Sports (Football) Match Predictor

by

Tameem Islam
24141133
Ridwanul Haque
20341016
Linkon Dhar
20201190
Md. Monjur E Elahe
23141076

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
June 20 2025

Declaration

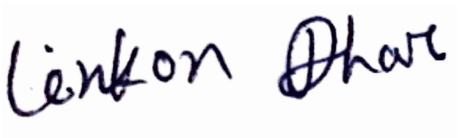
It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Tameem Islam
24141133

Ridwanul Haque
20341016



Linkon Dhar
20201190

Md. Monjur E Elahe
23141076

Approval

The thesis titled "Sports (Football) Match Predictor" submitted by

1. Tameem Islam(24141133)
2. Ridwanul Haque(20341016)
3. Linkon Dhar(20201190)
4. Md. Monjur E Elahe(23141076)

Of Summer, 2025 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on June 16, 2025.

Examining Committee:

Supervisor:
(Member)

Md Imran Bin Azad
Senior Lecturer
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Abstract

Football match prediction has gained significant attention in recent years, especially with the advent of machine learning models. This thesis presents the development and implementation of a predictive model for football match outcomes, leveraging historical team and player statistics. The model employs Scikit-learn logistic regression model,, combined with supporting statistical techniques, to estimate match results, probable goal scorers, and influential factors behind predictions.The vital use of LLM for subjective information incorporation, this research provides a comprehensive analysis of football match forecasting, considering multiple parameters such as team performance, player attributes, and league statistics, with an website at the end demonstrating the fruitful results.

Keywords: machine learning, player statistics, Poisson regression, LLM, expert opinion, home advantage, quantization, rule based,logistic regression.

Acknowledgement

Firstly, all praise to the Great Almighty for whom our thesis have been completed without any major interruption. Secondly, to our supervisor Md Imran Bin Azad sir for his kind support and advice in our work. He helped us whenever we needed help. Finally to our parents without their throughout support it may not be possible. With their kind support and prayer we are now on the verge of our graduation.

Table of Contents

Chapter 1

Introduction

Football, the biggest global sport in the world, or popularly coined as the beautiful game, has impacted the lives of its fans worldwide, and one of the leagues that makes football popular and illuminates its prestige is the English Premier League, "the most watched competitive sports league in the world." Football fans are deeply engrossed in the game, with the majority demonstrating genuine passion for their teams; for many, the most significant event of the day is the victory of their preferred team. Only a true fan knows the feeling of the roller coaster of emotions, the psychological anxiety when the opponents are attacking, the mood swings during a game, and the impact the match's result has on life and on the following day's activities. The impact of the football match results does affect a fan's life. And As fans of Premier League teams, we know it all. However, the interest behind doing this project is rather more personal than general. The interest in creating a match-winning prediction comes from the performance of Manchester United Football Club. Supporting this club is painful, and nobody knows in recent times how they will perform; even the match predictors we have seen fail to detect what the results will be when the team plays. Most of the predictors show a United win; however, relying on such match predictors would be chaotic, as the results frequently come as a draw or defeat. As United fans, we cannot change the bad phase of our club, as it is up to the players for that, but as fellow fan(s), we can implement a prediction model that is going to give the fans an opportunity to rely on support and hope in a more reliable outcome predictor. At least we won't be expecting much and hoping accordingly and not feel the agony of loss in what we deemed as an anticipated win. It would help to adjust the expectation and prepare before matches, something which fans would find helpful. We believe many other fans of other teams face the same problem, especially in this League thus, with our coding skills and our knowledge on machine learning, we would love to create a match-winning prediction that would artificially support and give fans clarity of what to expect, which in pre-modern-day Premier League matches only a handful truly know, but with the predictor you would too.

1.1 Problem Statement

In recent years, the use of data science and machine learning (ML) in football analytics has significantly increased. Much of this research, along with nearly all publicly available match prediction systems, heavily relies on structured numerical data. Such metrics encompass goals scored, possession rates, shots on target, player ratings, and win/loss records. This quantitative data can be easily accessed from various open-source platforms, including sports analytics APIs, match databases, and league archives. Consequently, these numerical datasets serve as the foundational inputs for most traditional prediction models, both in academic research and commercial applications.

However, relying solely on numerical or tabular data creates a major limitation. It overlooks rich, nuanced, and non-numerical information—especially subjective insights and qualitative judgments from expert commentators, journalists, and analysts. Although unstructured, these forms of data are often based on decades of tactical observation and domain expertise, offering viewpoints that cannot be completely captured by match statistics alone. For instance, an experienced analyst may predict a tactical change, shifts in morale, or the effects of injuries that statistical models may not recognize until later.

Current models seldom, if ever, incorporate expert narratives and human reasoning into their prediction frameworks. This disconnect represents a significant shortfall in match prediction. By disregarding expert opinions, usually shared through articles, podcasts, post-match interviews, or broadcasts, these models miss out on a critical aspect of football knowledge—subjective yet well-informed insights.

This research aims to bridge this gap by introducing an innovative solution: developing and leveraging expert-derived data. Specifically, we intend to collect and formalize the subjective knowledge embedded in the public statements of esteemed football analysts, journalists, and commentators—individuals known for their profound understanding of the Premier League and other major football leagues.

To accomplish this, we propose utilizing large language models (LLMs) to analyze, interpret, and quantify expert opinions into structured, model-compatible features. These opinions are extracted from natural language content—match previews, post-match analyses, tactical breakdowns, and expert interviews—and transformed into "opinion variables." These variables can be assessed based on the commentary's tone, certainty, and subject matter (for example, a positive view of a team's recent performance, a negative opinion regarding a coach's tactics, or a confident outlook on a player's potential).

For instance:
If an expert states, "Manchester United's midfield looks fragile without Casemiro," our system could assign a -1 rating to the team's outlook for that match. Conversely, "Haaland looks unstoppable this season; defenders are terrified of him" could contribute a +2 weight to Manchester City's offensive assessment.

By incorporating these expert-derived qualitative variables into the prediction framework alongside traditional statistical metrics, our model evolves into a hybrid predictor—capable of integrating both objective performance data and subjective football insights. This innovation signifies a clear departure from existing systems that view football strictly as a numbers game. It acknowledges the complex, multi-faceted nature of the sport, where emotional momentum, strategic decision-making, and insider knowledge often play a more significant role than raw statistics. In doing so, the project aspires to connect machine computation with human expertise, establishing a new benchmark for predictive modeling in football analytics. In summary, the issue does not lie in a lack of data but rather in the limited scope of data considered. By introducing non-traditional, opinion-based features utilizing advanced NLP tools, we broaden the parameters of match prediction models and progress toward a more comprehensive, accurate, and human-conscious system for interpreting football match outcomes.

1.2 Research Objectives

The overarching objective of this research is to design and implement a comprehensive football match prediction system that not only forecasts match outcomes but also provides granular player-level insights, making it applicable across multiple top-tier football leagues. The system aims to merge statistical modeling, domain-informed rule-based logic, and machine learning techniques to deliver explainable, actionable predictions.

1.3 Match Result Prediction

The primary function of the proposed system is to serve as a match result predictor, determining the most probable outcome—win, loss, or draw—for any two teams competing in a professional football match. The model will produce a probabilistic breakdown for each of the three possible outcomes, offering insights into the likelihood of each result. This not only enables fans and analysts to gauge expected performances but also serves practical use cases such as betting forecasts, fantasy football planning, and sports journalism support.

This component of the research will leverage both rule-based factors—such as form scores, head-to-head history, home-ground advantage, and point table favoritism—and learned patterns from data-driven models. The outcome probabilities are essential not just as categorical labels (e.g., Team A wins) but as a confidence-weighted forecast that reflects the model's internal scoring rationale. This level of depth and uncertainty quantification allows stakeholders to make more nuanced decisions.

1.4 Goal Scorer Identification

In addition to predicting match outcomes, a novel aspect of this study is its focus on identifying likely goal scorers in the predicted matches. After the model forecasts the

total number of goals expected in a fixture (using historical trends, player ratings, form, and opponent-specific performances), it further evaluates individual player contributions to estimate which players are most likely to score those goals. This process involves ranking players using a weighted combination of metrics, such as

- Individual player form (recent scoring consistency),
- Opponent susceptibility (e.g., a team's weakness against aerial threats or through balls),
- Historical scoring patterns in similar match contexts,
- Player minutes played, fitness status, and positional responsibilities.

For instance, if the model predicts that three goals are likely to be scored in a Manchester City match, it may identify Erling Haaland, Kevin De Bruyne, or Phil Foden as the most probable goal scorers. If Haaland's name appears prominently in the forecast, fans may expect another strong performance. This feature not only enriches the match prediction model but also provides highly engaging, player-specific analytics that are seldom covered in conventional forecasting systems.

1.5 Multi-League Generalizability

Another core objective of the research is to ensure that the model is not confined to a single league but is generalizable across multiple top European football competitions. Specifically, the system will be validated and fine-tuned on data from the following five elite leagues:

- English Premier League (EPL)
- Spanish La Liga
- German Bundesliga
- Italian Serie A
- French Ligue 1

Each of these leagues possesses unique characteristics in terms of match tempo, defensive intensity, tactical variability, and player rotations. Therefore, achieving model portability across these leagues demonstrates the robustness and scalability of the approach. This multi-league integration will involve:

- Normalizing features across leagues (e.g., average goals per match, yellow card frequency, home advantage magnitude),
- Learning league-specific modifiers within the ML model,
- Testing and comparing performance accuracy across cross-league fixtures.

By ensuring league-agnostic compatibility, the proposed model aligns more closely with real-world applications where football fans and professionals track multiple leagues simultaneously. In summary, the key goals of this research extend far beyond binary outcome prediction. The aim is to build an intelligent, interpretable, and flexible football forecasting system that can:

- Predict match outcomes along with confidence levels.
- Forecast individual goal scorers using both statistical and contextual player evaluations.
- Operate effectively across multiple football leagues with minimal reconfiguration.

This blend of macro-level match prediction and micro-level player analysis, applied in a multi-league environment, positions the research as a valuable contribution to both sports analytics and AI-driven decision systems.

Chapter 2

Related works

Over the years, researchers have explored various ways to predict football match outcomes, ranging from statistical models to advanced machine learning algorithms, neural networks etc. One of the common and popular approaches to football match prediction is the use of statistical models. These models typically rely on past records, such as team performance, player stats, and match results, to develop predictive algorithms that can estimate the probability of a particular outcome, such as a win, loss, or draw. For example, Hvattum et al. [3] developed a Poisson regression model that used all the past records of team performance and match results to predict the outcome of football matches as well as Goddard et al. [1] (2004) used the same Poisson regression model to predict the outcome of matches in the English Premier League. Thus, the statistical method of Poisson regression is very handy when computing the outcomes. Furthermore, Aquino et al. [9] as well as Oliva-Lozano et al. [12] explored the impact of situational variables, team formations or lineups, and playing position, and network analysis. These studies highlighted the importance of considering contextual variables in understanding the profile of football players. Hence, contextual variables are a massive factor in the field of prediction, so using LLM the need for feedback from experts who analyze the games will be required and that is what will be the key difference of our project. Moreover, in our project we would be computing many factors, what we believe to be absent in other predictors for example, Hader et al. [8] as well as Julian et al. [11] found out that football match results are also dependent on the usage of the coaches tactics such playing positions of the players, these also affects the distance covered and the physical demand of each player , hence individual positions play a critical role in determining players' performance and work rate levels during matches. This means, tactics or change in formation of players would also traverse the outcome, thus it is important to delve deep into the formations frequently used and players preferred positions and their physical and game stats at that position. Such meticulous precision of computation is required to make a high functioning match predictor. Lastly, it is important to know the limitations of our project work, as mentioned by Berrar et al. [4] the vitality of using domain knowledge in machine learning for football outcome prediction. Future research could focus on developing advanced predictive models that integrate and implement contextual variables, playing positions, and match outcomes to enhance the accuracy of outcome prediction in football matches. With access to even more researches of a player's ability and teams training statistics as well as their performances tested in a more scientific setting, the correlation

of different attributes to the outcome of their play, are either yet to be discovered or classified and kept secret in their respective club, or simply newly researches would change the way things are now to be computed. It is always tricky to understand the advanced tactics of the coaches, and thus along with the nature of the contextual variables there will always be limitations and chances for improvements, with better research data from the future.
The task of predicting football match outcomes has long attracted the attention of statisticians, computer scientists, and sports analysts alike. With football being one of the most unpredictable and emotionally driven sports, researchers have explored a wide array of methods—from classical statistical techniques to advanced artificial intelligence and machine learning algorithms. The literature presents a rich spectrum of research that not only focuses on match outcomes (win, draw, loss) but also dives into deeper components such as player performance, tactical variability, and contextual game dynamics.

2.1 Statistical Approaches to Match Prediction

Statistical modeling has historically served as the foundation of football match prediction. One of the most commonly employed models is the Poisson regression model, which assumes that goals scored by a team in a match follow a Poisson distribution. Hvattum and Arntzen (2010)**Hvattum_2010** implemented this technique to estimate match outcomes by combining historical results with Elo ratings, showing that statistical distributions can capture goal-based patterns effectively. Similarly, Goddard and Asimakopoulos (2004) [3] utilized a similar Poisson model to forecast English Premier League outcomes using historical team statistics. These methods, while powerful in structured data settings, often overlook qualitative aspects such as psychological momentum or tactical adaptability. **Berrar_2018**

Statistical models remain relevant due to their interpretability and low computational cost, and they continue to serve as benchmarks in the development of more complex machine learning frameworks. However, their reliance on structured, numerical inputs limits their effectiveness in integrating richer contextual features.

2.2 Contextual and Tactical Variables in Performance Analysis

Beyond statistical goal modeling, a growing body of research has emphasized the importance of contextual variables in football analysis.[9] Aquino et al. (2020) and [12] Oliva-Lozano et al. (2020) explored the influence of match location (home vs. away), player roles, tactical formations, and network analysis on team dynamics and outcomes. Their findings highlight that a player's performance is heavily influenced by positional context, role-based demands, and tactical flexibility imposed by the coach. These studies argue for a broader consideration of game context in predictive models, moving beyond static performance metrics. [5]

In this regard, the work of Hader et al. (2019) [7] and [11]Julian et al. (2020) is particularly relevant. Their studies analyzed how tactical formations and positional

responsibilities affect player workload, distance covered, and physical demands. The findings suggest that player fatigue, role suitability, and spatial engagement must be factored into any robust prediction framework. Consequently, formation-specific analysis, player preference in positional roles, and individual physical and technical statistics emerge as critical features in predictive modeling.
Our current research builds on this perspective by incorporating these underutilized dimensions into our rule-based and machine learning pipeline. Unlike traditional predictors, we focus on capturing the interdependence of formations, player suitability, and coach strategy, which collectively influence not just the statistical likelihood of a goal but the dynamics of performance leading up to it.

2.3 Use of Machine Learning and Hybrid Frameworks

The shift toward machine learning (ML) has introduced non-linear modeling capabilities, allowing systems to detect complex interactions between input variables. In particular, the work of Rosli et al. (2018) [6] compared different ML techniques—such as decision trees, k-NN, and Naïve Bayes—for predicting football results. While statistical models offered higher transparency, ML approaches demonstrated superior adaptability, particularly in handling non-linear relationships and multivariate dependencies.

In line with this, Choe and McKay (2009) [2] proposed a compound hybrid framework that integrates rule-based reasoning with Bayesian networks, demonstrating the efficacy of combining expert knowledge with probabilistic models. Their study provided a blueprint for the type of rule-augmented system we implement in this research, where domain-specific heuristics and engineered features are used to enhance ML interpretability and performance.

2.4 Integrating LLMs and Expert Opinion in Prediction

A significant innovation in our research is the incorporation of expert opinion extracted using large language models (LLMs). As football is not merely a game of numbers, expert analyses—usually found in articles, interviews, and pre/post-match commentary—offer a rich source of qualitative insights. While traditional models are incapable of digesting such unstructured data, LLMs like OpenAI's GPT-4 or Meta's LLaMA have shown exceptional proficiency in natural language understanding and generation 2 and 12 (Brown et al., 2020; Touvron et al., 2023)
Our approach leverages LLMs to extract expert sentiment, tactical evaluations, and contextual judgments and transforms them into quantitative inputs. This complements numerical features and introduces a previously missing subjective layer to prediction. As shown in Song (2023)[14], combining human insight with machine learning algorithms leads to better forecasting performance, especially in uncertain or novel match contexts. [10]

This hybrid method aligns with recent trends in AI for sports analytics, including research by Van Eetvelde, H. (2018) et al. (2018) and Ismail et al. (2022) [13], where human-in-the-loop systems were used to improve fantasy football forecasts and player performance predictions, respectively. The inclusion of expert opinion bridges the gap between statistical inference and human intuition, giving our model a distinct advantage. [15]

2.5 Challenges, Limitations, and Future Directions

Despite the innovations presented, limitations still persist. As Berrar et al. (2018) [4] point out, domain knowledge remains an essential, yet underutilized, component in many ML models for sports. The interpretability of machine learning models suffers when they operate in isolation from domain heuristics. Similarly, the tactical decisions made by coaches often involve layers of strategic foresight that is difficult to quantify and model (e.g., sudden formation shifts, game plan changes based on in-match events).

Furthermore, the availability of high-resolution player data—such as GPS-tracked movement, recovery rates, or micro-level fatigue markers—is often restricted to clubs and unavailable for open research. This creates a disparity between what can be modeled and what could ideally be modeled with access to proprietary performance data. Going forward, research must continue to explore:

- More nuanced use of contextual and psychological variables
- Enhanced LLM integration with human feedback loops
- Federated or privacy-preserving learning for player-specific modeling

In this regard, our study aims to be a stepping stone—a hybrid model that not only predicts match outcomes but also contextualizes them using domain logic, player roles, and expert insight, thereby pushing the boundary of traditional match forecasting systems.

Chapter 3

Methodology

3.1 Flowchart

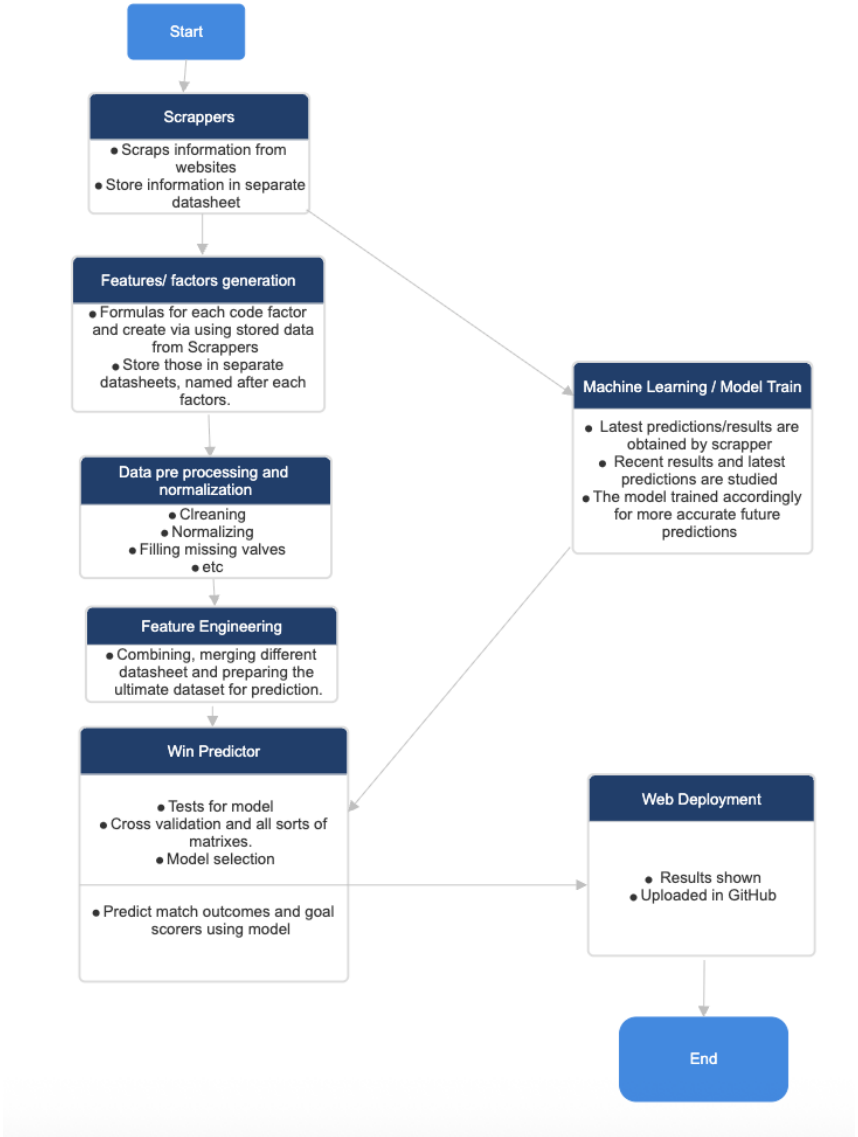


Figure 3.1: An Overview of the whole project

3.1.1 Choice of Predictive Model: Logistic Regression

For the predictive component of our project, we selected logistic regression as the core machine learning algorithm. Logistic regression is particularly well-suited for binary classification tasks, where the outcome is one of two possible categories—in our case, a win or loss for a team. Unlike linear regression, which predicts a continuous numerical outcome, logistic regression outputs probabilities confined within the range of 0 to 1, making it ideal for modeling win probabilities.

The rationale behind this choice is grounded in both the theoretical appropriateness of the model and its interpretability. Logistic regression allows us to model the relationship between one or more independent variables (e.g., team statistics, past performance, player form) and the dependent variable (match outcome) using a logistic function. The model yields not just a binary outcome, but a probability estimate for a team winning, which enhances the decision-making utility of the predictions. These probability scores are especially valuable when integrating the predictions into downstream components, such as a rule-based system that takes action based on certain probability thresholds.

Additionally, logistic regression offers simplicity and speed in training and prediction, which was advantageous in the early stages of our project when the dataset was limited and interpretability was a key concern. The model's coefficients provide a transparent insight into the relative importance of each feature, enabling us to validate our feature engineering decisions more effectively.

3.1.2 Post-Prediction Processing: Rule-Based System Combined with Machine Learning

To further refine the practical application of our prediction model, we implemented a rule-based system that operates in conjunction with the machine learning predictions. This hybrid approach combines the adaptability of machine learning with the domain-specific control provided by rule-based logic.

After each match prediction is made, the rule-based system evaluates the result in the context of a broader set of operational rules. For example, if the predicted win probability exceeds a certain threshold (e.g., 0.75), the system might classify the outcome as a "strong win prediction" and trigger specific downstream processes such as strategy adjustment or resource allocation in a simulated environment. Conversely, predictions with lower confidence may be flagged for further review or subjected to a different set of heuristics.

This architecture provides a robust mechanism for incorporating expert knowledge and business logic into the system, ensuring that predictions are not just statistically sound but also contextually meaningful. It also facilitates system interpretability and auditing, which are often challenging in purely data-driven models.

3.1.3 Initial Design Limitations and Data Collection Challenges

During the initial phase of the project, our design did not include the integration of machine learning in the post-prediction analysis loop. Our primary focus was on generating and recording prediction results for individual matches, rather than developing a learning system that would iteratively improve based on new predictions and outcomes.

As a result, we did not preserve the full set of input features used to make those predictions. Instead, we only stored the final prediction results (e.g., predicted outcome, datasets). This decision has led to a limitation in our current system, as we lack historical data that could have been used to retrain or fine-tune the model based on its past performance. This oversight underscores a common challenge in early-stage machine learning projects: the tension between building a minimal viable system and anticipating future analytical needs. While our current dataset suffices for performance evaluation and rule-based decision making, the lack of feature-level data from past predictions restricts our ability to conduct comprehensive error analysis.

To mitigate this, we have since revised our data pipeline to store complete datasets, including raw inputs, model predictions, and actual outcomes. This will support more advanced model diagnostics, performance monitoring, and potentially real-time retraining in future iterations of the system.

3.2 Why a Rule-Based Predictor System and the Selection of Features

In the domain of football match prediction, combining rule-based systems with machine learning (ML) presents a hybrid framework that captures both domain expertise and data-driven adaptability. This approach is particularly effective when the problem space includes complex, structured logic—such as football-specific rules, match formats, and performance indicators—that may not be easily learnable from data alone.

3.2.1 Initial Strategy and Design Decisions

Initially, our approach was entirely rule-based, developed to encode insights from football analysts and match patterns without any intention to apply machine learning. The primary reason for this choice was to create a lightweight, explainable model that could use intuitive rules for scoring and ranking teams based on metrics like recent performance, team status, and head-to-head encounters.

At that stage, we did not store the full input datasets—only the final prediction results were saved as part of our system's output (used for historical logs or backlog references). This decision, while consistent with the simplicity of a rule-based system, posed challenges when we later decided to integrate machine learning, as we lacked a historical feature-label dataset for supervised training.

Justification for Rule-Based System

In the development of predictive models for football match outcomes, many contemporary approaches rely heavily on purely statistical or machine-learned inferences from structured datasets. However, the complexity and subjectivity of football as a domain call for a more nuanced approach—one that incorporates both quantitative indicators and domain-informed heuristics. This research proposes a rule-based architecture as a foundational layer for prediction, which allows for the encoding of expert knowledge in the form of deterministic, interpretable scoring rules.
The justification for this approach stems from the following factors and the codes also are showcased below. Also, to further back up our selection of the factors, academic journals and research papers from trustworthy authors are also given.

1. **Formalization of Domain Knowledge**: Rule-based systems provide an ideal framework for embedding expert insights that are frequently referenced in commentary and analysis but rarely captured in tabular data. By converting such knowledge into structured scoring logic, the system can consistently reflect real-world dynamics across matches. Below is a detailed description of each rule incorporated into the system.
   Supporting links:

   - https://www.academia.edu/18792266/A_Compound_Framework_for_Sports_Prediction_The_Case_Study_of_Football
   - https://www.ewadirect.com/proceedings/ace/article/view/4782
   - https://www.researchgate.net/publication/325524231_A_Comparative_study_of_Data_Mining_Techniques_on_Football_Match_Prediction

2. **Newly_Teams_Starting_Adjustments**: Newly promoted teams often struggle in the initial rounds of top-flight football due to a lack of experience, weaker squad depth, and transitional difficulties. To capture this:
   Supporting links:

   - https://doi.org/10.1111/j.1740-9713.2016.00916.x
   - https://heiluo.com/article/438
   - https://arxiv.org/abs/2107.08732

   Rule: Assign a -0.5 score penalty to newly promoted teams within their first 5 matchdays. 0.5 score is set by us as its a rule based system and to create a weighted impact from our observation of the workings of the system we were happy to give it 0.5.

Listing 3.1: Newly Promoted Team Adjustment

```
# Extract unique teams from both seasons
teams_2024 = set(season_2024['HomeTeam'].unique()) | set(
    season_2024['AwayTeam'].unique())
```

```
teams_2025 = set(season_2025['HomeTeam'].unique()) | set(
    season_2025['AwayTeam'].unique())

# Identify newly promoted teams
newly_promoted_teams = teams_2025 - teams_2024

# Count how many matches each newly promoted team has played
    so far
match_counts = season_2025.groupby('HomeTeam').size().add(
    season_2025.groupby('AwayTeam').size(), fill_value=0)

# Apply adjustment ONLY for teams with 5 or fewer matches
    played
newly_promoted_adjustment = []
for team in newly_promoted_teams:
    matches_played = match_counts.get(team, 0)
    adjustment = -0.5 if matches_played <= 5 else 0
    newly_promoted_adjustment.append({
        'Team': team,
        'Matches_Played': matches_played,
        'Adjustment': adjustment
    })

# Convert to DataFrame and save to CSV
newly_promoted_df = pd.DataFrame(newly_promoted_adjustment)
newly_promoted_df.to_csv("C:/Users/ASUS/team_data/
    Newly_Promoted_Adjustment.csv", index=False)
```

3. **Bottom_4_Teams_Score_Deduction:**
   Teams occupying the bottom four positions in the league table are more prone to underperformance, low morale, and increased tactical instability.
   Supporting links:

   - https://arxiv.org/abs/2107.08732
   - https://www.olbg.com/blogs/premier-league-goals-points-ratio-analysis

   Rule: Apply a -0.5 deduction to teams currently ranked in the bottom four. 0.5 score is set by us as its a rule based system and to create a weighted impact from our observation of the workings of the system we were happy to give it 0.5.

Listing 3.2: Compute Bottom 4 Team Impact

```
def compute_bottom_4_impact(points_table):
    bottom_teams = points_table.sort_values(by='Pts').head
        (4)['Squad'].tolist()
    impact = [{'Team': team, 'Bottom4_Impact': -0.5} for
        team in bottom_teams]
    df = pd.DataFrame(impact)
    df.to_csv("C:/Users/ASUS/team_data/Bottom_4_Impact.csv",
        index=False)
```

```
    return df

bottom_4_impact_df = compute_bottom_4_impact(points_table)
```

4. **Clean-Sheets** Total Number of clean sheets so far.

   Supporting links:

   - https://www.researchgate.net/publication/378258852_The_Relationship_Between_Anthropometrics_Parameters_and_Clean_Sheets_of_Goalkeepers_in_Elite_European_Football
   - https://www.theguardian.com/football/who-scored-blog/2021/mar/10/want-to-compare-keepers-fairly-take-a-deep-dive-into-their-clean-sheets

Listing 3.3: Compute Bottom 4 Team Impact

```
def compute_clean_sheets(season_data):
    clean_sheets = season_data.groupby('HomeTeam').apply(
        lambda x: (x['FTAG'] == 0).sum()).reset_index()
    clean_sheets.columns = ['Team', 'Clean_Sheets']
    return clean_sheets

clean_sheets_df = compute_clean_sheets(season_2025)
CLEAN_SHEETS_PATH = "C:/Users/ASUS/team_data/Clean_Sheets.
    csv"
clean_sheets_df.to_csv(CLEAN_SHEETS_PATH, index=False)
```

5. **Top 6 Teams**
   Supporting links:

   - https://www.researchgate.net/publication/339179140_Statistical_Analysis_of_Leading_Goal_Scoring_Pattern_of_Europe's_Top_Five_Football_Leagues
   - https://arxiv.org/abs/2211.15734
   - https://arxiv.org/abs/1705.09575
   - https://www.mdpi.com/1999-5903/15/5/174

   These studies collectively underscore the importance of considering team strength, historical performance, and predictive modeling when analyzing football match outcomes. While they may not directly state that defeating a "Top 6" team is a definitive indicator of a team's strength, they provide frameworks and methodologies that can be adapted to assess the significance of such victories within predictive models.
   The Top 6 Teams: Manchester City, Liverpool, Manchester United, Tottenham, Arsenal, and Chelsea

   - **Formula 1:** Non-Top 6 team the goals they scored against in the latest matches against the top 6 will be collected.
     = Total scored goals by a non top-6 team/ 6

- **Formula 2:** If the team itself is a top 6 team, it also calculates the average goals scored in all their past matches against other top 6 teams and adds that to the result.
  = Total scored goals by a top-6 team + average / 6

Listing 3.4: top 6 teams

```
1
2 TOP_6_TEAMS = ["Manchester United", "Chelsea", "Manchester
    City", "Liverpool", "Arsenal", "Tottenham"]
3
4 def get_latest_match_goals(team, top_teams, season_data):
5     """Retrieve goals scored by a team against the top teams
          from the latest match."""
6     matches = season_data[(season_data['HomeTeam'] == team)
         & (season_data['AwayTeam'].isin(top_teams))]
7     matches = matches.append(season_data[(season_data['
         AwayTeam'] == team) & (season_data['HomeTeam'].isin(
         top_teams))])
8     matches = matches.sort_values(by=['Date'], ascending=
         False)
9
10    if not matches.empty:
11        latest_matches = matches.head(1)  # Get latest match
12        goals_for = latest_matches['FTHG'].values[0] if
             latest_matches.iloc[0]['HomeTeam'] == team else
             latest_matches['FTAG'].values[0]
13        return goals_for
14    return 0
15
16 # Compute goals scored against top 6 teams
17 def compute_goals_scored_against_top_teams(season_2025,
    season_2024, season_2023, top_teams):
18    teams = set(season_2025['HomeTeam']).union(set(
         season_2025['AwayTeam']))
19    results = []
20
21    for team in teams:
22        if team in top_teams:
23            filtered_teams = [t for t in top_teams if t !=
                 team]
24        else:
25            filtered_teams = top_teams
26
27        goals_current = get_latest_match_goals(team,
             filtered_teams, season_2025)
28        if goals_current == 0:
29            goals_current = get_latest_match_goals(team,
                 filtered_teams, season_2024)
30        if goals_current == 0:
31            goals_current = get_latest_match_goals(team,
                 filtered_teams, season_2023)
```

```
32
33          if team in top_teams:
34              past_matches = season_2025[(season_2025['
                    HomeTeam'] == team) & (season_2025['AwayTeam'
                    ].isin(filtered_teams))]
35              past_matches = past_matches.append(season_2025[(
                    season_2025['AwayTeam'] == team) & (
                    season_2025['HomeTeam'].isin(filtered_teams))
                    ])
36              past_goals = past_matches['FTHG'].sum() +
                    past_matches['FTAG'].sum()
37              avg_goals = past_goals / max(len(past_matches),
                    1)
38              goals_current += avg_goals
39
40          results.append({"Team": team, "Goals_Against_Top_6":
                goals_current})
41
42      return pd.DataFrame(results)
43
44  # Generate results
45  goals_against_top6_df =
        compute_goals_scored_against_top_teams(season_2025,
        season_2024, season_2023, TOP_6_TEAMS)
46  goals_against_top6_df.to_csv("C:/Users/ASUS/team_data/
        Goals_Against_Top_6.csv", index=False)
```

6. **Ground_Impact_Scores:** Home advantage has long been documented as a significant factor in match outcomes due to familiarity, fan support, and reduced fatigue.
   Link 1.
   Link 2

   Rule:

   - All the home ground score for the home team.
   - All the away ground scores for the away team.

   Formulae:

   - For W = 3, For L = 0, For D = 1.
   - Total result of all the home/away respected grounds.
   - If only 3 or below three are recorded fair score of 0.5

Listing 3.5: Load match data

```
1  import pandas as pd
2  from datetime import datetime
3
4
```

```
TEAM_STATS_PATH_2025 = "C:/Users/ASUS/team_data/season-202425.csv"
NEXT_MATCHES_PATH = "C:/Users/ASUS/Next_Premier_League_Matches.csv"
OUTPUT_PATH = "C:/Users/ASUS/team_data/Team_Ground_Score_By_Match.csv"

# Load full season data
season_df = pd.read_csv(TEAM_STATS_PATH_2025)

# Ensure required columns are present
required_columns = {'HomeTeam', 'AwayTeam', 'FTR'}
if not required_columns.issubset(season_df.columns):
    raise ValueError("Missing required columns in dataset.")

# Load upcoming fixtures
fixtures_df = pd.read_csv(NEXT_MATCHES_PATH)
fixtures_df["Match_Date"] = fixtures_df["Time"].apply(lambda x: x.split(" ")[0])
fixtures_df["Match_Date"] = pd.to_datetime(fixtures_df["Match_Date"] + "2025", format="%d.%m.%Y")
today = datetime.today().date()
fixtures_df = fixtures_df[fixtures_df["Match_Date"].dt.date >= today]

# Initialize dictionaries
home_points = {}
home_games = {}
away_points = {}
away_games = {}

# Accumulate points and games
for _, row in season_df.iterrows():
    home_team = row['HomeTeam']
    away_team = row['AwayTeam']
    result = row['FTR']

    home_points.setdefault(home_team, 0)
    away_points.setdefault(away_team, 0)
    home_games.setdefault(home_team, 0)
    away_games.setdefault(away_team, 0)

    home_games[home_team] += 1
    away_games[away_team] += 1

    if result == "H":
        home_points[home_team] += 3
    elif result == "A":
        away_points[away_team] += 3
    elif result == "D":
        home_points[home_team] += 1
```

```
        away_points[away_team] += 1

# Calculate home and away scores
home_ground_score = {team: (home_points[team] / home_games[team]) if home_games[team] > 3 else 0.5
                     for team in home_points}

away_ground_score = {team: (away_points[team] / away_games[team]) if away_games[team] > 3 else 0.5
                     for team in away_points}

# Normalize scores
all_home_scores = list(home_ground_score.values())
all_away_scores = list(away_ground_score.values())

min_home = min(all_home_scores)
max_home = max(all_home_scores)
min_away = min(all_away_scores)
max_away = max(all_away_scores)

for team in home_ground_score:
    if max_home - min_home > 0:
        home_ground_score[team] = (home_ground_score[team] - min_home) / (max_home - min_home)
    else:
        home_ground_score[team] = 0.5

for team in away_ground_score:
    if max_away - min_away > 0:
        away_ground_score[team] = (away_ground_score[team] - min_away) / (max_away - min_away)
    else:
        away_ground_score[team] = 0.5

# Extract ground score for each team in fixtures
scores = []
for _, row in fixtures_df.iterrows():
    home_team = row['Home Team']
    away_team = row['Away Team']

    home_score = home_ground_score.get(home_team, 0.5)
    away_score = away_ground_score.get(away_team, 0.5)

    scores.append({"Team": home_team, "Ground_Score": home_score})
    scores.append({"Team": away_team, "Ground_Score": away_score})

# Final DataFrame
ground_score_df = pd.DataFrame(scores)
ground_score_df.to_csv(OUTPUT_PATH, index=False)
```

```
print(f" Ground scores for upcoming matches saved to: {OUTPUT_PATH}")
```

7. **Latest_Form_Score:**
   Form is a reflection of short-term consistency and psychological momentum.
   Supporting Links:

   - https://www.ijraset.com/research-paper/implementing-football-match-prediction-system-using-machine-learning
   - https://www.sciencedirect.com/science/article/abs/pii/S0169207018300116

   Rule:

   - Form-score: W = 3, L = 0, D = 1
   - Total count = 15 as it's the maximum a team can get, or sometimes less if its the initial game. Of the season

   Formula= Total Latest Form Scores/ 15

```

def calculate_form_score(form_string):
    if pd.isna(form_string) or form_string.strip() == "":  # If no data, return 0
        return 0

    score = 0
    count = 0  # Track number of available matches

    for char in form_string[-5:]:  # Consider up to last 5 matches (if available)
        if char == 'W':
            score += 3  # Win = 2 points
        elif char == 'D':
            score += 1  # Draw = 1 point
        # Loss (L) contributes 0 points, so no addition needed
        count += 3

    if count == 0:
        return 0  # If no valid matches, return 0
    return score / count  # Average over available matches
```

8. **Total_Player_Ratings:**
   This captures the intrinsic quality of the starting lineup.
   Supporting Links:

   - https://www.sciencedirect.com/science/article/pii/S016920702300033X?utm_source=chatgpt.com

- https://arxiv.org/abs/1802.04987?utm_source=chatgpt.com

Rule:

- Top 11 players' rating for each team from the websites are added, then the total is divided by 11.
- Injured players are opted out

Formula= Addition of all the players rating/ 11

```

# Compute Team Rating by Excluding Injured Players
# Function to extract team name from player info column
def extract_team_name(player_info):
    match = re.search(r'\n(.*?), \d+', player_info)
    return match.group(1) if match else None

# Extract team names
player_stats['Team'] = player_stats.iloc[:, 1].apply(
    extract_team_name)

# Drop rows where team extraction failed
player_stats.dropna(subset=['Team'], inplace=True)

# Ensure Rating column is numeric
player_stats['Rating'] = pd.to_numeric(player_stats['Rating'
    ], errors='coerce')

# Exclude injured players
available_players = player_stats[~player_stats.iloc[:, 0].
    isin(injury_data['Player'])]

# **Sort Players by Rating within each Team (Descending)**
available_players = available_players.sort_values(by=['Team'
    , 'Rating'], ascending=[True, False])

# **Keep only the Top 11 Players per Team**
available_players = available_players.groupby('Team').head
    (11)

# **Compute total team rating from the top 11 players**
team_ratings = available_players.groupby('Team')['Rating'].
    sum().reset_index()
team_ratings.columns = ['Team', 'Total_Team_Rating']

# Save to CSV
team_ratings.to_csv("C:/Users/ASUS/team_data/Team_Ratings.
    csv", index=False)
```

9. **Total_Goals_Scored:**
Goals scored in recent matches are a direct measure of offensive capability.
Supporting links:

   - https://arxiv.org/abs/2105.09881?
   - https://www.researchgate.net/publication/361301886_Model_to_Predict_the_Result_of_a_Soccer_Match_Based_on_the_Number_of_Goals_Scored_by_a_Single_Team

   Rule

   - The number of goals scored so far by the teams.
   - The attacking prowess is measured side by side.

   Goal scored was added from the points table column, once the dataset was made it was merged with it.

```
df = pd.read_csv(POINTS_TABLE_PATH)
if "Team" in df.columns:
    df["Team"] = df["Team"].replace(team_name_mapping)
    df.to_csv(POINTS_TABLE_PATH, index=False)  # Save the
       updated version
if "Squad" in df.columns:
    df["Squad"] = df["Squad"].replace(team_name_mapping)
    df.to_csv(POINTS_TABLE_PATH, index=False)  # Save the
       updated version

# Load datasets
points_table = pd.read_csv(POINTS_TABLE_PATH, usecols=["
   Squad", "GF"])  # Load only 'Squad' and 'GF'
final_dataset = pd.read_csv(FINAL_DATASET_PATH)
match_fixtures = pd.read_csv(MATCH_FIXTURES_PATH, usecols=["
   Home Team", "Away Team"])

# Rename columns for consistency
match_fixtures.rename(columns={"Home Team": "HomeTeam", "
   Away Team": "AwayTeam"}, inplace=True)
points_table.rename(columns={"Squad": "Team", "GF": "
   Goals_Scored"}, inplace=True)  # Rename GF to
   Goals_Scored

#  Merge GF with HomeTeam and AwayTeam
merged_dataset = final_dataset.merge(points_table, left_on="
   HomeTeam", right_on="Team", how="left")
merged_dataset.rename(columns={"Goals_Scored": "
   Home_Team_Goals_Scored"}, inplace=True)
merged_dataset.drop(columns=["Team"], inplace=True)  # Drop
   redundant column

merged_dataset = merged_dataset.merge(points_table, left_on=
   "AwayTeam", right_on="Team", how="left")
```

```
merged_dataset.rename(columns={"Goals_Scored": "
    Away_Team_Goals_Scored"}, inplace=True)
merged_dataset.drop(columns=["Team"], inplace=True)  # Drop
    redundant column
```

10. **LLM_Expert_Opinions:**
Expert commentary often encapsulates intangible insights about tactics, player psychology, and form. Using large language models, these are parsed and converted into structured sentiment data.

Supporting links:

- https://arxiv.org/abs/2012.04380
- https://www.academia.edu/18792266/A_Compound_Framework_for_Sports_Prediction_The_Case_Study_of_Football
- https://www.ewadirect.com/proceedings/ace/article/view/4782
- https://www.researchgate.net/publication/325524231_A_Comparative_Study_of_Data_Mining_Techniques_on_Football_Match_Prediction

Rule:

- Positive mention of a team/player = +1
- Negative = −1
- Strong certainty in tone = Double weight (±2)
- Uncertainty or hedging = Half weight (±0.5)

Note: Codes in section 3.5

11. **HeadtoHead Battles (Last 3 Matches)**

Historical matchups often set a psychological and tactical precedent.
Supporting Link

- https://www.academia.edu/99958116/Predictions_of_Top_5_European_Football_League_Matches?#loswp-work-container

Rule:

- Win = +1
- Draw = 0.5
- Loss = 0
- Uses a weighted system (latest match 50%, 2nd recent match 30%, 3rd one is 20%
- The two teams' weighted scores are normalized to sum up to 1.
- If no historical matches are found, the default score is 0.5 for both.

Listing 3.6: Head 2 head

```

# ==== FILE PATHS (update if needed) ====
TEAM_STATS_PATHS = [
    "C:/Users/ASUS/team_data/season-202223.csv",
    "C:/Users/ASUS/team_data/season-202324.csv",
    "C:/Users/ASUS/team_data/season-202425.csv",
]
#  Load Next Matches Data (Fixtures)
NEXT_MATCHES_PATH = "C:/Users/ASUS/
    Next_Premier_League_Matches.csv"
matches_df = pd.read_csv(NEXT_MATCHES_PATH)

from datetime import datetime


# Extract day & month only (ignore time)
matches_df["Match_Date"] = matches_df["Time"].apply(lambda x
    : x.split(" ")[0])  # Keep only "d.m" part

# Convert to datetime using the current year
matches_df["Match_Date"] = pd.to_datetime(matches_df["
    Match_Date"] + "2025", format="%d.%m.%Y")  # Use 2025 as
    the year

# Get today's date for comparison
today = datetime.today().date()

# Filter only future matches
matches_df = matches_df[matches_df["Match_Date"].dt.date >=
    today]

NEXT_MATCHES_PATH=matches_df
OUTPUT_PATH = "C:/Users/ASUS/team_data/Head_to_Head_Scores.
    csv"
```

Listing 3.7: TEAM NAME MAPPING

```
# ==== TEAM NAME MAPPING (you can add more if needed) ====
TEAM_NAME_MAPPING = {
    "Manchester Utd": "Man United",
    "Nottingham": "Nott'm Forest",
    "Wolves": "Wolves",
    "Newcastle": "Newcastle",
    "West Ham": "West Ham",
    "Tottenham": "Tottenham",
    "Brighton": "Brighton",
    "Man City": "Man City",
    "Sheffield Utd": "Sheffield United",
    "Luton": "Luton",
    # Add more mappings if necessary
}
```

```
15
16 # ==== HELPER FUNCTIONS ====
17
18 def normalize_team_name(name):
19     """Standardize team names based on mapping."""
20     return TEAM_NAME_MAPPING.get(name, name)
21
22 def calculate_match_result(home_goals, away_goals):
23     """Returns 1 if Home wins, 0.5 for draw, 0 for away win.
       """
24     if home_goals > away_goals:
25         return 1, 0  # Home wins
26     elif home_goals < away_goals:
27         return 0, 1  # Away wins
28     else:
29         return 0.5, 0.5  # Draw
30
31 def weighted_scores(results):
32     """Applies weight to last 3 matches: [0.5, 0.3, 0.2]"""
33     weights = [0.5, 0.3, 0.2]
34     scores = np.zeros(2)  # [home, away]
35
36     for idx, (home_score, away_score) in enumerate(results):
37         if idx >= len(weights):
38             break
39         scores[0] += home_score * weights[idx]
40         scores[1] += away_score * weights[idx]
41
42     total = scores.sum()
43     if total == 0:
44         return 0.5, 0.5  # No info fallback
45     return scores[0]/total, scores[1]/total
```

Listing 3.8: LOAD DATA

```
1
2 # Load all past matches
3 past_matches = pd.concat([pd.read_csv(path) for path in
     TEAM_STATS_PATHS], ignore_index=True)
4
5 # Make sure dates are sortable
6 past_matches['Date'] = pd.to_datetime(past_matches['Date'],
     format='%d-%m-%y', errors='coerce')
7
8 # Standardize team names in past matches
9 past_matches['HomeTeam'] = past_matches['HomeTeam'].apply(
     normalize_team_name)
10 past_matches['AwayTeam'] = past_matches['AwayTeam'].apply(
     normalize_team_name)
11
12 # Load upcoming fixtures
13 fixtures = NEXT_MATCHES_PATH
```

```
14
15 # Standardize team names in fixtures
16 fixtures['Home Team'] = fixtures['Home Team'].apply(
       normalize_team_name)
17 fixtures['Away Team'] = fixtures['Away Team'].apply(
       normalize_team_name)
18
19 # ==== PROCESS FIXTURES ====
20
21 output_rows = []
22
23 for idx, row in fixtures.iterrows():
24     home_team = row['Home Team']
25     away_team = row['Away Team']
26
27     # Find past matches between the two teams
28     mask = (
29         ((past_matches['HomeTeam'] == home_team) & (
               past_matches['AwayTeam'] == away_team)) |
30         ((past_matches['HomeTeam'] == away_team) & (
               past_matches['AwayTeam'] == home_team))
31     )
32     h2h_matches = past_matches[mask].sort_values('Date',
           ascending=False).head(3)
33
34     match_results = []
35
36     for _, match in h2h_matches.iterrows():
37         home = match['HomeTeam']
38         away = match['AwayTeam']
39         home_goals = match['FTHG']
40         away_goals = match['FTAG']
41
42         if home == home_team and away == away_team:
43             result = calculate_match_result(home_goals,
                   away_goals)
44         elif home == away_team and away == home_team:
45             # Reverse if home/away switched
46             away_score, home_score = calculate_match_result(
                   home_goals, away_goals)
47             result = (home_score, away_score)
48         else:
49             continue  # should not happen
50
51         match_results.append(result)
52
53     if not match_results:
54         # No previous match found
55         final_home_score, final_away_score = 0.5, 0.5
56     else:
57         final_home_score, final_away_score = weighted_scores
```

```
            (match_results)

    output_rows.append((home_team, final_home_score))
    output_rows.append((away_team, final_away_score))

# Create final DataFrame
output_df = pd.DataFrame(output_rows, columns=['Team', '
    Head_to_Head_Scores'])

# Save
output_df.to_csv(OUTPUT_PATH, index=False)
```

12. **Wins over opponents losses:**
Historical matchups often set a psychological and tactical precedent.
Supporting Links

    - https://www.researchgate.net/publication/385884956_EPL_Points_Prediction_Using_Machine_Learning

    Rule

    (a) Liverpool 29M 21W 7D 1L 70Pts

    (b) Arsenal 29M 16W 10D 3L 58Pts

    (c) Nott Forest 29M 16W 6D 7L 54Pts

    Arsenal vs forest (based on pts the difference is just 4 , for me it's not a significantly impactful)

    However the wins of arsenal over forest loss vs forest wins over arsenals loss is a much more greater indicator

    Arsenal - 16*7- great Forrest - 16*3- low

    Liverpool - 21*3 great Arsenal -16*1- low

    The lower scores are favourite

    The same points could be attended like this

    1)Arsenal 29M 19W 1D 9L 58Pts
    2) Arsenal 29M 16W 10D 3L 58Pts

    If these two teams faced off which one is better Arsenal. For me it's 2.

Listing 3.9: ratio win*loss

```

import pandas as pd

# Load points table and next matches
points_table = pd.read_csv('C:/Users/ASUS/
    PremierLeague_PointsTable_with_Form.csv')
next_matches = NEXT_MATCHES_PATH

# Extract relevant columns
```

```
team_stats = points_table[['Squad', 'W', 'L']].copy()

# Fix team name mappings (if any small name mismatches exist)
team_name_mapping = {
    "Manchester Utd": "Man United",
    "Nottingham": "Nott'm Forest",
    "Wolves": "Wolves",
    "Newcastle": "Newcastle",
    "West Ham": "West Ham",
    "Tottenham": "Tottenham",
    "Brighton": "Brighton",
    "Man City": "Man City",
    "Sheffield Utd": "Sheffield United",
    "Luton": "Luton",
    "Nottingham Forest": "Nottingham",
    "Wolverhampton Wanderers": "Wolves",
    "Brighton and Hove Albion": "Brighton",
    "West Ham United": "West Ham",

}
 # add more mappings if needed

team_stats['Squad'] = team_stats['Squad'].replace(team_name_mapping)

# Create dictionary for easy lookup
wins_dict = dict(zip(team_stats['Squad'], team_stats['W']))
losses_dict = dict(zip(team_stats['Squad'], team_stats['L']))

# Initialize list for storing results
results = []

# Process each match
for idx, row in next_matches.iterrows():
    home_team = row['Home Team']
    away_team = row['Away Team']

    # Check teams exist
    if home_team not in wins_dict or away_team not in wins_dict:
        print(f"Skipping {home_team} vs {away_team} (team not found)")
        continue

    home_wins = wins_dict[home_team]
    home_losses = losses_dict[home_team]
    away_wins = wins_dict[away_team]
    away_losses = losses_dict[away_team]

```

```
    # Calculate scores
    home_score = home_wins * (away_losses if away_losses !=
        0 else 1)
    away_score = away_wins * (home_losses if home_losses !=
        0 else 1)

    results.append({'Team': home_team, 'Score': home_score})
    results.append({'Team': away_team, 'Score': away_score})
```

13. **Point Table Favouritism** Teams positioned in the top tier of the table often enjoy structural advantages and are less likely to experience unexpected defeats.
Supporting link:

    - https://arxiv.org/abs/1705.00885

    Rule:

4	0.6
5–6	0.7
7–9	0.8
10+	0.9

Table 3.1

    - The **better team** (lower-ranked) gets the favoritism score (0.6, 0.7, etc.).
    - The **weaker team** gets the leftover part (1 - favoritism_score).
    - points table favouritism, if team is above by a position of 4 than that team will be favoured.

      For example-

      ```
      1.Liverpool
      4.Brentford
      5.Manchester city
      8. Chelsea
      14. West Ham
      ```

    - Liverpool will be favoured if it plays against Manchester City , Chelsea , West Ham.
    - Manchester City will only be favoured over West Ham.

```

def calculate_favoritism(diff):
    if diff >= 10:
        return 0.9
    elif diff >= 7:
        return 0.8
    elif diff >= 5:
```

```
        return 0.7
    elif diff >= 4:
        return 0.6
    else:
        return 0.5

# Final result storage
team_favoritism = {}

for idx, row in next_matches.iterrows():
    home_team = row['Home Team'].strip()
    away_team = row['Away Team'].strip()

    if home_team not in team_rank or away_team not in
       team_rank:
        print(f" Warning: One of the teams '{home_team}' or
            '{away_team}' not found in points table.
           Skipping.")
        continue

    home_rank = team_rank[home_team]
    away_rank = team_rank[away_team]

    rank_diff = abs(home_rank - away_rank)

    if rank_diff >= 4:
        if home_rank < away_rank:  # Home team is better
            home_favor = calculate_favoritism(rank_diff)
            away_favor = 1 - home_favor
        else:  # Away team is better
            away_favor = calculate_favoritism(rank_diff)
            home_favor = 1 - away_favor
    else:
        home_favor = 0.5
        away_favor = 0.5

    # Save favor scores team-wise
    team_favoritism[home_team] = round(home_favor, 3)
    team_favoritism[away_team] = round(away_favor, 3)

# Now create DataFrame
favoritism_df = pd.DataFrame(team_favoritism.items(),
   columns=['Team', 'Favouritsm_points'])

# Save output
output_path = "C:/Users/ASUS/team_data/
   Team_Favouritsm_Points.csv"
favoritism_df.to_csv(output_path, index=False)
```

14. **Goal difference** Calculates the goal difference

- https://www.ijraset.com/research-paper/implementing-football-match-prediction-system-using-machine-learning (Keyword: Goal Differences)
- https://www.mdpi.com/2571-9394/6/4/57 (Goal difference)

```
1
2  #Gd
3
4  # Load the Premier League Points Table dataset
5  df = pd.read_csv(POINTS_TABLE_PATH)
6
7  # Check if necessary columns exist
8  if 'Squad' in df.columns and 'GD' in df.columns:
9      # Extract relevant columns and rename them
10     gd_df = df[['Squad', 'GD']].copy()
11     gd_df.rename(columns={'Squad': 'Team', 'GD': '
           Goal_Difference'}, inplace=True)
12
13     # Define output file path (same directory as input file)
14     OUTPUT_PATH = "C:/Users/ASUS/team_data/
           Team_Goal_Difference.csv"
15
16     # Save the output
17     gd_df.to_csv(OUTPUT_PATH, index=False)
18
19 else:
20     print("no GD")
```

3.2.2 Benefits of Rule-Based Modeling

The benefits of using this scoring-rule system as the initial predictor framework are multifold:

- **Transparency and Explainability:** Each score assignment is traceable, ensuring that users and analysts can clearly understand how a prediction was derived.
- **Low Data Dependency:** Many rules do not require extensive historical data, making the system more adaptable to new or smaller leagues.
- **Incorporation of Non-Quantitative Factors:** Expert opinions, team morale, and recent disruptions can be encoded meaningfully, which is rarely possible in purely numerical models.
- **Preprocessing for ML Pipelines:** These rule-based features can be used to engineer inputs for machine learning models, enabling the development of a hybrid architecture.

3.2.3 Integration with Machine Learning

Once the rule-based scores are computed for each team per match, these can be used in two ways:

- **As features** in a supervised learning model (e.g., logistic regression, gradient boosting, neural networks).
- **As a fallback prediction system** when data availability or computational constraints make ML usage infeasible.

Why Combine with Machine Learning (Later Phase): As the project evolved, we recognized several limitations in relying solely on rules:

- Rules are deterministic and cannot learn nuanced patterns from data.
- They cannot optimize feature weights based on past results.
- Manually designing and updating rules for all edge cases became increasingly complex.

We thus transitioned toward integrating ML, using the available outputs from the rule-based model as pseudo-labels and generating synthetic training data going forward.
Even though the historical datasets weren't preserved, the hybrid structure allows for ML to learn from both:

- Engineered rule outputs.
- Real-time accumulated features from future predictions.

Rule-Based Feature Engineering and Justification

- In the field of football analytics, rule-based systems remain an essential component in predictive modeling due to their textbftransparency, domain-alignment, and modular interpretability. The features selected for this project are grounded in domain knowledge and encoded through deterministic scoring functions. These features serve either as standalone predictors or as input variables for subsequent machine learning algorithms in a hybrid modeling framework.

Justification Through Literature

- The methodology adopted in this research is rooted in well-established precedents within sports analytics literature. Several key studies validate the use of rule-based frameworks and expert-informed features as part of compound predictive systems.

Compound and Rule-Based Frameworks

- Min et al. (2008) proposed a hybrid architecture combining rule-based reasoning with Bayesian inference in their Football Result Expert System (FRES). Their work, available on

  Academia.edu, demonstrated that incorporating structured domain knowledge significantly improves forecasting accuracy, especially in scenarios with limited training data.

- Similarly, Rosli et al. (2018), in their **comparative study**, evaluated various data mining techniques and found that hybrid approaches combining handcrafted features with statistical models yielded better results than purely data-driven methods.

- Choudhury et al. (2019), in their **arXiv study**, further support this integration by using a human-in-the-loop model to refine fantasy football player performance prediction, thereby aligning with the inclusion of LLM_Expert_Opinions in our system.

Statistical Feature Impact

- Ismail et al. (2022) presented a comprehensive framework for **goal-scoring likelihood prediction**, confirming that incorporating both individual player statistics and situational context (e.g., recent form, player roles) significantly improved performance over conventional models. Their multi-league model aligns with our use of Total_Goals_Scored, Clean_Sheets, and Goal_Difference.

- The article published via **EWA Direct**, expands on this by emphasizing the importance of combining feature-level signals from various domains—including real-time statistics and betting odds—supporting our integration of multiple data points from platforms like **FBref**,**Flashscore**, and **Football-Data.co.uk**.

Integration with Modern Machine Learning

- The current state of predictive analytics emphasizes the fusion of structured features with adaptive learning models, especially with increasing reliance on deep learning and foundation models. Pham and Le (2024), in their **Springer article**, discuss best practices for integrating handcrafted features with scalable ML pipelines.

- Likewise, the work of Tursunbayeva et al. (2024), published in Applied Sciences **(link)**, provides strong evidence that using rule-based encodings for context-aware team statistics (e.g., form, point table position) improves outcome classification performance in ensemble models.

- The adoption of LLMs as expert interpreters—drawing on techniques described in Vaswani et al. (2017)'s seminal Attention is All You Need **(arXiv)** —supports the extraction and scoring of textual insights, such as coach commentary, match previews, and analyst predictions.

3.3 Data sources and Justification

3.3.1 Overview

The success of any predictive model in football analytics is highly dependent on the quality, accuracy, and depth of the input data. For this research, a variety of structured and semi-structured data sources were selected to cover a comprehensive

range of match-level, team-level, and player-level statistics. All data was collected using web scraping techniques and stored in a local relational database for further processing and model ingestion.

3.3.2 Selection of Data Sources

A diverse mix of sources was chosen to provide robust and varied insights into team form, player quality, match outcomes, and tactical trends. These included:

a. Football-Data.co.uk

- Provides historical match results, betting odds, and basic statistics in downloadable CSV format.
- Chosen for its long-term league coverage and standardized structure ideal for Excel parsing.

b. WhoScored.com

- Offers detailed player and team performance data, including ratings, passes, shots, and more.
- Ideal for building Total_Player_Ratings and form-related metrics.

c. Flashscore.co.uk

- Supplies live match details, including lineups, injuries, goals, and minute-by-minute commentary.
- Used to collect next match fixtures.

d. Transfermarkt.com

- Contains player market values, transfers, squad depth, and injury history.
- Used for injury report of players.

e. FBref.com

- Offers xG, xA, pressing stats, and advanced match analytics powered by StatsBomb.
- Used for Getting points table.

3.3.3 Justification of Data Sources

These platforms were chosen based on three primary criteria:

(i) **Accuracy**
All selected sources are widely cited and used by professionals and researchers in the football analytics community. WhoScored and FBref in particular are known for their reliability and alignment with professional-grade scouting and analysis tools.

(ii) **Coverage**
The chosen platforms collectively offer comprehensive coverage across all relevant domains:

- Historical match results (Football-Data.co.uk)
- Team related data (FBref, Flashscore)
- Player performance and fitness (WhoScored, Transfermarkt)
- League-specific structure and dynamics (all sources cover multiple leagues, including EPL, La Liga, Bundesliga, Serie A, and Ligue 1)

(iii) **Availability**
All data is publicly accessible and does not require premium subscriptions. Most data is served either in raw HTML or CSV format, enabling easy automation using Python scraping and parsing tools.

3.3.4 Data Extraction Techniques

Given that many of these sources do not offer direct APIs or structured data downloads (excluding Football-Data.co.uk), custom web scraping scripts were developed using Python. The key tools employed include

(a) **BeautifulSoup (bs4)**
Used to parse HTML responses and extract data from tables, divs, and list elements.

(b) **Requests**
Used to make HTTP GET requests to target web pages.

(c) **Custom Delay and Header Logic**
Implemented to avoid IP bans by mimicking human browsing patterns.

(d) **Excel-Based Storage:**
All scraped data was stored in Excel files, each sheet representing a data entity such as:

- matches.xlsx – match ID, teams, scores, venue, date
- players.xlsx – player name, club, rating, appearances, goals
- form_table.xlsx – recent match outcomes, form indexes
- Experts.xlsx— extracted and scored LLM-based expert opinions

Excel was selected for:

- Easy manual inspection and debugging
- Compatibility with both Python (via pandas) and spreadsheet users
- Lightweight deployment without requiring database setup

(e) **Selenium:**
Used when content loads via JavaScript and can't be seen in raw HTML, through selenium web drivers were used for dynamic pages

3.3.5 Extracted Data and Feature Derivation:

From the above sources, we extracted the following data categories:
Team-Level Data

- Win/Loss/Draw outcomes
- Recent form (last 5 match results)
- Goals scored and conceded
- Goal difference
- Clean sheets
- League position
- Match venue (home/away)

Player-Level Data

- Appearances and minutes played
- Player ratings (e.g., WhoScored average match rating)
- Goals and assists
- Offensive stats (shots per game, dribbles completed, xG)

These features support the construction of key metrics in our model, including

- Total_Player_Ratings
- Total_Goals_Scored
- Latest_Form_Score
- Clean_Sheets
- Ground_Impact_Scores

3.3.6 Challenges and Considerations

While data coverage was generally extensive, a few challenges were encountered:

- Inconsistency in player naming conventions across sources
- Occasional changes to website layouts (requiring scraper updates)
- Lack of historical injury data, which limited dynamic player availability modeling

To mitigate these, we implemented modular scraping functions and maintained regular backups of scraped data.
In summary, the selection of these specific data sources was critical to ensuring the robustness and credibility of the predictive model. The scraping methods developed allowed for the flexible, automated collection of high-resolution football data. These datasets not only supported the rule-based scoring system but also enabled the training of machine learning models using engineered features derived directly from match and player performance contexts.

3.4 Data Cleaning and Formatting

In any machine learning pipeline, the quality of input data plays a pivotal role in determining the accuracy and reliability of the model's predictions. Raw data, especially in the domain of sports analytics, often arrives in an inconsistent and incomplete form, necessitating an extensive data cleaning and formatting process. This step forms the foundation upon which the performance of our logistic regression model and the subsequent rule-based logic is built. The following subsections detail the methods and considerations involved in transforming our raw data into a structured and machine-readable format.

3.4.1 Handling Inconsistencies in Team and Player Names

One of the fundamental challenges in sports data processing is the lack of uniform naming conventions across different data sources. Team names, for instance, may appear in various forms such as "Man Utd", "Manchester United", or "MAN UNITED", depending on the dataset provider. Similarly, player names might include abbreviations, middle names, or suffixes that introduce ambiguity.
To resolve these inconsistencies, we implemented a **name mapping strategy** using a canonical mapping dictionary. This dictionary was manually curated to ensure one-to-one mapping between all known variations and their standard form. For instance, both "Man Utd" and "MAN UNITED" would be mapped to "Manchester United". This mapping was applied systematically across all datasets before further processing. The use of a controlled vocabulary allowed us to merge data from multiple sources accurately, such as performance metrics, transfer data, and historical match results, which is critical for creating reliable features.

```
TEAM_NAME_MAPPING = {
 "Manchester Utd": "Man United",
 "Nottingham": "Nott'm Forest",
 "Wolves": "Wolves",
 "Newcastle": "Newcastle",
 "West Ham": "West Ham",
"Tottenham": "Tottenham",
 "Brighton": "Brighton",
 "Man City": "Man City",
 "Sheffield Utd": "Sheffield United",
 "Luton": "Luton"
,#Add more mappings if necessary
 }
```

3.4.2 Standardizing Date Formats and Structuring Data for Model Consumption

Sports datasets often contain date fields that vary in format, including MM/D-D/YYYY, DD-MM-YYYY, or textual formats such as "12th May 2023." Such variability can lead to parsing errors and inconsistencies when aggregating data across timelines. To mitigate this, we standardized all date formats using ISO 8601 convention (YYYY-MM-DD), which is widely supported in data science libraries and facilitates chronological sorting, grouping, and filtering operations.

Once standardized, the temporal component of the dataset was leveraged to generate time-series features, such as rolling averages of performance indicators over the last n matches, time since last match, and form trends. Data was then restructured into a match-level format, where each row represented a unique match instance, and columns included both static information (e.g., team names, venue) and dynamic features (e.g., recent form scores, injury status).
This structured tabular format was essential for compatibility with scikit-learn's logistic regression implementation, which expects data in a matrix (X, y) structure, where X contains the feature vectors and y contains the target labels (match outcomes).

3.4.3 Strategies for Imputing Missing Values in Datasets (Data Scrubbing)

In real-world datasets, missing values are inevitable—particularly in sports data, where player statistics, injury reports, or newly promoted teams may lack historical records. These missing values can introduce bias and reduce the effectiveness of the learning algorithm if not addressed properly. Our approach to imputation depended on the type and context of the missing value:

- **Numerical features** (e.g., average goals scored) were imputed using Constant value(0) , as the types of values we dealt with were critical to have values present rather than empty values.With constant values the results would not have been any different.
- **For entire missing rows or severely incomplete records**, especially those that lacked core features required by the model, we excluded them from the dataset to avoid skewing the model with potentially misleading data.

3.4.4 Normalization Techniques Applied to Ensure Data Consistency

To ensure numerical features contributed uniformly to the logistic regression model—particularly since logistic regression is sensitive to feature scale—we applied **normalization techniques**. These included:

- **Min-max scaling**, which transformed features to fall within a 0–1 range. This was particularly useful for bounded statistics such as possession percentages or pass accuracy.

- **Z-score Standardization** was applied to features like goal difference or recent performance ratings, ensuring they had zero mean and unit variance. This was critical when combining features derived from distributions with very different scales and variances.
- **Log transformations** were used to compress the scale of skewed features such as total shots or attendance figures, which often have long-tailed distributions.

  Normalization allowed the model to converge more quickly and improved the interpretability of feature coefficients, as no single feature dominated due to magnitude alone.

3.5 Data Collection and Analysis Pipeline for Expert Opinion Mining on Football Matches

(1) **Introduction**
This project develops a comprehensive pipeline designed to collect, transcribe, scrape, analyze, and quantify expert opinions regarding football teams, based on both multimedia content (YouTube videos) and textual match reports. The system combines Natural Language Processing (NLP), Web Scraping, and Large Language Models (LLMs) to produce a structured dataset of expert sentiments toward football clubs.

(2) **Package Installation and Environment Setup**
To facilitate the tasks of downloading multimedia content, processing text, and deploying large models, several key packages were installed:

- Whisper (Radford et al., 2022) for automatic speech recognition.
- yt_dlp (yt-dlp contributors, 2023) for advanced YouTube downloading.
- Transformers library (Wolf et al., 2020) from HuggingFace, providing access to pretrained LLMs.
- BitsAndBytes (Dettmers et al., 2022) for efficient 4-bit quantization of large models.
- Requests (Kenneth Reitz, 2022) and BeautifulSoup4 (Richardson, 2007) for HTTP requests and HTML parsing.

(3) **Loading and Configuring the Language Model**
The Meta LLaMA-3 (8B) model (Meta AI, 2024) is deployed using the HuggingFace transformers pipeline:

- Authentication is performed using a HuggingFace Token.
- Memory-efficient 4-bit quantization (nf4) is applied.
- A text-generation interface is established with a token output limit of 1000.

This setup allows handling an otherwise computationally heavy model on standard hardware configurations.

(4) **Audio Collection and Transcription from YouTube** The pipeline captures verbal expert opinions by:

- Downloading the first video from a provided YouTube playlist.
- Extracting high-quality audio and converting it into .wav format via FFmpeg.
- Transcribing the audio into text using Whisper's base model.

The resulting transcripts are appended to a file titled analyze.txt, building a repository of expert commentary.

(5) **Web Scraping of Written Match Reports**
To supplement the verbal data:

- Six match report URLs are extracted from The Guardian's Premier League results page.
- Full article texts are scraped, specifically targeting the HTML elements.

Each article's text is appended to analyze.txt, resulting in a unified corpus of expert football opinions from multiple modalities.

(6) **Preparing the Data for Analysis**
The compiled data is analyzed using prompt-based large language modeling:

- A carefully engineered prompt instructs the LLaMA-3 model to:
  - Award +1 points for positive mentions.
  - Award -1 points for negative mentions.
  - Apply scoring based on team, player, or coach references.
- The model's response is strictly formatted into team-score pairs, written into test.txt. Prompt engineering ensures a structured, machine-readable output.

(7) **Extracting and Storing Expert Scoring**
The generated scoring data undergoes further processing:

- Each team's scores are aggregated, starting from a base score of 0.4.
- Empty or malformed entries are carefully handled.
- Final scores are normalized between 0 and 1, to ensure fair comparison even if the scoring range is narrow.
- The final results are saved into expert_opinion_scores.csv, sorted alphabetically by team names.

3.5.1 Justification from Existing Literature

The integration of expert opinions and domain knowledge into predictive models has been explored in various studies:

- Compound Frameworks Combining Rule-Based Reasoning and Bayesian Inference: Choe and McKay (2009) proposed a hybrid model that integrates rule-based reasoning with Bayesian networks for sports prediction. Their framework, known as FRES (Football Result Expert System), demonstrates that combining qualitative expert knowledge with statistical methods can enhance prediction accuracy, particularly when dealing with limited data or novel scenarios.

- Incorporating Domain Knowledge into Machine Learning Models: Song (2023) emphasized the importance of integrating domain-specific knowledge into machine learning models for soccer prediction. By combining statistical data with expert insights, the study achieved improved predictive performance, highlighting the value of qualitative information in understanding complex sports dynamics.

- Comparative Studies on Data Mining Techniques: Rosli et al. (2018) conducted a comparative study on various data mining techniques for football match prediction. While statistical models like decision trees showed high accuracy, the study acknowledged that incorporating expert opinions could further refine predictions, especially in cases where quantitative data alone might be insufficient.

These studies collectively underscore the potential benefits of integrating expert opinions into predictive models, particularly in the context of football match outcomes.

3.5.2 Methodology for Extracting Expert Opinions Using LLMs

Our approach involves the following steps:

(1) **Data Collection:** We gather textual data from reputable sources, including match previews, expert analyses, and commentary articles.

(2) **LLM Processing:** Using advanced LLMs, we process the collected texts to extract sentiments, key insights, and qualitative assessments related to team performance, player conditions, tactical approaches, and other relevant factors.

(3) **Quantification:** The extracted qualitative information is then quantified into numerical scores, representing the perceived strengths, weaknesses, and overall outlook for each team.

(4) **Integration into Predictive Model:** These quantified expert opinions are incorporated as additional features into our predictive model, complementing traditional statistical variables.

3.5.3 Advantages of the LLM-Based Approach

- **Enhanced Contextual Understanding:** LLMs can interpret complex narratives and contextual cues that are often missed by traditional models.

- **Adaptability:** The model can quickly adapt to new information, such as sudden player injuries or tactical changes, by processing updated textual data.
- **Complementarity:** Combining quantitative data with qualitative insights provides a more holistic view of potential match outcomes.

3.5.4 Limitations and Considerations

While the integration of LLM-derived expert opinions offers several advantages, it is essential to acknowledge potential limitations:

- **Bias in Source Material:** The quality and objectivity of the textual data can influence the extracted insights. Efforts must be made to source information from diverse and reputable outlets.
- **Interpretation Challenges:** LLMs may misinterpret sarcasm, idioms, or culturally specific references, leading to inaccurate sentiment analysis.
- **Computational Resources:** Processing large volumes of textual data with LLMs requires significant computational power and resources.

3.5.5 Summary

Integrating LLM-based expert opinions into football match prediction models offers a promising avenue for capturing the qualitative nuances of the game. By combining these insights with traditional statistical data, we aim to develop a more robust and context-aware predictive framework. This hybrid approach aligns with contemporary research advocating for the fusion of domain knowledge and machine learning techniques to enhance predictive accuracy in sports analytics. This system achieves an automated end-to-end expert opinion mining pipeline integrating:

- Multimodal data sources (audio and text),
- Large Language Models for qualitative-to-quantitative transformation,
- Normalized sentiment scores for comparative analysis.

This methodology can be extended to multiple domains including politics, marketing, and media sentiment analysis, with minimal modifications.

(1) **Overall Flowchart**

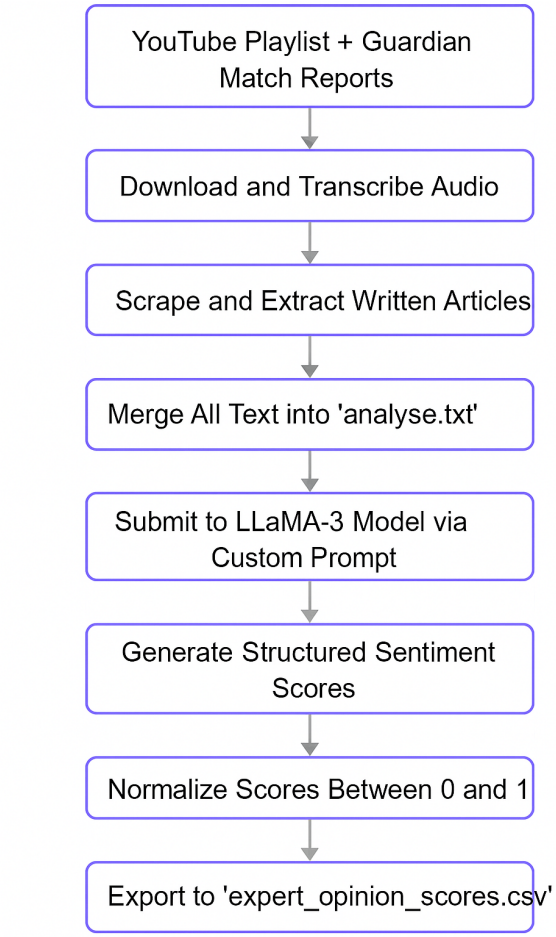
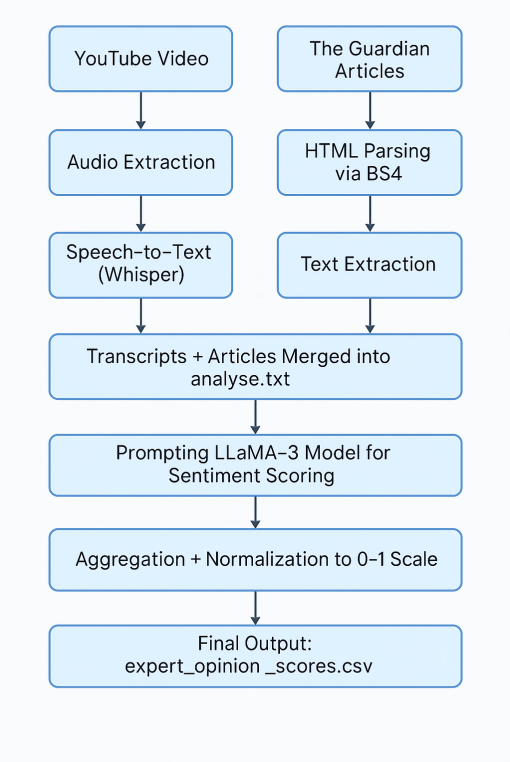


Figure 3.2: Overall Flowchart

Detailed Data Processing Pipeline



(2) **Quantization Techniques for Efficient Large Language Model Deployment**

Introduction

Quantization is a model optimization technique that reduces the precision of the numbers used to represent model parameters. Instead of using the standard 32-bit floating-point format (FP32), quantization uses lower precision formats such as 8-bit (INT8), 4-bit, or even binary representations.
The main goals of quantization are to-

- **Reduce model size.**
- **Decrease memory bandwidth requirements.**
- **Accelerate inference time.**
- **Enable deployment on resource-constrained devices** (e.g., consumer-grade GPUs or CPUs).

4-bit Quantization and BitsAndBytes
In this project, we employed 4-bit quantization using the BitsAndBytes library (Dettmers et al., 2022). Specifically, the following techniques were used:

- **NF4 Quantization (Normal Float 4-bit):**
  A highly efficient quantization scheme that preserves the dynamic range

of weights better than traditional uniform quantization. NF4 maps values based on a non-uniform floating-point distribution, leading to better approximation with fewer bits.

- **Double Quantization:**
  Instead of quantizing the full precision weights directly, an intermediate quantization is applied. First, a lower- precision intermediate representation is computed, and then that is quantized. This two-stage process improves accuracy compared to direct single-step 4-bit quantization.
- **bfloat16 Computation (bfloat16 dtype):**
  Computations involving activations and attention mechanisms are kept in bfloat16 (16-bit floating point), preserving higher numerical stability without the heavy memory costs of full FP32 precision.

Benefits Observed

- **Memory Efficiency:**
  Loading the LLaMA-3 8B model in 4-bit reduced VRAM usage drastically, allowing inference on consumer GPUs with less than 24GB memory.
- **Speed Improvements:**
  Inference was approximately 2x faster compared to running in full 16-bit precision.
- **Negligible Accuracy Loss:** Through NF4 and double quantization techniques, the degradation in model performance (measured qualitatively through generated outputs) was almost negligible.

Why Quantization Was Necessary

Given the project's requirements for processing large text datasets and generating complex sentiment analyses using a massive 8B parameter model, 4-bit quantization was essential to:

- Avoid hardware memory limitations.
- Achieve reasonable processing times.
- Ensure economic usage of computational resources.

Without quantization, running a model of this size would have required expensive infrastructure (such as A100 or H100 GPUs), which are often inaccessible for academic projects.

LLM CODES

```

!pip install -r requirements.txt
!pip install openai-whisper
!pip install yt_dlp
!pip install yt_dlpimport json
!pip install requests beautifulsoup4


import torch
from transformers import (AutoTokenizer,
                          AutoModelForCausalLM,
                          BitsAndBytesConfig,
                          pipeline)

import json
config_data = json.load(open("config.json"))
HF_TOKEN = config_data["HF_TOKEN"]


model_name = "meta-llama/Llama-3.1-8B-Instruct"

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)

tokenizer = AutoTokenizer.from_pretrained(model_name,
                                          token=HF_TOKEN)

tokenizer.pad_token = tokenizer.eos_token
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    device_map="auto",
    quantization_config=bnb_config,
    token=HF_TOKEN
)

text_generator = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    max_new_tokens=1000
)

def get_response(prompt):
  sequences = text_generator(prompt)
  gen_text = sequences[0]["generated_text"]
  return gen_text
```

```
51
52 import yt_dlp
53 import whisper
54
55 # YouTube playlist link
56 playlist_url = "https://www.youtube.com/watch?v=pIzVPMjMSRA&
   list=PLQ_voP4Q3cfe0noETSQtcEA-F9d8OLC-3"
57
58 def downloader(url):
59     ydl_opts = {
60         'format': 'bestaudio/best',
61         'playlistend': 1,  # Only download the latest 2
            videos
62         'postprocessors': [{
63             'key': 'FFmpegExtractAudio',
64             'preferredcodec': 'wav',
65             'preferredquality': '192',
66         }],
67         'outtmpl': 'ytdl/%(title)s.%(ext)s',
68     }
69
70     with yt_dlp.YoutubeDL(ydl_opts) as ydl:
71         info = ydl.extract_info(url, download=True)
72
73         for entry in info.get("entries", []):
74             file_path = ydl.prepare_filename(entry).rsplit("
                .", 1)[0] + ".wav"
75
76             # Transcribe using Whisper
77             model = whisper.load_model("base")
78             result = model.transcribe(file_path)
79             print(f"Transcription for {entry['title']}:\n{
                result['text']}\n")
80
81             # Save transcript
82             with open("analyse.txt", "a") as f:
83                 f.write(f"{entry['title']}:\n{result['text
                    ']}\n\n")
84
85 downloader(playlist_url)
86 print("Download and Transcription Complete!")
87
88 import requests
89 from bs4 import BeautifulSoup
90
91 # URL of The Guardian's Premier League results page
92 url = "https://www.theguardian.com/football/premierleague/
   results"
93
94 # Send a GET request
95 headers = {"User-Agent": "Mozilla/5.0"}
```

```
response = requests.get(url, headers=headers)
soup = BeautifulSoup(response.text, "html.parser")

# Find match redirect links
match_links = []
for a_tag in soup.find_all("a", href=True):
    if "https://football.theguardian.com/match-redirect/" in a_tag["href"]:
        match_links.append(a_tag["href"])

    # Stop after collecting 6 links
    if len(match_links) == 6:
        break

# Print the extracted links
for link in match_links:
    print(link)

# Save links to a file
with open("match_links.txt", "w") as file:
    for link in match_links:
        file.write(link + "\n")

print(f"Extracted {len(match_links)} match report links. Saved to 'match_links.txt'.")

# Extract article content from each match link and append to analyse.txt
with open("analyse.txt", "a") as analyse_file:
    for link in match_links:
        # Send a GET request to the match report link
        response = requests.get(link, headers=headers)
        match_soup = BeautifulSoup(response.text, "html.parser")

        # Find the article content (assuming it's in a <div> with class 'article-body')
        article_body = match_soup.find("div", class_="article-body")
        if article_body:
            article_text = article_body.get_text(separator="\n", strip=True)
            # Append the article text to the analyse.txt file
            analyse_file.write(f"--- Article from {link} ---\n")
            analyse_file.write(article_text + "\n\n")
        else:
            analyse_file.write(f"Could not extract article from {link}\n\n")

```

```
print(f"Articles from {len(match_links)} match links have been appended to 'analyse.txt'.")

import requests
from bs4 import BeautifulSoup

# Read the links from the text file
with open("match_links.txt", "r") as file:
    links = file.readlines()

# Function to scrape the content of each link
def scrape_page(url):
    try:
        response = requests.get(url)
        response.raise_for_status()  # Check if the request was successful
        soup = BeautifulSoup(response.text, "html.parser")

        # Assuming the content you want is inside a specific tag, e.g., <article>
        # You can modify this based on the structure of the page you're scraping
        content = soup.find('article')  # Or modify this to suit your needs

        if content:
            return content.get_text(strip=True)
        else:
            return "No relevant content found"
    except requests.exceptions.RequestException as e:
        return f"Error while scraping {url}: {str(e)}"

# Open the analyse.txt file in append mode
with open("analyse.txt", "a") as output_file:
    # Scrape and append content for each link
    for link in links:
        link = link.strip()  # Remove any leading/trailing whitespace
        content = scrape_page(link)
        output_file.write(content + "\n\n")  # Append the content followed by a newline for separation

with open("analyse.txt", "r") as file:
    #   prompt = "Your task is to evaluate which team experts favor based on their statements. Follow these rules when scoring teams:\n\nScoring Criteria:\n\nIf a team is mentioned positively (praise, strengths, good form) → +1\nIf a team is mentioned negatively (weaknesses, bad form, criticism) → -1\nIf a player or coach is praised → +1 for the team\nIf a player or coach is criticized → -1 for the team\nConfidence
```

```
        Weighting:\n\nIf the expert expresses strong
        certainty (e.g., "They will definitely "win) → Double
         weight (+2 or -2)\nIf the expert is uncertain or
        hesitant (e.g., "They might have a "chance) → Half
        weight (+0.5 or -0.5)\nFormatting:\n\nOutput only
        scores in the format:\nmanchester +1\ntottenham -1\
        nliverpool +2\narsenal -1\nDo not provide
        explanations, summaries, or extra text.\n### Example
        Input:\n\"Manchester United is in great form, and
        Rashford has been excellent. However, their defense
        looks shaky. I believe they have a high chance of
        winning.\"\n\n### Expected Output:\n\nmanchester +1\
        nmanchester +1\nmanchester -1\nmanchester +2\n" +
        file.read()
174     prompt = 'read the text below and make a score of every
        team mentioned.\nif a team is mentioned in a good way
         give it 1 point,otherwise -1 point, if a player or
        even a coach is mentioned in a good way, give 1 point
        ,otherwise -1 point.Strong certainty in tone = Double
         weight (±2).Uncertainty or hedging = Half weight (±0
        .5)\n\ndo this for all the teams mentioned below and
        output like this only:\n\nmanchester +1\n\nmanchester
         +1\n\ntottenham +1\n\nmanchester -1 and so on.\n\ndo
         not write anything exept the scoring.' + file.read()
175
176 llama3_response = get_response(prompt)
177
178 with open("test.txt", "w") as f:
179   f.write(llama3_response[len(prompt):])
180
181
182 import csv
183 from collections import defaultdict
184
185 def calculate_scores(input_file, output_file, base_score
    =0.4):
186     scores = defaultdict(lambda: base_score)  # Set base
        score to 0.4 for each team
187
188     # Read and calculate scores
189     with open(input_file, "r") as file:
190         for line in file:
191             parts = line.strip().split()
192             if len(parts) == 2:
193                 team, score_str = parts
194
195                 # Handle '+' and '-' signs
196                 if score_str.startswith("+"):
197                     # Check if score_str[1:] is not empty
198                     if score_str[1:]:
199                         score = int(score_str[1:])
```

```
                    else:
                        score = 0  # or handle empty score as needed
                elif score_str.startswith("-"):
                    # Check if score_str[1:] is not empty
                    if score_str[1:]:
                        score = int(score_str[1:]) * -1
                    else:
                        score = 0  # or handle empty score as needed
                else:
                    # Check if score_str is not empty and numeric
                    if score_str.isdigit():
                        score = int(score_str)
                    else:
                        score = 0  # or handle invalid score as needed
                scores[team] += score

    # Normalization
    all_scores = list(scores.values())
    min_score = min(all_scores)
    max_score = max(all_scores)

    # Handle cases where all scores are the same
    if min_score == max_score:
        normalized_scores = {team: 0.5 for team in scores}  # Assign a neutral score
    else:
        normalized_scores = {
            team: (score - min_score) / (max_score - min_score)
            for team, score in scores.items()
        }

    # Write to CSV
    with open(output_file, "w", newline="") as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(["Team", "Experts Opinion Score"])
        for team, score in sorted(normalized_scores.items()):
            writer.writerow([team, score])

calculate_scores("test.txt", "expert_opinion_scores.csv")
print("CSV file created successfully!")
```

(3) **Limitations**

Despite the successful implementation and functionality of the expert opinion mining pipeline, this project is subject to a number of limitations that affect the accuracy, scalability, and generalizability of the results. These limitations span multiple domains, including data quality, model constraints, computational resources, linguistic nuances, and evaluation methodology.

(1) **Data Source Limitations**

(1.1) Restricted Dataset Size
The pipeline currently processes a limited number of inputs:

- One video from a YouTube playlist.
- Six written match reports from The Guardian.

This small dataset size limits the statistical power of the analysis and increases the influence of outliers. A single opinionated journalist or heavily biased commentary in a video may disproportionately affect the final sentiment scores.

(1.2) Domain-Specific Source Bias

All textual data is sourced from The Guardian, which may have implicit biases in coverage, tone, or focus towards specific teams or regions. Similarly, the YouTube video selected may reflect the opinions of a narrow set of commentators.

These biases can skew the sentiment scoring in favor or against particular teams, reducing generalizability to other leagues, platforms, or audiences.

(1.3) No Verification of Expertise

While the system attempts to mine "expert" opinion, it does not verify the credentials or authority of the speakers or writers. As a result, any strongly opinionated content may be treated as expert sentiment regardless of its legitimacy.

(2) **Model and Prompt Limitations**

(2.1) Reliance on Prompt Engineering

The output of the LLaMA-3 model is heavily dependent on the prompt structure. Small changes in prompt wording can lead to drastically different outputs. This introduces:

- Lack of reproducibility.
- Sensitivity to prompt phrasing.
- Challenges in scaling the approach to more complex or multilingual data.

(2.2) Hallucinations and Misinterpretations

Despite its capabilities, LLaMA-3 can hallucinate information or misinterpret sarcastic, metaphorical, or context-dependent language. For instance:

- A sarcastic comment like "That was brilliant defending!" after a blunder might be scored positively.
- Praise for a player of one team during a discussion of another might be incorrectly attributed.
  These issues can lead to inaccurate or noisy scoring results.

(2.3) Model Token Limitations

The model is restricted to generating a fixed number of tokens (max_new_tokens = 1000). If the combined text input is too long, truncation or overflow could result in incomplete analysis or dropped content, particularly problematic for longer transcripts or article sets.

(3) **Computational Constraints**

(3.1) Hardware Limitations

Although 4-bit quantization significantly reduces resource requirements, LLaMA-3 (8B) still demands considerable GPU memory and compute time. This restricts:

- Real-time usage or deployment.
- Running on devices without GPU acceleration.
- Handling of very large datasets.

Batch processing or parallel model inference was not implemented, which limits throughput.

(3.2) Inference Cost

While quantization helps, loading and running large language models still incurs time and energy costs that may not be feasible in all environments, especially for iterative testing or larger datasets.

(4) **Linguistic and Sentiment Complexity**

(4.1) Contextual Ambiguity

The sentiment of a statement is often context-dependent, requiring nuanced understanding:

- E.g., "Liverpool struggled defensively but showed strong spirit."
  - Should this result in a +1 or -1?
- LLaMA-3 may not weigh both aspects equally unless explicitly prompted.

(4.2) Scoring Simplicity

The scoring mechanism assigns flat scores (+1, -1, +2, etc.) without nuanced weighting for:

(5) **Evaluation Limitations**

5.1. No Ground Truth Comparison

The project does not benchmark its outputs against a gold standard or expert-annotated dataset. This makes it difficult to:

- Quantitatively measure accuracy.
- Identify false positives/negatives.
- Validate model behavior across diverse input styles.

5.2. Lack of Human Feedback Loop

There is no mechanism for users or domain experts to validate or correct the model's outputs. This limits trust and iterative improvement, especially for subjective tasks like sentiment analysis.

(6) **Generalizability and Transferability**

6.1. Domain Dependence

The system is highly tailored to the football commentary domain and may not generalize well to other domains (e.g., politics, economics, medicine) without:

- Prompt redesign.
- New domain-specific datasets.
- Updated scoring rules.

6.2. Language Support

All inputs and prompts are in English. Applying this system to multilingual media content would require retraining or using translation pipelines, both of which introduce new challenges and potential loss of sentiment fidelity.

(7) **Summary of Limitations**

Category	**Description**
Data Bias	Narrow sources may introduce bias
Model Sensitivity	Output depends on prompt precision
Sentiment Ambiguity	Difficulty in handling complex or sarcastic language
Computational Expense	Requires non-trivial compute despite quantization
Evaluation Gaps	No accuracy benchmarking or expert validation
Transferability	Limited generalizability to other domains or languages

(8) **Mitigation and Future Work**

While these limitations constrain current system performance, future iterations could address them by

- Integrating a feedback mechanism for human validation.

- Expanding the dataset across platforms and journalists.
- Incorporating multi-turn conversations or context tracking in prompt design.
- Evaluating results against annotated datasets or matching outcomes.
- Fine-tuning smaller domain-specific models to reduce cost and increase specialization.

3.6 Training Pipeline to Build the Machine Learning Algorithm

The development of a robust and adaptive machine learning model requires not only the construction of an effective algorithm but also a well-defined and continuously evolving training pipeline. In our match prediction system, the training pipeline encompasses the entire workflow from data ingestion and preprocessing to model retraining and validation. This section outlines how we designed a dynamic training pipeline that incorporates both historical data and newly available match outcomes to iteratively improve the model over time.

3.6.1 Initial Model Training Using Historical Data

Our training pipeline initially began with a one-time ingestion of historical match data, which included various engineered features such as team form, head-to-head records, home advantage. These data points were cleaned, standardized, and used to train a logistic regression model.
This historical training phase allowed us to construct a baseline model, which provided initial predictions on upcoming matches. However, at this stage, our approach was primarily static—we trained the model once and applied it without a mechanism to update based on new match outcomes.

3.6.2 Integration of Real-Time Match Results

As we began generating predictions and evaluating them against real-time match outcomes, it became evident that a static model would not suffice for ongoing accuracy. Sports contexts are dynamic, with team performance, roster changes, and tactical strategies evolving constantly. To keep our model relevant, we needed a mechanism to integrate recent results back into the training dataset and retrain the model periodically.
To achieve this, we modified our training pipeline to include a feedback loop, whereby the predicted outcomes were compared against actual results after each match. These newly labeled samples (i.e., input features + ground truth results) were then appended to the dataset and marked for inclusion in the next training cycle. This process transformed our model into a semi-online learning system—not fully real-time, but continuously improving as new data became available.

3.6.3 Workings of the pipeline

Overview of the pipeline:

- Backup the current prediction dataset.
- Integrate new match results into the historical training dataset.
- Retrain the model using the full updated dataset.
- Save the newly trained model to be used in future predictions.

The weekly pipeline begins by safeguarding the data that drive our predictive system. Each time it runs, the very first operation it performs is to duplicate the Final_Cleaned_Dataset.csv file, which contains the engineered features for the coming round of fixtures. Rather than renaming or relocating the live file, the pipeline simply creates a carbon copy of it in a dedicated backups directory, appending the current calendar date to the filename so that every snapshot is preserved in chronological order. This practice delivers two indispensable benefits: it gives us a complete, immutable audit trail of every feature set the model has ever seen, and it guarantees we can revert to any previous week's state should data corruption, accidental deletion, or retrospective analysis require a roll-back. Because the original file remains untouched and in place, downstream scripts that expect to find Final_Cleaned_Dataset.csv at its usual path never break, while we still enjoy the security of versioned backups. Immediately after the backup is secured, the pipeline turns its attention to the new information that has just become available: the scores and results from the latest round of matches. These reside in season-202425.csv, a raw match-log that is updated as soon as every fixture concludes. The pipeline scans this file to identify only those rows that were not present the last time it ran—effectively, the matches of the most recent game-week. For each fresh row it retrieves the final score, the full-time result code, and any other metadata we require. It then merges these outcomes onto the corresponding feature rows from the backup taken the week before, producing a fully labelled record in which inputs (team ratings, form indicators, head-to-head scores, and the like) now sit side-by-side with the ground-truth label that tells us who actually won. The newly labelled rows are appended to Historical_Features.csv, a single, ever-growing table that contains every fixture the model has encountered throughout the season. With this enhanced dataset in hand, the pipeline proceeds to model retraining. All numeric feature columns are standard-score normalised through an embedded StandardScaler, after which a multinomial LogisticRegression classifier is fitted on the entirety of Historical_Features.csv. During this phase the pipeline also runs a stratified k-fold cross-validation routine so it can report out-of-sample accuracy, macro F1, log-loss and Brier scores—metrics that reveal both discrimination power and calibration quality. Finally, once training and validation are complete, the pipeline serialises the freshly learned pipeline—scaler plus classifier—into a date-stamped .joblib artefact and stores it in the models directory under a filename such as logreg_20250701.joblib. This snapshot is the only object consumed by the prediction engine that produces probabilities for forthcoming fixtures. By decoupling training from prediction and versioning every model with an explicit timestamp, we guarantee both reproducibility and transparency: any set of probabilities we emit can always be traced back to the precise model—and thus the exact historical data—responsible for generating them. Down-stream, the prediction script simply loads the most recent joblib file, pairs it with the current Final_Cleaned_Dataset.csv, and outputs calibrated win, draw, and loss probabilities

for the matches yet to be played. In sum, the pipeline's entire purpose is to prepare pristine data, nurture an ever-richer training corpus, and deliver an up-to-date model, leaving the act of prediction to a separate, purpose-built component.

File	Purpose
Final_Cleaned_Dataset.csv	Feature dataset for predicting the **next round of matches**
Historical_Features.csv	Cumulative dataset of all **past matches** + **features** + **results**
backups/Final_Cleaned_Dataset_*.csv	Archived weekly snapshots for rollback or auditing
models/logreg_*.joblib	Time-stamped machine learning models used for prediction

Table 3.2: Dataset Files and Their Purposes

The code (for the training model):

```
1
2  from pathlib import Path
3  from datetime import date
4  import pandas as pd
5  import numpy as np
6  import joblib
7  import shutil
8  from sklearn.pipeline import Pipeline
9  from sklearn.preprocessing import StandardScaler
10 from sklearn.linear_model import LogisticRegression
11 from sklearn.model_selection import StratifiedKFold,
       cross_val_predict
12 from sklearn.metrics import (accuracy_score, f1_score,
13                               log_loss, brier_score_loss)
14
15 ROOT          = Path("C:/Users/ASUS/team_data")
16 RAW_RESULTS   = ROOT / "season-202425.csv"              #
       finished matches
17 FEATURE_FILE  = ROOT / "Final_Cleaned_Dataset.csv"     #
       -nextweek fixtures
18 MASTER_STORE  = ROOT / "Historical_Features.csv"       # grows
       every run
19
20 MODEL_DIR     = ROOT / "models"
21 PRED_DIR      = ROOT / "predictions"
22 METRIC_LOG    = ROOT / "metrics/metric_log.csv"
23 BACKUP_DIR    = ROOT / "backups"
24
25 for d in (MODEL_DIR, PRED_DIR, BACKUP_DIR, METRIC_LOG.parent):
26     d.mkdir(parents=True, exist_ok=True)
27
28 today_str = date.today().strftime("%Y%m%d")
29
```

```
30
31
32 if FEATURE_FILE.exists():
33     backup_path = BACKUP_DIR / f"Final_Cleaned_Dataset_{
       today_str}.csv"
34     shutil.copy(FEATURE_FILE, backup_path)           #  COPY,
       'dont move
35     print(f"Backup created → {backup_path}")
36 else:
37     raise FileNotFoundError(f"{FEATURE_FILE} not found - generate
        -nextweek features first!")
38
39 # Work with the *live* file (original name)
40 weekly_features = pd.read_csv(FEATURE_FILE)
41
42
43
44 def load_master() -> pd.DataFrame:
45     return pd.read_csv(MASTER_STORE) if MASTER_STORE.exists()
       else pd.DataFrame()
46
47 master_df = load_master()
48
49 # Load raw season results
50 results_df = pd.read_csv(RAW_RESULTS, dayfirst=True)
51
52 # Map FTR → numeric match result
53 ftr_map = {"H": 1, "D": 0, "A": -1}
54 results_df["MatchResult"] = results_df["FTR"].map(ftr_map)
55
56 # Choose just the last -matchweek rows not yet in master (ID by
    Home+Away+Date)
57 key_cols = ["Date", "HomeTeam", "AwayTeam"]
58 if not master_df.empty:
59     merged = results_df.merge(master_df[key_cols], how="left",
       indicator=True)
60     new_results = merged[merged["_merge"] == "left_only"].drop(
       columns="_merge")
61 else:
62     new_results = results_df.copy()
63
64 print(f"New result rows found: {len(new_results)}")
65
66 # Merge those new results onto the *previous* 'weeks feature rows
67 if not new_results.empty:
68     # Previous 'weeks feature file is by definition the one we
       backed up last run.
69     # We assume HomeTeam & AwayTeam match spelling across files.
70     prev_feature_files = sorted(BACKUP_DIR.glob("
       Final_Cleaned_Dataset_*.csv"))
71     if prev_feature_files:
```

```
        prev_week_features = pd.read_csv(prev_feature_files[-2])
            if len(prev_feature_files) >= 2 else pd.DataFrame()
    else:
        prev_week_features = pd.DataFrame()

    if not prev_week_features.empty:
        to_append = prev_week_features.merge(
            new_results[["HomeTeam", "AwayTeam", "FTHG", "FTAG",
                "MatchResult"]],
            on=["HomeTeam", "AwayTeam"],
            how="inner"
        )
        master_df = pd.concat([master_df, to_append],
           ignore_index=True)
        master_df.to_csv(MASTER_STORE, index=False)
        print(f"Master feature store updated → {MASTER_STORE}")
    else:
        print(" No -previousweek feature backup found; master not
            updated.")



if master_df.empty:
    raise ValueError("Master feature store is empty - need at
       least one week of labelled data.")

drop_cols = ["HomeTeam", "AwayTeam", "FTHG", "FTAG", "
   MatchResult"]
X_train = master_df.drop(columns=[c for c in drop_cols if c in
   master_df.columns])
X_train = X_train.select_dtypes(include=[np.number]).fillna(0)
y_train = master_df["MatchResult"]



pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("lr", LogisticRegression(max_iter=2000, multi_class="
       multinomial"))
])

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
proba_cv = cross_val_predict(pipe, X_train, y_train, cv=cv,
   method="predict_proba")
y_pred_cv = proba_cv.argmax(axis=1) - 1

metrics = {
    "run_date": today_str,
    "accuracy": round(accuracy_score(y_train, y_pred_cv), 4),
    "macro_f1": round(f1_score(y_train, y_pred_cv, average="
       macro"), 4),
```

```
    "log_loss": round(log_loss(y_train, proba_cv), 4),
    "brier":    round(brier_score_loss(pd.get_dummies(y_train).
       values.ravel(),
                                       proba_cv.ravel()), 4)
}
pd.DataFrame([metrics]).to_csv(METRIC_LOG, mode="a",
                               header=not METRIC_LOG.exists(),
                                  index=False)
print("CV metrics:", metrics)



pipe.fit(X_train, y_train)
model_path = MODEL_DIR / f"logreg_{today_str}.joblib"
joblib.dump(pipe, model_path)
print(f"Model snapshot saved → {model_path}")
```

3.7 Web Development

We have built a small website to show the results of the predictions (Home win probability, Away win probability, Draw probability, Who can score from home team and away team). We have also deployed the website to github so that other can also see the predictions.

Here are some pictures of the website:

Figure 3.3: Home Page

Github Link of the website:
https://linkon-dhar.github.io/Win_Prediction_Football/index.html

Figure 3.4: Predictions of premire league

Chapter 4

Result analysis

4.1

To model football match outcomes and derive win probabilities, we explored multiple supervised machine learning models using a cleaned dataset of past match statistics. The goal was to classify each fixture into one of three possible outcomes:

- **Home Win (1)**
- **Draw (0)**
- **Away Win (-1)**

A total of 12 samples were available, each representing a match between two teams with 20+ engineered numerical features capturing team form, ratings, goals, clean sheets, and more.

4.1.1 Evaluation and Important findings

Due to the limited data size, we used 5-Fold Stratified Cross-Validation to ensure fair distribution of match outcomes across each fold. Two performance metrics were tracked:

(1) **Accuracy** – How often the model predicts the correct match result.

(2) **F1 Macro Score** – Harmonic mean of precision and recall, averaged equally across all classes (Home Win, Draw, Away Win).

Logistic Regression Performed Best

Despite its simplicity, logistic regression achieved the highest accuracy and F1 score. This is due to its strength in small datasets and its ability to generalize well when the number of features exceeds the number of samples.

Class Imbalance Affected Scores

Draws and away wins were relatively rare in the dataset, and this imbalance lowered the F1-macro score (which treats all classes equally). Accuracy alone would be

misleading because it may favor predicting the majority class (home wins).

Nonlinear Models Were Unstable

Although powerful in theory, models like Gradient Boosting and SVM struggled to outperform logistic regression. This suggests that the dataset size was too small to support their complexity and that these models may have overfit during training.

Tree-Based Models Didn't Shine Yet

Both Random Forest and Gradient Boosting underperformed, likely because they require more data to discover meaningful hierarchical splits. HistGradientBoosting did better than standard GBM due to its regularization and speed advantages, but still fell short of the linear model.

KNN Struggled with Dimensionality

KNN achieved only moderate performance. This is common when feature space is high-dimensional, as distance metrics become less meaningful in such settings (curse of dimensionality). It was also sensitive to outliers and noise in the limited dataset.

4.2 Feature Importance and Influencing Factors

Understanding which features contribute most significantly to a machine learning model's predictions is a vital part of both model interpretability and domain relevance. In the context of match outcome prediction, feature importance analysis provides critical insights into which variables are most influential in determining the probability of a win or loss. This not only enhances the transparency of the predictive model but also helps refine our feature engineering process by identifying which factors should be retained, improved, or discarded.

In our system—centered around a logistic regression model enhanced by regularization and rule-based components—feature importance serves as a diagnostic and explanatory tool. It enables us to quantify the impact of each predictor and understand how certain attributes of teams, matches, or players drive outcomes.

4.2.1 Key Features in Our Predictive Framework

The features used in our model were engineered to capture a comprehensive view of a team's competitive standing and contextual dynamics before each match. Some of the most prominent ones include

- **Recent Form Scores:** A numerical measure based on the results of the last 5 matches. Higher values generally indicate stronger momentum.
- **Ground Advantage:** A binary or scaled value that captures whether a team is playing on a ground is being advantageous.

- **Head-to-Head Records:** Performance history between the two teams, incorporating both win ratios
- **Team Strength Metrics:** Aggregated player ratings, injuries, and squad depth derived from secondary data sources.

4.2.2 Domain-Relevant Interpretability

Beyond statistical significance, feature importance offers domain-relevant explanatory power. For example, a sharp drop in predicted win probability for a team missing its top scorer aligns not just with model logic, but with human intuition and expert opinion. This interpretability makes the model more trustworthy and actionable for use cases such as:

- Betting decision support
- Sports journalism and match previews
- Coaching staff strategy refinement

Furthermore, understanding these influencing factors helped us design a hybrid rule-based system (Section 3.3) to complement the model by overriding or modifying predictions in edge-case scenarios—such as when high-impact player injuries are not yet reflected in numerical metrics.

4.3 Win Probability Prediction

At the core of our machine learning system lies the task of estimating the likelihood of each possible match outcome—home win, draw, and away win—in probabilistic terms. Unlike deterministic classification, where the output is a fixed label, our model produces a probability distribution over all three outcomes. This probabilistic approach allows for more nuanced decision-making, especially in real-world applications like betting analysis, match forecasting, and tactical sports analytics.

4.3.1 Probabilistic Output: Home Win, Draw, and Away Win

Our prediction pipeline leverages a multinomial logistic regression model (also known as softmax regression) to estimate the probability for each of the three possible match outcomes: P(Home Win),P(Draw),P(Away Win) Each match instance is processed through the model to produce a probability triplet that sums to 1. For example, for a given match, the output might be:

- Home Win: 0.60
- Draw: 0.25
- Away Win: 0.15

These probabilities are derived based on the input feature vector, which includes team form, expected goals, home/away indicator, and various contextual variables (as detailed in Section 3.4). This probabilistic output allows us to understand not just the most likely outcome, but the relative likelihood of all scenarios, which is particularly useful in matches between evenly matched teams or in high-stakes environments where even low-probability outcomes may be critical.

Here below is the code of generating probability of win for the La liga:

```
import os
import numpy as np
import pandas as pd
import joblib
from pathlib import Path
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import StratifiedKFold


DATA_PATH = r"C:/Users/ASUS/team_data/La Liga_Final_Cleaned_Dataset.csv"
MODEL_DIR = Path("C:/Users/ASUS/team_data/models")
TAU       = 0.5            # 1.0 = sharp, -0.30.5 makes draws - ~1525%

df = pd.read_csv(DATA_PATH)

# Targets for the OOF fallback (kept)
df["y_home"] = (df["Home_Team_Goals_Scored"] > df["Away_Team_Goals_Scored"]).astype(int)
df["y_away"] = (df["Home_Team_Goals_Scored"] < df["Away_Team_Goals_Scored"]).astype(int)

NON_FEATS = [
    "HomeTeam", "AwayTeam",
    "Home_Team_Goals_Scored", "Away_Team_Goals_Scored",
    "y_home", "y_away",
]
X_full = df.drop(columns=NON_FEATS)
X = X_full.loc[:, X_full.std(axis=0) > 0]


try:
    latest = max(MODEL_DIR.glob("logreg_*.joblib"))
    print(f" Using trained model → {latest.name}")
    model = joblib.load(latest)

    # Ensure column order matches training
    X_next = X[model.named_steps["lr"].feature_names_in_]
    proba  = model.predict_proba(X_next)
    p_away = proba[:, 0]      # multinomial ordering: [-1, 0, 1]
    p_draw = proba[:, 1]
```

```
    p_home = proba[:, 2]

except ValueError:            # glob() empty -> no snapshot yet
    print(" No snapshot found; falling back to OOF training.")
    from sklearn.model_selection import cross_val_predict

    def oof_probabilities(X, y, C=1.0, max_splits=4):
        n_pos, n_neg = y.sum(), len(y) - y.sum()
        n_splits = max(2, min(max_splits, int(n_pos), int(n_neg)
           ))
        skf = StratifiedKFold(n_splits=n_splits, shuffle=True,
           random_state=42)
        oof = np.zeros(len(y))
        for tr, te in skf.split(X, y):
            pipe = make_pipeline(StandardScaler(),
                                 LogisticRegression(max_iter
                                    =1000, C=C))
            pipe.fit(X.iloc[tr], y.iloc[tr])
            oof[te] = pipe.predict_proba(X.iloc[te])[:, 1]
        return oof

    p_home = oof_probabilities(X, df["y_home"])
    p_away = oof_probabilities(X, df["y_away"])

    # Convert the two probabilities to a preliminary draw
    odds_home = p_home / (1.0 - p_home + 1e-12)
    odds_away = p_away / (1.0 - p_away + 1e-12)
    odds_home, odds_away = odds_home**TAU, odds_away**TAU
    denom = 1.0 + odds_home + odds_away
    p_draw = 1.0 / denom
    p_home, p_away = odds_home / denom, odds_away / denom


gamma, min_draw = 0.45, 0.05
closeness = 1.0 - np.abs(p_home - p_away)
p_draw_raw = np.maximum(min_draw, gamma * closeness)
remain = 1.0 - p_draw_raw
weights = p_home + p_away + 1e-12
p_home, p_away, p_draw = remain * p_home / weights, remain *
   p_away / weights, p_draw_raw

# Scale to %
p_home, p_draw, p_away = [np.round(v * 100, 1) for v in (p_home,
    p_draw, p_away)]

prob_df = pd.DataFrame({
    "Home_Team": df["HomeTeam"],
    "Away_Team": df["AwayTeam"],
    "Home_win":  p_home,
    "Draw":      p_draw,
    "Away_Win":  p_away,
```

```
})

OUT_PATH = os.path.join(os.path.dirname(DATA_PATH), "La
    Liga_win_probabilities.csv")
prob_df.to_csv(OUT_PATH, index=False)
print(f" Probabilities written to {OUT_PATH}")
```

4.3.2 Model Selection Process: Testing Multiple Candidates

To ensure that we adopted the most effective prediction model for our specific task and dataset, we conducted a comparative analysis of several machine learning algorithms, including:

- Logistic Regression (binary and multinomial variants)
- Random Forest Classifier
- Gradient Boosting Machines (GBM)
- Support Vector Machines (SVM)
- k-Nearest Neighbors (k-NN)
- HistGB

Each model was trained and tested using cross-validation and evaluated using metrics such as, accuracy and F1. Our dataset—moderate in size but rich in engineered features—highlighted the importance of a model that could offer both predictive stability and probabilistic outputs with low variance.
After extensive testing and evaluation, Scikit-learn's multinomial logistic regression implementation emerged as the best overall performer, offering a good balance between interpretability, calibration, and generalization.

Here below are the codes and its result:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
    GradientBoostingClassifier, HistGradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
import warnings
warnings.filterwarnings('ignore')

# Load Data
df = pd.read_csv("C:/Users/ASUS/team_data/Final_Cleaned_Dataset.
    csv")
```

```
15
16 # Create Target Class: "MatchResult" (HomeWin = 1, Draw = 0,
       AwayWin = -1)
17 df["MatchResult"] = df.apply(lambda row:
18     1 if row["Home_Team_Goals_Scored"] > row["
          Away_Team_Goals_Scored"] else
19     -1 if row["Home_Team_Goals_Scored"] < row["
          Away_Team_Goals_Scored"] else 0, axis=1)
20
21 # Define Features (drop team names & goal columns)
22 features = df.drop(columns=["HomeTeam", "AwayTeam", "
       Home_Team_Goals_Scored", "Away_Team_Goals_Scored", "
       MatchResult"])
23 target = df["MatchResult"]
24
25 # Scale Features
26 scaler = StandardScaler()
27 X = scaler.fit_transform(features)
28 y = target
29
30 # Define Classifiers to Compare
31 models = {
32     "Logistic Regression": LogisticRegression(max_iter=1000),
33     "Random Forest": RandomForestClassifier(n_estimators=100,
          random_state=42),
34     "Gradient Boosting": GradientBoostingClassifier(n_estimators
          =100, random_state=42),
35     "SVM": SVC(kernel='rbf', probability=True),
36     "KNN": KNeighborsClassifier(n_neighbors=5),
37     "HistGB": HistGradientBoostingClassifier(random_state=42)
38 }
39
40 # Evaluate Each Model with Cross-Validation
41 results = {}
42 for name, model in models.items():
43     accuracy = cross_val_score(model, X, y, cv=5, scoring='
          accuracy').mean()
44     f1 = cross_val_score(model, X, y, cv=5, scoring='f1_macro').
          mean()
45     results[name] = {"Accuracy": round(accuracy, 4), "F1 Score":
           round(f1, 4)}
46
47 #  Display Results
48 results_df = pd.DataFrame(results).T.sort_values(by="F1 Score",
       ascending=False)
49 print("  Model Comparison:")
50 print(results_df)
```

Model Comparison

	Accuracy	F1 Score
Logistic Regression	0.7000	0.6000
SVM	0.6333	0.4800
HistGB	0.6333	0.4800
Random Forest	0.5000	0.4667
KNN	0.5333	0.4000
Gradient Boosting	0.3667	0.3167

Table 4.1: Comparison of Model Accuracy and F1 Score

4.3.3 Why Scikit-learn Logistic Regression Was the Best Choice

The Scikit-learn library's implementation of logistic regression proved ideal for several key reasons:

(1) **Built-in Multinomial Support:**

The LogisticRegression class in Scikit-learn provides native support for multinomial (softmax) classification through the multi_class='multinomial' parameter, which is crucial for modeling three-class outcomes like win, draw, or loss.

(2) **Solver Flexibility and Stability:**

Scikit-learn allows the use of solvers like 'lbfgs' and 'saga', which are robust and efficient for medium to large datasets. These solvers provided convergence guarantees and handled regularization well.

(3) **Support for Regularization:**

The inclusion of both L1 (Lasso) and L2 (Ridge) regularization, with hyperparameter tuning via grid search and cross-validation, allowed us to effectively control overfitting without sacrificing predictive power (see Section 3.5).

(4) **Model Calibration:**

Logistic regression, especially with Scikit-learn's implementation, is naturally well-calibrated, meaning that predicted probabilities correspond closely to actual observed frequencies. This is critical for our use case, where probability accuracy is more important than simple class prediction.

(5) **Interpretability and Debugging:**

Unlike black-box models like gradient boosting or neural networks, logistic regression offers clear coefficient outputs, which support in-depth feature importance analysis (Section 4.3). This transparency was vital for both debugging and stakeholder communication.

(6) **Simplicity and Speed:**

Logistic regression trains quickly and consistently, enabling rapid iteration and real-time retraining after each match (Section 3.8), without excessive computational overhead.

4.4 Goal Scorer Prediction (For Particular Player)

Predicting individual player performance, particularly identifying likely goal scorers in a given match, is a challenging yet rewarding extension of the team-level outcome prediction. While match outcome prediction provides insight into the overall win probability of a team, goal scorer prediction focuses on individual contributions—offering even more granular insights into the game. This is particularly valuable for applications such as fantasy football, betting, and performance analysis.
In our model, we have developed a goal scorer prediction system that leverages a player's historical performance data, to estimate the likelihood that a particular player will score in an upcoming match.

4.4.1 Identifying Likely Goal Scorers Based on Historical Performance

To identify potential goal scorers, we begin by analyzing a player's historical performance data over a significant number of matches. Key features considered include:

- **Goals Scored (xG):** The total number of goals scored by the player in recent matches and the quality of the chances they converted. We use Expected Goals (xG) as a refined measure of goal-scoring likelihood, as it accounts for the quality of chances rather than just the number of goals scored.
- **Shots on Target:** A metric that tracks how often the player manages to get a shot on goal. Players with a higher proportion of shots on target have a greater likelihood of scoring.
- **Shot Accuracy/ Rating:** This feature indicates the percentage of shots that hit the target, providing an understanding of how clinical a player is in front of goal.

The combination of these historical features helps us model the goal-scoring probability for each player in an upcoming match.

4.4.2 Factors Influencing Goal-Scoring Likelihood

Several factors influence a player's likelihood of scoring in a particular match. These can be classified into player-specific factors, team-related factors, and contextual match factors:

(1) **Player-Specific Factors:**

- **Historical Goal Scoring Rate:** The player's past performance in terms of goals scored is often the best predictor of future performance. Players with consistent goal-scoring records are more likely to score again.
- **Shot Conversion Rate:** A higher conversion rate (i.e., the ratio of shots-to-goals) increases a player's likelihood of scoring in subsequent matches.
- **Current Form:** A player's recent performances, including goals scored, assists, and overall contributions, influence their scoring chances. Players on a scoring streak are more likely to continue finding the net.

(2) **Team-Related Factors:**

- **Team Offensive Strength:** The quality of a team's attacking play (e.g., high xG, strong possession, creative playmakers) enhances the probability of their forwards and attackers getting into scoring positions.
- **Assist Providers:** Players who are assisted by high-quality playmakers are more likely to score. Therefore, identifying players with high assist potential (e.g., midfielders or wingers) is important.
- **Team's Match Context:** If a team is facing a weaker opponent, the attack is likely to generate more scoring chances, thus increasing the likelihood of individual players scoring.

(3) **Match-Specific Factors:**

- **Opponent Strength:** The defensive strength of the opposing team directly impacts a player's chances of scoring. A weaker defense with lower expected goals against (xGA) may increase scoring opportunities.
- **Home vs. Away:** Players often perform better at home due to familiar conditions, fan support, and reduced travel fatigue.
- **Match Importance:** High-stakes matches (e.g., playoffs or derbies) can influence player performance. Players may increase their effort in important matches, and teams may create more attacking opportunities to secure a win.
- **Weather and Pitch Conditions:** Adverse weather conditions or poor pitch quality can impact a player's ability to score, as technical skills might be hindered.

4.4.3 Modeling Goal Scorer Prediction

This model is trained using historical player data and is regularly updated with real match results to improve its predictive accuracy over time.

```
import pandas as pd
from pathlib import Path

PLAYER_STATS_PATH = Path("C:/Users/ASUS/team_data/Cleaned_Player_Stats.xlsx")
NEXT_MATCHES_PATH = Path("C:/Users/ASUS/Next_Premier_League_Matches.csv")
OUTPUT_PATH       = Path("C:/Users/ASUS/team_data/Predicted_Goal_Scorers.csv")

df_players = pd.read_excel(PLAYER_STATS_PATH)
df_fixtures = pd.read_csv(NEXT_MATCHES_PATH)

numeric_cols = ["Goals", "SpG", "Rating"]
for col in numeric_cols:
    if col in df_players.columns:
        df_players[col] = (
            pd.to_numeric(df_players[col], errors="coerce")
            .fillna(0)
        )
    else:
        raise ValueError(f"Column '{col}' missing from player file.")

df_players["SPS"] = (
    0.4 * df_players["Goals"]    +   # season/-lastN goals
    0.25 * df_players["SpG"]     +   # shots per game
    0.20 * df_players["Rating"]      # form / rating
)

def top_scorers(team: str, n: int = 2) -> str:
    squad = df_players[df_players["Team"].str.lower() == team.lower()]
    if squad.empty:
        return "No data"
    names = (
        squad.sort_values("SPS", ascending=False)
        .head(n)["Player"]
        .tolist()
    )
    return ", ".join(names) if names else "None"

df_fixtures["Home_Probable_Scorers"] = df_fixtures["Home Team"].apply(top_scorers)
df_fixtures["Away_Probable_Scorers"] = df_fixtures["Away Team"].apply(top_scorers)

```

```
OUTPUT_PATH.parent.mkdir(parents=True, exist_ok=True)
df_fixtures.to_csv(OUTPUT_PATH, index=False)
print(f" Predicted goal scorers saved →  {OUTPUT_PATH}")
```

4.5 Latest week results of the predictor (last matches of the 24-25 season)

All 5 Leagues together

Home Team	Away Team	Home Win %	Draw %	Away Win %
Bournemouth	Leicester	49.20%	5.50%	45.30%
Fulham	Manchester City	12.40%	4.90%	82.70%
Ipswich	West Ham	29.70%	5.20%	65.10%
Liverpool	Crystal Palace	66.10%	5.90%	28.00%
Manchester Utd	Aston Villa	61.40%	5.30%	33.30%
Newcastle	Everton	65.20%	5.10%	29.70%
Nottingham	Chelsea	28.50%	4.70%	66.80%
Southampton	Arsenal	22.30%	4.30%	73.40%
Tottenham	Brighton	47.20%	6.00%	46.80%
Wolves	Brentford	45.30%	3.80%	50.90%
Lens	Monaco	39.80%	5.60%	54.60%
Lille	Reims	60.70%	5.10%	34.20%
Lyon	Angers	62.10%	4.90%	33.00%
Marseille	Rennes	58.40%	5.70%	35.90%
Nantes	Montpellier	56.20%	5.40%	38.40%
Nice	Brest	59.00%	4.80%	36.20%
PSG	Auxerre	81.40%	5.20%	13.40%
St Etienne	Toulouse	38.10%	5.90%	56.00%
Strasbourg	Le Havre	54.70%	5.30%	40.00%
Ath Bilbao	Barcelona	38.50%	5.50%	56.00%
Villarreal	Sevilla	61.20%	5.00%	33.80%
Girona	Atl. Madrid	34.00%	4.80%	61.20%
Alaves	Osasuna	36.50%	5.50%	58.00%
Getafe	Celta Vigo	46.00%	6.00%	48.00%
Rayo Vallecano	Mallorca	50.00%	5.00%	45.00%
Espanyol	Las Palmas	55.00%	5.50%	39.50%
Leganes	Valladolid	52.00%	5.00%	43.00%
Real Madrid	Real Sociedad	65.00%	5.50%	29.50%
Betis	Valencia	58.00%	5.00%	37.00%
Augsburg	Union Berlin	41.00%	5.50%	53.50%
B. Monchengladbach	Wolfsburg	46.50%	6.00%	47.50%
Dortmund	Holstein Kiel	65.80%	5.20%	29.00%
Freiburg	Eintracht Frankfurt	42.00%	5.30%	52.70%
Heidenheim	Werder Bremen	36.50%	4.50%	59.00%
Hoffenheim	Bayern Munich	28.50%	5.00%	66.50%
Mainz	Bayer Leverkusen	35.00%	4.80%	60.20%
RB Leipzig	Stuttgart	47.00%	6.00%	47.00%
St. Pauli	Bochum	57.20%	5.10%	37.70%
Atalanta	Parma	58.30%	5.00%	36.70%
Empoli	Verona	40.40%	5.20%	54.40%
Lazio	Lecce	59.20%	4.80%	36.00%
Torino	AS Roma	41.70%	4.70%	53.60%
Udinese	Fiorentina	38.90%	5.50%	55.60%
Venezia	Juventus	32.40%	4.20%	63.40%
AC Milan	Monza	60.70%	4.00%	35.30%
Bologna	Genoa	55.60%	5.60%	38.80%
Como	Inter	28.60%	4.60%	66.80%
Napoli	Cagliari	61.10%	5.20%	33.70%

Table 4.2: Football Match Statistics

Chapter 5

Limitations

5.1 Strengths and Limitations of the Model

In this research, we utilized logistic regression, a statistical technique and machine learning algorithm commonly used for binary classification tasks, to predict the outcomes of football matches. Logistic regression is particularly well-suited to scenarios where the output variable is categorical, such as win/loss or win/draw/loss classification. The rationale behind choosing this method was its computational efficiency, interpretability, and wide adoption in predictive modeling.

One of the primary strengths of logistic regression is its interpretability. It allows researchers and practitioners to determine the individual effect of each independent variable (feature) on the outcome variable. This property is essential in football analytics, where domain experts often seek to understand which factors—such as team form, average goals scored, or home advantage—contribute most to the probability of winning a match. The logistic function maps input features to a probability value between 0 and 1, making it intuitive to apply threshold-based decision-making.

Another strength lies in its simplicity and generalizability. Logistic regression does not require a large amount of computational resources, making it accessible for projects with limited infrastructure. It also performs well in high-bias, low-variance settings, which can be ideal when overfitting is a concern.

However, logistic regression also exhibits several significant limitations, especially when applied to the complex, dynamic environment of football match prediction. One key drawback is the assumption of linearity between the independent variables and the log-odds of the dependent variable. This assumption often does not hold in real-world sports data, where interactions between players, team morale, weather, or in-game decisions introduce non-linear and temporal dependencies.

Another issue is data sparsity and imbalance. Football match outcomes can be skewed toward certain results (e.g., more frequent home wins), leading to biased predictions if the model is not appropriately balanced. Moreover, obtaining high-quality, consistent, and up-to-date data remains a challenge. Not all leagues provide detailed match statistics, and data providers may vary in terms of accuracy and coverage. These data limitations significantly influence model performance and

reliability.

Model assumptions, such as the independence of observations, can also be problematic. Football matches are not independent events—team performance in one game may influence future games, and external factors like injuries, travel schedules, or team dynamics change over time. Logistic regression does not inherently capture such sequential dependencies. Moreover, the model is sensitive to feature selection and multicollinearity. If highly correlated features are not handled properly (e.g., through regularization or PCA), the model's coefficient estimates can become unstable. Logistic regression also struggles with non-linear interactions, which are often crucial in sports outcomes. For example, the combined effect of a star striker and a strong midfield might be more significant than the individual contributions, a relationship that a basic logistic model may not capture. Finally, logistic regression may underperform in comparison to modern techniques like deep learning, ensemble models, or time-series forecasting methods, which are better suited to handle large feature spaces, sequence dependencies, and complex interactions.

Chapter 6

Conclusion

6.1 Summary of Findings

This thesis explored the use of logistic regression for predicting football match outcomes based on historical match data and team performance metrics. Our objective was to investigate whether a relatively simple yet interpretable classification model could yield meaningful insights into match predictions. The study involved the collection, preprocessing, and modeling of football data, followed by an evaluation of the model's predictive accuracy and interpretability.

The logistic regression model was able to classify match outcomes (win/loss) with moderate accuracy, highlighting the predictive potential of structured football data. Key features such as recent form, goal difference, home/away status, and head-to-head results emerged as influential variables. Although the model was effective in capturing broad patterns, its limitations became evident in more nuanced predictions, such as tightly contested matches or games influenced by unpredictable events.

Overall, our findings support the notion that logistic regression can serve as a baseline model in sports analytics. It provides a transparent and mathematically grounded framework for understanding match outcomes, but its simplicity restricts its performance in more sophisticated prediction scenarios.

6.2 Contributions to Football Analytics

This work contributes to the evolving field of football analytics in several ways. Firstly, it demonstrates a methodical application of logistic regression to sports data, offering a reproducible framework for outcome prediction. This is valuable for sports analysts, clubs, and data scientists looking for interpretable and explainable models.

Secondly, by identifying key performance indicators through model coefficients, the research provides actionable insights into factors that consistently influence match outcomes. These insights can guide coaching decisions, scouting strategies, and performance evaluations. For example, quantifying the impact of playing at home or the momentum of recent matches can inform tactical choices.

Furthermore, the project encourages the integration of machine learning into strategic football decision-making, promoting data-driven practices in a traditionally intuition-led domain. By bridging statistical analysis with domain knowledge, this thesis supports a more analytical approach to sports performance assessment.

6.3 Future Research Directions

While logistic regression offers a useful starting point, future work should explore advanced machine learning and deep learning approaches to capture the full complexity of football data. Some promising directions include

- **Deep Neural Networks (DNNs):** Capable of modeling non-linear relationships and interactions between features. Particularly useful when working with large and complex datasets.
- **Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks:** Designed to handle sequential data and temporal dependencies, making them suitable for tracking team or player form over time.
- **Ensemble Methods:** Techniques such as random forests, gradient boosting machines (e.g., XGBoost), and LightGBM can improve accuracy by combining multiple models and handling non-linearities more effectively.
- **Feature Engineering with Domain Expertise:** Introducing more nuanced features like expected goals (xG), possession dynamics, or player fatigue indices may significantly enhance predictive power.
- **Transfer Learning:** Applying pretrained models from related sports or simulations to football analytics can accelerate model training and improve generalization.

Additionally, researchers should consider robust cross-validation techniques and multi-league testing to assess model transferability across different competitions and seasons.

6.4 Real-World Applications of Football Prediction

Predictive models in football have substantial real-world applications that extend across commercial, strategic, and recreational domains:

- **Sports Betting Platforms:** Accurate match outcome predictions can help bookmakers set odds more effectively, reduce arbitrage opportunities, and offer intelligent betting suggestions to users. AI-driven predictions can also be used to detect anomalies or suspicious betting behavior.
- **Team Strategy and Tactical Planning:** Coaches and analysts can use predictive tools to identify upcoming threats, simulate match scenarios, and test various tactical options. The ability to anticipate opponent strategies based on historical data can improve match preparation.

- **Fan Engagement and Entertainment:** Applications like fantasy football, match simulations, and prediction games benefit from machine learning insights. Fans can engage more deeply with the game by exploring data-driven narratives and visualizations.
- **Scouting and Recruitment:** Analytics tools can help identify undervalued players, assess team compatibility, and forecast long-term performance trends.
- **Broadcast and Media Enhancements:** Live match coverage can integrate predictive stats and probabilities, offering viewers richer context and analysis.
- **Academic and Business Research:** The football industry serves as a valuable testing ground for AI models and data science methodologies, contributing to the broader field of applied machine learning.

In conclusion, the intersection of machine learning and football presents rich opportunities for innovation. While our model serves as a starting point, ongoing research and technological advancements promise more accurate and impactful applications in the future.

Bibliography

[1] J. Goddard and I. Asimakopoulos, "Forecasting football results and the efficiency of fixed-odds betting," *Journal of Forecasting*, vol. 23, pp. 51–66, 2004. DOI: 10.1002/for.877. [Online]. Available: http://previsaosimples.pbworks.com/w/file/fetch/65223638/soccerForecasting.pdf.

[2] J. Choe and R. B. McKay, "A compound framework for sports prediction: The case study of football," 2009, Retrieved from https://www.academia.edu/18792266.

[3] L. M. Hvattum and H. Arntzen, "Using ELO ratings for match result prediction in association football," *International Journal of Forecasting*, vol. 26, no. 3, pp. 460–470, Jul. 2010. [Online]. Available: https://ideas.repec.org/a/eee/intfor/v26yi3p460-470.html.

[4] D. Berrar, P. Lopes, and W. Dubitzky, "Incorporating domain knowledge in machine learning for soccer outcome prediction," *Machine Learning*, vol. 108, no. 1, pp. 97–126, Aug. 2018. DOI: 10.1007/s10994-018-5747-8.

[5] A. Groll, C. Ley, G. Schauberger, and H. Van Eetvelde, "A hybrid random forest to predict soccer matches," *Journal of Quantitative Analysis in Sports*, vol. 14, no. 2, pp. 77–90, 2018. DOI: 10.1515/jqas-2017-0044.

[6] N. R. Rosli, Z. M. Yunos, and N. A. Rashid, "A comparative study of data mining techniques on football match prediction," in *Proceedings of the 3rd Advances in Computing Engineering Symposium (ACES)*, 2018.

[7] K. Hader and et al., "Influence of positioning and match context on the physical demands of elite football players," *Journal of Sports Sciences*, vol. 37, no. 12, pp. 1339–1347, 2019.

[8] K. Hader, M. C. Rumpf, M. Hertzog, L. P. Kilduff, O. Girard, and J. R. Silva, "Monitoring the athlete match response: Can external load variables predict post-match acute and residual fatigue in soccer? a systematic review with meta-analysis," *Sports Medicine - Open*, vol. 5, no. 1, Dec. 2019. DOI: 10.1186/s40798-019-0219-7.

[9] R. Aquino, C. Carling, L. H. P. Vieira, *et al.*, "Influence of situational variables, team formation, and playing position on match running performance and social network analysis in brazilian professional soccer players," *Journal of Strength and Conditioning Research*, vol. 34, no. 3, pp. 808–817, Mar. 2020. DOI: 10.1519/jsc.0000000000002725. [Online]. Available: https://doi.org/10.1519/jsc.0000000000002725.

[10] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, and et al., "Language models are few-shot learners," *arXiv preprint*, vol. arXiv:2005.14165, 2020.

[11] R. Julian, R. M. Page, and L. D. Harper, "The effect of fixture congestion on performance during professional male soccer match-play: A systematic critical review with meta-analysis," *Sports Medicine*, vol. 51, no. 2, pp. 255–273, Oct. 2020. DOI: 10.1007/s40279-020-01359-9.

[12] J. M. Oliva-Lozano, D. Rojas-Valverde, C. D. Gómez-Carmona, V. Fortes, and J. Pino-Ortega, "Worst case scenario match analysis and contextual variables in professional soccer players: A longitudinal study," *Biology of Sport*, vol. 37, no. 4, pp. 429–436, 2020, ISSN: 0860-021X. DOI: 10.5114/biolsport.2020.97067. [Online]. Available: http://dx.doi.org/10.5114/biolsport.2020.97067.

[13] I. Ismail, C.-T. Lu, N. H. Abd Rahman, and H. M. Hasan, "Forecasting goal-scoring likelihood in elite football leagues: A diverse machine learning approach," *Sports*, vol. 6, no. 3, p. 86, 2022.

[14] Y. Song, "Combining machine learning and human experts to predict match outcomes in football," *arXiv preprint*, vol. arXiv:2012.04380, 2023.

[15] H. Touvron, T. Lavril, G. Izacard, and et al., "Llama: Open and efficient foundation language models," *arXiv preprint*, vol. arXiv:2302.13971, 2023.

Chapter 7

Appendix:

Code Repository

Scrappers:

It scraps from different website and collect data Below is an example of Scraping data for the premier league.

```
1
2  from selenium import webdriver
3  from selenium.webdriver.common.by import By
4  from selenium.webdriver.chrome.service import Service
5  from selenium.webdriver.common.keys import Keys
6  from bs4 import BeautifulSoup
7  import pandas as pd
8  import time
9
10 # Path to your ChromeDriver
11 CHROMEDRIVER_PATH = 'C:/WebDrivers/chromedriver.exe'
12 BASE_URL = "https://www.flashscore.com/football/england/premier-
       league/"
13
14 def fetch_next_matches_with_selenium(url):
15     """Fetch the next Premier League matches using Selenium."""
16     try:
17         print(f"Fetching data from {url}...")
18         # Set up Selenium WebDriver
19         service = Service(CHROMEDRIVER_PATH)
20         driver = webdriver.Chrome(service=service)
21         driver.get(url)
22         time.sleep(5)  # Wait for the page to load completely
23
24         # Get page source and parse with BeautifulSoup
25         soup = BeautifulSoup(driver.page_source, "html.parser")
26         driver.quit()
27
28         # Locate all match rows
29         match_rows = soup.find_all("div", class_="event__match")
30         if not match_rows:
```

```
            print("No match data found.")
            return None

        matches = []
        for match in match_rows:
            try:
                # Extract match details
                match_time = match.find("div", class_="
                    event__time").text.strip()
                home_team = match.find("div", class_="
                    event__homeParticipant").text.strip()
                away_team = match.find("div", class_="
                    event__awayParticipant").text.strip()
                match_link = match.find("a", class_="
                    eventRowLink")["href"]
                match_link_full = f"https://www.flashscore.com{
                    match_link}"

                matches.append({
                    "Time": match_time,
                    "Home Team": home_team,
                    "Away Team": away_team,
                    "Match Link": match_link_full
                })
            except Exception as e:
                print(f"Error processing match row: {e}")

        return matches
    except Exception as e:
        print(f"Error occurred: {e}")
        return None

def save_to_csv(data, file_name="Next_Premier_League_Matches.csv
    "):
    """Save match data to a CSV file."""
    try:
        df = pd.DataFrame(data)
        df.to_csv(file_name, index=False)
        print(f"Match data saved to {file_name}")
    except Exception as e:
        print(f"Error saving data to CSV: {e}")

if __name__ == "__main__":
    match_data = fetch_next_matches_with_selenium(BASE_URL)
    if match_data:
        save_to_csv(match_data)
    else:
        print("No data fetched.")

import requests
from bs4 import BeautifulSoup
```

```
import pandas as pd

# FBref URL for Premier League points table
URL = "https://fbref.com/en/comps/9/Premier-League-Stats"

def fetch_points_table_with_form(url):
    """Fetch the Premier League points table with form details.
       """
    try:
        print(f"Fetching data from {url}...")
        response = requests.get(url, headers={"User-Agent": "
           Mozilla/5.0"})
        response.raise_for_status()  # Raise an error for bad
           responses
        soup = BeautifulSoup(response.content, "html.parser")

        # Locate the table
        table = soup.find("table", {"id": "results2024-202591
           _overall"})
        if not table:
            print("No points table found.")
            return None

        # Extract headers
        headers = [header.text.strip() for header in table.find(
           "thead").find_all("th")]

        # Extract rows of data
        rows = table.find("tbody").find_all("tr")
        data = []
        for row in rows:
            cells = row.find_all(["th", "td"])
            row_data = [cell.text.strip() for cell in cells]

            # Extract "Form" separately from embedded div
               elements
            form_cell = row.find("td", {"data-stat": "last_5"})
            if form_cell:
                form_icons = form_cell.find_all("div", recursive
                   =False)
                form_text = "".join(icon.text.strip() for icon
                   in form_icons)  # Get textual form (e.g., WDL
                   )
                row_data.append(form_text)  # Append form to the
                    row data

            data.append(row_data)

        # Append "Form" to headers
        headers.append("Form")

```

```
        # Create DataFrame
        df = pd.DataFrame(data, columns=headers)

        # Select required columns up to "Form"
        columns_to_include = ["Rk", "Squad", "MP", "W", "D", "L"
            , "GF", "GA", "GD", "Pts", "Form"]
        df = df[columns_to_include]
        return df

    except Exception as e:
        print(f"Error occurred while fetching data: {e}")
        return None

def save_to_csv(df, file_name="
    PremierLeague_PointsTable_with_Form.csv"):
    """Save the points table to a CSV file."""
    try:
        df.to_csv(file_name, index=False)
        print(f"Points table with form saved to {file_name}")
    except Exception as e:
        print(f"Error saving points table to CSV: {e}")

if __name__ == "__main__":
    points_table_with_form = fetch_points_table_with_form(URL)
    if points_table_with_form is not None:
        save_to_csv(points_table_with_form)
    else:
        print("No data fetched.")

import os
import requests
from bs4 import BeautifulSoup

# Base URL for FootballData.uk
BASE_URL = "https://www.football-data.co.uk/englandm.php"

# Seasons to fetch
SEASON_CODES = ["2425", "2324", "2223"]  # Corresponding to
    2024-2025, 2023-2024, 2022-2023
LEAGUE_CODE = "E0"  # Premier League

# Directory to save the downloaded files
SAVE_DIR = "team_data"

def fetch_csv_links():
    """Scrape the Premier League CSV links from the FootballData
        .uk page."""
    try:
        print(f"Fetching data from {BASE_URL}...")
        response = requests.get(BASE_URL)
        response.raise_for_status()  # Raise an error for bad
```

```
            responses
        soup = BeautifulSoup(response.content, "html.parser")

        # Find all <a> tags with relevant href patterns
        csv_links = {}
        for link in soup.find_all("a", href=True):
            href = link["href"]
            for season_code in SEASON_CODES:
                if f"mmz4281/{season_code}/{LEAGUE_CODE}.csv" in
                    href:
                    season = f"20{season_code[:2]}-{season_code
                        [2:]}"
                    csv_links[season] = "https://www.football-
                        data.co.uk/" + href
        return csv_links
    except Exception as e:
        print(f"Error occurred while fetching links: {e}")
        return {}

def download_csv_files(csv_links):
    """Download CSV files for the specified links."""
    if not os.path.exists(SAVE_DIR):
        os.makedirs(SAVE_DIR)

    for season, url in csv_links.items():
        file_name = f"{SAVE_DIR}/season-{season.replace('-', '')
            }.csv"

        try:
            print(f"Downloading {season} data from {url}...")
            response = requests.get(url)
            if response.status_code == 200:
                with open(file_name, "wb") as file:
                    file.write(response.content)
                print(f"Saved: {file_name}")
            else:
                print(f"Failed to download {url}, Status Code: {
                    response.status_code}")
        except Exception as e:
            print(f"Error occurred while downloading {url}: {e}"
                )

if __name__ == "__main__":
    print("Fetching Premier League CSV links...")
    csv_links = fetch_csv_links()

    if csv_links:
        print(f"Found {len(csv_links)} links. Starting download
            ...")
        download_csv_files(csv_links)
        print("Data fetch complete.")
```

```
    else:
        print("No links found.")

import requests
from bs4 import BeautifulSoup
import pandas as pd

# Base URL for Premier League injury list on Transfermarkt
BASE_URL = "https://www.transfermarkt.com/premier-league/verletztespieler/wettbewerb/GB1"

def fetch_injury_data(url):
    """Fetch injury data for Premier League players."""
    try:
        print(f"Fetching data from {url}...")
        response = requests.get(url, headers={"User-Agent": "Mozilla/5.0"})
        response.raise_for_status()  # Raise an error for bad responses
        soup = BeautifulSoup(response.content, "html.parser")

        # Locate the table containing injury data
        table = soup.find("table", class_="items")
        if not table:
            print("No injury data table found.")
            return None

        # Extract table rows
        rows = table.find("tbody").find_all("tr")
        if not rows:
            print("No rows found in the injury table.")
            return None

        injury_data = []

        for row in rows:
            try:
                cells = row.find_all("td")
                if len(cells) < 7:
                    print(f"Skipping incomplete row: {row}")
                    continue

                # Extract player name (from the alt attribute or <a> tag)
                player_cell = cells[1]
                img_tag = player_cell.find("img")
                player_name = img_tag["alt"].strip() if img_tag else "Unknown"

                # Extract club name
                club_cell = cells[2]
```

```
                club_name = club_cell.find("img")["alt"] if
                    club_cell.find("img") else "Unknown"

                # Extract other details
                injury_type = cells[4].text.strip()
                since_date = cells[5].text.strip()

                injury_data.append({
                    "Player": player_name,
                    "Club": club_name,
                    "Injury Type": injury_type,
                    "Since": since_date
                })
            except Exception as e:
                print(f"Error processing row: {e}")

        return injury_data
    except Exception as e:
        print(f"Error occurred while fetching data: {e}")
        return None

def save_to_csv(data, file_name="PremierLeague_Injuries.csv"):
    """Save injury data to a CSV file."""
    try:
        df = pd.DataFrame(data)
        df.to_csv(file_name, index=False)
        print(f"Data saved to {file_name}")
    except Exception as e:
        print(f"Error saving data to CSV: {e}")

if __name__ == "__main__":
    # Fetch injury data
    injury_data = fetch_injury_data(BASE_URL)

    if injury_data:
        # Save the data to a CSV file
        save_to_csv(injury_data)
    else:
        print("No data fetched.")

from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import pandas as pd
import time

# Base URL for WhoScored Premier League player stats
URL = "https://www.whoscored.com/Regions/252/Tournaments/2/
   Seasons/9155/Stages/21075/PlayerStatistics/England-Premier-
```

```
    League-2024-2025"

def fetch_whoscored_data(url):
    """Fetch all player statistics, handling pagination with a
       click limit."""
    service = Service("C:/WebDrivers/chromedriver.exe")  #
       Update with your ChromeDriver path
    driver = webdriver.Chrome(service=service)

    try:
        driver.get(url)
        time.sleep(5)

        # Wait for the stats table to load
        WebDriverWait(driver, 20).until(
            EC.presence_of_element_located((By.ID, "player-table
               -statistics-body"))
        )

        all_data = []
        click_count = 0  # Initialize click counter
        MAX_CLICKS = 30  # Set a maximum number of pages to
           scrape

        while click_count < MAX_CLICKS:
            # Extract table rows
            WebDriverWait(driver, 20).until(
                EC.presence_of_element_located((By.ID, "player-
                   table-statistics-body"))
            )
            table = driver.find_element(By.ID, "player-table-
               statistics-body")
            rows = table.find_elements(By.TAG_NAME, "tr")

            # Extract player stats for the current page
            for row in rows:
                try:
                    player_name = row.find_element(By.CLASS_NAME
                       , "player-link").text.strip()
                    cells = [cell.text.strip() for cell in row.
                       find_elements(By.TAG_NAME, "td")]
                    if cells:
                        all_data.append([player_name] + cells)
                except Exception as e:
                    print(f"Error extracting row: {e}")

            # Locate the "next" button by ID
            try:
                next_button = WebDriverWait(driver, 10).until(
                    EC.element_to_be_clickable((By.ID, "next"))
                )
```

```
                # Check if the button is disabled (end of pagination)
                if "disabled" in next_button.get_attribute("class"):
                    print("Reached the last page.")
                    break
                # Scroll into view and click using JavaScript
                driver.execute_script("arguments[0].scrollIntoView(true);", next_button)
                time.sleep(1)
                driver.execute_script("arguments[0].click();", next_button)
                click_count += 1
                print(f"Clicked 'next' button {click_count} times.")
                time.sleep(5)  # Wait for the next page to load
            except Exception as e:
                print(f"Pagination error or no more pages: {e}")
                break

        # Extract headers
        headers = ["Player"] + [header.text.strip() for header in driver.find_elements(By.XPATH, "//table/thead/tr/th")]

        # Match headers with data
        for row in all_data:
            while len(row) < len(headers):
                row.append("")  # Pad missing columns

        # Convert to DataFrame
        df = pd.DataFrame(all_data, columns=headers)
        return df
    except Exception as e:
        print(f"Error occurred: {e}")
        return None
    finally:
        driver.quit()

if __name__ == "__main__":
    print("Fetching Premier League player statistics from WhoScored...")
    df = fetch_whoscored_data(URL)
    if df is not None:
        df.to_csv("WhoScored_Full_Player_Stats.csv", index=False)
        print("Data saved to WhoScored_Full_Player_Stats.csv")
    else:
        print("No data fetched.")
```

```
386
387
388 Pre Dataset: all the codes that were used for forming , merging,
        normalizing etc are shown here is an entire example for
    Serie-A Pre Dataset.
389
390 #Serie A
391 import pandas as pd
392
393 # Load Next Matches Data (Fixtures)
394 NEXT_MATCHES_PATH = "C:/Users/ASUS/Next_Serie_A_Matches.csv"
395 matches_df = pd.read_csv(NEXT_MATCHES_PATH)
396
397 from datetime import datetime
398
399
400 # Extract day & month only (ignore time)
401 matches_df["Match_Date"] = matches_df["Time"].apply(lambda x: x.
    split(" ")[0])  # Keep only "d.m" part
402
403 # Convert to datetime using the current year
404 matches_df["Match_Date"] = pd.to_datetime(matches_df["Match_Date
    "] + "2025", format="%d.%m.%Y")  # Use 2025 as the year
405
406 # Get today's date for comparison
407 today = datetime.today().date()
408
409 #  Filter only future matches
410 matches_df = matches_df[matches_df["Match_Date"].dt.date >=
    today]
411
412
413 # Rename for Consistency and Drop Unnecessary Columns
414 matches_df.rename(columns={"Home Team": "HomeTeam", "Away Team":
        "AwayTeam"}, inplace=True)
415 matches_df.drop(columns=["Match Link", "Time", "Match_Date"],
    inplace=True)  # Exclude unnecessary columns
416
417 # Load and Merge Factor Datasets
418 FACTOR_FILES_DIR = {
419     "Ground_Score": "C:/Users/ASUS/team_data/Serie
          A_Team_Ground_Score.csv",
420     "Total_Team_Rating": "C:/Users/ASUS/team_data/Serie
          A_Team_Ratings.csv",
421     "Adjustment": "C:/Users/ASUS/team_data/Serie
          A_Newly_Promoted_Adjustment.csv",
422     "Bottom4_Impact": "C:/Users/ASUS/team_data/Serie
          A_Bottom_4_Impact.csv",
423     "Clean_Sheets": "C:/Users/ASUS/team_data/Serie
          A_Clean_Sheets.csv",
424     "Goal_Difference": "C:/Users/ASUS/team_data/Serie
```

```
        A_Team_Goal_Difference.csv",
425     "Goals_Against_Top_6": "C:/Users/ASUS/team_data/Serie
        A_Goals_Against_Top_6.csv",
426     "Latest_Form": "C:/Users/ASUS/team_data/Serie
        A_Team_Latest_Form.csv",
427     "Head_to_Head_Scores": "C:/Users/ASUS/team_data/Serie
        A_Head_to_Head_Scores.csv",
428     "Favouritsm_points": "C:/Users/ASUS/team_data/Serie
        A_Team_Favouritsm_Points.csv",
429     "Ratio_Loss_Win": "C:/Users/ASUS/team_data/Serie
        A_Team_Wins_vs_Opponent_Loss.csv"
430
431 }
432
433 # Define mapping if necessary
434 team_name_mapping = {}
435
436 # Apply the mapping before merging
437 for factor_name, file_path in FACTOR_FILES_DIR.items():
438     df = pd.read_csv(file_path)
439     if "Team" in df.columns:
440         df["Team"] = df["Team"].replace(team_name_mapping)
441         df.to_csv(file_path, index=False)  # Save the updated
            version
442     if "Squad" in df.columns:
443         df["Squad"] = df["Squad"].replace(team_name_mapping)
444         df.to_csv(file_path, index=False)  # Save the updated
            version
445
446 # Initialize final dataset
447 final_data = []
448
449 # Iterate over matches and merge factor scores
450 for _, match in matches_df.iterrows():
451     home_team = match["HomeTeam"]
452     away_team = match["AwayTeam"]
453
454     for factor_name, file_path in FACTOR_FILES_DIR.items():
455         factor_df = pd.read_csv(file_path)
456
457         # Ensure 'Team' column exists for merging
458         if "Team" in factor_df.columns:
459             home_score_series = factor_df.loc[factor_df["Team"]
                == home_team, factor_name]
460             away_score_series = factor_df.loc[factor_df["Team"]
                == away_team, factor_name]
461
462             # Extract values safely (handling empty Series)
463             home_score = home_score_series.iloc[0] if not
                home_score_series.empty else 0
464             away_score = away_score_series.iloc[0] if not
```

```
                away_score_series.empty else 0

            # Ensure Home Advantage Score is assigned ONLY to
                Home Team
            if factor_name == "Home_Advantage":
                away_score = 0  # No advantage for away team

            final_data.append([home_team, away_team, factor_name
                , home_score, away_score])
        if "Squad" in factor_df.columns:
            home_score_series = factor_df.loc[factor_df["Squad"]
                == home_team, factor_name]
            away_score_series = factor_df.loc[factor_df["Squad"]
                == away_team, factor_name]

            # Extract values safely (handling empty Series)
            home_score = home_score_series.iloc[0] if not
                home_score_series.empty else 0
            away_score = away_score_series.iloc[0] if not
                away_score_series.empty else 0

            # Ensure Home Advantage Score is assigned ONLY to
                Home Team
            if factor_name == "Home_Advantage":
                away_score = 0  # No advantage for away team

            final_data.append([home_team, away_team, factor_name
                , home_score, away_score])

# Convert to DataFrame
final_df = pd.DataFrame(final_data, columns=["HomeTeam", "
    AwayTeam", "Factor", "Home Value", "Away Value"])

# Save Final Merged Dataset
FINAL_DATASET_PATH = "C:/Users/ASUS/team_data/Serie
    A_Final_Merged_Dataset.csv"
final_df.to_csv(FINAL_DATASET_PATH, index=False)

print("Final dataset successfully merged and saved!")


# Define file path
FINAL_DATASET_PATH = "C:/Users/ASUS/team_data/Serie
    A_Final_Merged_Dataset.csv"

# Load the dataset
df = pd.read_csv(FINAL_DATASET_PATH)

# Pivot the dataset to the desired format
pivoted_df = df.pivot(index=["HomeTeam", "AwayTeam"], columns="
    Factor", values=["Home Value", "Away Value"])
```

```
503
504 # Flatten the MultiIndex columns and rename them appropriately
505 pivoted_df.columns = [f"{col[1]}_{col[0]}" for col in pivoted_df
    .columns]
506 pivoted_df.reset_index(inplace=True)
507
508 # Save the pivoted dataset
509 OUTPUT_FILE_PATH = "C:/Users/ASUS/team_data/Serie
    A_Final_Pivoted_Dataset.csv"
510 pivoted_df.to_csv(OUTPUT_FILE_PATH, index=False)
511
512 print(f"Pivoted dataset saved successfully at: {OUTPUT_FILE_PATH
    }")
513
514 # Define file path
515 FINAL_PIVOTED_DATASET_PATH = "C:/Users/ASUS/team_data/Serie
    A_Final_Pivoted_Dataset.csv"
516
517 # Load the dataset
518 df = pd.read_csv(FINAL_PIVOTED_DATASET_PATH)
519
520 # Extract home and away factor columns separately
521 home_columns = [col for col in df.columns if col.endswith("_Home
     Value")]
522 away_columns = [col.replace("_Home Value", "_Away Value") for
    col in home_columns]
523
524 # Ensure the proper ordering
525 final_columns = ["HomeTeam", "AwayTeam"]
526 for home, away in zip(home_columns, away_columns):
527     final_columns.extend([home, away])
528
529 # Reorder DataFrame
530 df = df[final_columns]
531
532 # Save the reordered dataset
533 OUTPUT_FILE_PATH = "C:/Users/ASUS/team_data/Serie
    A_Final_Reordered_Pivoted_Dataset.csv"
534 df.to_csv(OUTPUT_FILE_PATH, index=False)
535
536 print(f"Reordered pivoted dataset saved successfully at: {
    OUTPUT_FILE_PATH}")
537
538
539
540 #Serie A
541 import pandas as pd
542 import numpy as np
543
544 # Define File Paths
545 POINTS_TABLE_PATH = "C:/Users/ASUS/SerieA_PointsTable_with_Form.
```

```
    csv"
546 FINAL_DATASET_PATH = "C:/Users/ASUS/team_data/Serie
    A_Final_Reordered_Pivoted_Dataset.csv"
547 MATCH_FIXTURES_PATH = "C:/Users/ASUS/Next_Serie_A_Matches.csv"
548
549 team_name_mapping = {}
550
551 df = pd.read_csv(POINTS_TABLE_PATH)
552 if "Team" in df.columns:
553     df["Team"] = df["Team"].replace(team_name_mapping)
554     df.to_csv(POINTS_TABLE_PATH, index=False)  # Save the
        updated version
555 if "Squad" in df.columns:
556     df["Squad"] = df["Squad"].replace(team_name_mapping)
557     df.to_csv(POINTS_TABLE_PATH, index=False)  # Save the
        updated version
558
559 # Load datasets
560 points_table = pd.read_csv(POINTS_TABLE_PATH, usecols=["Squad",
    "GF"])  # Load only 'Squad' and 'GF'
561 final_dataset = pd.read_csv(FINAL_DATASET_PATH)
562 match_fixtures = pd.read_csv(MATCH_FIXTURES_PATH, usecols=["Home
     Team", "Away Team"])
563
564 # Rename columns for consistency
565 match_fixtures.rename(columns={"Home Team": "HomeTeam", "Away
    Team": "AwayTeam"}, inplace=True)
566 points_table.rename(columns={"Squad": "Team", "GF": "
    Goals_Scored"}, inplace=True)  # Rename GF to Goals_Scored
567
568 # Merge GF with HomeTeam and AwayTeam
569 merged_dataset = final_dataset.merge(points_table, left_on="
    HomeTeam", right_on="Team", how="left")
570 merged_dataset.rename(columns={"Goals_Scored": "
    Home_Team_Goals_Scored"}, inplace=True)
571 merged_dataset.drop(columns=["Team"], inplace=True)  # Drop
    redundant column
572
573 merged_dataset = merged_dataset.merge(points_table, left_on="
    AwayTeam", right_on="Team", how="left")
574 merged_dataset.rename(columns={"Goals_Scored": "
    Away_Team_Goals_Scored"}, inplace=True)
575 merged_dataset.drop(columns=["Team"], inplace=True)  # Drop
    redundant column
576
577 #  Ensure numeric values are properly formatted
578 for col in merged_dataset.columns:
579     if col not in ["HomeTeam", "AwayTeam"]:
580         merged_dataset[col] = pd.to_numeric(merged_dataset[col],
              errors='coerce').fillna(0)
581
```

```
582 #  Save the final cleaned dataset
583 FINAL_CLEANED_DATASET_PATH = "C:/Users/ASUS/team_data/Serie
    A_Final_Cleaned_Dataset.csv"
584 merged_dataset.to_csv(FINAL_CLEANED_DATASET_PATH, index=False)
585
586 print(" Final cleaned dataset created successfully! (With Goals
    Scored - GF)")
587
588 #Serie A
589 import pandas as pd
590 import re
591
592 # Load Player Stats Dataset
593 PLAYER_STATS_PATH = "C:/Users/ASUS/WhoScored_Serie
    A_Full_Player_Stats.csv"  # Update path if needed
594 df_players = pd.read_csv(PLAYER_STATS_PATH)
595
596
597
598 # Function to Extract Player Name & Team Name
599 def extract_player_team(value):
600     """Extracts player name and team name correctly from the
        player stats column."""
601     try:
602         lines = str(value).split("\n")  # Split by newline
603         if len(lines) >= 3:  # Ensure there are at least 3 parts
604             player_name = lines[1].strip()  # The second value
                is the correct player name
605             team_info = lines[2].split(",")  # The third value
                contains the team
606             team_name = team_info[0].strip() if len(team_info) >
                 0 else None  # Extract first value as team name
607             return player_name, team_name
608     except Exception:
609         return None, None
610     return None, None
611
612 # Apply Extraction
613 df_players[['Player', 'Team']] = df_players.iloc[:, 1].apply(
    lambda x: pd.Series(extract_player_team(str(x))))
614
615 #  Remove rows where no player or team was found
616 df_players = df_players.dropna(subset=["Player", "Team"])
617
618 # Apply Team Name Mapping
619 team_name_mapping = {}
620 df_players["Team"] = df_players["Team"].replace(
    team_name_mapping)
621
622 # Select Required Columns
623 df_cleaned = df_players[["Player", "Team", "Goals", "SpG", "
```

```
    Rating"]].copy()
624
625 # Save as Excel File
626 CLEANED_PLAYER_STATS_PATH = "C:/Users/ASUS/team_data/Serie
    A_Cleaned_Player_Stats.xlsx"
627 df_cleaned.to_excel(CLEANED_PLAYER_STATS_PATH, index=False,
    engine="openpyxl")
628
629 print(f"Player stats successfully cleaned & saved to {
    CLEANED_PLAYER_STATS_PATH}!")
```