = X\_all['AM2'].astype('str')

    X\_all['AM3'] = X\_all['AM3'].astype('str')

    # Function to preprocess features

    def preprocess\_features(X):

        ''' Preprocesses the football data and converts categorical variables into dummy variables. '''

        output = pd.DataFrame(index=X.index)

        for col, col\_data in X.items():

            if col\_data.dtype == object:

                col\_data = pd.get\_dummies(col\_data, prefix=col)

            output = output.join(col\_data)

        return output

    # Apply preprocessing

    X\_all = preprocess\_features(X\_all)

    print("Processed feature columns ({} total features):\n{}".format(len(X\_all.columns), list(X\_all.columns)))

Columns in X\_all: Index(['Unnamed: 0', 'HM1', 'AM1', 'HM2', 'AM2', 'HM3', 'AM3'], dtype='object')

Processed feature columns (25 total features):

['Unnamed: 0', 'HM1\_D', 'HM1\_L', 'HM1\_M', 'HM1\_W', 'AM1\_D', 'AM1\_L', 'AM1\_M', 'AM1\_W', 'HM2\_D', 'HM2\_L', 'HM2\_M', 'HM2\_W', 'AM2\_D', 'AM2\_L', 'AM2\_M', 'AM2\_W', 'HM3\_D', 'HM3\_L', 'HM3\_M', 'HM3\_W', 'AM3\_D', 'AM3\_L', 'AM3\_M', 'AM3\_W']

X\_all.head(10)

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

# Shuffle and split the dataset into training and testing set.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_all, y\_all,

test\_size = 0.3,

random\_state = 2,

stratify = y\_all)

from sklearn.impute import SimpleImputer

from sklearn.linear\_model import LogisticRegression

from sklearn.pipeline import Pipeline

# Create an imputer object with a strategy (mean, median, most\_frequent, etc.)

imputer = SimpleImputer(strategy='mean')

# Create a pipeline that first imputes missing values, then applies Logistic Regression

pipeline = Pipeline([

('imputer', imputer),

('classifier', LogisticRegression(random\_state=0))

])

# Fit the pipeline on the training set

pipeline.fit(X\_train, y\_train)

Pipeline

SimpleImputer

LogisticRegression

from sklearn.impute import SimpleImputer

from sklearn.linear\_model import LogisticRegression

from sklearn.pipeline import Pipeline

# Define the columns with missing values

missing\_columns = ['HTP', 'ATP']

# Create a pipeline with an imputer and a logistic regression classifier

pipeline = Pipeline([

('imputer', SimpleImputer(strategy='mean')), # Impute missing values with the mean

('classifier', LogisticRegression(random\_state=0))

])

# Fit the pipeline on the training set

pipeline.fit(X\_train, y\_train)

# Optional: You can also transform the data to check the imputed values

X\_train\_imputed = pipeline.named\_steps['imputer'].transform(X\_train)

X\_test\_imputed = pipeline.named\_steps['imputer'].transform(X\_test)

# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from datetime import datetime as dt

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

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion\_matrix, classification\_report

from xgboost import XGBClassifier

from sklearn.model\_selection import GridSearchCV

from sklearn.preprocessing import scale

from sklearn.metrics import f1\_score, make\_scorer

# Extract relevant columns and concatenate data

columns\_req = ['Date', 'HomeTeam', 'AwayTeam', 'FTHG', 'FTAG', 'FTR']

playing\_statistics = [df[columns\_req] for df in datasets]

# Function to calculate goals scored

def get\_goals\_scored(playing\_stat):

teams = {team: [] for team in playing\_stat['HomeTeam'].unique()}

for i in range(len(playing\_stat)):

HTGS = playing\_stat.iloc[i]['FTHG']

ATGS = playing\_stat.iloc[i]['FTAG']

teams[playing\_stat.iloc[i].HomeTeam].append(HTGS)

teams[playing\_stat.iloc[i].AwayTeam].append(ATGS)

GoalsScored = pd.DataFrame(data=teams, index=[i for i in range(1, 39)]).T

GoalsScored[0] = 0

for i in range(2, 39):

GoalsScored[i] = GoalsScored[i] + GoalsScored[i-1]

return GoalsScored

# Function to calculate goals conceded

def get\_goals\_conceded(playing\_stat):

teams = {team: [] for team in playing\_stat['HomeTeam'].unique()}

for i in range(len(playing\_stat)):

ATGC = playing\_stat.iloc[i]['FTHG']

HTGC = playing\_stat.iloc[i]['FTAG']

teams[playing\_stat.iloc[i].HomeTeam].append(HTGC)

teams[playing\_stat.iloc[i].AwayTeam].append(ATGC)

GoalsConceded = pd.DataFrame(data=teams, index=[i for i in range(1, 39)]).T

GoalsConceded[0] = 0

for i in range(2, 39):

GoalsConceded[i] = GoalsConceded[i] + GoalsConceded[i-1]

return GoalsConceded

# Function to get goals scored and conceded statistics

def get\_gss(playing\_stat):

GC = get\_goals\_conceded(playing\_stat)

GS = get\_goals\_scored(playing\_stat)

HTGS, ATGS, HTGC, ATGC = [], [], [], []

j = 0

for i in range(len(playing\_stat)):

ht = playing\_stat.iloc[i].HomeTeam

at = playing\_stat.iloc[i].AwayTeam

HTGS.append(GS.loc[ht][j])

ATGS.append(GS.loc[at][j])

HTGC.append(GC.loc[ht][j])

ATGC.append(GC.loc[at][j])

if ((i + 1) % 10) == 0:

j += 1

playing\_stat['HTGS'] = HTGS

playing\_stat['ATGS'] = ATGS

playing\_stat['HTGC'] = HTGC

playing\_stat['ATGC'] = ATGC

return playing\_stat

# Apply the function to each dataset

playing\_statistics = [get\_gss(df) for df in playing\_statistics]

# Function to calculate cumulative points

def get\_points(result):

if result == 'W':

return 3

elif result == 'D':

return 1

else:

return 0

def get\_cuml\_points(matchres):

matchres\_points = matchres.applymap(get\_points)

for i in range(2, 39):

matchres\_points[i] = matchres\_points[i] + matchres\_points[i-1]

matchres\_points.insert(column=0, loc=0, value=[0] \* len(matchres\_points))

return matchres\_points

def get\_matchres(playing\_stat):

teams = {team: [] for team in playing\_stat['HomeTeam'].unique()}

for i in range(len(playing\_stat)):

if playing\_stat.iloc[i].FTR == 'H':

teams[playing\_stat.iloc[i].HomeTeam].append('W')

teams[playing\_stat.iloc[i].AwayTeam].append('L')

elif playing\_stat.iloc[i].FTR == 'A':

teams[playing\_stat.iloc[i].AwayTeam].append('W')

teams[playing\_stat.iloc[i].HomeTeam].append('L')

else:

teams[playing\_stat.iloc[i].AwayTeam].append('D')

teams[playing\_stat.iloc[i].HomeTeam].append('D')

return pd.DataFrame(data=teams, index=[i for i in range(1, 39)]).T

def get\_agg\_points(playing\_stat):

matchres = get\_matchres(playing\_stat)

cum\_pts = get\_cuml\_points(matchres)

HTP, ATP = [], []

j = 0

for i in range(len(playing\_stat)):

ht = playing\_stat.iloc[i].HomeTeam

at = playing\_stat.iloc[i].AwayTeam

HTP.append(cum\_pts.loc[ht][j])

ATP.append(cum\_pts.loc[at][j])

if ((i + 1) % 10) == 0:

j += 1

playing\_stat['HTP'] = HTP

playing\_stat['ATP'] = ATP

return playing\_stat

# Apply the function to each dataset

playing\_statistics = [get\_agg\_points(df) for df in playing\_statistics]

# Function to get team form

def get\_form(playing\_stat, num):

form = get\_matchres(playing\_stat)

form\_final = form.copy()

for i in range(num, 39):

form\_final[i] = ''

j = 0

while j < num:

form\_final[i] += form[i-j]

j += 1

return form\_final

def add\_form(playing\_stat, num):

form = get\_form(playing\_stat, num)

h = ['M' for \_ in range(num \* 10)]

a = ['M' for \_ in range(num \* 10)]

j = num

for i in range(num \* 10, len(playing\_stat)):

ht = playing\_stat.iloc[i].HomeTeam

at = playing\_stat.iloc[i].AwayTeam

h.append(form.loc[ht][j][num-1])

a.append(form.loc[at][j][num-1])

if ((i + 1) % 10) == 0:

j += 1

playing\_stat[f'HM{num}'] = h

playing\_stat[f'AM{num}'] = a

return playing\_stat

def add\_form\_df(playing\_statistics):

for num in range(1, 6):

playing\_statistics = [add\_form(df, num) for df in playing\_statistics]

return playing\_statistics

# Add form data to datasets

playing\_statistics = add\_form\_df(playing\_statistics)

# Rearrange columns

cols = ['Date', 'HomeTeam', 'AwayTeam', 'FTHG', 'FTAG', 'FTR', 'HTGS', 'ATGS', 'HTGC', 'ATGC', 'HTP', 'ATP',

'HM1', 'HM2', 'HM3', 'HM4', 'HM5', 'AM1', 'AM2', 'AM3', 'AM4', 'AM5']

playing\_statistics = [df[cols] for df in playing\_statistics]

# Function to add matchweek

def get\_mw(playing\_stat):

j = 1

MatchWeek = []

for i in range(len(playing\_stat)):

MatchWeek.append(j)

if ((i + 1) % 10) == 0:

j += 1

playing\_stat['MW'] = MatchWeek

return playing\_stat

# Add matchweek to datasets

playing\_statistics = [get\_mw(df) for df in playing\_statistics]

# Combine all data into a single DataFrame

playing\_stat = pd.concat(playing\_statistics, ignore\_index=True)

# Add form points

playing\_stat['HTFormPtsStr'] = playing\_stat['HM1'] + playing\_stat['HM2'] + playing\_stat['HM3'] + playing\_stat['HM4'] + playing\_stat['HM5']

playing\_stat['ATFormPtsStr'] = playing\_stat['AM1'] + playing\_stat['AM2'] + playing\_stat['AM3'] + playing\_stat['AM4'] + playing\_stat['AM5']

playing\_stat['HTFormPts'] = playing\_stat['HTFormPtsStr'].apply(get\_form\_points)

playing\_stat['ATFormPts'] = playing\_stat['ATFormPtsStr'].apply(get\_form\_points)

# Identify Win/Loss Streaks

playing\_stat['HTWinStreak3'] = playing\_stat['HTFormPtsStr'].apply(get\_3game\_ws)

playing\_stat['HTWinStreak5'] = playing\_stat['HTFormPtsStr'].apply(get\_5game\_ws)

playing\_stat['HTLossStreak3'] = playing\_stat['HTFormPtsStr'].apply(get\_3game\_ls)

playing\_stat['HTLossStreak5'] = playing\_stat['HTFormPtsStr'].apply(get\_5game\_ls)

playing\_stat['ATWinStreak3'] = playing\_stat['ATFormPtsStr'].apply(get\_3game\_ws)

playing\_stat['ATWinStreak5'] = playing\_stat['ATFormPtsStr'].apply(get\_5game\_ws)

playing\_stat['ATLossStreak3'] = playing\_stat['ATFormPtsStr'].apply(get\_3game\_ls)

playing\_stat['ATLossStreak5'] = playing\_stat['ATFormPtsStr'].apply(get\_5game\_ls)

# Calculate Goal Difference

playing\_stat['HTGD'] = playing\_stat['HTGS'] - playing\_stat['HTGC']

playing\_stat['ATGD'] = playing\_stat['ATGS'] - playing\_stat['ATGC']

# Calculate DiffPts and DiffFormPts

playing\_stat['DiffPts'] = playing\_stat['HTP'] - playing\_stat['ATP']

playing\_stat['DiffFormPts'] = playing\_stat['HTFormPts'] - playing\_stat['ATFormPts']

# Scale features by Matchweek

playing\_stat.MW = playing\_stat.MW.astype(float)

cols = ['HTGD', 'ATGD', 'DiffPts', 'DiffFormPts', 'HTP', 'ATP']

for col in cols:

playing\_stat[col] = playing\_stat[col] / playing\_stat.MW

# Transform target variable

def only\_hw(string):

if string == 'H':

return 'H'

else:

return 'NH'

playing\_stat['FTR'] = playing\_stat.FTR.apply(only\_hw)

# Save final dataset

playing\_stat.to\_csv('/content/final\_dataset.csv', index=False)

# Load the dataset

dataset = pd.read\_csv('/content/final\_dataset.csv')

# Drop columns to prevent multicollinearity

dataset2 = dataset.copy().drop(columns=['Date', 'HomeTeam', 'AwayTeam', 'FTHG', 'FTAG', 'HTGS', 'ATGS', 'HTGC', 'ATGC',

'HM4', 'HM5', 'AM4', 'AM5', 'MW', 'HTFormPtsStr', 'ATFormPtsStr',

'HTWinStreak3', 'HTWinStreak5', 'HTLossStreak3', 'HTLossStreak5',

'ATWinStreak3', 'ATWinStreak5', 'ATLossStreak3', 'ATLossStreak5',

'DiffPts'])

# Split the dataset into features and target variable

X\_all = dataset2.drop(['FTR'], axis=1)

y\_all = dataset2['FTR']

# Standardizing numerical features

cols = [['HTGD', 'ATGD', 'HTP', 'ATP']]

for col in cols:

X\_all[col] = scale(X\_all[col])

# Convert categorical features to dummy variables

X\_all = pd.get\_dummies(X\_all, columns=['HM1', 'HM2', 'HM3', 'AM1', 'AM2', 'AM3'], drop\_first=True)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_all, y\_all, test\_size=0.3, random\_state=2, stratify=y\_all)

# Function to evaluate model performance

def evaluate\_model(model, X\_test, y\_test):

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, annot=True, fmt='d')

plt.show()

print(classification\_report(y\_test, y\_pred))

from sklearn.preprocessing import LabelEncoder

# Encode target labels with value between 0 and n\_classes-1

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# Logistic Regression

lr\_model = LogisticRegression(random\_state=0)

lr\_model.fit(X\_train, y\_train\_encoded)

evaluate\_model(lr\_model, X\_test, y\_test\_encoded)

# SVM

svm\_model = SVC(kernel='rbf', random\_state=0)

svm\_model.fit(X\_train, y\_train\_encoded)

evaluate\_model(svm\_model, X\_test, y\_test\_encoded)

# Random Forest

rf\_model = RandomForestClassifier(criterion='gini', n\_estimators=700, min\_samples\_split=10, min\_samples\_leaf=1,

max\_features='sqrt', oob\_score=True, random\_state=1, n\_jobs=-1)

rf\_model.fit(X\_train, y\_train\_encoded)

evaluate\_model(rf\_model, X\_test, y\_test\_encoded)

# XGBoost

xgb\_model = XGBClassifier(seed=82)

xgb\_model.fit(X\_train, y\_train\_encoded)

evaluate\_model(xgb\_model, X\_test, y\_test\_encoded)

# Hyperparameter Tuning for XGBoost

parameters = {

'learning\_rate': [0.1],

'n\_estimators': [40],

'max\_depth': [3],

'min\_child\_weight': [3],

'gamma': [0.4],

'subsample': [0.8],

'colsample\_bytree': [0.8],

'scale\_pos\_weight': [1],

'reg\_alpha': [1e-5]

}

f1\_scorer = make\_scorer(f1\_score, pos\_label=1)

grid\_obj = GridSearchCV(XGBClassifier(seed=2), param\_grid=parameters, scoring=f1\_scorer, cv=5)

grid\_obj = grid\_obj.fit(X\_train, y\_train\_encoded)

best\_xgb = grid\_obj.best\_estimator\_

print("Best XGBoost model:", best\_xgb)

# Evaluate the best model

evaluate\_model(best\_xgb, X\_test, y\_test\_encoded)

# Extract relevant columns and concatenate data

columns\_req = ['Date', 'HomeTeam', 'AwayTeam', 'FTHG', 'FTAG', 'FTR']

playing\_statistics = [df[columns\_req] for df in datasets]

# Function to calculate goals scored

def get\_goals\_scored(playing\_stat):

    teams = {team: [] for team in playing\_stat['HomeTeam'].unique()}

    for i in range(len(playing\_stat)):

        HTGS = playing\_stat.iloc[i]['FTHG']

        ATGS = playing\_stat.iloc[i]['FTAG']

        teams[playing\_stat.iloc[i].HomeTeam].append(HTGS)

        teams[playing\_stat.iloc[i].AwayTeam].append(ATGS)

    GoalsScored = pd.DataFrame(data=teams, index=[i for i in range(1, 39)]).T

    GoalsScored[0] = 0

    for i in range(2, 39):

        GoalsScored[i] = GoalsScored[i] + GoalsScored[i-1]

    return GoalsScored

# Function to calculate goals conceded

def get\_goals\_conceded(playing\_stat):

    teams = {team: [] for team in playing\_stat['HomeTeam'].unique()}

    for i in range(len(playing\_stat)):

        ATGC = playing\_stat.iloc[i]['FTHG']

        HTGC = playing\_stat.iloc[i]['FTAG']

        teams[playing\_stat.iloc[i].HomeTeam].append(HTGC)

        teams[playing\_stat.iloc[i].AwayTeam].append(ATGC)

    GoalsConceded = pd.DataFrame(data=teams, index=[i for i in range(1, 39)]).T

    GoalsConceded[0] = 0

    for i in range(2, 39):

        GoalsConceded[i] = GoalsConceded[i] + GoalsConceded[i-1]

    return GoalsConceded

# Function to get goals scored and conceded statistics

def get\_gss(playing\_stat):

    GC = get\_goals\_conceded(playing\_stat)

    GS = get\_goals\_scored(playing\_stat)

    HTGS, ATGS, HTGC, ATGC = [], [], [], []

    j = 0

    for i in range(len(playing\_stat)):

        ht = playing\_stat.iloc[i].HomeTeam

        at = playing\_stat.iloc[i].AwayTeam

        HTGS.append(GS.loc[ht][j])

        ATGS.append(GS.loc[at][j])

        HTGC.append(GC.loc[ht][j])

        ATGC.append(GC.loc[at][j])

        if ((i + 1) % 10) == 0:

            j += 1

    playing\_stat['HTGS'] = HTGS

    playing\_stat['ATGS'] = ATGS

    playing\_stat['HTGC'] = HTGC

    playing\_stat['ATGC'] = ATGC

    return playing\_stat

# Apply the function to each dataset

playing\_statistics = [get\_gss(df) for df in playing\_statistics]

# Function to calculate cumulative points

def get\_points(result):

    if result == 'W':

        return 3

    elif result == 'D':

        return 1

    else:

        return 0

def get\_cuml\_points(matchres):

    matchres\_points = matchres.applymap(get\_points)

    for i in range(2, 39):

        matchres\_points[i] = matchres\_points[i] + matchres\_points[i-1]

    matchres\_points.insert(column=0, loc=0, value=[0] \* len(matchres\_points))

    return matchres\_points

def get\_matchres(playing\_stat):

    teams = {team: [] for team in playing\_stat['HomeTeam'].unique()}

    for i in range(len(playing\_stat)):

        if playing\_stat.iloc[i].FTR == 'H':

            teams[playing\_stat.iloc[i].HomeTeam].append('W')

            teams[playing\_stat.iloc[i].AwayTeam].append('L')

        elif playing\_stat.iloc[i].FTR == 'A':

            teams[playing\_stat.iloc[i].AwayTeam].append('W')

            teams[playing\_stat.iloc[i].HomeTeam].append('L')

        else:

            teams[playing\_stat.iloc[i].AwayTeam].append('D')

            teams[playing\_stat.iloc[i].HomeTeam].append('D')

    return pd.DataFrame(data=teams, index=[i for i in range(1, 39)]).T

def get\_agg\_points(playing\_stat):

    matchres = get\_matchres(playing\_stat)

    cum\_pts = get\_cuml\_points(matchres)

    HTP, ATP = [], []

    j = 0

    for i in range(len(playing\_stat)):

        ht = playing\_stat.iloc[i].HomeTeam

        at = playing\_stat.iloc[i].AwayTeam

        HTP.append(cum\_pts.loc[ht][j])

        ATP.append(cum\_pts.loc[at][j])

        if ((i + 1) % 10) == 0:

            j += 1

    playing\_stat['HTP'] = HTP

    playing\_stat['ATP'] = ATP

    return playing\_stat

# Apply the function to each dataset

playing\_statistics = [get\_agg\_points(df) for df in playing\_statistics]

# Function to get team form

def get\_form(playing\_stat, num):

    form = get\_matchres(playing\_stat)

    form\_final = form.copy()

    for i in range(num, 39):

        form\_final[i] = ''

        j = 0

        while j < num:

            form\_final[i] += form[i-j]

            j += 1

    return form\_final

def add\_form(playing\_stat, num):

    form = get\_form(playing\_stat, num)

    h = ['M' for \_ in range(num \* 10)]

    a = ['M' for \_ in range(num \* 10)]

    j = num

    for i in range(num \* 10, len(playing\_stat)):

        ht = playing\_stat.iloc[i].HomeTeam

        at = playing\_stat.iloc[i].AwayTeam

        h.append(form.loc[ht][j][num-1])

        a.append(form.loc[at][j][num-1])

        if ((i + 1) % 10) == 0:

            j += 1

    playing\_stat[f'HM{num}'] = h

    playing\_stat[f'AM{num}'] = a

    return playing\_stat

def add\_form\_df(playing\_statistics):

    for num in range(1, 6):

        playing\_statistics = [add\_form(df, num) for df in playing\_statistics]

    return playing\_statistics

# Add form data to datasets

playing\_statistics = add\_form\_df(playing\_statistics)

# Rearrange columns

cols = ['Date', 'HomeTeam', 'AwayTeam', 'FTHG', 'FTAG', 'FTR', 'HTGS', 'ATGS', 'HTGC', 'ATGC', 'HTP', 'ATP',

        'HM1', 'HM2', 'HM3', 'HM4', 'HM5', 'AM1', 'AM2', 'AM3', 'AM4', 'AM5']

playing\_statistics = [df[cols] for df in playing\_statistics]

# Function to add matchweek

def get\_mw(playing\_stat):

    j = 1

    MatchWeek = []

    for i in range(len(playing\_stat)):

        MatchWeek.append(j)

        if ((i + 1) % 10) == 0:

            j += 1

    playing\_stat['MW'] = MatchWeek

    return playing\_stat

# Add matchweek to datasets

playing\_statistics = [get\_mw(df) for df in playing\_statistics]

# Combine all data into a single DataFrame

playing\_stat = pd.concat(playing\_statistics, ignore\_index=True)

# Add form points

playing\_stat['HTFormPtsStr'] = playing\_stat['HM1'] + playing\_stat['HM2'] + playing\_stat['HM3'] + playing\_stat['HM4'] + playing\_stat['HM5']

playing\_stat['ATFormPtsStr'] = playing\_stat['AM1'] + playing\_stat['AM2'] + playing\_stat['AM3'] + playing\_stat['AM4'] + playing\_stat['AM5']

playing\_stat['HTFormPts'] = playing\_stat['HTFormPtsStr'].apply(get\_form\_points)

playing\_stat['ATFormPts'] = playing\_stat['ATFormPtsStr'].apply(get\_form\_points)

# Identify Win/Loss Streaks

playing\_stat['HTWinStreak3'] = playing\_stat['HTFormPtsStr'].apply(get\_3game\_ws)

playing\_stat['HTWinStreak5'] = playing\_stat['HTFormPtsStr'].apply(get\_5game\_ws)

playing\_stat['HTLossStreak3'] = playing\_stat['HTFormPtsStr'].apply(get\_3game\_ls)

playing\_stat['HTLossStreak5'] = playing\_stat['HTFormPtsStr'].apply(get\_5game\_ls)

playing\_stat['ATWinStreak3'] = playing\_stat['ATFormPtsStr'].apply(get\_3game\_ws)

playing\_stat['ATWinStreak5'] = playing\_stat['ATFormPtsStr'].apply(get\_5game\_ws)

playing\_stat['ATLossStreak3'] = playing\_stat['ATFormPtsStr'].apply(get\_3game\_ls)

playing\_stat['ATLossStreak5'] = playing\_stat['ATFormPtsStr'].apply(get\_5game\_ls)

# Calculate Goal Difference

playing\_stat['HTGD'] = playing\_stat['HTGS'] - playing\_stat['HTGC']

playing\_stat['ATGD'] = playing\_stat['ATGS'] - playing\_stat['ATGC']

# Calculate DiffPts and DiffFormPts

playing\_stat['DiffPts'] = playing\_stat['HTP'] - playing\_stat['ATP']

playing\_stat['DiffFormPts'] = playing\_stat['HTFormPts'] - playing\_stat['ATFormPts']

# Scale features by Matchweek

playing\_stat.MW = playing\_stat.MW.astype(float)

cols = ['HTGD', 'ATGD', 'DiffPts', 'DiffFormPts', 'HTP', 'ATP']

for col in cols:

    playing\_stat[col] = playing\_stat[col] / playing\_stat.MW

# Transform target variable

def only\_hw(string):

    if string == 'H':

        return 'H'

    else:

        return 'NH'

playing\_stat['FTR'] = playing\_stat.FTR.apply(only\_hw)

# Save final dataset

playing\_stat.to\_csv('/content/final\_dataset.csv', index=False)

# Load the dataset

dataset = pd.read\_csv('/content/final\_dataset.csv')

# Drop columns to prevent multicollinearity

dataset2 = dataset.copy().drop(columns=['Date', 'HomeTeam', 'AwayTeam', 'FTHG', 'FTAG', 'HTGS', 'ATGS', 'HTGC', 'ATGC',

                                        'HM4', 'HM5', 'AM4', 'AM5', 'MW', 'HTFormPtsStr', 'ATFormPtsStr',

                                        'HTWinStreak3', 'HTWinStreak5', 'HTLossStreak3', 'HTLossStreak5',

                                        'ATWinStreak3', 'ATWinStreak5', 'ATLossStreak3', 'ATLossStreak5',

                                        'DiffPts'])

# Split the dataset into features and target variable

X\_all = dataset2.drop(['FTR'], axis=1)

y\_all = dataset2['FTR']

# Standardizing numerical features

cols = [['HTGD', 'ATGD', 'HTP', 'ATP']]

for col in cols:

    X\_all[col] = scale(X\_all[col])

# Convert categorical features to dummy variables

X\_all = pd.get\_dummies(X\_all, columns=['HM1', 'HM2', 'HM3', 'AM1', 'AM2', 'AM3'], drop\_first=True)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_all, y\_all, test\_size=0.3, random\_state=2, stratify=y\_all)

# Function to evaluate model performance

def evaluate\_model(model, X\_test, y\_test):

    y\_pred = model.predict(X\_test)

    cm = confusion\_matrix(y\_test, y\_pred)

    sns.heatmap(cm, annot=True, fmt='d')

    plt.show()

    print(classification\_report(y\_test, y\_pred))

from sklearn.preprocessing import LabelEncoder

# Encode target labels with value between 0 and n\_classes-1

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# Logistic Regression

lr\_model = LogisticRegression(random\_state=0)

lr\_model.fit(X\_train, y\_train\_encoded)

evaluate\_model(lr\_model, X\_test, y\_test\_encoded)

# SVM

svm\_model = SVC(kernel='rbf', random\_state=0)

svm\_model.fit(X\_train, y\_train\_encoded)

evaluate\_model(svm\_model, X\_test, y\_test\_encoded)

# Random Forest

rf\_model = RandomForestClassifier(criterion='gini', n\_estimators=700, min\_samples\_split=10, min\_samples\_leaf=1,

                                  max\_features='sqrt', oob\_score=True, random\_state=1, n\_jobs=-1)

rf\_model.fit(X\_train, y\_train\_encoded)

evaluate\_model(rf\_model, X\_test, y\_test\_encoded)

# XGBoost

xgb\_model = XGBClassifier(seed=82)

xgb\_model.fit(X\_train, y\_train\_encoded)

evaluate\_model(xgb\_model, X\_test, y\_test\_encoded)

# Hyperparameter Tuning for XGBoost

parameters = {

    'learning\_rate': [0.1],

    'n\_estimators': [40],

    'max\_depth': [3],

    'min\_child\_weight': [3],

    'gamma': [0.4],

    'subsample': [0.8],

    'colsample\_bytree': [0.8],

    'scale\_pos\_weight': [1],

    'reg\_alpha': [1e-5]

}

f1\_scorer = make\_scorer(f1\_score, pos\_label=1)

grid\_obj = GridSearchCV(XGBClassifier(seed=2), param\_grid=parameters, scoring=f1\_scorer, cv=5)

grid\_obj = grid\_obj.fit(X\_train, y\_train\_encoded)

best\_xgb = grid\_obj.best\_estimator\_

print("Best XGBoost model:", best\_xgb)

# Evaluate the best model

evaluate\_model(best\_xgb, X\_test, y\_test\_encoded)

precision recall f1-score support

0 0.64 0.55 0.59 953

1 0.65 0.74 0.69 1099

accuracy 0.65 2052

macro avg 0.65 0.64 0.64 2052

weighted avg 0.65 0.65 0.65 2052

precision recall f1-score support

0 0.68 0.50 0.57 953

1 0.65 0.79 0.71 1099

accuracy 0.66 2052

macro avg 0.66 0.65 0.64 2052

weighted avg 0.66 0.66 0.65 2052

precision recall f1-score support

0 0.62 0.55 0.58 953

1 0.64 0.72 0.68 1099

accuracy 0.64 2052

macro avg 0.63 0.63 0.63 2052

weighted avg 0.64 0.64 0.63 2052

precision recall f1-score support

0 0.57 0.53 0.55 953

1 0.62 0.66 0.64 1099

accuracy 0.60 2052

macro avg 0.60 0.59 0.59 2052

weighted avg 0.60 0.60 0.60 2052

Best XGBoost model: XGBClassifier(base\_score=None, booster=None, callbacks=None,

colsample\_bylevel=None, colsample\_bynode=None,

colsample\_bytree=0.8, device=None, early\_stopping\_rounds=None,

enable\_categorical=False, eval\_metric=None, feature\_types=None,

gamma=0.4, grow\_policy=None, importance\_type=None,

interaction\_constraints=None, learning\_rate=0.1, max\_bin=None,

max\_cat\_threshold=None, max\_cat\_to\_onehot=None,

max\_delta\_step=None, max\_depth=3, max\_leaves=None,

min\_child\_weight=3, missing=nan, monotone\_constraints=None,

multi\_strategy=None, n\_estimators=40, n\_jobs=None,

num\_parallel\_tree=None, random\_state=None, ...)

precision recall f1-score support

0 0.65 0.52 0.57 953

1 0.64 0.76 0.69 1099

accuracy 0.64 2052

macro avg 0.64 0.64 0.63 2052

weighted avg 0.64 0.64 0.64 2052

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import mean\_absolute\_error, r2\_score, mean\_absolute\_percentage\_error

# Function to evaluate model performance with additional metrics

def evaluate\_model\_with\_metrics(model, X\_test, y\_test):

y\_pred = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, annot=True, fmt='d')

plt.show()

print(classification\_report(y\_test, y\_pred))

# Calculate additional metrics

mae = mean\_absolute\_error(y\_test, y\_pred)

mape = mean\_absolute\_percentage\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Absolute Error (MAE): {mae}")

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

print(f"R-squared (R²): {r2}")

# Encode target labels with value between 0 and n\_classes-1

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# Logistic Regression

lr\_model = LogisticRegression(random\_state=0)

lr\_model.fit(X\_train, y\_train\_encoded)

evaluate\_model\_with\_metrics(lr\_model, X\_test, y\_test\_encoded)

# SVM

svm\_model = SVC(kernel='rbf', random\_state=0)

svm\_model.fit(X\_train, y\_train\_encoded)

evaluate\_model\_with\_metrics(svm\_model, X\_test, y\_test\_encoded)

# Random Forest

rf\_model = RandomForestClassifier(criterion='gini', n\_estimators=700, min\_samples\_split=10, min\_samples\_leaf=1,

max\_features='sqrt', oob\_score=True, random\_state=1, n\_jobs=-1)

rf\_model.fit(X\_train, y\_train\_encoded)

evaluate\_model\_with\_metrics(rf\_model, X\_test, y\_test\_encoded)

# XGBoost

xgb\_model = XGBClassifier(seed=82)

xgb\_model.fit(X\_train, y\_train\_encoded)

evaluate\_model\_with\_metrics(xgb\_model, X\_test, y\_test\_encoded)

# Gradient Boosting Machines (GBM)

gbm\_model = GradientBoostingClassifier(random\_state=0)

gbm\_model.fit(X\_train, y\_train\_encoded)

evaluate\_model\_with\_metrics(gbm\_model, X\_test, y\_test\_encoded)

# Hyperparameter Tuning for XGBoost

parameters = {

'learning\_rate': [0.1],

'n\_estimators': [40],

'max\_depth': [3],

'min\_child\_weight': [3],

'gamma': [0.4],

'subsample': [0.8],

'colsample\_bytree': [0.8],

'scale\_pos\_weight': [1],

'reg\_alpha': [1e-5]

}

f1\_scorer = make\_scorer(f1\_score, pos\_label=1)

grid\_obj = GridSearchCV(XGBClassifier(seed=2), param\_grid=parameters, scoring=f1\_scorer, cv=5)

grid\_obj = grid\_obj.fit(X\_train, y\_train\_encoded)

best\_xgb = grid\_obj.best\_estimator\_

print("Best XGBoost model:", best\_xgb)

# Evaluate the best XGBoost model.

evaluate\_model\_with\_metrics(best\_xgb, X\_test, y\_test\_encoded)

precision recall f1-score support

0 0.64 0.55 0.59 953

1 0.65 0.74 0.69 1099

accuracy 0.65 2052

macro avg 0.65 0.64 0.64 2052

weighted avg 0.65 0.65 0.65 2052

Mean Absolute Error (MAE): 0.3508771929824561

Mean Absolute Percentage Error (MAPE): 945931500680645.1

R-squared (R²): -0.41064995650916103

precision recall f1-score support

0 0.68 0.50 0.57 953

1 0.65 0.79 0.71 1099

accuracy 0.66 2052

macro avg 0.66 0.65 0.64 2052

weighted avg 0.66 0.66 0.65 2052

Mean Absolute Error (MAE): 0.34405458089668617

Mean Absolute Percentage Error (MAPE): 1049084123724706.1

R-squared (R²): -0.3832206517992607

precision recall f1-score support

0 0.62 0.55 0.58 953

1 0.64 0.72 0.68 1099

accuracy 0.64 2052

macro avg 0.63 0.63 0.63 2052

weighted avg 0.64 0.64 0.63 2052

Mean Absolute Error (MAE): 0.3635477582846004

Mean Absolute Percentage Error (MAPE): 950320974001669.0

R-squared (R²): -0.46159009382754723

precision recall f1-score support

0 0.57 0.53 0.55 953

1 0.62 0.66 0.64 1099

accuracy 0.60 2052

macro avg 0.60 0.59 0.59 2052

weighted avg 0.60 0.60 0.60 2052

Mean Absolute Error (MAE): 0.4010721247563353

Mean Absolute Percentage Error (MAPE): 978852550588324.1

R-squared (R²): -0.6124512697319993

precision recall f1-score support

0 0.63 0.52 0.57 953

1 0.64 0.74 0.69 1099

accuracy 0.64 2052

macro avg 0.64 0.63 0.63 2052

weighted avg 0.64 0.64 0.63 2052

Mean Absolute Error (MAE): 0.36208576998050684

Mean Absolute Percentage Error (MAPE): 1000799917193443.5

R-squared (R²): -0.4557123856754257

Best XGBoost model: XGBClassifier(base\_score=None, booster=None, callbacks=None,

colsample\_bylevel=None, colsample\_bynode=None,

colsample\_bytree=0.8, device=None, early\_stopping\_rounds=None,

enable\_categorical=False, eval\_metric=None, feature\_types=None,

gamma=0.4, grow\_policy=None, importance\_type=None,

interaction\_constraints=None, learning\_rate=0.1, max\_bin=None,

max\_cat\_threshold=None, max\_cat\_to\_onehot=None,

max\_delta\_step=None, max\_depth=3, max\_leaves=None,

min\_child\_weight=3, missing=nan, monotone\_constraints=None,

multi\_strategy=None, n\_estimators=40, n\_jobs=None,

num\_parallel\_tree=None, random\_state=None, ...)

precision recall f1-score support

0 0.65 0.52 0.57 953

1 0.64 0.76 0.69 1099

accuracy 0.64 2052

macro avg 0.64 0.64 0.63 2052

weighted avg 0.64 0.64 0.64 2052

Mean Absolute Error (MAE): 0.3562378167641326

Mean Absolute Percentage Error (MAPE): 1013968337156515.1

R-squared (R²): -0.43220155306693986

from sklearn.model\_selection import RandomizedSearchCV

import numpy as np

# Define the parameter grid for each model with reduced search space

param\_grid\_rf = {

'n\_estimators': [100, 200],

'max\_depth': [10, None],

'min\_samples\_split': [2, 10],

'min\_samples\_leaf': [1, 2],

'max\_features': ['sqrt', 'log2']

}

param\_grid\_gbm = {

'n\_estimators': [100, 200],

'learning\_rate': [0.01, 0.1],

'max\_depth': [3, 4],

'subsample': [0.8, 1.0],

'min\_samples\_split': [2, 5],

'min\_samples\_leaf': [1, 2]

}

param\_grid\_xgb = {

'learning\_rate': [0.1, 0.05],

'n\_estimators': [40, 100],

'max\_depth': [3, 4],

'min\_child\_weight': [1, 3],

'gamma': [0.1, 0.3],

'subsample': [0.8, 1.0],

'colsample\_bytree': [0.8, 1.0],

'reg\_alpha': [1e-5, 1e-2]

}

# Hyperparameter tuning for Random Forest using RandomizedSearchCV

grid\_rf = RandomizedSearchCV(RandomForestClassifier(random\_state=1), param\_distributions=param\_grid\_rf,

scoring=f1\_scorer, cv=3, n\_iter=10, n\_jobs=-1, random\_state=1)

grid\_rf.fit(X\_train, y\_train\_encoded)

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

print("Best Random Forest model:", best\_rf)

evaluate\_model\_with\_metrics(best\_rf, X\_test, y\_test\_encoded)

# Hyperparameter tuning for Gradient Boosting Machines (GBM) using RandomizedSearchCV

grid\_gbm = RandomizedSearchCV(GradientBoostingClassifier(random\_state=0), param\_distributions=param\_grid\_gbm,

scoring=f1\_scorer, cv=3, n\_iter=10, n\_jobs=-1, random\_state=1)

grid\_gbm.fit(X\_train, y\_train\_encoded)

best\_gbm = grid\_gbm.best\_estimator\_

print("Best GBM model:", best\_gbm)

evaluate\_model\_with\_metrics(best\_gbm, X\_test, y\_test\_encoded)

# Hyperparameter tuning for XGBoost using RandomizedSearchCV.

grid\_xgb = RandomizedSearchCV(XGBClassifier(seed=2), param\_distributions=param\_grid\_xgb,

scoring=f1\_scorer, cv=3, n\_iter=10, n\_jobs=-1, random\_state=1)

grid\_xgb.fit(X\_train, y\_train\_encoded)

best\_xgb = grid\_xgb.best\_estimator\_

print("Best XGBoost model:", best\_xgb)

evaluate\_model\_with\_metrics(best\_xgb, X\_test, y\_test\_encoded)

Best Random Forest model: RandomForestClassifier(max\_depth=10, max\_features='log2', min\_samples\_leaf=2,

min\_samples\_split=10, random\_state=1)

precision recall f1-score support

0 0.64 0.52 0.57 953

1 0.64 0.75 0.69 1099

accuracy 0.64 2052

macro avg 0.64 0.63 0.63 2052

weighted avg 0.64 0.64 0.64 2052

Mean Absolute Error (MAE): 0.3591617933723197

Mean Absolute Percentage Error (MAPE): 1005189390514467.4

R-squared (R²): -0.4439569693711829

Best GBM model: GradientBoostingClassifier(learning\_rate=0.01, max\_depth=4, min\_samples\_leaf=2,

random\_state=0, subsample=0.8)

precision recall f1-score support

0 0.67 0.46 0.55 953

1 0.63 0.80 0.71 1099

accuracy 0.64 2052

macro avg 0.65 0.63 0.63 2052

weighted avg 0.65 0.64 0.63 2052

Mean Absolute Error (MAE): 0.35721247563352826

Mean Absolute Percentage Error (MAPE): 1121510433521600.1

R-squared (R²): -0.43612002516835413

Best XGBoost model: XGBClassifier(base\_score=None, booster=None, callbacks=None,

colsample\_bylevel=None, colsample\_bynode=None,

colsample\_bytree=1.0, device=None, early\_stopping\_rounds=None,

enable\_categorical=False, eval\_metric=None, feature\_types=None,

gamma=0.1, grow\_policy=None, importance\_type=None,

interaction\_constraints=None, learning\_rate=0.05, max\_bin=None,

max\_cat\_threshold=None, max\_cat\_to\_onehot=None,

max\_delta\_step=None, max\_depth=3, max\_leaves=None,

min\_child\_weight=1, missing=nan, monotone\_constraints=None,

multi\_strategy=None, n\_estimators=40, n\_jobs=None,

num\_parallel\_tree=None, random\_state=None, ...)

precision recall f1-score support

0 0.66 0.50 0.56 953

1 0.64 0.77 0.70 1099

accuracy 0.64 2052

macro avg 0.65 0.63 0.63 2052

weighted avg 0.65 0.64 0.64 2052

Mean Absolute Error (MAE): 0.35526315789473684

Mean Absolute Percentage Error (MAPE): 1055668333706242.0

R-squared (R²): -0.42828308096552536