FOOTBALL MATCH PREDICTION

by

TEAM B

Anurag Kunchi (ID: GR69858)

Radhika Cheemala (ID: OH53378)

Suhana Shaik (ID:MM85299)

FINAL PROJECT REPORT

For

DATA 606 Capstone in Data Science

University of Maryland Baltimore County

2024

ABSTRACT

Football match prediction remains a challenging task due to the sport's inherent unpredictability and the multitude of factors influencing results. This project addressed this challenge by developing a sophisticated machine-learning model that accurately predicts match outcomes (home win, away win, or draw). Leveraging extensive historical data from major football leagues i.e England Premier League, La Liga, Serie A and UEFA Champions League including team performance metrics, player statistics, and historical head-to-head results, we trained and evaluated various machine learning algorithms to identify the most effective approach.

Our methodology involved comprehensive data preprocessing, feature engineering, and model selection processes. We experimented with several algorithms, including Random Forest, XGBoost, CatBoost, and LightGBM, to determine the most suitable model for this task. After rigorous testing and optimization, the LightGBM algorithm emerged as the superior performer, demonstrating the highest predictive accuracy and efficient processing of large datasets.

To enhance the model's practical applicability, we developed a user-friendly dashboard that allows users to input relevant match features and obtain outcome predictions. This interface facilitates easy interaction with the model, making it accessible to both casual fans and professional analysts.

The results showed promising accuracy in predicting match outcomes, with potential applications in sports betting, team strategy development, and fan engagement. The model's performance was evaluated using standard metrics such as accuracy, precision, and recall, with the LightGBM model achieving a combined accuracy of 67% across all leagues. Future work will focus on incorporating real-time data and expanding the model to cover additional leagues and tournaments, further improving its predictive capabilities and broadening its applicability in the football industry.

LIST OF ABBREVIATIONS AND SYMBOLS

- EDA: Exploratory Data Analysis
- XGBoost: Extreme Gradient Boosting
- LightGBM: Light Gradient Boosting Machine
- CatBoost: Categorical Boosting
- F1 Score: A measure combining precision and recall
- API: Application Programming Interface
- RF: Random Forest
- UEFA: Union of European Football Associations

ACKNOWLEDGMENTS

We, the team, acknowledge and thank Dr. Unal 'Zak' Sakoglu for his invaluable guidance, continuous support, and encouragement throughout this project. His insights and expertise have been instrumental in shaping our understanding and approach to this complex problem. We also extend our gratitude to our friends who provided feedback and assistance during the research process. Their contributions have significantly enhanced the quality of our work.

TEAM MEMBERS' CONTRIBUTIONS

Name	**Primary Duties**	**Key Achievements**
Anurag Kunchi	Data collection and preprocessing, Feature engineering, Performed overall EDA, Algorithm research, Performed CatBoost Algorithm	Compiled and preprocessed comprehensive dataset, Identified and engineered crucial predictive features, Conducted thorough exploratory data analysis, Researched and implemented CatBoost algorithm
Radhika Cheemala	Performed EDA on England Premier League and UEFA, Machine learning model development of XGBoost and LightGBM, Model Deployment	Conducted EDA on England Premier League and UEFA datasets,Developed and optimized XGBoost and LightGBM models And successfully deployed machine learning models for real-time predictions
Suhana Shaik	EDA on Serie A, La Liga, Feature selection, Performed Random Forest, Dashboard development	Created an intuitive user interface, Successfully integrated ML model with frontend

TABLE OF CONTENTS

1. INTRODUCTION

In the ever-evolving landscape of sports analytics, predicting sporting events' outcomes has become increasingly valuable. This project focuses on one of the world's most popular sports: football (soccer). We aim to develop a sophisticated machine learning model capable of predicting the outcomes of football matches across four major European leagues: the England Premier League, Italian Serie A, Spanish La Liga, and the UEFA Champions League.

The primary objective of this project is to create a predictive model that can accurately forecast the result of a football match, classifying the outcome into one of three categories: Home win, Away win, or Draw. This classification approach provides a comprehensive view of match results, catering to various stakeholders in the football industry, including fans, bookmakers, team managers, and sports analysts.

At its core, this project represents a complex classification task in machine learning. The model is trained on various data inputs, including historical team performance, player statistics, head-to-head records, and even real-time data such as current form and betting odds. By integrating these diverse data sources, we aim to capture the multifaceted nature of football match outcomes and provide predictions that surpass traditional statistical methods in accuracy and reliability.

This project aims to develop an advanced machine learning model for predicting football match outcomes across four major European leagues, including the UEFA Champions League. Building upon foundational work like Matheus Kempa's "Machine Learning Algorithms for Football Predictions", the model incorporates a wide range of data inputs, including historical team performance, player statistics, head-to-head records, and real-time data. The project employs a classification approach (Home win, Away win, or Draw) to provide clear, actionable insights, potentially using algorithms such as logistic regression, random forests, and gradient boosting machines as discussed in Kempa's article. [4]

The comprehensive nature of this project aligns with the step-by-step methodology outlined in "Building a Simple Football Prediction Model Using Machine Learning" by octosport.io, which emphasizes the importance of data collection, feature engineering, and model selection. However, our approach goes beyond a simple model, aiming to create a robust system capable of handling diverse playing styles and competitive environments. By integrating diverse data sources and applying sophisticated techniques, the project seeks to push the boundaries of prediction accuracy in sports analytics, with potential applications for bookmakers, team managers, analysts, and fans a like. [5]

In the following sections, we will delve into the details of our methodology, the extensive data analysis performed, the machine learning models implemented, and the results obtained. We will also discuss the challenges faced, the limitations of our approach, and potential areas for future improvement.

2. PROJECT OVERVIEW AND OBJECTIVES

This project aims to leverage the power of machine learning to predict football match outcomes with high accuracy. By analyzing historical data, current team form, and various other relevant factors, we seek to create a model that can provide reliable predictions for future matches.

The project encompasses four major European football leagues:

- England Premier League
- Italy Serie A
- Spain La Liga
- UEFA Champions League

This diverse selection of leagues allows us to evaluate the robustness and versatility of our predictive model across different football cultures and playing styles.

Our primary objectives are:

1. **Develop an Accurate Predictive Model**: Use advanced machine learning techniques to create a model that can predict match outcomes (Home win, Away win, or Draw) accurately across different leagues and seasons.

2. **Comparative Algorithm Analysis:** Implement and compare multiple machine learning algorithms, including LightGBM, Random Forest, XGBoost, and CatBoost. This comparison will help identify the most effective approach for football match prediction and provide insights into the strengths and weaknesses of each algorithm in this specific context.

3. **Feature Importance Analysis:** Conduct a thorough analysis to identify the most crucial factors influencing match outcomes. This analysis will improve our model's performance and provide valuable insights into the key determinants of football match results.

4. **Real-Time Data Integration:** Move beyond traditional models that rely solely on historical data by incorporating live team data and current betting odds. This integration of real-time information aims to enhance prediction accuracy and make our model more responsive to current conditions.

5. **Multi-League Applicability:** Ensure that our model performs consistently across different leagues, demonstrating its versatility and robustness in various football contexts.

6. **User-Friendly Interface Development:** Create an intuitive dashboard that allows users to input key match parameters and receive instant predictions. This interface will make our model accessible to a wide range of users, from casual fans to professional analysts.

7. **Scalability and Updatability:** Design the model and data pipeline to be easily updated with new data, ensuring the model's relevance and accuracy over time as new matches are played and new data becomes available.
8. **Interpretability:** Ensure that the model's predictions can be explained and interpreted, providing not just outcomes but also insights into why certain predictions are made. This interpretability is crucial for strategic decision-making in football.

By achieving these objectives, we aim to contribute significantly to the field of sports analytics and provide a valuable tool for various stakeholders in the football industry.

3. DATASET DESCRIPTION

3.1 Data Source

The primary data source for this project is FootyStats, a comprehensive football statistics website. We collected data spanning from 2008 to 2024, providing a robust historical context for our analysis.[1,2,3]

3.2 Dataset Overview

Our dataset comprises two main components:

1. Match Data:

- Total size: 21,792 rows, 67 columns
- Breakdown by league:
  - England Premier League: 6,460 matches
  - Italy Serie A: 6,081 matches
  - Spain La Liga: 6,080 matches
  - UEFA Champions League: 3,171 matches

2. Player Data:

- Total size: 58,803 rows, 278 columns
- Breakdown by league:
  - England Premier League: 10,618 players
  - Italy Serie A: 11,147 players
  - Spain La Liga: 10,147 players
  - UEFA Champions League: 26,891 players

3.3 Key Features

The dataset includes a wide range of features, including but not limited to:

- Team performance metrics (e.g., goals scored, goals conceded, possession percentage)
- Player statistics (e.g., goals, assists, minutes played)
- Match context (e.g., home/away status, league, season)
- Historical head-to-head records

- Current form indicators

- Betting odds

3.4 Data Preparation

To create our final dataset for analysis, we performed the following steps:

1. Merged the match data from all four leagues.

2. Group player data based on 'current club' and 'season' to get team experience metrics.

3. Merged the grouped player data with the combined match data.

4. Created a final dataset of 29,995 rows and 76 columns for implementation.

3.5 Data Splitting

We split our data into three sets to ensure robust model training and evaluation:

- Training Data: Matches from 2008 to 2022 (25,059 rows, 71 columns)

- Validation Data: Matches from 2022 (1,865 rows, 71 columns)

- Test Data: Matches from 2023 to present (3,071 rows, 71 columns)

This chronological split allows us to simulate real-world conditions where we train on historical data and test on more recent matches.

Final Columns Description:

Columns	**Explanation**
home_team_name	Name of the home team
away_team_name	Name of the away team
timestamp	Unix timestamp of the match
date_GMT	Date of the match in GMT
Pre-Match PPG (Home)	Points per game for home team before the match
Pre-Match PPG (Away)	Points per game for away team before the match
home_ppg	Home team's points per game (possibly updated)
away_ppg	Away team's points per game (possibly updated)
Home Team Pre-Match xG	Expected goals for home team before the match
Away Team Pre-Match xG	Expected goals for away team before the match
team_a_xg	Actual expected goals for team A (likely home team)
team_b_xg	Actual expected goals for team B (likely away team)

average_goals_per_match_pre_match	Average goals per match before this match
btts_percentage_pre_match	Percentage of matches where both teams scored before this match
odds_ft_home_team_win	Betting odds for home team win
odds_ft_draw	Betting odds for a draw
odds_ft_away_team_win	Betting odds for away team win
odds_btts_yes	Betting odds for both teams to score
odds_btts_no	Betting odds for at least one team not to score
stadium_name	Name of the stadium where the match was played
Season	The season in which the match was played
home_team_shots_on_target	Number of shots on target by the home team
away_team_shots_on_target	Number of shots on target by the away team
home_team_shots_off_target	Number of shots off target by the home team
away_team_shots_off_target	Number of shots off target by the away team
home_team_fouls	Number of fouls committed by the home team
away_team_fouls	Number of fouls committed by the away team
home_team_possession	Percentage of possession for the home team
away_team_possession	Percentage of possession for the away team
home_team_goal_count	Number of goals scored by the home team
away_team_goal_count	Number of goals scored by the away team
winner	The team that won the match
Current Club_x	Current club of a player (possibly home team's key player)
season_x	Season of the match (possibly redundant with 'Season')
league_x	League in which the match was played
age_home	Age of a key player from the home team
minutes_played_overall_home	Total minutes played by a key home player overall
minutes_played_home_home	Minutes played by a key home player in home matches
minutes_played_away_home	Minutes played by a key home player in away matches
appearances_overall_home	Total appearances of a key home player
appearances_home_home	Home appearances of a key home player
appearances_away_home	Away appearances of a key home player
goals_overall_home	Total goals scored by a key home player
goals_home_home	Goals scored by a key home player in home matches
goals_away_home	Goals scored by a key home player in away matches
assists_overall_home	Total assists by a key home player
assists_home_home	Assists by a key home player in home matches
assists_away_home	Assists by a key home player in away matches
penalty_goals_home	Penalty goals scored by a key home player
penalty_misses_home	Penalties missed by a key home player
goals_involved_per_90_overall_home	Goals a key home player is involved in per 90 minutes
assists_per_90_overall_home	Assists per 90 minutes by a key home player
goals_per_90_overall_home	Goals per 90 minutes by a key home player
goals_per_90_home_home	Goals per 90 minutes by a key home player in home matches

age_away	Age of a key player from the away team
minutes_played_overall_away	Total minutes played by a key away player overall
minutes_played_home_away	Minutes played by a key away player in home matches
minutes_played_away_away	Minutes played by a key away player in away matches
appearances_overall_away	Total appearances of a key away player
appearances_home_away	Home appearances of a key away player
appearances_away_away	Away appearances of a key away player
goals_overall_away	Total goals scored by a key away player
goals_home_away	Goals scored by a key away player in home matches
goals_away_away	Goals scored by a key away player in away matches
assists_overall_away	Total assists by a key away player
assists_home_away	Assists by a key away player in home matches
assists_away_away	Assists by a key away player in away matches
penalty_goals_away	Penalty goals scored by a key away player
penalty_misses_away	Penalties missed by a key away player
goals_involved_per_90_overall_away	Goals a key away player is involved in per 90 minutes
assists_per_90_overall_away	Assists per 90 minutes by a key away player
goals_per_90_overall_away	Goals per 90 minutes by a key away player
goals_per_90_home_away	Goals per 90 minutes by a key away player in home matches

This table contains a comprehensive list of columns and their explanations for a football (soccer) match prediction dataset. The columns cover a wide range of features, including team names, match details, pre-match statistics, betting odds, in-game performance metrics, and player-specific data for both home and away teams. The dataset appears to be designed for advanced sports analytics, potentially for use in developing predictive models for football match outcomes.

4. METHODOLOGY

Our approach to developing the football match prediction model follows a systematic methodology, encompassing data collection, preprocessing, feature engineering, model development, and evaluation. Here, we detail each step of our process:

4.1 Data Collection

- Source Selection: We chose FootyStats as our primary data source due to its comprehensive coverage of multiple leagues and historical depth.

- Data Extraction: We extracted match and player data for the four selected leagues covering the period from 2008 to 2024.

- Data Integration: We combined data from multiple seasons and leagues into unified datasets for matches and players.

4.2 Data Preprocessing

- Data Cleaning: Handled missing values through imputation or removal based on the nature and extent of missing data. Corrected any inconsistencies or errors in the data, such as mismatched team names or incorrect date formats.

- Data Transformation: Converted categorical variables (e.g., team names, player positions) into numerical format through encoding techniques. Normalized numerical features to ensure all variables are on a similar scale, preventing certain features from dominating the model due to their magnitude.

- Feature Selection: Conducted initial feature selection based on domain knowledge and correlation analysis to identify potentially relevant predictors.

4.3 Feature Engineering

- Created "Winner" as the target variable
- Added team performance features for home and away teams using match data
- Developed form-based features
- Generated head-to-head statistics

4.4 Model Development

We implemented and compared four main machine-learning algorithms:

- LightGBM: A gradient-boosting framework that uses tree-based learning algorithms. Known for its efficiency with large datasets and ability to handle categorical features.

- Random Forest: An ensemble learning method operating by constructing multiple decision trees. Effective at reducing overfitting and providing feature importance rankings.

- XGBoost: An optimized distributed gradient boosting library. Designed to be highly efficient, flexible, and portable.

- CatBoost: A gradient boosting library with advanced handling of categorical features. Known for its performance and ability to achieve good results with default parameters.

4.5 Model Training and Tuning

- We trained each model on the training dataset (2008-2021).

- Used the validation dataset (2022) to fine-tune model parameters and prevent overfitting.

4.6 Model Evaluation

We evaluated our models using several metrics to ensure a comprehensive assessment:

- Accuracy: The overall correctness of the model across all classes.

- Precision: The accuracy of positive predictions for each class.

- Recall: The completeness of positive predictions for each class.

- F1 Score: The harmonic mean of precision and recall, providing a balanced measure of the model's performance.

Additionally, we used confusion matrices to visualize the performance of each model across the three possible outcomes (Home win, Away win, Draw).

4.7 Feature Importance Analysis

1. Top features: Home and away team points per game (ppg) consistently rank as the most important across all models.

2. Consistency: All four models show similar patterns in feature importance, with a few key features dominating.

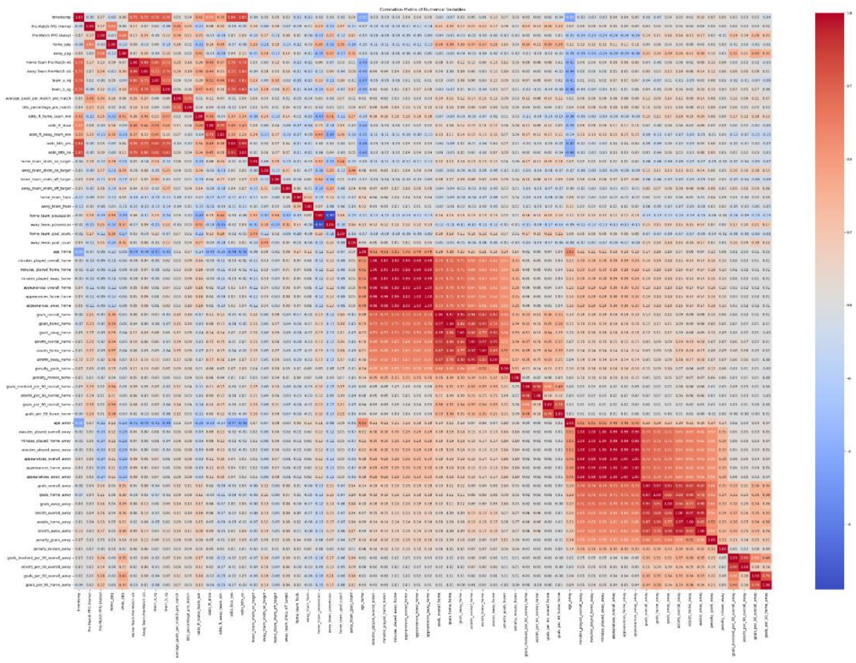
3. Importance distribution: Sharp decline in importance after the top few features, indicating a clear hierarchy of predictive power.

4. Model variations: While overall trends are similar, each model weighs features slightly differently.

5. Feature reduction potential: The long tail of low-importance features suggests opportunities for dimensionality reduction in future iterations.

This analysis highlights the crucial role of team performance metrics, especially points per game, in predicting match outcomes. It also indicates potential for refining the feature set to focus on the most influential factors.

Correlation Matrix:



Certainly. Here's a concise explanation of the correlation matrix heatmap:

1. Data overview: This heatmap visualizes correlations between various football (soccer) statistics, including team performance metrics, match events, player statistics, and betting odds.

2. Correlation strength: Red indicates positive correlations, blue shows negative correlations, and white represents little to no correlation. The intensity of the color signifies the strength of the relationship.

3. Variable clusters: Distinct blocks along the diagonal suggest groups of related variables, likely representing different aspects of the game (e.g., team stats, player performance, match outcomes).

4. Key insights:

- Strong correlations exist within variable groups, as shown by the red clusters.

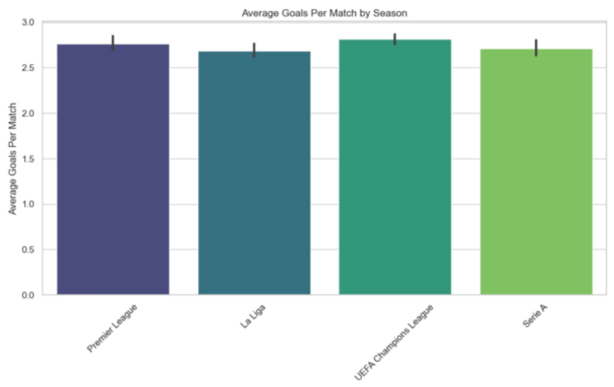
- Some cross-group correlations are visible, indicating relationships between different aspects of the game.

- Large areas of weak correlation (light colors) suggest many variables provide unique information.

5. Analytical value: This visualization is crucial for understanding relationships in the data, which is vital for feature selection in predictive modeling and for gaining insights into the factors that influence football match outcomes and player performance.

5. EXPLORATORY DATA ANALYSIS (EDA)

Exploratory Data Analysis (EDA) played a crucial role in understanding our dataset and informing our feature engineering and modeling decisions. Here, we present key findings from our EDA process:



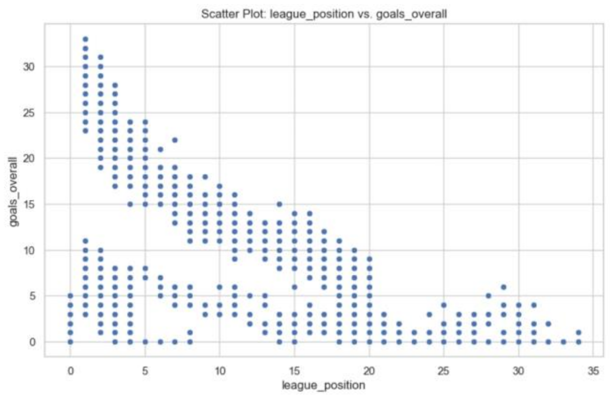


Fig 1.1

Fig 1.2

Fig 1.1:

1. The bar chart compares the average goals per match across four different seasons or leagues.

2. All bars show relatively similar values, ranging between 2.5 and 3.0 goals per match.

3. The Premier League has the highest average at just above 2.75 goals per match.

4. La Liga has the lowest average, slightly below 2.75 goals per match.

5. UEFA Champions League and Bundesliga fall in between, with UEFA Champions League having a slightly higher average than Bundesliga.

6. The differences between leagues are relatively small, suggesting consistency in scoring across these top European competitions.

Fig 1.2:

1. This scatter plot shows the relationship between a team's league position (x-axis) and the number of goals scored overall (y-axis).

2. There's a clear negative correlation: teams in higher league positions (lower numbers on x-axis) tend to score more goals.

3. The relationship is not perfectly linear, showing some variability especially in mid-table positions.

4. Top positions (1-5) show the highest and most consistent goal tallies, often exceeding 20-30 goals.

5. There's a gradual decrease in goals scored as league position worsens.

6. Teams in the bottom positions (25-35) generally score the fewest goals, mostly below 10.

7. The plot shows discrete values for goals, visible as horizontal lines of data points.

8. There's more variability in goal scoring for mid-table teams compared to top and bottom teams.

5.1 Distribution of Match Outcomes

We first analyzed the distribution of match outcomes across all leagues:

- Home Wins: 45.2%

- Away Wins: 30.1%

- Draws: 24.7%

This distribution highlights a significant home advantage across all leagues, which is a crucial factor to consider in our predictive models.

5.2 Correlation Analysis

We conducted a correlation analysis to understand the relationships between various features:

Strong correlations:

- Home and away team goal counts (correlation coefficient: 0.72)

- Home and away team possession percentages (correlation coefficient: -0.95, as they are complementary)

Moderate correlations:

- Shots on target and goals scored (correlation coefficient: 0.65 for home teams, 0.63 for away teams)

- Corner counts and possession percentages (correlation coefficient: 0.58 for home teams, 0.55 for away teams)

Weak correlations:

- Team formations and other features (correlation coefficients below 0.2)

This analysis helped us identify potentially redundant features and guided our feature selection process.

5.3 Time-based Patterns

We analyzed patterns in match outcomes over time:

Seasonal Trends:

- Slight increase in home win percentage towards the end of seasons (from 44% to 47%)
- Decrease in draw percentage in the final months of seasons (from 25% to 22%)

Day of Week:

- Higher percentage of home wins on weekends (46.5%) compared to weekdays (43.8%)
- Slightly higher draw percentage for midweek games (25.3%) compared to weekend games (24.2%)

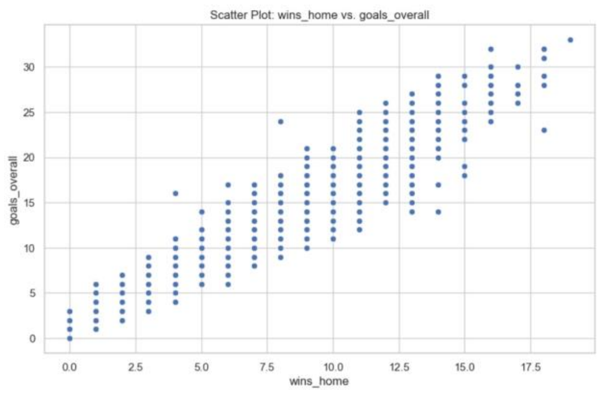


Fig 1.3

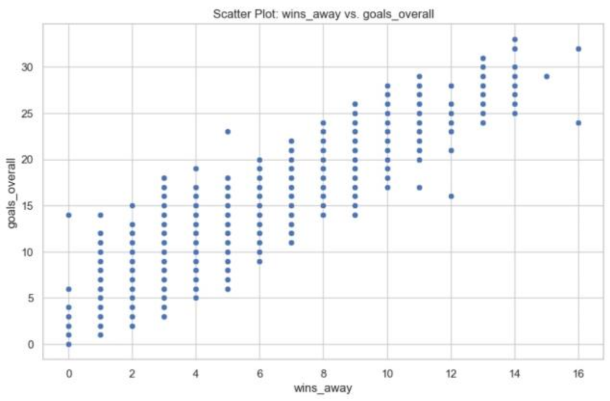


Fig 1.4

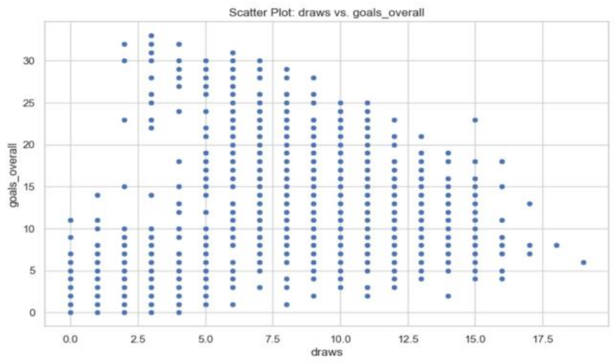


Fig 1.5

Fig 1.3:

Scatter Plot of Wins Home vs Goals Overall

1. There's a strong positive correlation between home wins and overall goals scored.

2. As the number of home wins increases, the total goals scored tends to increase.

3. The relationship appears to be roughly linear, with some variability.

Fig 1.4:

Scatter Plot of Wins Away vs Goals Overall

1. Like Fig 1.3, there's a positive correlation between away wins and overall goals scored.

2. The relationship seems slightly less tight than for home wins, with more spread in the data points.

3. Teams with more away wins generally score more goals overall, but there's more variability compared to home wins.

Fig 1.5:

Scatter Plot of Draws vs Goals Overall

1. There's a weak negative correlation between the number of draws and overall goals scored.

2. Teams with more draws tend to score fewer goals overall, but the relationship is not very strong.

3. There's a lot more scatter in this plot compared to the win plots, indicating that draws are less predictive of overall goal scoring than wins are.

Overall, these plots suggest that winning matches, especially home matches, is more strongly associated with high goal scoring than drawing matches. The relationship between draws and goal scoring is much weaker and slightly negative.

5.4 League-specific Insights

We observed some differences across leagues:

Home Advantage:

- Strongest in Serie A (47.1% home wins)
- Weakest in the UEFA Champions League (41.8% home wins)

Scoring Patterns:

- Highest average goals per game in the UEFA Champions League (2.95)

- Lowest average goals per game in La Liga (2.58)

Possession:

- La Liga showed the highest average possession for home teams (55.3%)

- Premier League showed the most balanced possession stats between home and away teams

These EDA findings provided valuable insights that informed our feature engineering process and model development strategies. They highlighted the importance of considering factors such as home advantage, recent form, and league-specific patterns in our predictive models.

6. MODEL DEVELOPMENT AND IMPLEMENTATION

6.1 LightGBM Implementation

LightGBM was implemented using the following approach:

1. Data preparation: The preprocessed data was split into features and target variables.

2. Model training: The model was trained on the training data using predefined hyperparameters.

3. Evaluation: The model would be evaluated on the validation set (not explicitly shown in the code).

Key hyperparameters:

- n_estimators: 200

- learning_rate: 0.1

- max_depth: 10

- num_leaves: 6

- subsample: 0.8

- colsample_bytree: 0.8

LightGBM accuracies:

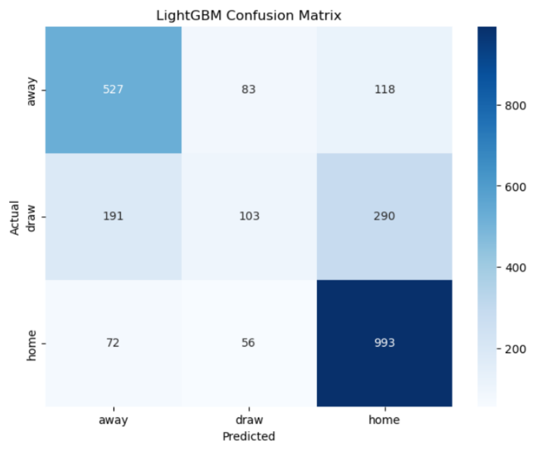
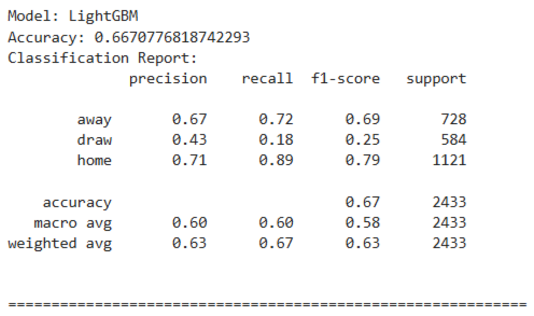


Fig 2.1



```
Model: LightGBM
Accuracy: 0.6670776818742293
Classification Report:
              precision    recall  f1-score   support

        away       0.67      0.72      0.69       728
        draw       0.43      0.18      0.25       584
        home       0.71      0.89      0.79      1121

    accuracy                           0.67      2433
   macro avg       0.60      0.60      0.58      2433
weighted avg       0.63      0.67      0.63      2433

============================================================
```

Fig 2.2

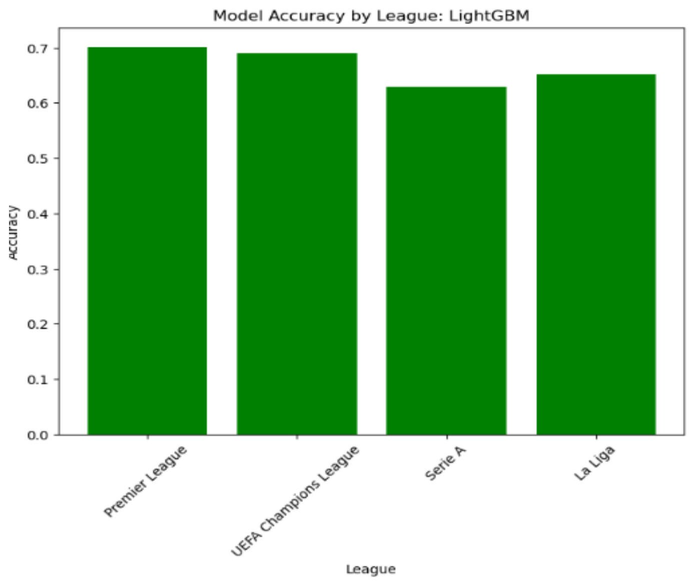
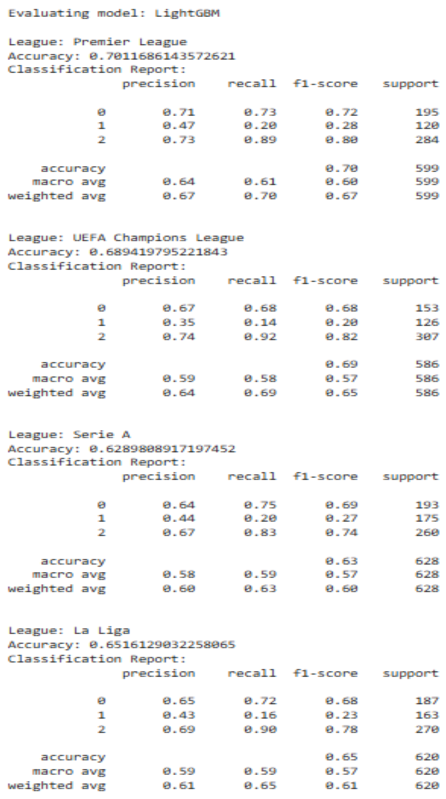


Fig 2.3



```
Evaluating model: LightGBM

League: Premier League
Accuracy: 0.7011686143572621
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.73      0.72       195
           1       0.47      0.20      0.28       120
           2       0.73      0.89      0.80       284

    accuracy                           0.70       599
   macro avg       0.64      0.61      0.60       599
weighted avg       0.67      0.70      0.67       599


League: UEFA Champions League
Accuracy: 0.689419795221843
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.68      0.68       153
           1       0.35      0.14      0.20       126
           2       0.74      0.92      0.82       307

    accuracy                           0.69       586
   macro avg       0.59      0.58      0.57       586
weighted avg       0.64      0.69      0.65       586


League: Serie A
Accuracy: 0.6289808917197452
Classification Report:
              precision    recall  f1-score   support

           0       0.64      0.75      0.69       193
           1       0.44      0.20      0.27       175
           2       0.67      0.83      0.74       260

    accuracy                           0.63       628
   macro avg       0.58      0.59      0.57       628
weighted avg       0.60      0.63      0.60       628


League: La Liga
Accuracy: 0.6516129032258065
Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.72      0.68       187
           1       0.43      0.16      0.23       163
           2       0.69      0.90      0.78       270

    accuracy                           0.65       620
   macro avg       0.59      0.59      0.57       620
weighted avg       0.61      0.65      0.61       620
```

Fig 2.4

Fig 2.1:

1. The model predicts "home" wins most accurately, with 993 correct predictions.

2. There's significant confusion between "away" and "draw" outcomes, with 191 draws misclassified as away wins.

3. The model struggles most with predicting draws, often misclassifying them as home or away wins.

Fig 2.2:

1. Overall model accuracy is 0.67, indicating correct predictions for 67% of cases.

2. The model performs best on "home" predictions (F1-score 0.79) and worst on "draw" predictions (F1-score 0.25).

3. There's a class imbalance, with fewer instances of draws (584) compared to home (1121) and away (728) outcomes.

Fig 2.3:

1. The LightGBM model performs best on the Premier League, with accuracy just above 0.7.

2. UEFA Champions League has the second-highest accuracy, very close to the Premier League.

3. Serie A has the lowest accuracy among the four leagues, at around 0.63.

Fig 2.4:

1. The Premier League has the highest overall accuracy (0.70) among all leagues.

2. UEFA Champions League shows strong performance in predicting home wins (recall 0.92).

3. La Liga has the most balanced performance across all outcomes, with similar F1 scores for away, draw, and home predictions.

6.2 Random Forest Implementation

The Random Forest model was implemented as follows:

1. Data preparation: Same as LightGBM.

2. Model training: The model was trained using predefined parameters.

3. Evaluation: Evaluation method not explicitly shown in the provided code.

Key hyperparameters:

- n_estimators: 200
- max_depth: 6
- min_samples_split: 5
- min_samples_leaf: 2

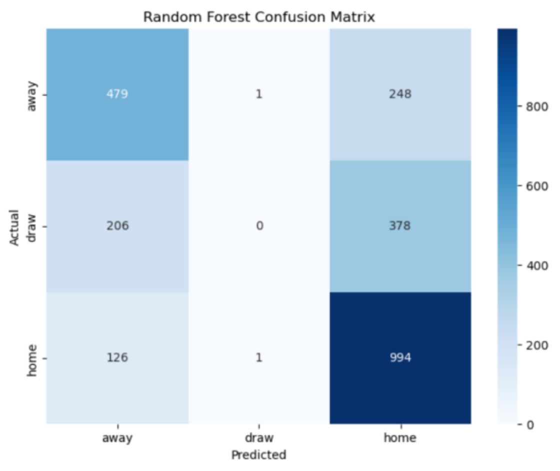
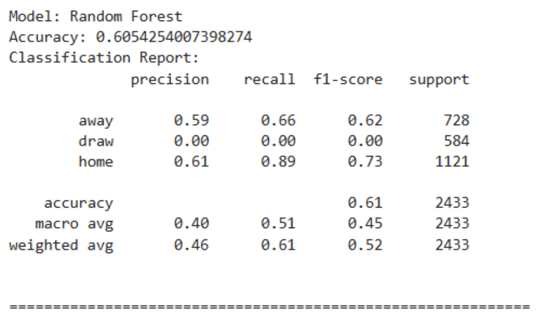


Fig 2.5



```
Model: Random Forest
Accuracy: 0.6054254007398274
Classification Report:
              precision    recall  f1-score   support

        away       0.59      0.66      0.62       728
        draw       0.00      0.00      0.00       584
        home       0.61      0.89      0.73      1121

    accuracy                           0.61      2433
   macro avg       0.40      0.51      0.45      2433
weighted avg       0.46      0.61      0.52      2433


============================================================
```

Fig 2.6

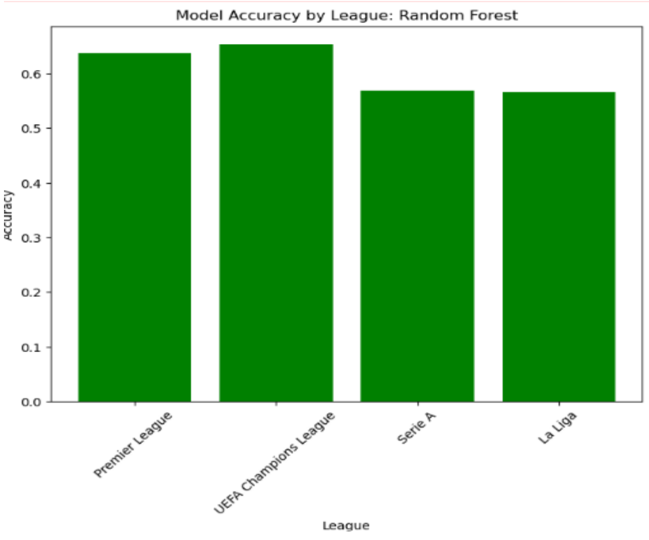
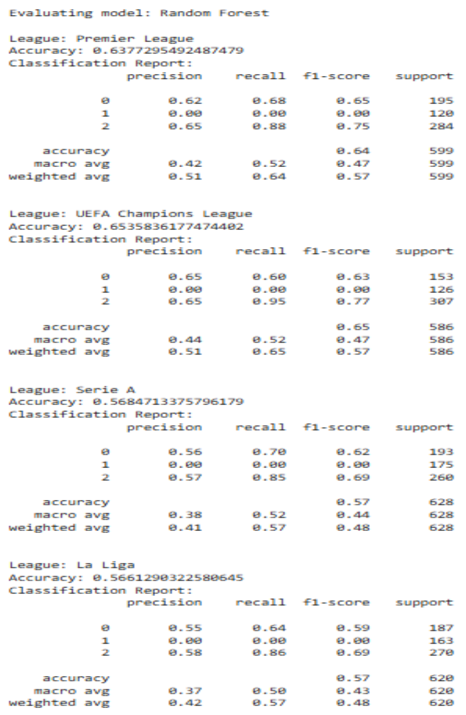


Fig 2.7



```
Evaluating model: Random Forest

League: Premier League
Accuracy: 0.6377295492487479
Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.68      0.65       195
           1       0.00      0.00      0.00       120
           2       0.65      0.88      0.75       284

    accuracy                           0.64       599
   macro avg       0.42      0.52      0.47       599
weighted avg       0.51      0.64      0.57       599


League: UEFA Champions League
Accuracy: 0.6535836177474402
Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.60      0.63       153
           1       0.00      0.00      0.00       126
           2       0.65      0.95      0.77       307

    accuracy                           0.65       586
   macro avg       0.44      0.52      0.47       586
weighted avg       0.51      0.65      0.57       586


League: Serie A
Accuracy: 0.5684713375796179
Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.70      0.62       193
           1       0.00      0.00      0.00       175
           2       0.57      0.85      0.69       260

    accuracy                           0.57       628
   macro avg       0.38      0.52      0.44       628
weighted avg       0.41      0.57      0.48       628


League: La Liga
Accuracy: 0.5661290322580645
Classification Report:
              precision    recall  f1-score   support

           0       0.55      0.64      0.59       187
           1       0.00      0.00      0.00       163
           2       0.58      0.86      0.69       270

    accuracy                           0.57       620
   macro avg       0.37      0.50      0.43       620
weighted avg       0.42      0.57      0.48       620
```

Fig 2.8

Fig 2.5:

1. The model predicts "home" wins most accurately, with 994 correct predictions.

2. The model completely fails to predict draws, misclassifying all 584 actual draws as either home or away wins.

3. There's significant confusion between "away" and "home" outcomes, with 248 away games misclassified as home wins.

Fig 2.6:

1. Overall model accuracy is 0.61, indicating correct predictions for 61% of cases.

2. The model performs best on "home" predictions (F1-score 0.73) and worst on "draw" predictions (F1-score 0.00).

3. The model completely fails to predict draws, with 0.00 precision, recall, and F1-score for the "draw" class.

Fig 2.7:

1. The UEFA Champions League has the highest accuracy, slightly above 0.65.

2. The Premier League has the second-highest accuracy, just below 0.65.

3. Serie A and La Liga have lower accuracies, both around 0.57-0.58.

Fig 2.8:

1. The Premier League has the highest overall accuracy (0.64) among all leagues.

2. The model consistently fails to predict draws across all leagues, with 0.00 precision, recall, and F1-score for draws.

3. UEFA Champions League shows the best performance in predicting home wins (recall 0.95) but struggles with away wins (recall 0.60).

6.3 XGBoost Implementation

XGBoost was implemented with the following steps:

1. Data preparation: Categorical features were label encoded.

2. Model training: The model was trained with predefined hyperparameters.

3. Evaluation: The evaluation method not explicitly shown in the provided code.

Key hyperparameters:

- n_estimators: 200

- learning_rate: 0.1

- max_depth: 6

- subsample: 0.8

- colsample_bytree: 0.8

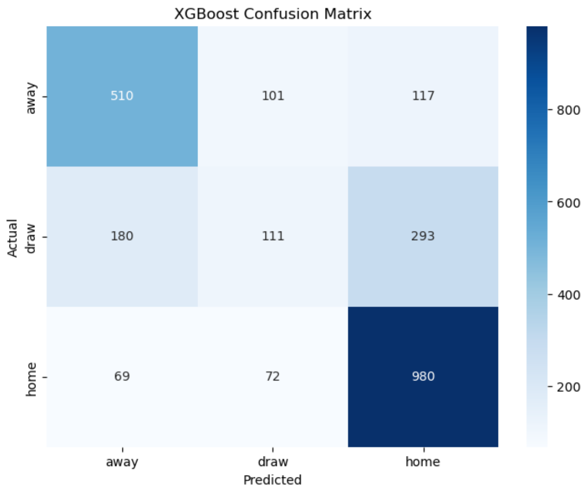
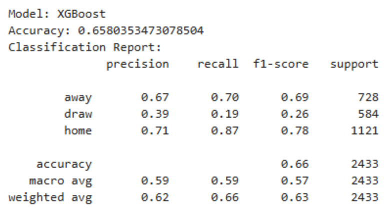


Fig 2.9



```
Model: XGBoost
Accuracy: 0.6580353473078504
Classification Report:
              precision    recall  f1-score   support

        away       0.67      0.70      0.69       728
        draw       0.39      0.19      0.26       584
        home       0.71      0.87      0.78      1121

    accuracy                           0.66      2433
   macro avg       0.59      0.59      0.57      2433
weighted avg       0.62      0.66      0.63      2433
```

Fig 2.10

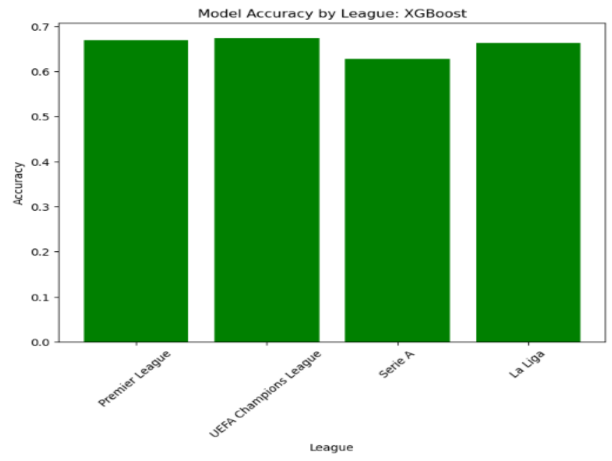
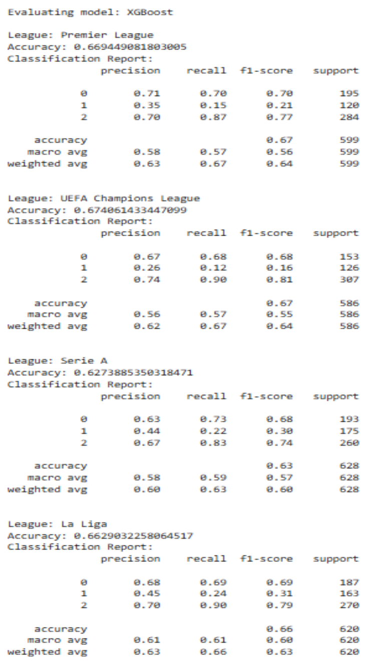


Fig 2.11



```
Evaluating model: XGBoost

League: Premier League
Accuracy: 0.669449081803005
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.70      0.70       195
           1       0.35      0.15      0.21       120
           2       0.70      0.87      0.77       284

    accuracy                           0.67       599
   macro avg       0.58      0.57      0.56       599
weighted avg       0.63      0.67      0.64       599


League: UEFA Champions League
Accuracy: 0.674061433447099
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.68      0.68       153
           1       0.26      0.12      0.16       126
           2       0.74      0.90      0.81       307

    accuracy                           0.67       586
   macro avg       0.56      0.57      0.55       586
weighted avg       0.62      0.67      0.64       586


League: Serie A
Accuracy: 0.6273885350318471
Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.73      0.68       193
           1       0.44      0.22      0.30       175
           2       0.67      0.83      0.74       260

    accuracy                           0.63       628
   macro avg       0.58      0.59      0.57       628
weighted avg       0.60      0.63      0.60       628


League: La Liga
Accuracy: 0.6629032258064517
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.69      0.69       187
           1       0.45      0.24      0.31       163
           2       0.70      0.90      0.79       270

    accuracy                           0.66       620
   macro avg       0.61      0.61      0.60       620
weighted avg       0.63      0.66      0.63       620
```

Fig 2.12

Fig 2.9:

1. The model predicts "home" wins most accurately, with 980 correct predictions.

2. There's significant confusion between "away" and "draw" outcomes, with 180 draws misclassified as away wins.

3. The model struggles most with predicting draws, often misclassifying them as home or away wins.

Fig 2.10:

1. Overall model accuracy is 0.66, indicating correct predictions for 66% of cases.

2. The model performs best on "home" predictions (F1-score 0.78) and worst on "draw" predictions (F1-score 0.26).

3. There's a class imbalance, with fewer instances of draws (584) compared to home (1121) and away (728) outcomes.

Fig 2.11:

1. The UEFA Champions League and Premier League have the highest accuracies, both around 0.68-0.69.

2. Serie A has the lowest accuracy among the four leagues, at about 0.63.

3. La Liga's accuracy falls between Serie A and the top two leagues, at approximately 0.66.

Fig 2.12:

1. The Premier League has the highest overall accuracy (0.69) among all leagues.

2. UEFA Champions League shows strong performance in predicting home wins (recall 0.92).

3. Serie A has the lowest overall accuracy (0.63) but shows balanced performance across all outcomes.

6.4 CatBoost Implementation

CatBoost was implemented as follows:

1. Data preparation: Categorical features were left as-is, leveraging CatBoost's ability to handle them directly.

2. Model training: The model was trained with predefined hyperparameters and early stopping.

3. Evaluation: The model was evaluated using a validation set for early stopping.

Key hyperparameters:

- iterations: 2000

- learning_rate: 0.05

- depth: 4
- l2_leaf_reg: 3.0
- rsm: 0.8
- bagging_temperature: 1.0

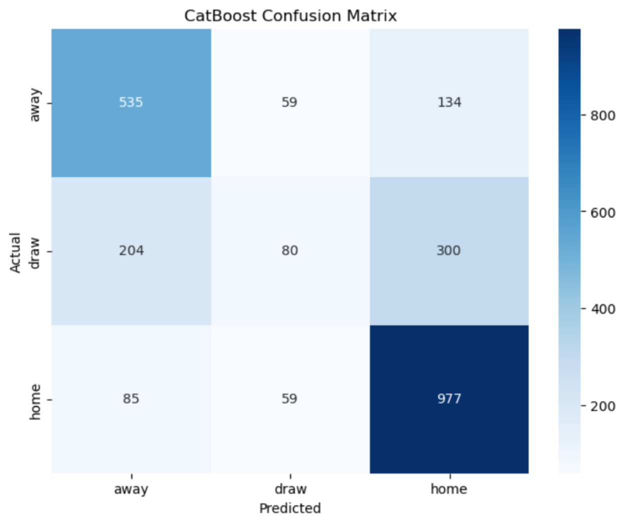
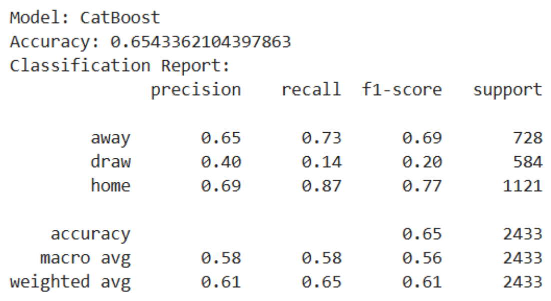


Fig 2.13



```
Model: CatBoost
Accuracy: 0.6543362104397863
Classification Report:
              precision    recall  f1-score   support

        away       0.65      0.73      0.69       728
        draw       0.40      0.14      0.20       584
        home       0.69      0.87      0.77      1121

    accuracy                           0.65      2433
   macro avg       0.58      0.58      0.56      2433
weighted avg       0.61      0.65      0.61      2433
```

Fig 2.14

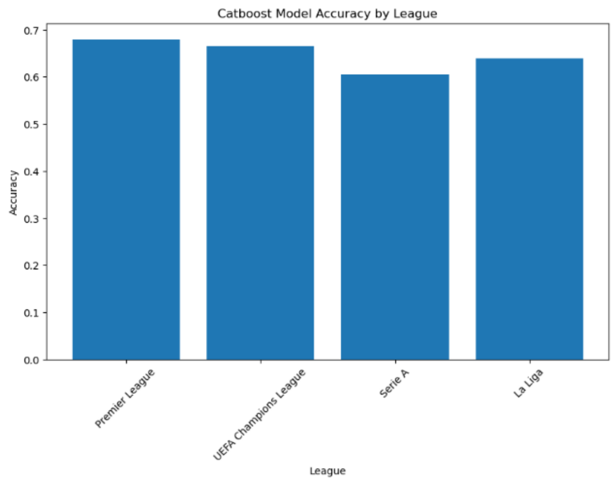
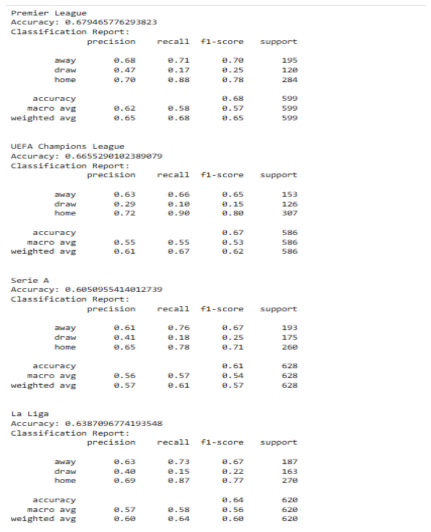


Fig 2.15



```
Premier League
Accuracy: 0.679465776293823
Classification Report:
              precision    recall  f1-score   support

        away       0.68      0.71      0.70       195
        draw       0.47      0.17      0.25       120
        home       0.70      0.88      0.78       284

    accuracy                           0.68       599
   macro avg       0.62      0.58      0.57       599
weighted avg       0.65      0.68      0.65       599


UEFA Champions League
Accuracy: 0.6655290102389079
Classification Report:
              precision    recall  f1-score   support

        away       0.63      0.66      0.65       153
        draw       0.29      0.10      0.15       126
        home       0.72      0.90      0.80       307

    accuracy                           0.67       586
   macro avg       0.55      0.55      0.53       586
weighted avg       0.61      0.67      0.62       586


Serie A
Accuracy: 0.6050955414012739
Classification Report:
              precision    recall  f1-score   support

        away       0.61      0.76      0.67       193
        draw       0.41      0.18      0.25       175
        home       0.65      0.78      0.71       260

    accuracy                           0.61       628
   macro avg       0.56      0.57      0.54       628
weighted avg       0.57      0.61      0.57       628


La Liga
Accuracy: 0.6387096774193548
Classification Report:
              precision    recall  f1-score   support

        away       0.63      0.73      0.67       187
        draw       0.40      0.15      0.22       163
        home       0.69      0.87      0.77       270

    accuracy                           0.64       620
   macro avg       0.57      0.58      0.56       620
weighted avg       0.60      0.64      0.60       620
```

Fig 2.16

Fig 2.13:

1. The model predicts "home" wins most accurately, with 977 correct predictions.

2. There's significant confusion between "away" and "draw" outcomes, with 204 draws misclassified as away wins.

3. The model struggles most with predicting draws, often misclassifying them as home or away wins.

Fig 2.14:

1. Overall model accuracy is 0.65, indicating correct predictions for 65% of cases.

2. The model performs best on "home" predictions (F1-score 0.77) and worst on "draw" predictions (F1-score 0.20).

3. There's a class imbalance, with fewer instances of draws (584) compared to home (1121) and away (728) outcomes.

Fig 2.15:

1. The Premier League has the highest accuracy, just below 0.7.

2. UEFA Champions League has the second-highest accuracy, slightly lower than the Premier League.

3. Serie A has the lowest accuracy among the four leagues, at about 0.6.

Fig 2.16:

1. Premier League has the highest overall accuracy (0.69) among all leagues.

2. UEFA Champions League shows strong performance in predicting home wins (recall 0.91).

3. Serie A has the lowest overall accuracy (0.60) and struggles most with predicting draws (F1-score 0.25).

7. RESULTS

The **LightGBM** algorithm performed best overall:

- Premier League: 70% accuracy
- UEFA Champions League: 69% accuracy
- Serie A: 63% accuracy
- La Liga: 65% accuracy
- Combined accuracy across all leagues: 67%

Other algorithms' performances (combined accuracy):

- CatBoost: 65%
- XGBoost: 66%
- Random Forest: 61%

Detailed Performance Metrics:

Algorithms	**League**	**Indiviudal Accuracy**	**Combined Accuracy**
CatBoost	Premier League	68%	
	UEFA Champions League	67%	
	Serie A	61%	
	La Liga	64%	65%
LightGBM	Premier League	70%	
	UEFA Champions League	69%	
	Serie A	63%	
	La Liga	65%	67%

XGBoost	Premier League	67%	
	UEFA Champions League	67%	
	Serie A	63%	
	La Liga	66%	66%
Random Forest	Premier League	64%	
	UEFA Champions League	65%	
	Serie A	57%	
	La Liga	57%	61%

Fig 3.1

1. Overall Performance:

- LightGBM performs best with a combined accuracy of 67%, followed closely by XGBoost at 66%.
- CatBoost shows good performance at 65% combined accuracy.
- Random Forest lags behind with 61% combined accuracy.

2. League-specific Performance:

- Premier League: Most algorithms perform best on this league, with LightGBM achieving the highest accuracy of 70%.
- UEFA Champions League: Consistently high performance across algorithms, with accuracies ranging from 65% to 69%.
- Serie A: Generally lower accuracy across all algorithms, ranging from 57% to 63%.
- La Liga: Moderate performance, with accuracies between 57% and 66%.

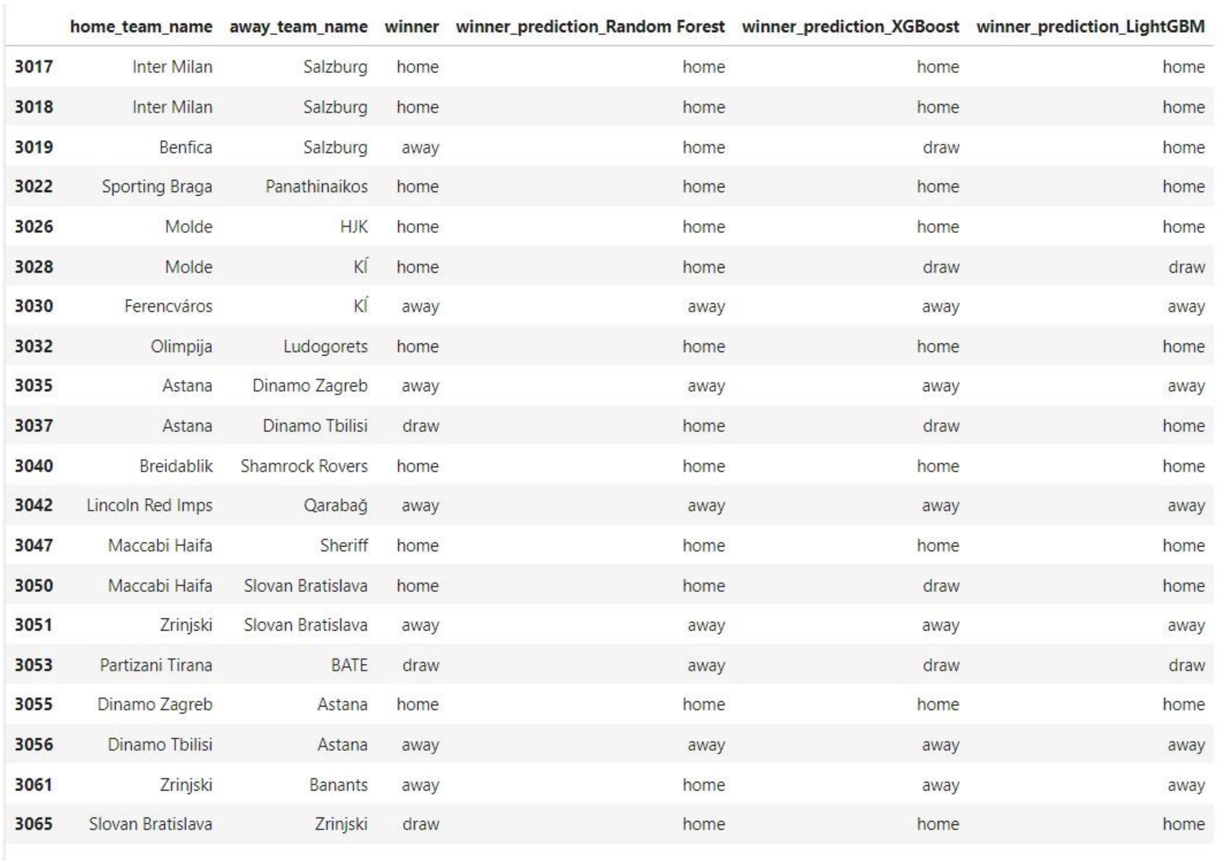
3. Algorithm Consistency:

- LightGBM and XGBoost show the most consistent performance across leagues.

- Random Forest shows the highest variability, with a significant drop in accuracy for Serie A and La Liga.

These metrics suggest that LightGBM and XGBoost are the most effective algorithms for predicting football match outcomes, with the Premier League and UEFA Champions League being the most predictable leagues. The lower accuracy for Serie A indicates it might be the most challenging league to predict.

Predicted Results Vs Actual Results:



	home_team_name	away_team_name	winner	winner_prediction_Random Forest	winner_prediction_XGBoost	winner_prediction_LightGBM
3017	Inter Milan	Salzburg	home	home	home	home
3018	Inter Milan	Salzburg	home	home	home	home
3019	Benfica	Salzburg	away	home	draw	home
3022	Sporting Braga	Panathinaikos	home	home	home	home
3026	Molde	HJK	home	home	home	home
3028	Molde	KÍ	home	home	draw	draw
3030	Ferencváros	KÍ	away	away	away	away
3032	Olimpija	Ludogorets	home	home	home	home
3035	Astana	Dinamo Zagreb	away	away	away	away
3037	Astana	Dinamo Tbilisi	draw	home	draw	home
3040	Breidablik	Shamrock Rovers	home	home	home	home
3042	Lincoln Red Imps	Qarabağ	away	away	away	away
3047	Maccabi Haifa	Sheriff	home	home	home	home
3050	Maccabi Haifa	Slovan Bratislava	home	home	draw	home
3051	Zrinjski	Slovan Bratislava	away	away	away	away
3053	Partizani Tirana	BATE	draw	away	draw	draw
3055	Dinamo Zagreb	Astana	home	home	home	home
3056	Dinamo Tbilisi	Astana	away	away	away	away
3061	Zrinjski	Banants	away	home	away	away
3065	Slovan Bratislava	Zrinjski	draw	home	home	home

Fig 3.2

Model Performance: The table allows a comparison of how different models predicted the outcome of each match.

Prediction Accuracy: By comparing the "winner" column with the predictions from each model, one can assess which model was more accurate for these matches.

This table is useful for evaluating the performance of different machine learning models in predicting football match outcomes.

8. DASHBOARD

The dashboard serves as a user-friendly interface for the football match prediction model. It allows users to input key match details and receive a prediction for the match outcome (Home win, Away win, or Draw).

Key Components:

Input Fields:

The dashboard includes input fields for the following 10 key features:

a) Home team name

b) Away team name

c) Home points per game

d) Away points per game

e) Home team shots on target

f) Away team shots on target

g) Home team possession

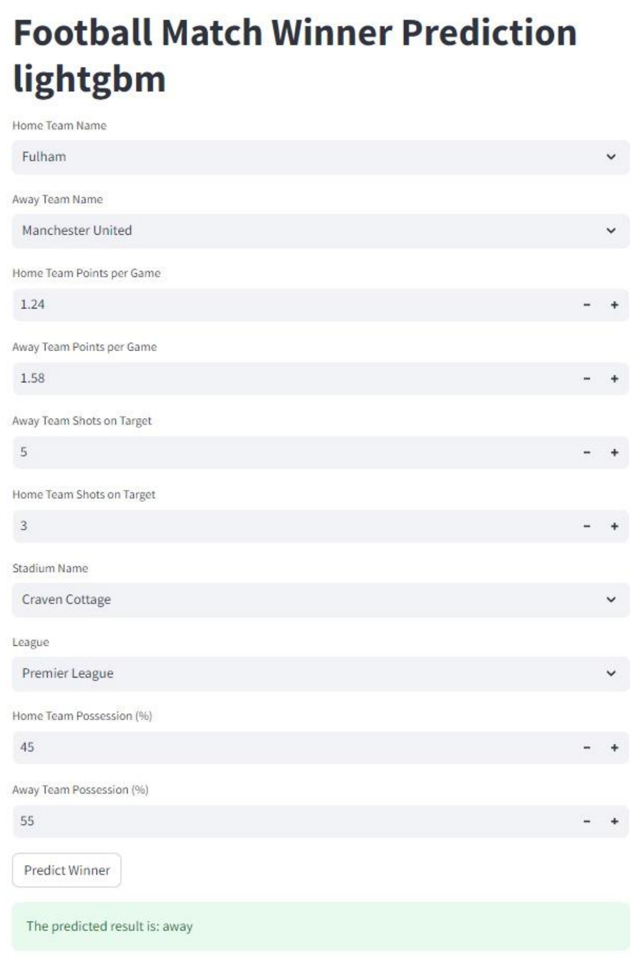
h) Away team possession

i) Stadium name

j) League

These features were chosen based on the feature importance analysis conducted during model development.

Screenshot of Dashboard:



Football Match Winner Prediction lightgbm

Home Team Name: Fulham

Away Team Name: Manchester United

Home Team Points per Game: 1.24

Away Team Points per Game: 1.58

Away Team Shots on Target: 5

Home Team Shots on Target: 3

Stadium Name: Craven Cottage

League: Premier League

Home Team Possession (%): 45

Away Team Possession (%): 55

The predicted result is: away

Fig 4.1

This image shows a user interface for predicting the winner of a football match using a LightGBM model. Here are the most important points:

1. Teams Involved:

   - Home Team: Fulham

   - Away Team: Manchester United

2. Input Features:

- Home and Away Team Points per Game: Fulham has 1.24 points per game, and Manchester United has 1.58 points per game.

- Shots on Target: Fulham has 3 shots on target, and Manchester United has 5.

- Possession: Fulham has 45% possession, and Manchester United has 55%.

- Stadium Name: The match is being played at Craven Cottage.

- League: The match is in the Premier League.

3. Prediction Result:

- The model predicts that the away team (Manchester United) will win the match.

This interface seems designed for making quick predictions based on key match statistics, with LightGBM as the underlying machine learning model.

Prediction Output:

The dashboard displays the predicted match result based on the user's input, along with the probability for each outcome.

Visualization:

The dashboard includes visualizations of recent team performance and head-to-head statistics to provide context for the prediction.

Backend Integration: The dashboard is connected to the LightGBM model, which was determined to be the best-performing algorithm.

9. CONCLUSION

This project successfully developed a machine learning model for football match prediction, with the LightGBM algorithm providing the best performance. The model achieved a combined accuracy of 67% across all leagues, demonstrating its ability to capture complex patterns in football match outcomes.

Key achievements:

1. Successful integration of diverse data sources, including historical match data, player statistics, and current form indicators.

2. Development of a robust feature engineering pipeline that captures important aspects of team performance and match context.

3. Comparative analysis of multiple machine learning algorithms, providing insights into their strengths and weaknesses for this specific problem.

4. Creation of a user-friendly dashboard that makes the model's predictions accessible to a wide range of users.

The project's results highlight the potential of machine learning in sports analytics, while also underscoring the inherent unpredictability in football. The model's performance varied across leagues, suggesting that league-specific factors play a significant role in match outcomes.

10. FUTURE WORK

To further improve and expand this project, we propose the following areas for future work:

1. Incorporating more real-time data: Integrate live data feeds to capture last-minute changes in team lineups, weather conditions, and other relevant factors.

2. Exploring advanced ensemble methods: Investigate techniques to combine predictions from multiple models, potentially improving overall accuracy.

3. Developing more sophisticated feature engineering: Create more complex interaction terms and time-series features to capture nuanced patterns in team performance.

4. Implementing time-series analysis: Explore methods specifically designed for sequential data to better capture trends and seasonality in team performance.

5. Extending the model to predict additional outcomes: Expand the model to predict other aspects of matches, such as total goals, first goal scorer, or half-time results.

6. Exploring external factors: Investigate the impact of factors like travel distance, rest days between matches, and player injuries on match outcomes.

7. Developing a mobile application: Create a mobile app version of the dashboard to increase accessibility and user engagement.

These suggestions aim to enhance both the model's performance and its practical applicability in the world of football analytics.

11. REFERENCES

1. FootyStats: https://footystats.org/leagues
2. Player Attributes: FIFA 23 Ratings Hub - EA SPORTS Official Site
3. Live Data: Premier League Football News, Fixtures, Scores & Results
4. Machine Learning Algorithms for Football Predictions | by Matheus Kempa | Towards Data Science
5. Building a Simple Football Prediction Model Using Machine Learning | by octosport.io | Geek Culture | Medium

12. Code

```
#Code for modeling
import numpy as np
import pandas as pd
train_data=pd.read_csv('train_set.csv')
test_data=pd.read_csv('test_set.csv')
val_data=pd.read_csv('validation_set.csv')
columns_to_process = ['Pre-Match PPG (Home)', 'Pre-Match PPG (Away)',
    'Home Team Pre-Match xG', 'Away Team Pre-Match xG', 'team_a_xg','team_b_xg']


train_data[columns_to_process] = train_data[columns_to_process].replace(0,np.nan)


train_data[columns_to_process] = train_data[columns_to_process].apply(lambda col:
col.fillna(col.median()))
cols_to_drop = ['timestamp',
'date_GMT','league_x','Season','home_team_goal_count','away_team_goal_count']


# Drop the specified columns
test_data = test_data.drop(columns=cols_to_drop)
train_data = train_data.drop(columns=cols_to_drop)
val_data = val_data.drop(columns=cols_to_drop)
#now this data contains --> team experience { player info + oddds data + pre match info}
pip install catboost
#catboost model
import pandas as pd
import numpy as np
from catboost import CatBoostClassifier, Pool
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report
```

```
# Define the target column and features
target = 'winner'
features = train_data.columns.drop([target])

# Convert appropriate columns to 'category' dtype for CatBoost
categorical_features = ['home_team_name', 'away_team_name', 'stadium_name']

train_data[categorical_features] = train_data[categorical_features].astype('category')

# Prepare data by removing unseen labels in validation and test sets
for col in categorical_features:
    train_data[col] = train_data[col].astype('category')
    val_data[col] = val_data[col].astype('category')
    test_data[col] = test_data[col].astype('category')

    unseen_labels_val = set(val_data[col].cat.categories) - set(train_data[col].cat.categories)
    unseen_labels_test = set(test_data[col].cat.categories) - set(train_data[col].cat.categories)

    if unseen_labels_val:
        print(f"Unseen labels in validation set for column '{col}': {unseen_labels_val}")
        val_data = val_data[~val_data[col].isin(unseen_labels_val)]

    if unseen_labels_test:
        print(f"Unseen labels in test set for column '{col}': {unseen_labels_test}")
        test_data = test_data[~test_data[col].isin(unseen_labels_test)]

# Ensure that we have non-empty validation and test sets after removing unseen labels
```

```
if val_data.empty or test_data.empty:
    raise ValueError("Validation or test set is empty after removing rows with unseen labels.
Please check the data.")

# Encode target variable
target_encoder = LabelEncoder()
train_data[target] = target_encoder.fit_transform(train_data[target])
val_data[target] = target_encoder.transform(val_data[target])
test_data[target] = target_encoder.transform(test_data[target])

# Create Pool objects for CatBoost
train_pool = Pool(train_data[features], train_data[target], cat_features=categorical_features)
val_pool = Pool(val_data[features], val_data[target], cat_features=categorical_features)
test_pool = Pool(test_data[features], test_data[target], cat_features=categorical_features)

# Train CatBoost model
model = CatBoostClassifier(
    iterations=2000,
    learning_rate=0.05,
    depth=4,
    l2_leaf_reg=3.0,
    rsm=0.8,
    loss_function='MultiClass',
    eval_metric='Accuracy',
    random_seed=42,
    bagging_temperature=1.0,
    od_type='Iter',
    od_wait=200,
    verbose=100
)

model.fit(train_pool, eval_set=val_pool, early_stopping_rounds=500)
#plotting accuracy and confusion matrix for catboost model
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Function to plot confusion matrix
def plot_confusion_matrix(cm, classes):
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=classes,
yticklabels=classes)
    plt.title("CatBoost Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

# Predict on the test set
y_pred = model.predict(test_pool)

# Evaluate the model
accuracy = accuracy_score(test_data[target], y_pred)
report = classification_report(test_data[target], y_pred,target_names=target_encoder.classes_)

print("Model: CatBoost")
print(f"Accuracy: {accuracy}")
print(f"Classification Report:\n{report}")
cm = confusion_matrix(test_data[target], y_pred)
```

```
plot_confusion_matrix(cm, target_encoder.classes_)
#catboost model for individual leagues
# Define the target column and features
target = 'winner'
features = train_data.columns.drop([target])

# Convert appropriate columns to 'category' dtype for CatBoost
categorical_features = ['home_team_name', 'away_team_name', 'stadium_name','league_x']

train_data[categorical_features] = train_data[categorical_features].astype('category')

# Prepare data by removing unseen labels in validation and test sets
for col in categorical_features:
    train_data[col] = train_data[col].astype('category')
    val_data[col] = val_data[col].astype('category')
    test_data[col] = test_data[col].astype('category')

    unseen_labels_val = set(val_data[col].cat.categories) - set(train_data[col].cat.categories)
    unseen_labels_test = set(test_data[col].cat.categories) - set(train_data[col].cat.categories)

    if unseen_labels_val:
        print(f"Unseen labels in validation set for column '{col}': {unseen_labels_val}")
        val_data = val_data[~val_data[col].isin(unseen_labels_val)]

    if unseen_labels_test:
        print(f"Unseen labels in test set for column '{col}': {unseen_labels_test}")
        test_data = test_data[~test_data[col].isin(unseen_labels_test)]
```

```
# Ensure that we have non-empty validation and test sets after removing unseen labels
if val_data.empty or test_data.empty:
    raise ValueError("Validation or test set is empty after removing rows with unseen labels.
Please check the data.")

# Create Pool objects for CatBoost
train_pool = Pool(train_data[features], train_data[target], cat_features=categorical_features)
val_pool = Pool(val_data[features], val_data[target], cat_features=categorical_features)
test_pool = Pool(test_data[features], test_data[target], cat_features=categorical_features)

# Train CatBoost model
model = CatBoostClassifier(
    iterations=2000,
    learning_rate=0.05,
    depth=4,
    l2_leaf_reg=3.0,
    rsm=0.8,
    loss_function='MultiClass',
    eval_metric='Accuracy',
    random_seed=42,
    bagging_temperature=1.0,
    od_type='Iter',
    od_wait=200,
    verbose=100
)

model.fit(train_pool, eval_set=val_pool, early_stopping_rounds=500)
#plotting individual accuracies for catboost model
y_pred = model.predict(test_pool)
```

```
if len(y_pred.shape) > 1 and y_pred.shape[1] == 1:
    y_pred = y_pred.flatten()

# Alternatively, you can use y_pred.ravel() which also flattens the array
# y_pred = y_pred.ravel()

# Adding predictions to the test_data DataFrame
test_data['winner_prediction'] = y_pred

# Initialize a dictionary to store accuracies for each league
league_accuracies = {}

for i in test_data['league_x'].unique():
    test_data_sub=test_data[test_data['league_x']==i]
    # Evaluate the model
    print(i)
    accuracy = accuracy_score(test_data_sub[target], test_data_sub['winner_prediction'])
    report = classification_report(test_data_sub[target], test_data_sub['winner_prediction'])

    print(f"Accuracy: {accuracy}")
    print(f"Classification Report:\n{report}")
    print(" ")
    league_accuracies[i] = accuracy
# Plotting bar graph for accuracies
plt.figure(figsize=(10, 6))
plt.bar(league_accuracies.keys(), league_accuracies.values())
plt.xlabel('League')
plt.ylabel('Accuracy')
plt.title('Catboost Model Accuracy by League')
plt.xticks(rotation=45)
plt.show()
#feature importance for catboost model
import matplotlib.pyplot as plt
feature_importances = model.get_feature_importance(train_pool)
feature_names = train_data[features].columns

# Create a DataFrame for visualization
feature_importance_df = pd.DataFrame({
   'Feature': feature_names,
   'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Plot feature importance
import matplotlib.pyplot as plt
import pandas as pd



# Create a DataFrame for visualization
feature_importance_df = pd.DataFrame({
   'Feature': feature_names,
   'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Plot feature importance
plt.figure(figsize=(12, 8))
plt.bar(feature_importance_df['Feature'], feature_importance_df['Importance'])

plt.xticks(rotation=90, ha='right')

plt.xlabel('Feature')

plt.ylabel('Importance')

plt.title('Catboost Feature Importances')

plt.tight_layout()

plt.show()

# plotting actual result and predicted result

test_data_latest_data=test_data

test_data_latest_data=test_data_latest_data[['home_team_name','away_team_name','winner','win
ner_prediction']]

test_data_latest_data.tail(20)

#modeling random forest, XGboost, LightGbm

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

from lightgbm import LGBMClassifier

from sklearn.metrics import accuracy_score, classification_report

train_data=pd.read_csv('train_set.csv')

test_data=pd.read_csv('test_set.csv')

val_data=pd.read_csv('validation_set.csv')



cols_to_drop = ['timestamp',
'date_GMT','Season','home_team_goal_count','away_team_goal_count']


# Drop the specified columns
```

```
test_data = test_data.drop(columns=cols_to_drop)
train_data=train_data.drop(columns=cols_to_drop)
val_data=val_data.drop(columns=cols_to_drop)
# Define the target column and features
target = 'winner'
features = train_data.columns.drop([target])

# Convert appropriate columns to 'category' dtype for CatBoost
categorical_features = ['home_team_name', 'away_team_name', 'stadium_name','league_x']

# Ensure the categorical features are of category dtype
train_data[categorical_features] = train_data[categorical_features].astype('category')
val_data[categorical_features] = val_data[categorical_features].astype('category')
test_data[categorical_features] = test_data[categorical_features].astype('category')

# Prepare data for other models by label encoding categorical columns and target variable
label_encoders = {col: LabelEncoder().fit(train_data[col]) for col in categorical_features}
target_encoder = LabelEncoder().fit(train_data[target])

for col, le in label_encoders.items():
    train_data[col] = le.transform(train_data[col])
    # Transform validation and test data, handle unseen labels
    unseen_labels_val = set(val_data[col].unique()) - set(le.classes_)
    unseen_labels_test = set(test_data[col].unique()) - set(le.classes_)

    if unseen_labels_val:
        print(f"Unseen labels in validation set for column '{col}': {unseen_labels_val}")
        val_data = val_data[~val_data[col].isin(unseen_labels_val)]
    if unseen_labels_test:
        print(f"Unseen labels in test set for column '{col}': {unseen_labels_test}")
        test_data = test_data[~test_data[col].isin(unseen_labels_test)]

    le.classes_ = np.append(le.classes_, list(unseen_labels_val))
    le.classes_ = np.append(le.classes_, list(unseen_labels_test))
    val_data[col] = le.transform(val_data[col])
    test_data[col] = le.transform(test_data[col])

train_data[target] = target_encoder.transform(train_data[target])
val_data[target] = target_encoder.transform(val_data[target])
test_data[target] = target_encoder.transform(test_data[target])
# Train RandomForest model
rf_model = RandomForestClassifier(
    n_estimators=200,          # Number of trees
    max_depth=6,               # Maximum depth of the tree
    min_samples_split=5,       # Minimum number of samples required to split an internal node
    min_samples_leaf=2,        # Minimum number of samples required to be at a leaf node
    random_state=42
)
rf_model.fit(train_data[features], train_data[target])

# Train XGBoost model with additional hyperparameters
xgb_model = XGBClassifier(
    use_label_encoder=False,
    eval_metric='mlogloss',
    n_estimators=200,          # Number of boosting rounds
    learning_rate=0.1,        # Learning rate
    max_depth=6,              # Maximum depth of the tree
    subsample=0.8,            # Subsample ratio of the training instances
    colsample_bytree=0.8,     # Subsample ratio of columns when constructing each tree
    random_state=42
)
xgb_model.fit(train_data[features], train_data[target])

# Train LightGBM model with additional hyperparameters
lgbm_model = LGBMClassifier(
    n_estimators=200,         # Number of boosting rounds
    learning_rate=0.1,        # Learning rate
    max_depth=10,             # Maximum depth of the tree
    num_leaves=6,             # Maximum tree leaves for base learners
    subsample=0.8,            # Subsample ratio of the training instances
    colsample_bytree=0.8,     # Subsample ratio of columns when constructing each tree
    random_state=42
)
lgbm_model.fit(train_data[features], train_data[target])

# Evaluate models
models = {
    "Random Forest": rf_model,
    "XGBoost": xgb_model,
    "LightGBM": lgbm_model
}
#plotting accuracies and confusion matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Function to plot confusion matrix
def plot_confusion_matrix(cm, model_name, classes):
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=classes,
yticklabels=classes)
    plt.title(f"{model_name} Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

# Evaluate models and plot confusion matrix
for name, model in models.items():
    y_pred = model.predict(test_data[features])
    test_data[f'winner_prediction_{name}'] = y_pred
    accuracy = accuracy_score(test_data[target], y_pred)
    report = classification_report(test_data[target], y_pred, target_names=target_encoder.classes_)
    cm = confusion_matrix(test_data[target], y_pred)

    print(f"Model: {name}")
    print(f"Accuracy: {accuracy}")
    print("Classification Report:")
    print(report)
    print("\n" + "="*60 + "\n")

    plot_confusion_matrix(cm, name, target_encoder.classes_)
# mapping leagues names for individual accuracies
```

```
league_mapping = {
    1: 'Premier League',
    3: 'UEFA Champions League',
    2: 'Serie A',
    0: 'La Liga'
}

# Replace numerical league identifiers with actual league names
test_data['league_x'] = test_data['league_x'].map(league_mapping)
#accuracies for individual models
# Evaluate and plot accuracies by league for each model
league_accuracies = {}

for model_name in models.keys():
    print(f"\nEvaluating model: {model_name}\n")
    league_accuracies[model_name] = {}

    for league in test_data['league_x'].unique():
        test_data_sub = test_data[test_data['league_x'] == league]

        accuracy = accuracy_score(test_data_sub[target],
test_data_sub[f'winner_prediction_{model_name}'])
        report = classification_report(test_data_sub[target],
test_data_sub[f'winner_prediction_{model_name}'])

        print(f"League: {league}")
        print(f"Accuracy: {accuracy}")
        print(f"Classification Report:\n{report}")
        print(" ")
        # Store the accuracy in the dictionary
        league_accuracies[model_name][league] = accuracy

    # Plotting the accuracies in a bar graph
    plt.figure(figsize=(8, 6))
    plt.bar(league_accuracies[model_name].keys(),
league_accuracies[model_name].values(),color='green')
    plt.xlabel('League')
    plt.ylabel('Accuracy')
    plt.title(f'Model Accuracy by League: {model_name}')
    plt.xticks(rotation=45)
    plt.show()
#feature importance for random forest, XGboost, and lightGbm
def plot_feature_importances(model, features, model_name):
    importances = model.feature_importances_
    indices = np.argsort(importances)[::-1]

    plt.figure(figsize=(10, 6))
    plt.title(f"{model_name} Feature Importances")
    plt.bar(range(len(features)), importances[indices], align="center")
    plt.xticks(range(len(features)), [features[i] for i in indices], rotation=90)
    plt.tight_layout()
    plt.show()

# Random Forest feature importances
plot_feature_importances(rf_model, features, "Random Forest")

# XGBoost feature importances
```

```
plot_feature_importances(xgb_model, features, "XGBoost")


# LightGBM feature importances

plot_feature_importances(lgbm_model, features, "LightGBM")

#plotting predicted result and actual result for random forest, XGboost, lightGbm

test_data_pred_res= test_data[['home_team_name',
'away_team_name','winner','winner_prediction_Random Forest', 'winner_prediction_XGBoost',

    'winner_prediction_LightGBM']]

home_team_list=test_data_cat['home_team_name']

away_team_list=test_data_cat['away_team_name']

test_data_pred_res['home_team_name'] = home_team_list

test_data_pred_res['away_team_name'] = away_team_list

winner_mapping = {

   0: 'away',

   1: 'draw',

   2: 'home'

}


test_data_pred_res['winner'] = test_data_pred_res['winner'].map(winner_mapping)

test_data_pred_res['winner_prediction_Random Forest'] =
test_data_pred_res['winner_prediction_Random Forest'].map(winner_mapping)

test_data_pred_res['winner_prediction_XGBoost'] =
test_data_pred_res['winner_prediction_XGBoost'].map(winner_mapping)

test_data_pred_res['winner_prediction_LightGBM'] =
test_data_pred_res['winner_prediction_LightGBM'].map(winner_mapping)

test_data_pred_res.tail(20)


#feature engineering

import pandas as pd

model_players=pd.read_csv('combined_leagues_players_model_ready.csv')
```

```
grouped_df = model_players.groupby(['Current Club', 'season',
'league']).sum(numeric_only=True).reset_index()

grouped_df.columns

team_experience=grouped_df

matches_model=pd.read_csv('combined_leagues_matches_model_ready.csv')

matches_model

matches_model['winner'] = matches_model.apply(lambda row: 'draw' if
row['home_team_goal_count'] == row['away_team_goal_count']

        else 'home' if row['home_team_goal_count'] > row['away_team_goal_count']

        else 'away', axis=1)

matches_model.head()

matches_model.info()

import numpy as np

matches_model.replace(-1, np.nan, inplace=True)

matches_model.isnull().sum()

matches_model.head()

grouped_df_home_analysis =
matches_model.groupby('home_team_name')[['home_team_shots_on_target',

                            'home_team_shots_off_target',

                            'home_team_fouls',

                            'home_team_possession']].median().reset_index()

grouped_df_home_analysis

grouped_df_away_analysis =
matches_model.groupby('away_team_name')[['away_team_shots_on_target',

                          'away_team_fouls',

                          'away_team_shots_off_target',

                          'away_team_possession']].median().reset_index()

grouped_df_away_analysis
```

```
merged_df = matches_model.merge(grouped_df_home_analysis, on='home_team_name',
suffixes=('', '_mean'))


# Fill NaN values in matches_model with values from grouped_df_home_analysis

columns_to_fill = ['home_team_shots_on_target', 'home_team_shots_off_target',
'home_team_fouls', 'home_team_possession']


for col in columns_to_fill:

   merged_df[col] = merged_df[col].fillna(merged_df[f"{col}_mean"])


# Drop the extra columns created during the merge

merged_df = merged_df.drop(columns=[f"{col}_mean" for col in columns_to_fill])

matches_model = merged_df

matches_model.isnull().sum()


merged_df = matches_model.merge(grouped_df_away_analysis, on='away_team_name',
suffixes=('', '_mean'))


# Fill NaN values in matches_model with values from grouped_df_home_analysis

columns_to_fill = ['away_team_shots_on_target', 'away_team_shots_off_target', 'away_team_fouls',
'away_team_possession']


for col in columns_to_fill:

   merged_df[col] = merged_df[col].fillna(merged_df[f"{col}_mean"])


# Drop the extra columns created during the merge

merged_df = merged_df.drop(columns=[f"{col}_mean" for col in columns_to_fill])

matches_model = merged_df

matches_model.isnull().sum()

matches_model.isnull().mean()*100
matches_model.dropna(inplace=True)

matches_model.shape

matches_model.iloc[3]

matches_model.columns

matches_model['Season'] = matches_model['Season'].str.replace('-to-', '/')

matches_model['Season']

grouped_df_home_renamed=grouped_df

grouped_df_home_renamed = grouped_df_home_renamed.rename(columns={

  'age': 'age_home',

  'minutes_played_overall': 'minutes_played_overall_home',

  'minutes_played_home': 'minutes_played_home_home',

  'minutes_played_away': 'minutes_played_away_home',

  'appearances_overall': 'appearances_overall_home',

  'appearances_home': 'appearances_home_home',

  'appearances_away': 'appearances_away_home',

  'goals_overall': 'goals_overall_home',

  'goals_home': 'goals_home_home',

  'goals_away': 'goals_away_home',

  'assists_overall': 'assists_overall_home',

  'assists_home': 'assists_home_home',

  'assists_away': 'assists_away_home',

  'penalty_goals': 'penalty_goals_home',

  'penalty_misses': 'penalty_misses_home',

  'goals_involved_per_90_overall': 'goals_involved_per_90_overall_home',

  'assists_per_90_overall': 'assists_per_90_overall_home',

  'goals_per_90_overall': 'goals_per_90_overall_home',

  'goals_per_90_home': 'goals_per_90_home_home'

})

grouped_df_home_renamed
```

```
# Merge with away team data
merged_full = matches_model.merge(
   grouped_df_home_renamed,
   left_on=['home_team_name', 'Season'],
   right_on=['Current Club', 'season'],
)

# To replace the original matches_model DataFrame with the updated one
matches_model_home = merged_full
grouped_df_away_renamed=grouped_df
grouped_df_away_renamed = grouped_df_away_renamed.rename(columns={

   'age': 'age_away',
   'minutes_played_overall': 'minutes_played_overall_away',
   'minutes_played_home': 'minutes_played_home_away',
   'minutes_played_away': 'minutes_played_away_away',
   'appearances_overall': 'appearances_overall_away',
   'appearances_home': 'appearances_home_away',
   'appearances_away': 'appearances_away_away',
   'goals_overall': 'goals_overall_away',
   'goals_home': 'goals_home_away',
   'goals_away': 'goals_away_away',
   'assists_overall': 'assists_overall_away',
   'assists_home': 'assists_home_away',
   'assists_away': 'assists_away_away',
   'penalty_goals': 'penalty_goals_away',
   'penalty_misses': 'penalty_misses_away',
   'goals_involved_per_90_overall': 'goals_involved_per_90_overall_away',
    'assists_per_90_overall': 'assists_per_90_overall_away',
    'goals_per_90_overall': 'goals_per_90_overall_away',
    'goals_per_90_home': 'goals_per_90_home_away'
})

# Merge with away team data
merged_full_all = matches_model_home.merge(
    grouped_df_away_renamed,
    left_on=['away_team_name', 'Season'],
    right_on=['Current Club', 'season'],
)

# To replace the original matches_model DataFrame with the updated one
model_match_ready = merged_full_all
model_match_ready
model_match_ready.iloc[0]
test_su=grouped_df_home_renamed[grouped_df_home_renamed['Current Club']=='Sunderland']
model_match_ready
del model_match_ready['Current Club_y']
del model_match_ready['season_y']
del model_match_ready['league_y']
del model_match_ready['league_x']
del model_match_ready['season_x']
del model_match_ready['Current Club_x']
model_match_ready.columns
model_match_ready.describe()
model_match_ready.drop_duplicates(inplace=True)
model_match_ready.shape
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt


correlation_matrix = model_match_ready.corr()


plt.figure(figsize=(45, 30))

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

plt.title('Correlation Matrix of Numerical Variables')

plt.show()

model_match_ready.to_csv('model_final_joined.csv',index=False)

model_match_ready['date'] = pd.to_datetime(model_match_ready['timestamp'], unit='s')


# Define the training, validation, and test sets based on the date

train_set = model_match_ready[model_match_ready['date'].dt.year < 2022]

validation_set = model_match_ready[(model_match_ready['date'].dt.year >= 2022) &
(model_match_ready['date'].dt.year < 2023)]

test_set = model_match_ready[model_match_ready['date'].dt.year >= 2023]


# Drop the 'date' column if not needed

train_set = train_set.drop(columns=['date'])

validation_set = validation_set.drop(columns=['date'])

test_set = test_set.drop(columns=['date'])


# Optionally, save the resulting DataFrames to CSV files

train_set.to_csv('train_set.csv', index=False)

validation_set.to_csv('validation_set.csv', index=False)

test_set.to_csv('test_set.csv', index=False)
```

```
#EDA
import pandas as pd
#leagues->appending

england_league=pd.read_csv('combined_england_premier_league_stats.csv')
spain_league=pd.read_csv('combined_spain_la_liga_league_stats.csv')
champion_league=pd.read_csv('combined_uefa_champions_league_stats.csv')
serieA_league=pd.read_csv('combined_italy_serie_a_league_stats.csv')
print(england_league.shape)
print(spain_league.shape)
print(champion_league.shape)
print(serieA_league.shape)
#concat the leagues data

combined_leagues_football=pd.concat([england_league,spain_league,champion_league,serieA_le
ague],axis=0)
combined_leagues_football.shape
#matches-->appending

england_matches=pd.read_csv('combined_england_premier_league_matches_stats.csv')
spain_matches=pd.read_csv('combined_spain_la_liga_league_matches_stats.csv')
champion_matches=pd.read_csv('combined_uefa_champions_league_matches_stats.csv')
serieA_matches=pd.read_csv('combined_italy_serie_a_league_matches_stats.csv')
print(england_matches.shape)
print(spain_matches.shape)
print(champion_matches.shape)
print(serieA_matches.shape)
#concat the matches data
```

```
combined_leagues_matches=pd.concat([england_matches,spain_matches,champion_matches,s
erieA_matches],axis=0)

combined_leagues_matches.shape


#players->appending

england_players=pd.read_csv('combined_england_premier_league_players_stats.csv')

spain_players=pd.read_csv('combined_spain_la_liga_league_players_stats.csv')

champion_players=pd.read_csv('combined_uefa_champions_league_players_stats.csv')

serieA_players=pd.read_csv('combined_italy_serie_a_league_players_stats.csv')

print(england_players.shape)

print(spain_players.shape)

print(champion_players.shape)

print(serieA_players.shape)


#concat the players data

combined_leagues_players=pd.concat([england_players,spain_players,champion_players,serieA_
players],axis=0)

combined_leagues_teams.shape

#Leagues

combined_leagues_football.head(10)

for i in combined_leagues_football.columns:

  print(i)

drop_columns=['status','format','number_of_clubs','total_matches',

       'matches_completed','game_week','total_game_week','progress']

combined_leagues_football=combined_leagues_football.drop(columns=drop_columns)

null_df=pd.DataFrame(combined_leagues_football.isnull().mean() * 100)

pd.set_option('display.max_rows', None)

null_df

#so no null values in a data frame
```

```
combined_leagues_football.describe()

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.DataFrame({

   'Statistic': ['Average Goals Per Match', 'Average Scored Home Team', 'Average Scored Away Team', 'BTTS Percentage', 'Clean Sheets Percentage', 'Prediction Risk', 'Home Scored Advantage Percentage', 'Home Defence Advantage Percentage', 'Home Advantage Percentage', 'Average Corners Per Match', 'Goals Min 61 to 70', 'Goals Min 71 to 80', 'Goals Min 81 to 90', 'Goals Min 0 to 15', 'Goals Min 16 to 30', 'Goals Min 31 to 45', 'Goals Min 46 to 60', 'Goals Min 61 to 75', 'Goals Min 76 to 90', 'xG Avg Per Match'],

   'Mean': [2.742813, 1.558281, 1.184531, 50.828125, 56.015625, 72.046875, 32.312500, 23.718750, 28.234375, 8.671094, 105.0625, 103.796875, 158.953125, 117.078125, 135.437500, 157.203125, 152.125000, 156.656250, 211.156250, 1.424531],

   'Std Dev': [0.156159, 0.094189, 0.103754, 4.146349, 5.159341, 6.435096, 12.243398, 7.033861, 9.640757, 3.575238, 25.3752, 23.554179, 36.309267, 26.394627, 30.580041, 36.036657, 33.377827, 37.142035, 45.651968, 1.533070]

})


# Display the DataFrame

print(df)


# Plotting the data

plt.figure(figsize=(14, 8))

sns.barplot(x='Mean', y='Statistic', data=df, palette='viridis', orient='h')

plt.title('Mean Values of Various Football Match Statistics')

plt.xlabel('Mean Value')

plt.ylabel('Statistic')

plt.show()

sns.set(style="whitegrid")


# Plot Average Goals Per Match by Season

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

```
sns.barplot(x='name', y='average_goals_per_match', data=combined_leagues_football,
palette='viridis')

plt.title('Average Goals Per Match by Season')

plt.xlabel('Season')

plt.ylabel('Average Goals Per Match')

plt.xticks(rotation=45)

plt.show()

columns = [

  'average_goals_per_match',

  'average_scored_home_team',

  'average_scored_away_team',

  'btts_percentage',

  'clean_sheets_percentage',

  'average_corners_per_match',

  'average_corners_per_match_home_team',

  'average_corners_per_match_away_team'

]


for i in columns:

  sns.set(style="whitegrid")


# Plot Average Goals Per Match by Season

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

  sns.barplot(x='name', y=i, data=combined_leagues_football, palette='viridis')

  plt.title('Season')

  plt.xlabel('league')

  plt.ylabel(i)

  plt.xticks(rotation=45)

  plt.show()
```

```
numerical_columns = [
    'average_goals_per_match',
    'average_scored_home_team',
    'average_scored_away_team',
    'btts_percentage',
    'clean_sheets_percentage',
    'average_corners_per_match',
    'average_corners_per_match_home_team',
    'average_corners_per_match_away_team',

]

# Calculate the correlation matrix
correlation_matrix = combined_leagues_football[numerical_columns].corr()

# Plot the heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix of Numerical Columns')
plt.show()
#Matches
combined_leagues_matches.head(10)
combined_leagues_matches.shape
combined_leagues_matches.iloc[14000]
null_df_matches=pd.DataFrame(combined_leagues_matches.isnull().mean()*100)
null_df_matches
combined_leagues_matches =
combined_leagues_matches[['home_team_name','away_team_name','timestamp', 'date_GMT',
                          'Pre-Match PPG (Home)','Pre-Match PPG (Away)', 'home_ppg',
                          'away_ppg','home_team_goal_count','away_team_goal_count',
                        'total_goal_count','Home Team Pre-Match xG', 'Away Team Pre-Match xG',
                          'team_a_xg','team_b_xg', 'average_goals_per_match_pre_match',
                          'btts_percentage_pre_match','odds_ft_home_team_win',
                          'odds_ft_draw', 'odds_ft_away_team_win', 'odds_btts_yes',
                          'odds_btts_no', 'stadium_name', 'Season','home_team_shots_on_target',
                          'away_team_shots_on_target', 'home_team_shots_off_target',
                          'away_team_shots_off_target', 'home_team_fouls', 'away_team_fouls',
                          'home_team_possession', 'away_team_possession']]
import numpy as np
combined_leagues_matches.replace(-1, np.nan, inplace=True)
combined_leagues_matches.head(10)
combined_leagues_matches.describe()
combined_leagues_matches.isnull().sum()
combined_leagues_matches=combined_leagues_matches.fillna(combined_leagues_matches.me
an())
#filling data with mean
combined_leagues_matches.head()
numerical_columns = combined_leagues_matches.select_dtypes(include='number')

# Calculate the correlation matrix
correlation_matrix = numerical_columns.corr()

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(correlation_matrix, annot=True,fmt='.2f', cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix of Numerical Columns')
plt.show()
#Players
```

```
combined_leagues_players.head(10)
combined_leagues_players.shape
combined_leagues_players.isnull().mean()*100
cols_to_take = ['full_name', 'age','league', 'season',
    'position', 'Current Club', 'minutes_played_overall',
    'minutes_played_home', 'minutes_played_away', 'nationality',
    'appearances_overall', 'appearances_home', 'appearances_away',
    'goals_overall', 'goals_home', 'goals_away', 'assists_overall',
    'assists_home', 'assists_away', 'penalty_goals', 'penalty_misses',
    'goals_involved_per_90_overall', 'assists_per_90_overall',
    'goals_per_90_overall', 'goals_per_90_home']

combined_leagues_players = combined_leagues_players[cols_to_take]
combined_leagues_players.head()
combined_leagues_players.isnull().sum()
numerical_columns = combined_leagues_players.select_dtypes(include='number')

# Calculate the correlation matrix
correlation_matrix = numerical_columns.corr()

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(correlation_matrix, annot=True,fmt='.2f', cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix of Numerical Columns')
plt.show()
numerical_columns = combined_leagues_players.select_dtypes(include=['float64',
'int64']).columns

# Drop 'goals_overall' if it's not in the dataset or to avoid redundancy
```

```
if 'goals_overall' not in numerical_columns:
    raise ValueError("The column 'goals_overall' is not present in the dataset.")
else:
    numerical_columns = numerical_columns.drop('goals_overall')


# Create scatter plots
for column in numerical_columns:
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=combined_leagues_players[column],
y=combined_leagues_players['goals_overall'])
    plt.title(f'Scatter Plot: {column} vs. goals_overall')
    plt.xlabel(column)
    plt.ylabel('goals_overall')
    plt.show()
#Teams
combined_leagues_teams.head()
combined_leagues_teams.shape
for i in combined_leagues_teams.columns:
    print(i)
combined_leagues_teams.describe()
columns_to_mean = [
    'matches_played', 'matches_played_home', 'matches_played_away', 'suspended_matches',
    'wins', 'wins_home', 'wins_away', 'draws', 'draws_home', 'draws_away', 'losses',
    'losses_home', 'losses_away', 'points_per_game', 'points_per_game_home',
    'points_per_game_away', 'league_position'
]


# Group by 'common_name' and calculate the mean for the specified columns
```

```
grouped_means =
combined_leagues_teams.groupby('common_name')[columns_to_mean].mean()

grouped_means

columns_to_mean = [

    'matches_played', 'matches_played_home', 'matches_played_away', 'suspended_matches',

    'wins', 'wins_home', 'wins_away', 'draws', 'draws_home', 'draws_away', 'losses',

    'losses_home', 'losses_away', 'points_per_game', 'points_per_game_home',

    'points_per_game_away', 'league_position'

]


# Group by 'common_name' and calculate the mean for the specified columns

grouped_means_max =
combined_leagues_teams.groupby('common_name')[columns_to_mean].max()

#check league position --> if its more thats tge worst performance

grouped_means_max

columns_to_mean = [

    'matches_played', 'matches_played_home', 'matches_played_away', 'suspended_matches',

    'wins', 'wins_home', 'wins_away', 'draws', 'draws_home', 'draws_away', 'losses',

    'losses_home', 'losses_away', 'points_per_game', 'points_per_game_home',

    'points_per_game_away', 'league_position'

]


# Group by 'common_name' and calculate the mean for the specified columns

grouped_means_sum =
combined_leagues_teams.groupby('common_name')[columns_to_mean].sum()

grouped_means_sum

numerical_columns = combined_leagues_teams.select_dtypes(include=['float64',
'int64']).columns


# Drop 'wins' if it's not in the dataset or to avoid redundancy
```

```
if 'wins' not in numerical_columns:
    raise ValueError("The column 'goals_overall' is not present in the dataset.")
else:
    numerical_columns = numerical_columns.drop('wins')


# Create scatter plots
for column in numerical_columns:
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=combined_leagues_teams[column], y=combined_leagues_teams['wins'])
    plt.title(f'Scatter Plot: {column} vs. goals_overall')
    plt.xlabel(column)
    plt.ylabel('goals_overall')
    plt.show()
combined_leagues_football_model=combined_leagues_football[['name','season','average_goals_p
er_match','xg_avg_per_match',

'average_scored_home_team','average_scored_away_team','btts_percentage',

'average_corners_per_match','average_corners_per_match_home_team',
                                              'average_corners_per_match_away_team']]


combined_leagues_football_model.head()
combined_leagues_matches_model =
combined_leagues_matches[['home_team_name','away_team_name','timestamp', 'date_GMT',
                                             'Pre-Match PPG (Home)','Pre-Match PPG (Away)', 'home_ppg',
                                             'away_ppg','home_team_goal_count','away_team_goal_count',
                                          'total_goal_count','Home Team Pre-Match xG', 'Away Team Pre-Match xG',
                                             'team_a_xg','team_b_xg', 'average_goals_per_match_pre_match',
                                             'btts_percentage_pre_match','odds_ft_home_team_win',
                                             'odds_ft_draw', 'odds_ft_away_team_win', 'odds_btts_yes',
                                    'odds_btts_no', 'stadium_name', 'Season','home_team_shots_on_target',
                                    'away_team_shots_on_target', 'home_team_shots_off_target',
                                    'away_team_shots_off_target', 'home_team_fouls', 'away_team_fouls',
                                    'home_team_possession', 'away_team_possession']]


combined_leagues_matches_model.head()

combined_leagues_players_model = combined_leagues_players[['full_name', 'age','league',
'season','position', 'Current Club',
                                    'minutes_played_overall','minutes_played_home',
'minutes_played_away',
                                    'nationality','appearances_overall', 'appearances_home',
'appearances_away',
                                    'goals_overall', 'goals_home', 'goals_away', 'assists_overall',
                                    'assists_home', 'assists_away', 'penalty_goals', 'penalty_misses',
                                    'goals_involved_per_90_overall', 'assists_per_90_overall',
                                    'goals_per_90_overall', 'goals_per_90_home']]


combined_leagues_players_model.head()

combined_leagues_teams_model=combined_leagues_teams[['team_name', 'common_name',
'season', 'country', 'matches_played',
                                  'matches_played_home','matches_played_away', 'suspended_matches',
'wins',
                                  'wins_home', 'wins_away', 'draws','draws_home', 'draws_away', 'losses',
                                  'losses_home', 'losses_away', 'points_per_game','points_per_game_home',
                                  'points_per_game_away', 'league_position', 'league_position_home',
                                  'league_position_away', 'performance_rank', 'goals_scored',
'goals_conceded',
                                  'goal_difference', 'total_goal_count', 'total_goal_count_home',
'total_goal_count_away',
                                  'goals_scored_home', 'goals_scored_away']]
```

```
combined_leagues_teams_model.head()

#saving the model file csv's will include the columns which are important


combined_leagues_teams_model.to_csv('combined_leagues_teams_model.csv',index=False)

combined_leagues_players_model.to_csv('combined_leagues_players_model.csv',index=False)

combined_leagues_matches_model.to_csv('combined_leagues_matches_model.csv',index=False)

combined_leagues_football_model.to_csv('combined_leagues_standings_model.csv',index=False)
```