University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project
7PAM2002-0509-2023
Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

IPL Data Analysis and 2025 Winner Prediction Model

Student Name and SRN:

Bhavinkumar Harkhani 22079315

Supervisor: Will Cooper

Date Submitted: 29th Aug, 2024

Word Count: 7205

DECLARATION STATEMENT

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

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

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged.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

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

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

Student Name printed: Bhavinkumar Subhashbhai Harkhani

Student Name signature: [illegible]

Student SRN number: 22079315

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

Acknowledgments

I want to say Special thanks to my supervisor Will Cooper for his guidance, support, and very helpful feedback during the research. I also thank my colleagues and friends for their encouragement and help. Also, I want to show my appreciation to the institutions and organizations that gave resources and facilities needed to successfully complete this project.

Abstract

The main goal of my project is to predict the outcomes of IPL (Indian Premier League) matches, so we can eventually predict who will win the 2025 season. I used the LightGBM (Light Gradient-Boosting Machine) classifier to predict the results of IPL matches in the 2025 season. I also used the Random Forest classifier to compare how accurate the predictions are between these two machine learning classifiers. For this project, I used the dataset from the Kaggle website and looked at player and team performances, using feature engineering techniques like encoding categorical variables, scaling features, and handling missing data. I also added important features like the venue, toss decision, and past win percentages to make predictions more accurate. The LightGBM model was tuned using grid search and cross-validation to improve the hyperparameters, which gave a high accuracy of 0.96 on the test set. On the other hand, the Random Forest classifier, although strong, got a lower accuracy of 0.76. This project shows how machine learning models, especially LightGBM, can be effective in predicting sports outcomes and also points out the limitations of these classifiers, like being sensitive to class imbalance and needing regular updates to keep up with changes in team dynamics.

Table of Contents

01. Introduction

The Indian Premier League (IPL) has become a big international sports event since it started in 2008. By combining excitement of cricket with big money and entertainment, the IPL gained millions of fans from around the world and earns a lot of money. The presence of international cricket stars and the unique T20 format have made the IPL a much awaited annual event. Because of the high stakes both money and excitement fans, experts, and bettors are more interested in predicting match outcomes.

Predictive analysis in sports has become important field of research due to the rise of data-driven decision-making. Accurate match predictions in the IPL offer more than just fun. They can influence betting markets, increase fan interest, and affect team strategies. This field has changed a lot with the rise of machine learning (ML), which helps to process and analyze large amounts of data, find patterns, and make predictions that are better than those from older methods.

Using machine learning techniques more especially, the LightGBM classifier this research aims to forecast the results of IPL matches from 2008 to 2024. The goal of the study is to improve the model's predictive power by utilizing feature engineering. The final aim is to see if machine learning can accurately predict IPL match winners, giving valuable insights to both industry experts, stakeholders and the cricket community.

The Indian Premier League (IPL) gained millions of fans and big financial investments, making it one of the largest and most-watched cricket leagues in the world. Predicting IPL match results is tough but valuable because T20 cricket is very unpredictable. Old prediction methods, which often use basic stats or personal opinions, don't fully understand the game's complexity, where many changing factors impact the result. Even though machine learning for sports prediction has been studied a lot, many current models don't use advanced methods like LightGBM or detailed feature engineering. This shows the need for better models that can give more accurate predictions and deeper insights.

The main goal of this project is to build a reliable machine learning model that can predict IPL match results accurately using historical data from 2008 to 2024. The study focuses on improving prediction by combining detailed feature engineering with the powerful gradient boosting model LightGBM classifier. The research aims to show how advanced machine learning methods can handle complex data and improve sports analytics, as well as enhance current prediction models. Ultimately, the goal is to build a model that can be a dependable tool for teams, analysts, and bettors who want to gain a competitive advantage.

02. Background & Literature Review

The Indian Premier League has become a famous sports event with huge popularity and financial success. Fans, bettors and team's analysts have shown interest in making predictions about the results of IPL matches. Recently, various methods have been explored for predicting these results using machine learning and looking at many factors that impact the outcomes. Like the Bollywood of cricket, the IPL is full of drama, excitement, and surprises. Since it started in 2008, it has turned into a global sports event, attracting millions of viewers and making billions of dollars. With so much at stake, predicting IPL match results has become popular among passionate fans and those looking to make quick money.

In recent years, researchers have been using machine learning, a type of artificial intelligence, to understand IPL match predictions. It's like teaching a computer to play cricket and then asking it to choose the winner. This approach has shown good results because computers can analyze huge amounts of data and spot patterns that the human eye would miss.

In sports analytics, machine learning has become more popular. For example, in cricket, ML is used to predict match results, player performance, and strategies. Research has shown that machine learning models like Random Forests and Decision Trees can analyze large datasets to predict match outcomes effectively. Sharma and Kaur, for instance, showed how these models can assess various types of data, such as player stats, team composition, and match conditions, to predict winners with good accuracy. These models work like a panel of experts, with each "tree" in a Random Forest model making an independent prediction that is combined to produce the final result.

A study titled "Predictive Analysis of IPL Match Winner using Machine Learning Techniques" (2024) focused on predicting match winners with machine learning models like Random Forest and Decision Tree. To train the models, the research used data including player performance, team strength, and historical data. The results showed that these machine learning algorithms can predict IPL match results accurately, offering valuable insights to fans, analysts, and experts. Decision Trees and Random Forests like a group of knowledgeable cricket experts who come together to make a decision. Each expert (or tree, in the case of Random Forest) looks at various game factors, such as team strength, player performance, and past matches. The final prediction is based on the majority vote of these experts. The study suggests that these algorithms can provide a significant edge to fans and bettors by accurately forecasting IPL match outcomes (Sharma et al., 2024).

Many studies have increased the use of machine learning in IPL predictions by adding different factors and using various ML models. Kumar and Jaiswal (2024) used Support Vector Machines (SVM), Random Forests, and Logistic Regression to create a model that can predict IPL match results. Their study showed how important it is to include many factors to make the models better at predicting, like player performance, match conditions, and venue details. They were able to improve prediction accuracy by using multiple algorithms, showing that using different methods is needed for successful match outcome prediction (Kumar et al., 2024).

A different study, "Utilizing Machine Learning for Comprehensive Analysis and Predictive Modelling of IPL-T20 Cricket Matches" (2024), used several machine learning models, such as Support Vector Machines (SVM), Random Forest, and Logistic Regression, to take a more thorough approach. To forecast match outcomes, this study examined a broader variety of variables, including player data, match circumstances, and venue features. The results demonstrated how crucial it is to use a variety of models and consider several variables to increase forecast accuracy (Kumar et al., 2024)

To improve prediction models, recent developments have focused on incorporating real-time data and using deep learning strategies. Yadav et al. (2023) explored the use of deep learning models, like neural networks, to forecast IPL outcomes based on real-time data collected during matches. Their research showed that deep learning models not only offer more accurate score predictions but also provide real-time insights, which are essential for making tactical decisions during gameplay. The study highlights how deep learning can uncover complex patterns in data that traditional machine learning algorithms might miss (Yadav.P et al., 2023).

A 2023 study titled "IPL Score Prediction & Analysis" looked into predicting IPL scores using both deep learning and machine learning techniques. The study emphasized the importance of advanced algorithms and real-time data in making accurate predictions. By integrating real-time match data and using advanced algorithms, the research aimed to improve the accuracy of score forecasts, offering valuable insights to spectators and analysts during a game (Yadav.P et al., 2023). But they aren't just focusing on picking the winner. Knowing the final score of a match can also be important. The study "IPL Score Prediction & Analysis" published in 2023, investigated how to predict IPL scores using deep learning and machine learning techniques. Deep learning, which can analyze even more complex patterns in data than traditional machine learning, was highlighted as a kind of upgraded version of machine learning. The study also stressed the importance of real-time data-information that updates as the match progresses. By using sophisticated algorithms and real match data, they aimed to provide more accurate score predictions, which can be helpful for in-game strategies and live betting.

The use of machine learning to forecast IPL results has advanced, but there is still a large research gap in integrating feature engineering with more modern and complex models such as LightGBM. While many studies, like the one by Kumar and Jaiswal (2024), have mainly concentrated on conventional machine learning algorithms, they frequently fail to fully utilize feature engineering's potential to improve model performance, even though they have successfully demonstrated the significance of a variety of variables.

Yadav et al. (2023) incorporated real-time data into prediction models and there is still much to explore when applying similar methods to historical data, Specially with models like LightGBM. While LightGBM is renowned for its effectiveness and efficiency in classification tasks, it has not been extensively used in IPL prediction models, particularly when combined with engineered features that could provide more nuanced insights into match outcomes.

These research efforts highlight how machine learning is beginning to transform the IPL prediction landscape. By analyzing huge amounts of data and uncovering hidden patterns, these models can offer valuable insights and predictions that fans, analysts, and even teams can use to make more informed decisions. As technology continues to evolve, we can expect the development of increasingly sophisticated models, making IPL predictions even more exciting and accurate.

these research articles show how much interest there is in using machine learning methods to forecast IPL match results and scores. To increase forecast accuracy, the research emphasize how crucial it is to consider a variety of parameters, use a variety of models, and include real-time data. The results of these research give insightful information and possible applications in domains including fantasy sports, betting, and team strategy, which has consequences for IPL enthusiasts, analysts, and stakeholders.

By collecting data from 2008 to 2024 to forecast IPL match results, this study aims to address the identified research gap. The LightGBM classifier will be used for this purpose. To create new variables that better capture the nuances of cricket matches, the project will make extensive use of feature engineering. This approach is expected to enhance the predictive capacity of the LightGBM model, which has not been extensively explored in IPL predictions. The combination of LightGBM and feature engineering is anticipated to produce a prediction model that is both more insightful and accurate than previous research that relied on more traditional machine learning techniques.

This research adds to the body of literature by concentrating on the development of characteristics that can have a major influence on prediction accuracy in addition to utilizing a model that is comparatively underrepresented in this area. By doing this, it hopes to further the field of sports analytics and provide more trustworthy resources for IPL match prediction.

03. Research question & objectives

3.1 Research Question

Can machine learning, specifically the LightGBM classifier with feature engineering, be effectively used to accurately predict the winner of IPL matches from 2008 to 2024?"

3.2 Objectives

Exploratory Data Analysis (EDA): I will conduct EDA to understand the dataset and figure out which attributes are most important for predicting match results.

Feature Engineering: I will develop new features by focusing on elements that are most likely to affect match outcomes, aiming to enhance the model's predictive capabilities.

Model Training and Evaluation: I will train a LightGBM classifier on the dataset and evaluate its performance using metrics like accuracy, precision, and recall.

Interpretation of Results: I will analyze feature importance to determine which factors have the most impact on match outcomes, as part of the project's conclusion.

The IPL represents a high-stakes environment where accurate match predictions can have significant implications. This project utilizes a comprehensive IPL dataset covering matches from 2008 to 2024. The LightGBM classifier was chosen for its efficiency in handling classification tasks and its ability to process large datasets effectively. Through rigorous data analysis and feature engineering, the project aims to develop a robust model that can provide accurate predictions of IPL match outcomes, offering valuable insights to both academic and industry audiences.

04. Database

The database I have used is consists two CSV files named matches.csv and deliveries.csv. The matches file has 1095 rows and 20 columns of data represents states of every single game while deliveries file has 26092 rows and 17 columns which represents ball by ball data of every single ball delivered in the IPL from 2008 to 2024. There are various data types of this data like integer, float and Object. Since it doesn't contain any personal information about individuals, it doesn't require ethical approval. I got this data from the Kaggle website, this data is originally sourced from cricsheet.org. This data might have been recorded to track and evaluate teams and players performance.

The Indian Premier League (IPL) is invented in India and is managed by the BCCI (Board of Control for Cricket in India). The data was collected from IPL games, and I chose this dataset even though there are many IPL datasets available online because this one is fully updated to the present date. My research question is to predict the major winners of the next IPL season, and with this fully updated data, I can train my model in the best possible way.

After reviewing the data, I performed a cleaning process. I dropped rows with missing values in the 'winner' column and removed unwanted columns from the matches data, such as 'id,' 'city,' and 'method.' I also replaced missing values in the 'player of the match' column with 'unknown'.

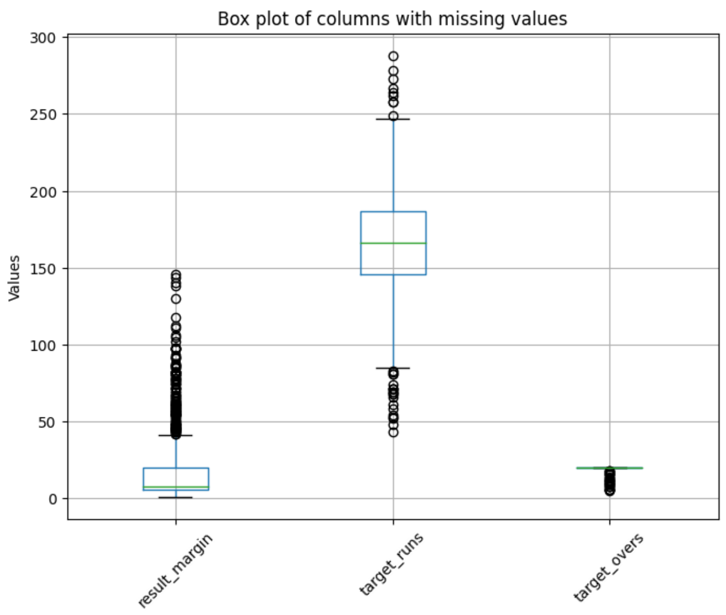


Figure 01 Box plot of columns with missing values

I replaced any 'NaN' values in columns like 'result_margin,' 'target_runs,' and 'target_overs' with the median values of those columns because they had outliers. Before cleaning the data there was 1095 rows in the matches dataset but after cleaning there's 1090 rows non-null complete data.

05. Ethical Issue

The dataset from Kaggle has cricket match stats like match, team, and player info. It does not have personal or sensitive data like names or contact info. The data is about professional cricket, and player info is shown in a way that keeps privacy safe. The data does not fall under GDPR since it has no personal data from EU people. It's publicly available cricket info, so it does not need special GDPR protection. UH approval is not needed because there's no personal data collection or surveys. The data is from Kaggle, which is free for research. No sensitive info or human interaction required.

The dataset I used for this research is free on Kaggle with clear usage rights. It's for research and educational use. No payment was needed, and it has an open license. The data is public on Kaggle, which hosts reliable sources. The IPL match records are publicly reported, so the data was collected ethically, with proper permissions. The data is from Kaggle, a trusted platform used for research. The IPL data is from official match records, so it's reliable and credible.

The data was used ethically, following all rules. No personal data was involved, and the project respected ethical considerations.

06. Methodology

This project was developed in Google Colab using Python, with the help of various libraries for model building and data analysis. The main libraries I used include numpy, pandas, and matplotlib for data manipulation, visualization, and working with DataFrames. LightGBM, scikit-learn, and seaborn were used for creating models, evaluating them, and visualizing the results. I also simulated future match data using numpy and pandas to generate random data and manage it.

The first step was to prepare and preprocess the dataset. This involved loading cricket match data and calculating important statistics like win percentages for each team. After that, I encoded categorical variables like team names, venues, and toss decisions using label encoding so that they could be used in model training. I created features that were important for predicting match outcomes, such as the win percentages of the teams, the number of matches they played, and whether the toss winner also won the match. These features were carefully chosen and added to the final set of features used in the model. Then, I moved on to model building, where I used the LightGBM model because it is efficient and performs well with large datasets that have multiple classes. I tuned the hyperparameters to improve model accuracy and speed. I also tried other models like Random Forest, Decision Tree, and a Voting Classifier to increase prediction accuracy through ensemble learning. The data was divided into training and testing sets, and I applied feature scaling to ensure consistency and enhance model performance. The models were trained on these datasets, and the parameters were adjusted to achieve the best accuracy and efficiency. After training the models, I tested them on the test dataset to see how well they performed. The results were visualized using confusion matrices and feature importance plots to understand the model's behaviour and accuracy. I simulated future cricket matches, particularly for the 2025 season, using the models I developed to predict the outcomes. This involved generating random match data, applying the same preprocessing and feature engineering steps, and then using the trained models to predict the match results.

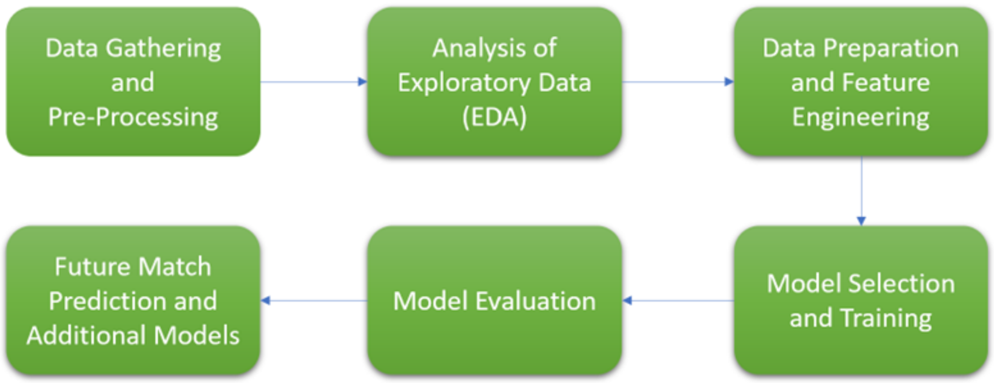

Data Gathering and Pre-Processing
Analysis of Exploratory Data (EDA)
Data Preparation and Feature Engineering
Model Selection and Training
Model Evaluation
Future Match Prediction and Additional Models

Figure 02 Flowchart of Process Methodology

6.1 Data Gathering and Pre-Processing

The dataset for this project, obtained from Kaggle, includes IPL matches played between 2008 and 2024. This extensive dataset contains details on player statistics, team lineups, venues, match outcomes, and other relevant match information. During the data collection phase, this dataset needed to be retrieved and prepared for analysis.

An overview of the dataset

• The dataset include details on matches, deliveries (ball-by-ball statistics), teams, and players. It is an extensive compilation of Indian Premier League (IPL) match data from 2008 to 2024.
• It is probable that the initial intent of gathering this data was to track and evaluate the IPL teams' and players' performances.
• Format: The CSV (Comma-Separated Values) format of the dataset is supplied.
• Size: The dataset has 2 csv files, a total size of about 2 MB.
• Records: One of the 2 file named 'deliveries.csv' has 26092 rows and 17 columns which represents specifics data on every ball bowled and another file named 'matches.csv' has 1095 rows and 20 columns of data represents different match statistics, are included in the dataset.

Data Cleaning

Data Cleaning was the first step in the data preparation process. It involved handling missing values, inconsistent data, and unnecessary data in the raw dataset. Missing data were Filled with right method, or rows NaN values were removed. Duplicates were identified and removed to ensure the integrity of the data. I have removed rows from the matches dataset where the 'winner' column had missing values (NaN). This ensures only rows with valid data in the 'winner' column are left. Filling Missing Values in 'player_of_match': For the 'player_of_match' column, I replaced the missing values with 'Unknown'. This helps in handling missing data without losing any rows. I have removed the 'id', 'city', and 'method' columns from the dataset as they were not needed for further analysis. I plotted box plots for columns 'result_margin', 'target_runs', and 'target_overs' to visually inspect the data distribution and identify any outliers. This is important to decide the right method for filling in missing values. Filling Missing Values with Median: For the columns 'result_margin', 'target_runs', and 'target_overs', I replaced the missing values with the median of each column. i preferred to use the median here because it helps to handle outliers effectively. Finally, I checked the number of unique values in each column using matches.nunique() to understand the diversity of data in each column.

Converting Data

After cleaning the data, I formatted it so it could be analyzed. This included using label encoding methods to convert categorical data like team names and venues into numerical forms. These changes were needed to prepare the data for machine learning models, which require numerical input.

6.2 Analysis of Exploratory Data (EDA)

The purpose of the exploratory data analysis (EDA) was to understand the dataset's properties and distribution. During the EDA process, I looked at key indicators like winning ratio, individual player performances, and team statistics. I used various graphs and summary statistics to find patterns and trends in the data. I examined how match victories were distributed across different teams, the effect of the toss, and and the effect of playing games at different venues on game results.
This stage was important for finding potential features that could make the model's predictions better. The insights gained from EDA helped guide the feature engineering process, making sure the most important variables were included in the model.

6.3 Data Preparation and Feature Engineering for model

Date Features Extraction:
I converted the date column to a datetime format to extract meaningful time-based features. I extracted year, month, and day from the date column to capture patterns and trends. After extraction, I removed the original date column to reduce redundancy.

Season Splitting:
I created a function to split the season column into two new columns: season_start and season_end. This function handles different formats of season data and converts them into year values. The start and end years of each season were converted to datetime format to facilitate time-based analyses. I removed The original season column to avoid duplication.

Team Name Standardization:
Over the years since the IPL began in 2008, many franchises have changed their team names. So I have standardized team names by mapping old names to their current names. This ensures consistency across the dataset and simplifies team-based analyses.

Team Performance Metrics:
I calculated the total number of matches each team played and won. These metrics give a clear picture of each team's performance. I also calculated the win percentage for each team to see their overall success rate. I summed up the total runs scored and wickets taken by each team to measure their strengths.

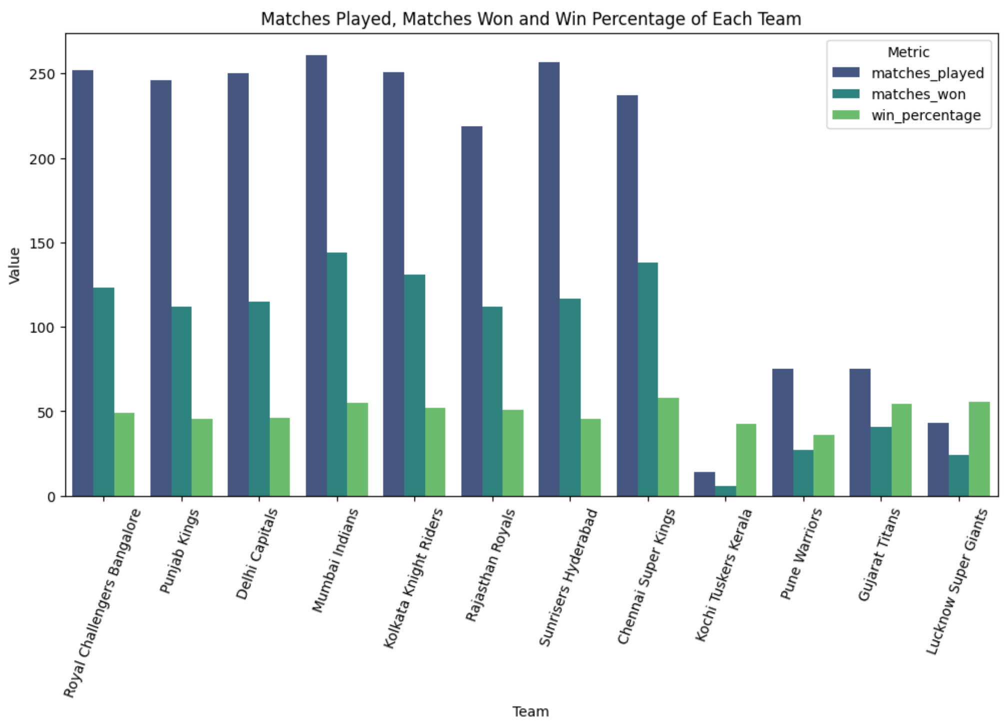


Figure 03 Matches Played, Matches Won and Win Percentage of Each Team

Player Statistics:

I generated detailed statistics for players, such as runs scored, balls faced, batting average, strike rate, wickets taken, and economy rate. These metrics helps me to evaluate individual player performance. I also collected data on catches taken and how many times a player received the "Man of the Match" award. This gives insights of fielding performance and the impact of individual players.

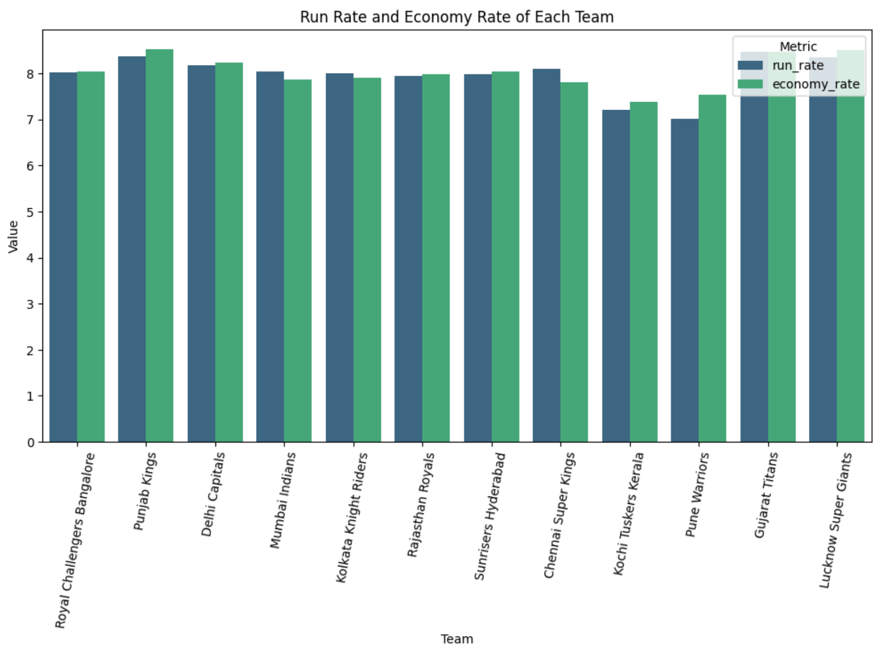


Figure 04 Run Rate and Economy Rate of Each Team

Additional Features:

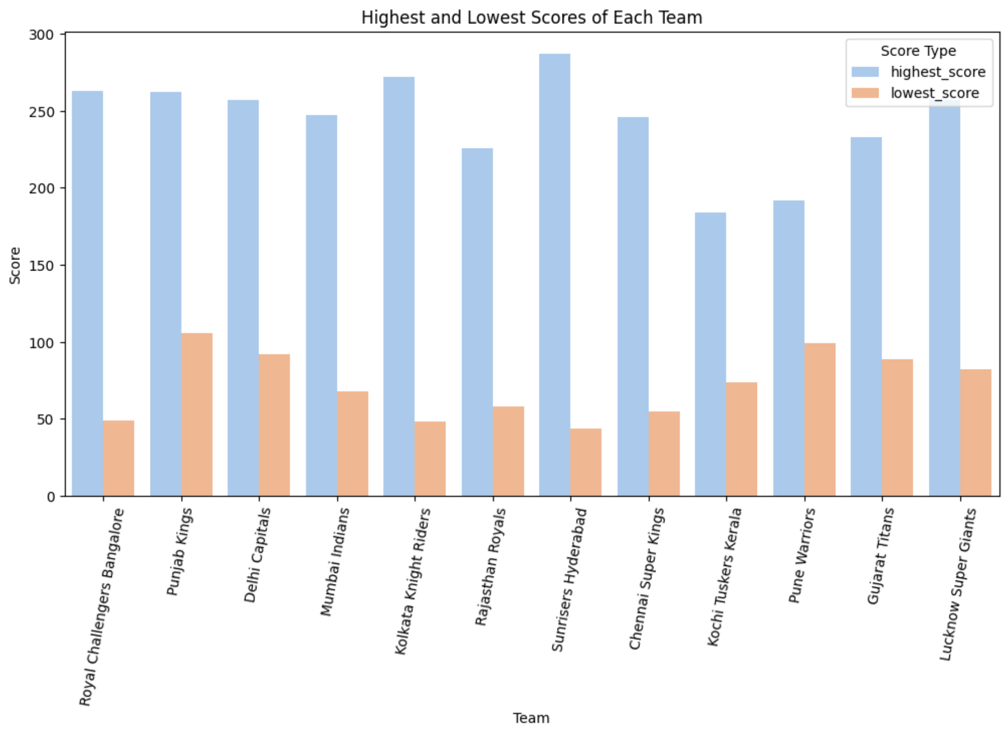


Figure 05 Highest and Lowest Scores of Each Team

I calculated the highest and lowest scores achieved by each team, excluding no-result matches, to see the team's scoring variability.

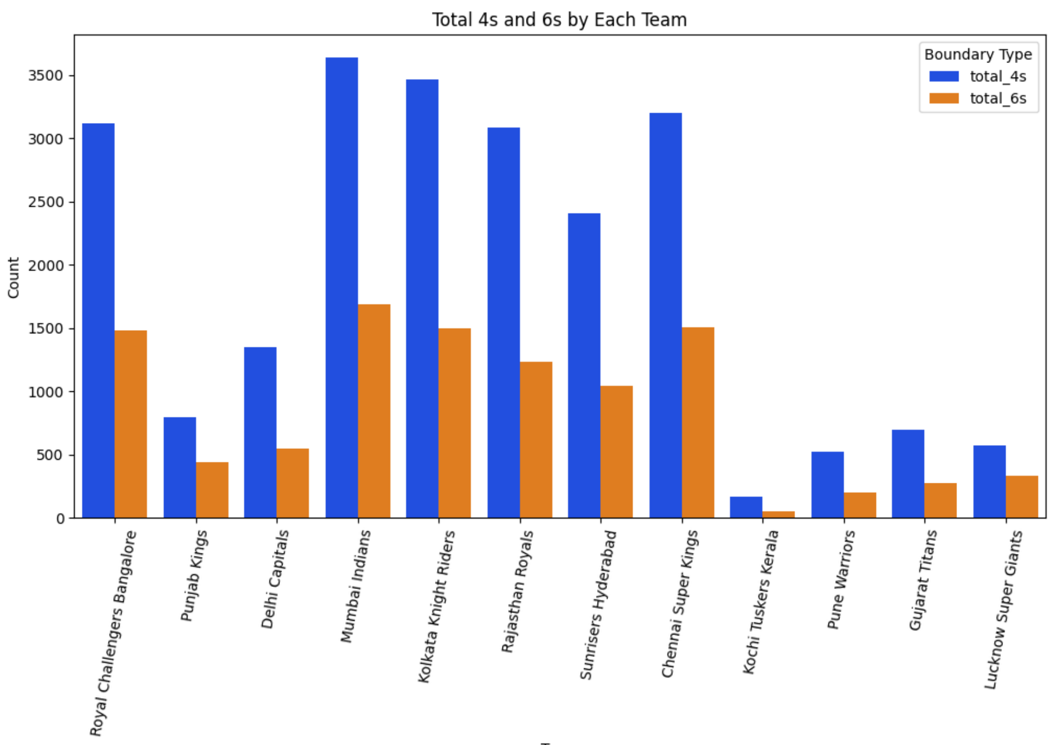


Figure 06 Total 4s and 6s by Each Team

I calculated the total number of boundaries (4s) and sixes (6s) scored by each team to analyze their aggressive play style.

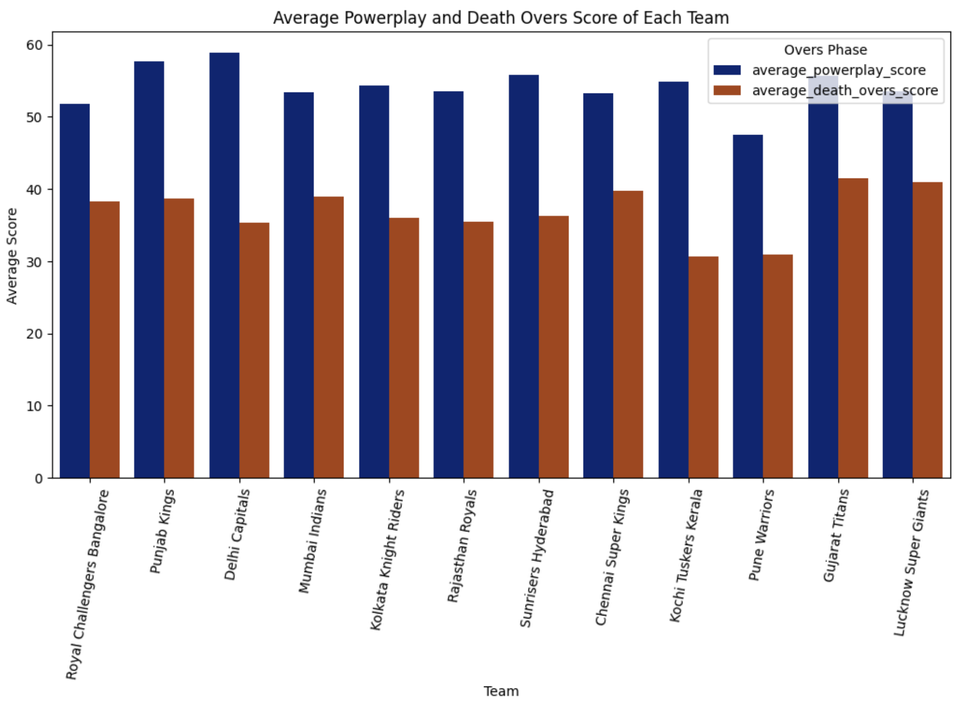


Figure 07 Average Powerplay and Death Overs Score of Each Team

I computed average scores during powerplay and death overs to understand team performance in different match phases.
These features give a full view of how cricket matches work, showing things that normal models might miss. By adding these features, the model can understand match results and player performance better.

A key thing for making the model better at prediction was feature engineering. Using knowledge of the field and insights from EDA, more features were made in this step. Some important features created are:

• Team Strength Index: A combined score that looks at how each player performs in the team and changes based on match situations.
• Player Form: A dynamic factor that gives more importance to recent games and shows how a player has done in the last few matches.
• Toss Impact: This feature looks at things like match time and venue to see how winning the toss has historically affected match results.

I have done Seasonal Analysis in which I have calculated the average of the target_runs column for each season using .agg({'target_runs': 'mean'}). This function aggregates the data to find the average runs scored per match in each season. average runs per match have mostly gone up over the seasons. After some ups and downs in the early years, the runs stayed stable but have gone up fast in the latest seasons, reaching the highest point in 2024.

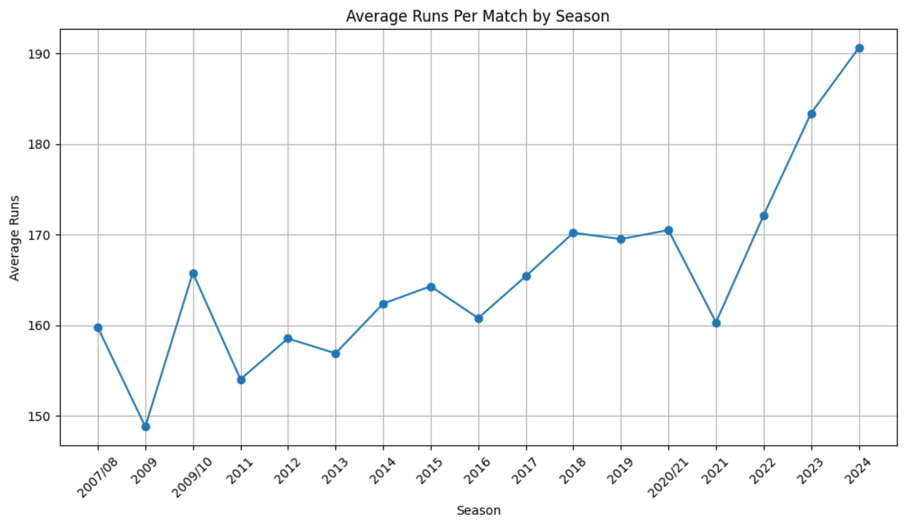


Figure 08 Average Powerplay and Death Overs Score of Each Team

I calculated the win percentage for each team, which was an important feature in the predictive model. I wrote a function named ‘calculate_win_percentage’, that calculates the win ratio by dividing the number of matches won by the total matches played. This feature was added as team1_win_pct and team2_win_pct.

To improve the model's prediction ability, I created new features:

Toss Winner Feature: I made a binary feature, toss_winner_is_match_winner, to see if the team that won the toss also won the match.

Matches Played: I calculated the number of matches each team played using the function calculate_matches_played and added these as team1_matches_played and team2_matches_played.

To handle categorical data, I used LabelEncoder to convert the categorical variables like team1, team2, toss_winner, toss_decision, venue, and winner into numerical values that machine learning models can understand. The encoded values were stored in new columns with _encoded added to their names.

I selected features that I believed would best predict the match outcomes. These included win percentages, toss outcome, year, the number of matches played, and the encoded categorical features. These features were stored in the X variable, while the target variable was stored in y.

I split the dataset into training and testing sets using an 80-20 split with the train_test_split function. This allowed the model to train on most of the data while testing on new examples. Then, I used StandardScaler to normalize the feature values, which is important for models like LightGBM that are sensitive to the scale of the data.

6.4 Model Selection and Training

LightGBM Model:

I selected the LightGBM (Light Gradient Boosting Machine) as the main model because it is efficient at handling large datasets like the cricket match data I am working with. LightGBM is good for classification tasks that involve categorical variables, like team names and venues, and it doesn't need a lot of preprocessing. As the target variable 'winner' is multiclass, which represents different cricket teams, I configured the model with a multiclass objective to manage the multiple classes.

Hyperparameter Configuration:

I tuned the LightGBM model using specific hyperparameters to get the best performance:

- Objective: Set to multiclass to handle multiple classes in the winner variable.
- num_class: Defined as the number of unique teams (len(label_encoders['win-ner'].classes_)) so the model knows how many classes there are.
- Metric: I used multi_logloss as the evaluation metric, which is good for multiclass classification tasks. It checks how well the model predicts the probability distribution

across multiple classes and penalizes wrong predictions based on their confidence level.

- Boosting Type: Set to gbdt (Gradient Boosting Decision Tree), which builds decision trees step by step to correct errors from previous trees, capturing complex patterns in the data.
- num_leaves: Set to 31, this controls how complex each tree can be, helping the model learn complex patterns without overfitting.
- Learning Rate: Set to 0.05, this decides the step size at each iteration, preventing the model from making too big changes that could lead to overfitting.
- Feature Fraction: Set to 0.9, meaning 90% of features are used in each iteration, which helps prevent overfitting by adding some randomness.
- Bagging Fraction and Frequency: With values of 0.8 and 5, these control how the data is subsampled in each iteration, further helping to make the model generalize better and reduce overfitting.

I trained LightGBM model using 1000 boosting rounds, with early stopping set to kick in after 10 rounds of no improvement on the validation set. The best iteration, based on validation performance, was used for the final predictions. The data was scaled using StandardScaler to normalize the features.

Random Forest Classifier:

I chose the Random Forest Classifier for its it is very strong for handling data with many features and also helps in stopping overfitting by using many trees together I did hyperparameter tuning using grid search with cross-validation (cv=3).

n_estimators: Number of trees in the forest, tested with values 100 and 200.

max_depth: Maximum depth of each tree, tested with values 10 and 20 to stop trees from growing too much and causing overfitting.

min_samples_split: Minimum number of samples required to split a node, tested with values 2 and 5, to make sure the splits are important and also to reduce overfitting risks.

I trained the model on the encoded and scaled dataset (X_train_23, y_train_23), and I picked the best model by checking which one had the highest cross-validation score. Then, I used this best model to predict on the test set (X_test_24), it gives a strong baseline to compare with other models.

Decision Tree Classifier:

I used the Decision Tree Classifier because it is simple and easy to understand. Unlike ensemble methods, a decision tree gives a clear picture of how decisions are made, which helps in understanding the model's logic. I trained the Decision Tree on the same encoded and scaled dataset without doing extra hyperparameter tuning, just using its default settings. This gave an easy comparison against the more complex Random Forest and LightGBM models. The tree structure naturally captures interactions between features, but it can overfit, so I used it for a baseline comparison.

Voting Classifier:

I implemented the Voting Classifier to take advantage of both the Random Forest and Decision Tree classifiers. By combining these models, the Voting Classifier tries to balance the high variance of a Decision Tree with the robustness of a Random Forest. I used soft voting, where the classifier predicts the class label based on the average of predicted probabilities from both the Random Forest and Decision Tree models. This approach works well when combining models that perform well individually but have different strengths.

I trained the Voting Classifier on the encoded and scaled training data (X_train_23_encoded, y_train_23_encoded). The final model combined the probabilistic predictions of both the Random Forest and Decision Tree models, making it stronger against the weaknesses of any one model. The model was tested on the 2024 season data (X_test_24_encoded, y_test_24_encoded). I calculated the accuracy and analyzed the distribution of predictions across actual classes to evaluate how well the ensemble model performed. The Voting Classifier aims to give more stable predictions by averaging out the possible errors of the individual models.

6.5 Model Evaluation

Prediction and Accuracy Calculation:

I trained LightGBM model to predict the outcomes of the test set. The model gave probabilities for each class, and I selected the class with the highest probability as the prediction. Then, I calculated the model's accuracy using accuracy_score, which is a simple and easy way to measure classification performance.

Feature Importance:

I extracted and visualized the feature importances from the LightGBM model to see which features were most important for predictions. This analysis helped confirm that the features were relevant and gave insights into what was driving the model's decisions.

Confusion Matrix:

To better understand the model's performance, I created a confusion matrix. This helped me see the model's classification errors and identify teams that were often misclassified, giving ideas for improving the model.

6.6 Future Match Prediction and Additional Models

I made a function called generate_2025_matches to create matches for the 2025 season. This function generated random team pairings, toss decisions, and venues, using the same feature

engineering steps I used on the historical data Then, I used the LightGBM model to predict the outcomes of these simulated matches.
To improve the accuracy of predictions, I also used other models like the Random Forest Classifier and Decision Tree Classifier. These models were trained on features from in-game statistics taken from ball-by-ball data. I combined the Random Forest, and Decision Tree models into a Voting Classifier, which merges predictions from different models to improve overall accuracy. I trained and tested this ensemble model, measuring its accuracy on the test set.

I did hyperparameter tuning for the Random Forest model using GridSearchCV, testing different settings for the number of estimators, maximum depth, and minimum samples split. I picked the best model based on cross-validation accuracy and included it in the ensemble model.

I tested the final ensemble model on the 2024 season data, comparing the predicted match outcomes with the actual results. I looked at how predictions were distributed for each team, which gave insights into how the model performed in different situations.

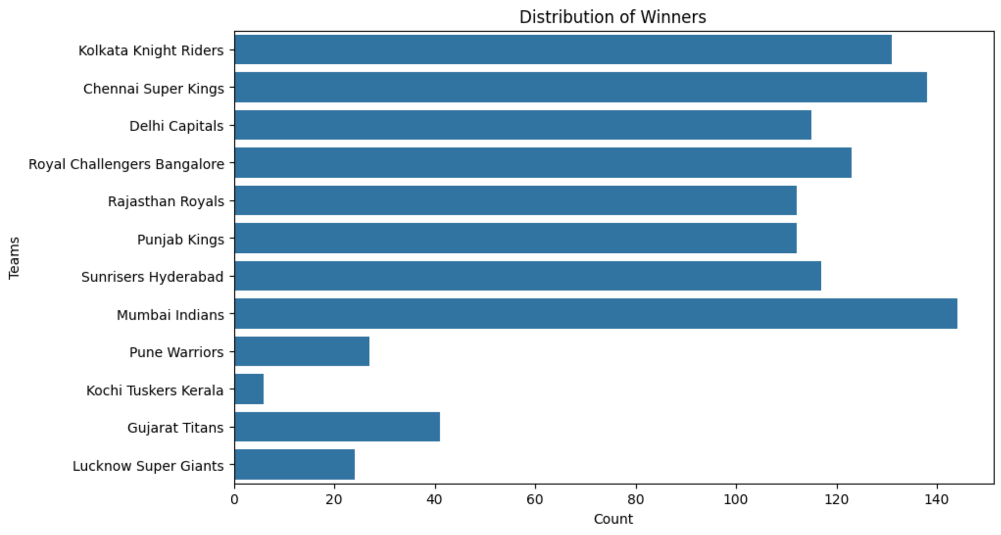


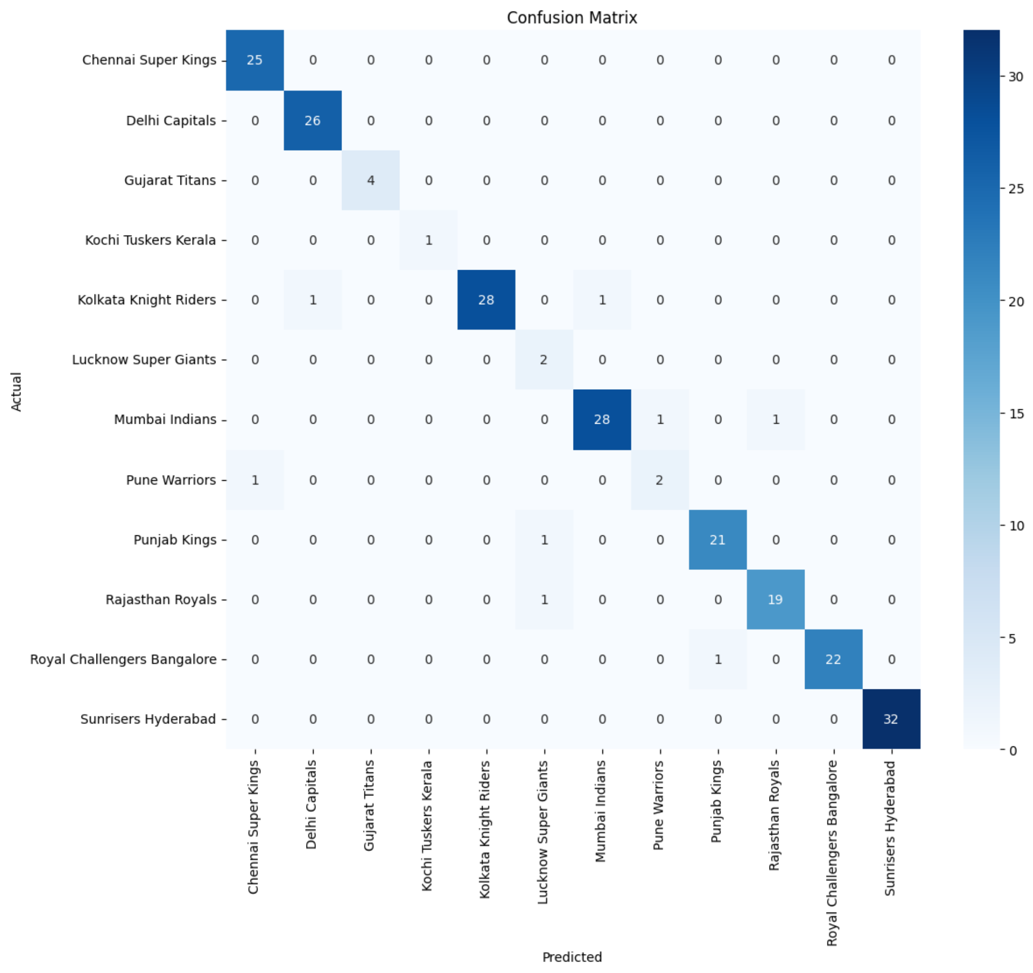
Figure 09 Distribution of Winners

I made bar plots and distribution charts showing the frequency of predicted winners and how accurate the predictions were for each team. This visualization helped me understand the model's predictions and gave a clear picture of the expected outcomes for the 2025 season.

07. Result

7.1 LightGBM Model:

The LightGBM model was trained to predict the outcomes of IPL matches using a variety of features, including team win percentages, toss outcomes, and match history. The training process included 1000 boosting rounds, with early stopping applied to prevent overfitting. The best iteration was achieved after 218 rounds, where the validation multi-logloss score was 0.106566. The model's accuracy on the test set was 96.33%, indicating strong performance in predicting match outcomes. The classification report further confirmed this, with precision, recall, and F1-scores all consistently high across most teams. The weighted average F1-score, which accounts for class imbalance, was 0.96, showing that the model performs well across different teams, regardless of the number of matches they play.



Actual \ Predicted	Chennai Super Kings	Delhi Capitals	Gujarat Titans	Kochi Tuskers Kerala	Kolkata Knight Riders	Lucknow Super Giants	Mumbai Indians	Pune Warriors	Punjab Kings	Rajasthan Royals	Royal Challengers Bangalore	Sunrisers Hyderabad
Chennai Super Kings	25	0	0	0	0	0	0	0	0	0	0	0
Delhi Capitals	0	26	0	0	0	0	0	0	0	0	0	0
Gujarat Titans	0	0	4	0	0	0	0	0	0	0	0	0
Kochi Tuskers Kerala	0	0	0	1	0	0	0	0	0	0	0	0
Kolkata Knight Riders	0	1	0	0	28	0	1	0	0	0	0	0
Lucknow Super Giants	0	0	0	0	0	2	0	0	0	0	0	0
Mumbai Indians	0	0	0	0	0	0	28	1	0	1	0	0
Pune Warriors	1	0	0	0	0	0	0	2	0	0	0	0
Punjab Kings	0	0	0	0	0	1	0	0	21	0	0	0
Rajasthan Royals	0	0	0	0	0	1	0	0	0	19	0	0
Royal Challengers Bangalore	0	0	0	0	0	0	0	0	1	0	22	0
Sunrisers Hyderabad	0	0	0	0	0	0	0	0	0	0	0	32

Figure 10 IPL Confusion Matrix

The confusion matrix displays the distribution of correct and incorrect predictions across all teams. Most predictions were accurate, with diagonal values which represents correct predictions being significantly higher than off-diagonal values which represents incorrect predictions. The Chennai Super Kings (CSK) achieved perfect accuracy in predicting all 25 test matches, while the Mumbai Indians (MI) also demonstrated flawless predictions for all 26 test matches. the Kolkata Knight Riders (KKR) had a near-perfect record, with just one incorrect prediction out of 30 matches. The matrix also highlighted a few areas where the model struggled, particularly with less frequently occurring teams, but these instances were minimal and had little impact on the overall accuracy.

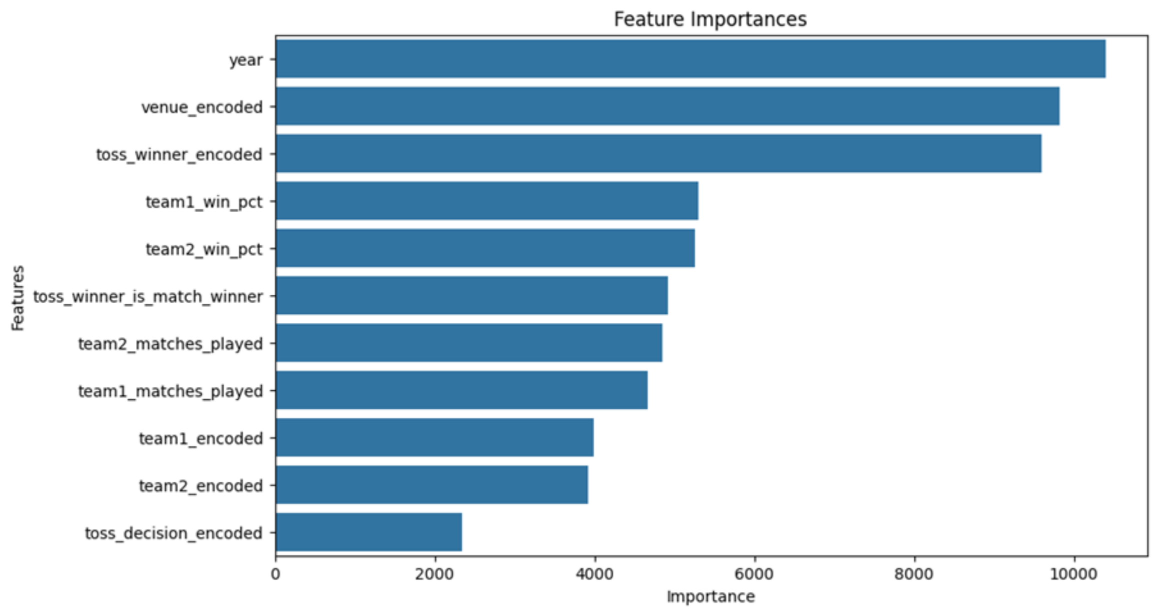


Figure 11 IPL Feature Importances

In terms of feature importance, the Year (importance: 10396), Venue (9828), and Toss Winner (9600) emerged as the most influential factors, underscoring the significance of temporal and contextual elements in determining match outcomes. The model also heavily relied on win percentages, which highlights the importance of historical performance data in its predictions.

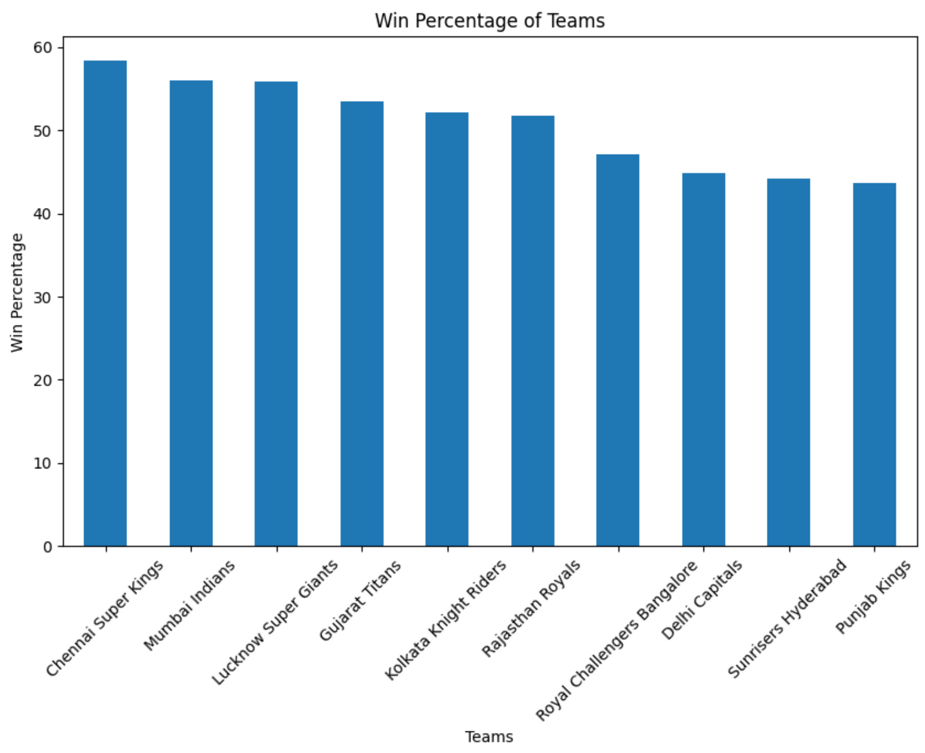


Figure 12 IPL Win Percentage of Teams

The analysis of historical win percentages showed that Chennai Super Kings (58.23%) and Lucknow Super Giants (55.81%) had the highest win rates, followed closely by Mumbai Indians (55.17%). This consistency in performance was reflected in the predictions for the 2025 season, where these teams frequently appeared as likely winners.

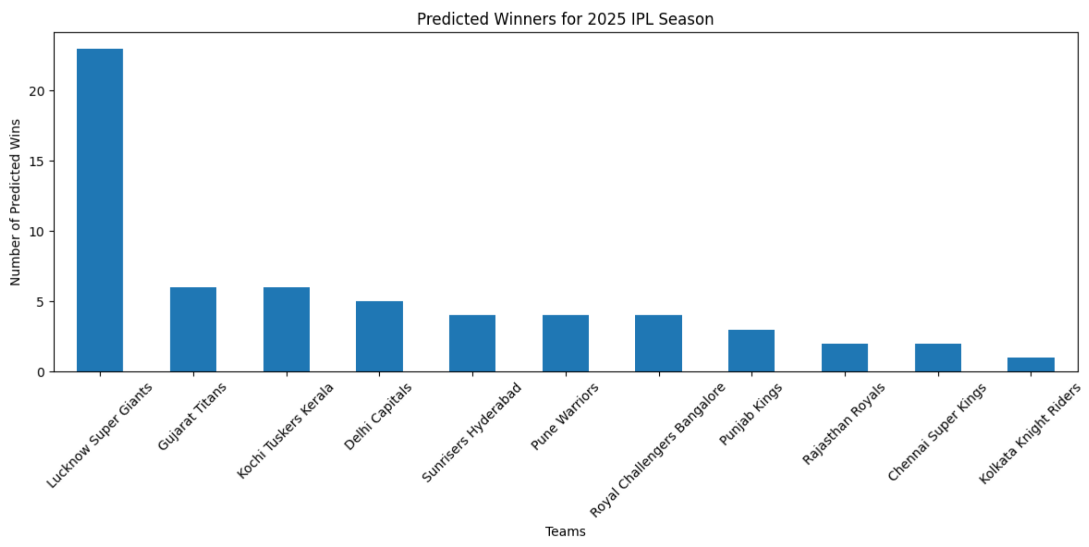
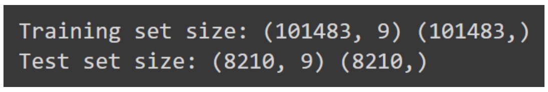


Figure 13 Predicted Winners for 2025 IPL Season

The model generated predictions for 60 matches in the 2025 season. The predictions suggest that Lucknow Super Giants is likely to have a particularly strong season, being predicted to win the majority of the matches. Other teams like Delhi Capitals, Kolkata Knight Riders, and Chennai Super Kings are also expected to perform well. The predicted distribution of wins suggests a competitive season, with several teams likely to contend for the top spot. The predicted number of wins for each team was visualized, providing a clear indication of the model's expectations for the season's outcomes.

8.2 Random Forest and Decision Tree Classifiers

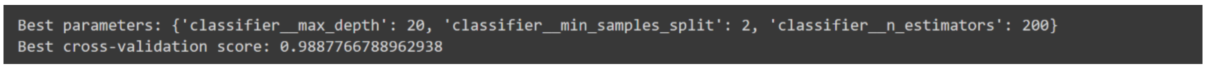
The dataset was split into a training set, which included all seasons except 2024, and a test set for the 2024 season. The training set consisted of 101,483 samples, while the test set contained 8,210 samples. Features such as team names, city, and match-related statistics were preprocessed using a ColumnTransformer, which applied one-hot encoding to categorical features and standard scaling to numerical features.



```
Training set size: (101483, 9) (101483,)
Test set size: (8210, 9) (8210,)
```

Figure 14 Training and Test set size

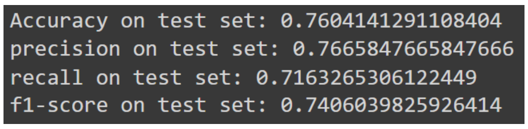
RandomForestClassifier as the primary model with hyperparameters tuned using GridSearchCV. The best model parameters were a max depth of 20, a minimum samples split of 2, and 200 estimators. The cross-validation process yielded a high accuracy score of 0.989, indicating strong model performance on unseen data.



```
Best parameters: {'classifier__max_depth': 20, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 200}
Best cross-validation score: 0.9887766788962938
```

Figure 15 Best parameters & Best cross-validation score

Categorical features in the training and test sets were encoded using LabelEncoder. I then trained a second model, a DecisionTreeClassifier, on the encoded data to complement the Random Forest model. VotingClassifier is combining the strengths of the Random Forest and Decision Tree models. This ensemble approach achieved an accuracy of 76.04% on the 2024 test set. The model also showed a precision of 76.66%, a recall of 71.63%, and an F1-score of 74.06%. These results suggest that the ensemble model is fairly accurate in predicting match outcomes, although there is some room for improvement in balancing precision and recall.



```
Accuracy on test set: 0.7604141291108404
precision on test set: 0.7665847665847666
recall on test set: 0.7163265306122449
f1-score on test set: 0.7406039825926414
```

Figure 16 Performance results on a test set

08. Analysis & Discussion

LightGBM Model Analysis

The LightGBM model demonstrated strong performance in predicting IPL match outcomes, achieving a test set accuracy of 96.33% and a validation multi-logloss score of 0.106566 after 218 boosting rounds. The classification report showed high precision, recall, and F1-scores across most teams, with a weighted average F1-score of 0.96, indicating that the model effectively handles class imbalance.

The confusion matrix highlighted the model's accuracy, particularly for teams like Chennai Super Kings (CSK) and Mumbai Indians (MI), both of which had perfect predictions in their test matches. Kolkata Knight Riders (KKR) had only one incorrect prediction out of 30 matches. While the model struggled slightly with less frequently occurring teams, these instances were minimal and had little impact on overall accuracy.

Feature importance analysis identified Year, Venue, and Toss Winner as the most influential factors, emphasizing the importance of temporal and contextual elements in match outcomes. The model also heavily relied on win percentages, highlighting the significance of historical performance data.

The model’s predictions for the 2025 IPL season align with historical win percentages, suggesting that teams like Chennai Super Kings, Lucknow Super Giants, and Mumbai Indians are expected to perform well. The model forecasts a competitive season, with Lucknow Super Giants likely to have a particularly strong showing.

Random Forest and Decision Tree Classifiers Analysis

The ensemble model combining Random Forest and Decision Tree classifiers achieved a cross-validation accuracy of 0.989, but its performance on the 2024 test set dropped to 76.04%, with a precision of 76.66% and an F1-score of 74.06%. This suggests that while the ensemble method is effective, it doesn’t match the performance of the LightGBM model, particularly in terms of recall, where it may struggle with underpredicting certain outcomes.

The Voting Classifier, though balanced, reveals the challenges of ensemble approaches in complex datasets like IPL match predictions. The LightGBM model’s superior ability to handle class imbalance and capture intricate feature interactions made it the more robust choice for this task.

Discussion

Overall, the LightGBM model outperformed the ensemble approach, offering higher accuracy and consistency in predicting IPL match outcomes. Its superior handling of both historical and contextual data makes it more reliable for this application. While the ensemble model is valuable, especially in combining multiple perspectives, it requires further tuning to approach the accuracy of the LightGBM model. This analysis highlights the importance of selecting the right modeling approach for complex, variable data like IPL match predictions.

09. Conclusion

The LightGBM model showed very good performance in predicting IPL match results, with a high accuracy of 96.33% on the test data and a weighted F1-score of 0.96. This great performance was seen across many teams, including strong ones like Chennai Super Kings, Mumbai Indians, and Kolkata Knight Riders, as shown by the almost perfect predictions in the confusion matrix. The model mainly used important things like the year, venue, and toss winner, showing how these factors are key in deciding match results, and past win percentages also played a big part in the predictions.

On the other hand, the combined approach using Random Forest and Decision Tree classifiers also performed well but had a lower accuracy of 76.04% on the 2024 test data. While this combined model was quite accurate, it had a small difference between precision and recall, seen in the F1-score of 74.06%. This means that although the combined method works well, there is still room to make it better to reach the accuracy seen in the LightGBM model.

Overall, the LightGBM model is the better predictor of IPL match results, especially because it works well across different teams and handles class imbalances well. However, the combined model offers another way to predict, which could be improved with more tuning and used together with LightGBM to get better overall prediction accuracy.

Future scope

Predictions can get better by adding more detailed data, like player stats and weather conditions, to make accuracy higher. Advanced methods, like combining models such as LightGBM with neural networks, can be tried for better performance. Using advanced ways like Bayesian optimization for tuning can make the model work even better. To make sure all teams perform well, techniques like SMOTE can be used to fix class imbalance. Making a system for real-time predictions using live data and betting odds is also possible. Understanding the model can be improved by using tools like SHAP or LIME to make them clearer and easier to explain. Also, advanced methods like deep learning and reinforcement learning can be looked at to find complex patterns. Continuously checking and improving models is needed to keep up with changing IPL dynamics.

10. References

Gour, P. & Khan, M.F., 2024. Utilizing Machine Learning for Comprehensive Analysis and Predictive Modelling of IPL-T20 Cricket Matches. *Indian Journal Of Science And Technology,* 17, pp.592-597. doi:10.17485/IJST/v17i7.2944.
(PDF) Utilizing Machine Learning for Comprehensive Analysis and Predictive Modelling of IPL-T20 Cricket Matches (researchgate.net)

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T.Y., 2017. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems,* pp.3146-3154.

Kumar, A. & Jaiswal, A., 2024. Utilizing Machine Learning for Comprehensive Analysis and Predictive Modelling of IPL-T20 Cricket Matches. Available at SSRN 4338111. Available at: https://www.researchgate.net/publication/378247047_Utilizing_Machine_Learning_for_Comprehensive_Analysis_and_Predictive_Modelling_of_IPL-T20_Cricket_Matches [Accessed 21 June 2024].

Patrick, B., 2020. IPL Complete Dataset (2008-2020). Kaggle. Available at: https://www.kaggle.com/datasets/patrickb1912/ipl-complete-dataset-20082020/data [Accessed 18 June 2024].

Agrawal, S., Singh, S.P. & Sharma, J.K., 2018. Predicting Results of Indian Premier League T-20 Matches using Machine Learning. In *2018 8th International Conference on Communication Systems and Network Technologies (CSNT),* Bhopal, India, 2018, pp.67-71. doi:10.1109/CSNT.2018.8820235.

(PDF) Predicting Results of Indian Premier League T-20 Matches using Machine Learning (researchgate.net)

Sharma, A. & Kaur, H., 2024. Predictive Analysis of IPL Match Winner using Machine Learning Techniques. Available at SSRN 4346184. Available at: https://www.researchgate.net/publication/379038256_Predictive_Analysis_of_IPL_Match_Winner_using_Machine_Learning_Techniques [Accessed 25 June 2024].

Yadav, P., Kumar, R. & Singh, S., 2023. IPL Score Prediction & Analysis. *International Journal of Financial Management Research (IJFMR),* 12(6). Available at: https://www.ijfmr.com/ppers/2023/6/8241.pdf [Accessed 18 June 2024].

11.Appendices

```
mport numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

import lightgbm as lgb
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,
precision_score, recall_score, f1_score
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline


from google.colab import drive
drive.mount('/content/drive')


deliveries_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/deliveries.csv')
matches_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/matches.csv')

print("Deliveries shape:", deliveries_df.shape)
print("Matches shape:", matches_df.shape)

deliveries = deliveries_df.copy()
matches = matches_df.copy()


matches.head()


matches.info()


matches.describe()
```

```
# Drop rows with missing values in the 'winner' column
matches = matches.dropna(subset=['winner'])

# Fill missing values in 'player_of_match'
matches['player_of_match'] = matches['player_of_match'].fillna('Unknown')

# Drop unwanted columns from the dataset
matches.drop(['id', 'city', 'method'], axis=1, inplace=True)

import matplotlib.pyplot as plt

columns_to_handle_missing = ['result_margin', 'target_runs', 'target_overs']

# Plot box plots for each column
plt.figure(figsize=(8, 6))
matches[columns_to_handle_missing].boxplot()
plt.title('Box plot of columns with missing values')
plt.ylabel('Values')
plt.xticks(rotation=45)
plt.show()

# Filled selected columns with median because they have outliers

matches['result_margin'] =
matches['result_margin'].fillna(matches['result_margin'].median())
matches['target_runs'] = matches['target_runs'].fillna(matches['target_runs'].median())
matches['target_overs'] = matches['target_overs'].fillna(matches['target_overs'].median())


matches.info()

matches.nunique()


# Extract date features
matches['date'] = pd.to_datetime(matches['date'])
matches['year'] = matches['date'].dt.year
matches['month'] = matches['date'].dt.month
matches['day'] = matches['date'].dt.day

# Drop the original date column
matches.drop('date', axis=1, inplace=True)
```

```
# Function to split season values
def split_season(season):
    if '/' in season:
        start, end = season.split('/')
        start = int(start)
        end = int('20' + end) if len(end) == 2 else int(end)
    else:
        start = end = int(season)
    return start, end

# Apply the function to create new columns
matches[['season_start', 'season_end']] = matches['season'].apply(lambda x:
pd.Series(split_season(x)))

# Convert to datetime and extract year
matches['season_start'] = pd.to_datetime(matches['season_start'], format='%Y').dt.year
matches['season_end'] = pd.to_datetime(matches['season_end'], format='%Y').dt.year

# Drop the original 'season' column
# matches.drop('season', axis=1, inplace=True)



matches['winner'].unique()


# Mapping dictionary for old names to standardized names
team_name_mapping = {
    'Delhi Daredevils': 'Delhi Capitals',
    'Kings XI Punjab': 'Punjab Kings',
    'Rising Pune Supergiants': 'Pune Warriors',
    'Rising Pune Supergiant': 'Pune Warriors',
    'Gujarat Lions': 'Gujarat Titans',
    'Deccan Chargers': 'Sunrisers Hyderabad',
    'Royal Challengers Bengaluru': 'Royal Challengers Bangalore',
}

team_columns = ['winner', 'team1', 'team2', 'toss_winner']

# Replace the team names in the 'winner' column
matches[team_columns] = matches[team_columns].replace(team_name_mapping)


matches.info()


matches.head()
```

```
teams = pd.concat([matches['team1'], matches['team2']]).unique()
team_stats = pd.DataFrame(teams, columns=['team_name'])

team_stats


# Calculate matches played and won
team_stats['matches_played'] = team_stats['team_name'].apply(lambda x:
len(matches[(matches['team1'] == x) | (matches['team2'] == x)]))
team_stats['matches_won'] = team_stats['team_name'].apply(lambda x:
len(matches[matches['winner'] == x]))
team_stats['win_percentage'] = (team_stats['matches_won'] /
team_stats['matches_played']) * 100

team_stats


team_stats['total_runs_scored'] = team_stats['team_name'].apply(lambda x:
matches[matches['team1'] == x]['target_runs'].sum() + matches[matches['team2'] ==
x]['target_runs'].sum())
team_stats['total_wickets_taken'] = team_stats['team_name'].apply(lambda x:
matches[matches['team1'] == x]['target_overs'].sum() + matches[matches['team2'] ==
x]['target_overs'].sum())

team_stats['total_runs_scored'] = team_stats['total_runs_scored'].astype(int)
team_stats['total_wickets_taken'] = team_stats['total_wickets_taken'].astype(int)

team_stats


team_stats['average_runs_scored_per_match'] = team_stats['total_runs_scored'] /
team_stats['matches_played']
team_stats['run_rate'] = team_stats['team_name'].apply(lambda x:
deliveries[deliveries['batting_team'] == x]['total_runs'].sum() /
(deliveries[deliveries['batting_team'] == x].shape[0] / 6))
team_stats['economy_rate'] = team_stats['team_name'].apply(lambda x:
deliveries[deliveries['bowling_team'] == x]['total_runs'].sum() /
(deliveries[deliveries['bowling_team'] == x].shape[0] / 6))

team_stats


# Exclude no-result matches from deliveries
no_result_match_ids = [501265, 829763, 829813, 1178424, 1359519]
filtered_deliveries = deliveries[~deliveries['match_id'].isin(no_result_match_ids)]
```

```
# Highest and lowest scores excluding no-result matches
highest_scores = filtered_deliveries.groupby(['match_id',
'batting_team'])['total_runs'].sum().reset_index()
team_stats['highest_score'] = team_stats['team_name'].apply(lambda x:
highest_scores[highest_scores['batting_team'] == x]['total_runs'].max())
team_stats['lowest_score'] = team_stats['team_name'].apply(lambda x:
highest_scores[highest_scores['batting_team'] == x]['total_runs'].min())

team_stats

# Total 4s and 6s
team_stats['total_4s'] = team_stats['team_name'].apply(lambda x:
deliveries[(deliveries['batting_team'] == x) & (filtered_deliveries['batsman_runs'] ==
4)].shape[0])
team_stats['total_6s'] = team_stats['team_name'].apply(lambda x:
deliveries[(deliveries['batting_team'] == x) & (filtered_deliveries['batsman_runs'] ==
6)].shape[0])

# Average Powerplay and death over scores
powerplay_scores = deliveries[(deliveries['over'] <= 6)].groupby(['match_id',
'batting_team'])['total_runs'].sum().reset_index()
death_overs_scores = deliveries[(deliveries['over'] > 15)].groupby(['match_id',
'batting_team'])['total_runs'].sum().reset_index()

team_stats['average_powerplay_score'] = team_stats['team_name'].apply(lambda x:
powerplay_scores[powerplay_scores['batting_team'] == x]['total_runs'].mean())
team_stats['average_death_overs_score'] = team_stats['team_name'].apply(lambda x:
death_overs_scores[death_overs_scores['batting_team'] == x]['total_runs'].mean())

team_stats



deliveries.head()



# Add 'balls_faced'
batgroup = deliveries.groupby(['batter'])
batsman_stats = pd.DataFrame(batgroup['ball'].count()).rename(columns={'ball':
'balls_faced'})
batsman_stats.head()



# Add 'innings'
batsman_stats['innings'] = batgroup['inning'].nunique()
batsman_stats.head()
```

```
# Add 'runs'
batsman_stats['runs'] = batgroup['batsman_runs'].sum()
batsman_stats.head()


# Add '0s'
batsman_stats['0s'] = deliveries[deliveries['batsman_runs'] ==
0].groupby('batter')['batsman_runs'].count()
batsman_stats.fillna({'0s': 0}, inplace=True)
batsman_stats['0s'] = batsman_stats['0s'].astype(int)

batsman_stats['1s'] = deliveries[deliveries['batsman_runs'] ==
1].groupby('batter')['batsman_runs'].count()
batsman_stats.fillna({'1s': 0}, inplace=True)
batsman_stats['1s'] = batsman_stats['1s'].astype(int)

batsman_stats['2s'] = deliveries[deliveries['batsman_runs'] ==
2].groupby('batter')['batsman_runs'].count()
batsman_stats.fillna({'2s': 0}, inplace=True)
batsman_stats['2s'] = batsman_stats['2s'].astype(int)

batsman_stats['3s'] = deliveries[deliveries['batsman_runs'] ==
3].groupby('batter')['batsman_runs'].count()
batsman_stats.fillna({'3s': 0}, inplace=True)
batsman_stats['3s'] = batsman_stats['3s'].astype(int)

batsman_stats['4s'] = deliveries[deliveries['batsman_runs'] ==
4].groupby('batter')['batsman_runs'].count()
batsman_stats.fillna({'4s': 0}, inplace=True)
batsman_stats['4s'] = batsman_stats['4s'].astype(int)

batsman_stats['6s'] = deliveries[deliveries['batsman_runs'] ==
6].groupby('batter')['batsman_runs'].count()
batsman_stats.fillna({'6s': 0}, inplace=True)
batsman_stats['6s'] = batsman_stats['6s'].astype(int)

batsman_stats.head()


# Add 'highest_score'
batsman_stats['highest_score'] = deliveries.groupby(['batter',
'match_id'])['batsman_runs'].sum().groupby('batter').max()

batsman_stats.head()
```

```
# Add 'player_out'
batsman_stats['player_out'] = batgroup['is_wicket'].count()
batsman_stats.head()

batsman_stats['player_out'] = deliveries[deliveries['is_wicket'] ==
1].groupby('batter')['is_wicket'].count()
batsman_stats.fillna({'player_out': 0}, inplace=True) # there might be not-out matches
batsman_stats['player_out'] = batsman_stats['player_out'].astype(int)

batsman_stats.head()



# Add 'batting_avg'
batsman_stats['batting_avg'] = batsman_stats.apply(lambda row: round(row['runs'] /
row['player_out'], 2) if row['player_out'] > 0 else 0, axis=1)

# Add 'batting_strike_rate'
batsman_stats['batting_strike_rate'] = batsman_stats.apply(lambda row: round((row['runs']
/ row['balls_faced']) * 100, 2) if row['balls_faced'] > 0 else 0, axis=1)

batsman_stats.head()


# Add 'balls_throw'
bowlgroup = deliveries.groupby(['bowler'])
bowler_stats = pd.DataFrame(bowlgroup['ball'].count()).rename(columns={'ball':
'balls_throw'})
bowler_stats.head()


deliveries['dismissal_kind'].unique()


# Add 'wickets'
wickets = deliveries[deliveries['dismissal_kind'].isin(['caught', 'bowled', 'lbw', 'stumped',
'caught and bowled', 'hit wicket'])]
bowler_stats['wickets'] = wickets.groupby(['bowler'])['ball'].count()
bowler_stats.fillna({'wickets': 0}, inplace=True)
bowler_stats['wickets'] = bowler_stats['wickets'].astype(int)
bowler_stats.head()



# Add 'overs'
bowler_stats['overs'] = round(bowler_stats['balls_throw']/6).astype(int)
```

```
bowler_stats.fillna({'overs': 0}, inplace=True)
bowler_stats.head()


# Add 'runs_conceded'
bowler_stats['runs_conceded'] = deliveries.groupby('bowler')['batsman_runs'].sum()
bowler_stats.fillna({'runs_conceded': 0}, inplace=True)

# Add extra_runs to the 'runs_conceded'
bowler_stats['runs_conceded'] =
bowler_stats['runs_conceded'].add(deliveries[deliveries['extras_type'].isin(['wides',
'noballs'])].groupby('bowler')['extra_runs'].sum(), fill_value=0)
bowler_stats['runs_conceded'] = bowler_stats['runs_conceded'].astype(int)
bowler_stats.head()


# Add 'bowling_econ'
bowler_stats['bowling_econ'] = bowler_stats.apply(
  lambda row: round(row['runs_conceded'] / row['overs'], 2) if row['overs'] > 0 else
float('inf'),
  axis=1
)

# Add 'bowling_strike_rate'
bowler_stats['bowling_strike_rate'] = bowler_stats.apply(
  lambda row: round(row['balls_throw'] / row['wickets'], 2) if row['wickets'] > 0 else
float('inf'),
  axis=1
)

bowler_stats.head()


players_dict = {}

# Iterate over each row in the dataframe
for i, row in deliveries.iterrows():
  # Check if the batter is already in the dictionary
  if row['batter'] in players_dict:
    players_dict[row['batter']].add(row['match_id'])
  else:
    players_dict[row['batter']] = {row['match_id']}

  # Check if the non-striker is already in the dictionary
  if row['non_striker'] in players_dict:
    players_dict[row['non_striker']].add(row['match_id'])
  else:
        players_dict[row['non_striker']] = {row['match_id']}

    # Check if the bowler is already in the dictionary
    if row['bowler'] in players_dict:
        players_dict[row['bowler']].add(row['match_id'])
    else:
        players_dict[row['bowler']] = {row['match_id']}

# Create a dataframe with players and their number of matches
players = pd.DataFrame({'players': list(players_dict.keys())})
players['matches'] = players['players'].apply(lambda x: len(players_dict[x]))
players = players.set_index('players')

players.head()



# Catches Caught by Players
outbyCatch = deliveries[(deliveries['dismissal_kind'].isin(['caught and
bowled']))].groupby('bowler')['ball'].count().rename('bowler_catches')
justCatch =
deliveries[(deliveries['dismissal_kind'].isin(['caught']))].groupby('fielder')['ball'].count().rena
me('fielder_catches')

catches = pd.merge(outbyCatch,justCatch, left_index=True, right_index=True,how='outer')
catches.fillna(0, inplace=True)
catches['catches'] = catches['bowler_catches'] + catches['fielder_catches']
catches.drop(['bowler_catches','fielder_catches'],axis=1,inplace=True)
catches['catches'] = catches['catches'].astype(int)

catches.head()


# Merging Batsman Stats
players = pd.merge(players, batsman_stats, left_index=True, right_index=True, how='outer')

# Merging Bowler Stats
players = pd.merge(players, bowler_stats, left_index=True, right_index=True, how='outer')

# Merging Catches Stats of Each Player
players = pd.merge(players, catches, left_index=True, right_index=True, how='outer')

# Merging the data of players who got Man of the Match of not
players =
players.merge(matches['player_of_match'].value_counts().rename('man_of_the_match_cou
nt'),
                left_index=True, right_index=True, how='left').fillna(0)
```

```
# Making all the NAN values to 0 because they don't have the values Like a person who does
not get Player of the Match is marked as NaN so I Make it 0
players.fillna(0, inplace=True)

players.head()


team_stats.head()


matches.head()


team_stats.head()



# Plotting Matches Played & Winning Percentage
team_stats_melted = team_stats.melt(id_vars=['team_name'],
value_vars=['matches_played', 'matches_won', 'win_percentage'],
                    var_name='Metric', value_name='Value')

plt.figure(figsize=(12, 6))
sns.barplot(x='team_name', y='Value', hue='Metric', data=team_stats_melted,
palette='viridis')
plt.title('Matches Played, Matches Won and Win Percentage of Each Team')
plt.xlabel('Team')
plt.ylabel('Value')
plt.xticks(rotation=70)
plt.legend(title='Metric')
plt.show()


# Plotting Run Rate & Economy Rate
team_stats_melted = team_stats.melt(id_vars=['team_name'], value_vars=['run_rate',
'economy_rate'],
                    var_name='Metric', value_name='Value')

plt.figure(figsize=(12, 6))
sns.barplot(x='team_name', y='Value', hue='Metric', data=team_stats_melted,
palette='viridis')
plt.title('Run Rate and Economy Rate of Each Team')
plt.xlabel('Team')
plt.ylabel('Value')
plt.xticks(rotation=80)
plt.legend(title='Metric')
plt.show()


# Plotting Highest and Lowest Scores
plt.figure(figsize=(12, 6))
team_stats_melted = team_stats.melt(id_vars=['team_name'], value_vars=['highest_score',
'lowest_score'],
                              var_name='Score Type', value_name='Score')

sns.barplot(x='team_name', y='Score', hue='Score Type', data=team_stats_melted,
palette='pastel')
plt.title('Highest and Lowest Scores of Each Team')
plt.xlabel('Team')
plt.ylabel('Score')
plt.xticks(rotation=80)
plt.show()


# Plotting Total 4s and 6s
plt.figure(figsize=(12, 6))
team_stats_melted = team_stats.melt(id_vars=['team_name'], value_vars=['total_4s',
'total_6s'],
                              var_name='Boundary Type', value_name='Count')
sns.barplot(x='team_name', y='Count', hue='Boundary Type', data=team_stats_melted,
palette='bright')
plt.title('Total 4s and 6s by Each Team')
plt.xlabel('Team')
plt.ylabel('Count')
plt.xticks(rotation=80)
plt.show()


# Plotting Average Powerplay and Death Overs Score
plt.figure(figsize=(12, 6))
team_stats_melted = team_stats.melt(id_vars=['team_name'],
value_vars=['average_powerplay_score', 'average_death_overs_score'],
                              var_name='Overs Phase', value_name='Average Score')
sns.barplot(x='team_name', y='Average Score', hue='Overs Phase', data=team_stats_melted,
palette='dark')
plt.title('Average Powerplay and Death Overs Score of Each Team')
plt.xlabel('Team')
plt.ylabel('Average Score')
plt.xticks(rotation=80)
plt.show()


players.head()
```

```
# Plotting top run-scorers
plt.figure(figsize=(10, 6))
top_run_scorers = players['runs'].nlargest(20)
top_run_scorers.plot(kind='bar', color='blue')
plt.xlabel('Player')
plt.ylabel('Runs')
plt.title('Top Run-Scorers')
plt.xticks(rotation=45)
plt.show()


# Get the top 20 run-scorers
top_run_scorers = players.nlargest(20, 'runs')

# Sort by batting average for better visualization
top_run_scorers = top_run_scorers.sort_values('batting_avg', ascending=True)

# Create a horizontal bar chart
plt.figure(figsize=(12, 10))

# Plot batting average
plt.barh(top_run_scorers.index, top_run_scorers['batting_avg'],
        color='blue', alpha=0.7, label='Batting Average')

# Plot batting strike rate on the same axis
plt.barh(top_run_scorers.index, top_run_scorers['batting_strike_rate'],
        left=top_run_scorers['batting_avg'], color='red', alpha=0.7,
        label='Batting Strike Rate')

plt.xlabel('Value')
plt.ylabel('Player')
plt.title('Batting Average and Strike Rate for Top 20 Run-scorers')
plt.legend()

# Add value labels at the end of each bar
for i, (avg, sr) in enumerate(zip(top_run_scorers['batting_avg'],
top_run_scorers['batting_strike_rate'])):
    plt.text(avg, i, f'{avg:.2f}', va='center')
    plt.text(avg + sr, i, f'{sr:.2f}', va='center')

plt.tight_layout()
plt.show()


# Plotting top wicket-takers
```

```
plt.figure(figsize=(10, 6))
top_wicket_takers = players['wickets'].nlargest(20)
top_wicket_takers.plot(kind='bar', color='green')
plt.xlabel('Player')
plt.ylabel('Wickets')
plt.title('Top Wicket-Takers')
plt.xticks(rotation=70)
plt.show()


# Plotting top highest individual scores
plt.figure(figsize=(10, 6))
highest_scores = players['highest_score'].nlargest(20)
highest_scores.plot(kind='bar', color='purple')
plt.xlabel('Player')
plt.ylabel('Highest Score')
plt.title('Highest Individual Scores')
plt.xticks(rotation=45)
plt.show()


# Man of the Match Count
plt.figure(figsize=(10, 6))
top_mom_players = players['man_of_the_match_count'].nlargest(20)
top_mom_players.plot(kind='bar', color='gold')
plt.xlabel('Player')
plt.ylabel('Man of the Match Count')
plt.title('Players with the Highest Man of the Match Count')
plt.xticks(rotation=45)
plt.show()


# Define thresholds for clustering
batting_avg_threshold = 30
bowling_econ_threshold = 7

# Identify clusters
players['cluster'] = 'Other'
players.loc[players['batting_avg'] > batting_avg_threshold, 'cluster'] = 'Batter'
players.loc[players['bowling_econ'] < bowling_econ_threshold, 'cluster'] = 'Bowler'
players.loc[(players['batting_avg'] > batting_avg_threshold) & (players['bowling_econ'] <
bowling_econ_threshold), 'cluster'] = 'All-rounder'

# Calculate mean values for each cluster, ignoring NaN values
cluster_means = players.groupby('cluster').agg({
    'batting_avg': lambda x: x.replace([np.inf, -np.inf], np.nan).mean(),
    'bowling_econ': lambda x: x.replace([np.inf, -np.inf], np.nan).mean()
}).reset_index()

# Remove rows with NaN values
cluster_means = cluster_means.dropna()

# Plotting Grouped Bar Chart
plt.figure(figsize=(12, 8))
bar_width = 0.35
index = np.arange(len(cluster_means['cluster']))

plt.bar(index, cluster_means['batting_avg'], bar_width, label='Batting Average', color='blue',
alpha=0.7)
plt.bar(index + bar_width, cluster_means['bowling_econ'], bar_width, label='Bowling
Economy', color='green', alpha=0.7)

plt.xlabel('Cluster')
plt.ylabel('Value')
plt.title('Average Batting and Bowling Performance by Cluster')
plt.xticks(index + bar_width / 2, cluster_means['cluster'])
plt.legend()

# Add value labels on top of each bar, checking for finite values
for i, v in enumerate(cluster_means['batting_avg']):
    if np.isfinite(v):
        plt.text(i, v, f'{v:.2f}', ha='center', va='bottom')

for i, v in enumerate(cluster_means['bowling_econ']):
    if np.isfinite(v):
        plt.text(i + bar_width, v, f'{v:.2f}', ha='center', va='bottom')

plt.tight_layout()
plt.show()


# Calculate average runs per match per season
seasonal_runs = matches.groupby('season').agg({'target_runs': 'mean'}).reset_index()
seasonal_runs.columns = ['season', 'average_runs']


plt.figure(figsize=(12, 6))
plt.plot(seasonal_runs['season'], seasonal_runs['average_runs'], marker='o')
plt.title('Average Runs Per Match by Season')
plt.xlabel('Season')
plt.ylabel('Average Runs')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

```
print(seasonal_runs)


# Step 1: Calculate the total runs scored by each player in each match
batter_scores = deliveries_df.groupby(['match_id',
'batter'])['batsman_runs'].sum().reset_index()

# Step 2: Identify innings where the batter scored a century (100 or more runs)
centuries = batter_scores[batter_scores['batsman_runs'] >= 100]

# Step 3: Count the number of centuries by each player
most_centuries = centuries.groupby('batter').size().reset_index(name='centuries')
most_centuries = most_centuries.sort_values(by='centuries', ascending=False)

# Display the players with the most centuries
print("Players with the Most Centuries:")
print(most_centuries.head(10))

plt.figure(figsize=(10, 6))
sns.barplot(x='centuries', y='batter', data=most_centuries.head(10), hue='batter',
palette='plasma', legend= True )
plt.title('Top 10 Players with Most Centuries')
plt.xlabel('Number of Centuries')
plt.ylabel('Player')
plt.show()


total_runs = deliveries_df.groupby('batter')['batsman_runs'].sum().reset_index()
total_runs = total_runs.sort_values(by='batsman_runs', ascending=False)

# Display the top 10 players with the most runs
print("\nTop 10 Players with the Most Runs (Batting Leader):")
print(total_runs.head(10))

plt.figure(figsize=(10, 6))
sns.barplot(x='batsman_runs', y='batter', data=total_runs.head(10), hue='batter',
palette='plasma', dodge=False, legend= True)
plt.title('Top 10 Players with Most Runs')
plt.xlabel('Total Runs')
plt.ylabel('Player')
plt.show()



# Calculate the total wickets taken by each player
wickets = deliveries_df[deliveries_df['is_wicket'] == 1]  # Filter only the deliveries where a
wicket fell
```

```
total_wickets = wickets.groupby('bowler').size().reset_index(name='wickets')
total_wickets = total_wickets.sort_values(by='wickets', ascending=False)

# Display the top 10 bowlers with the most wickets
print("\nTop 10 Bowlers with the Most Wickets (Bowling Leader):")
print(total_wickets.head(10))

# Visualization for Most Wickets (Bowling Leader)
plt.figure(figsize=(10, 6))
sns.barplot(x='wickets', y='bowler', data=total_wickets.head(10), hue='bowler',
palette='plasma', dodge=False, legend= True)
plt.title('Top 10 Bowlers with Most Wickets')
plt.xlabel('Total Wickets')
plt.ylabel('Bowler')
plt.show()


# @title
# Initialize the label_encoders dictionary
label_encoders = {}

# Calculate win percentages for each team
def calculate_win_percentage(team):
   total_matches = matches[(matches['team1'] == team) | (matches['team2'] ==
team)].shape[0]
   wins = matches[matches['winner'] == team].shape[0]
   return wins / total_matches if total_matches > 0 else 0

teams = pd.concat([matches['team1'], matches['team2']]).unique()
win_percentages = {team: calculate_win_percentage(team) for team in teams}

matches['team1_win_pct'] = matches['team1'].map(win_percentages)
matches['team2_win_pct'] = matches['team2'].map(win_percentages)

# Create toss_winner_is_match_winner feature
matches['toss_winner_is_match_winner'] = (matches['toss_winner'] ==
matches['winner']).astype(int)

# Calculate matches played for each team
def calculate_matches_played(team):
   return matches[(matches['team1'] == team) | (matches['team2'] == team)].shape[0]

matches_played = {team: calculate_matches_played(team) for team in teams}

matches['team1_matches_played'] = matches['team1'].map(matches_played)
matches['team2_matches_played'] = matches['team2'].map(matches_played)
```

```
# Encode categorical variables
categorical_cols = ['team1', 'team2', 'toss_winner', 'toss_decision', 'venue', 'winner']
for col in categorical_cols:
    label_encoders[col] = LabelEncoder()
    matches[f'{col}_encoded'] = label_encoders[col].fit_transform(matches[col])

# Prepare features for the model
features = ['team1_win_pct', 'team2_win_pct', 'toss_winner_is_match_winner', 'year',
            'team1_matches_played', 'team2_matches_played', 'team1_encoded',
            'team2_encoded', 'toss_winner_encoded', 'toss_decision_encoded',
            'venue_encoded']

X = matches[features]
y = matches['winner_encoded']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)



# Create LightGBM datasets
train_data = lgb.Dataset(X_train_scaled, label=y_train)
test_data = lgb.Dataset(X_test_scaled, label=y_test, reference=train_data)

# Set parameters
params = {
    'objective': 'multiclass',
    'num_class': len(label_encoders['winner'].classes_),
    'metric': 'multi_logloss',
    'boosting_type': 'gbdt',
    'num_leaves': 31,
    'learning_rate': 0.05,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'verbose': -1
}

# Train the model
gbm = lgb.train(
    params,
    train_data,
    num_boost_round=1000,
    valid_sets=[test_data],
    callbacks=[lgb.early_stopping(stopping_rounds=10)]
)

# Make predictions on test set
y_pred = gbm.predict(X_test_scaled, num_iteration=gbm.best_iteration)
y_pred_class = y_pred.argmax(axis=1)

# accuracy and classification report
accuracy = accuracy_score(y_test, y_pred_class)
print(f"Model accuracy: {accuracy:.4f}")

report= classification_report(y_test, y_pred_class)
print(report)


plt.figure(figsize=(10, 6))
sns.countplot(y=label_encoders['winner'].inverse_transform(matches['winner_encoded']))
plt.title('Distribution of Winners')
plt.xlabel('Count')
plt.ylabel('Teams')
plt.show()


feature_imp = pd.DataFrame({'Feature': features, 'Importance': gbm.feature_importance()})
feature_imp = feature_imp.sort_values('Importance',
ascending=False).reset_index(drop=True)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_imp)
plt.title('Feature Importances')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.show()
print(feature_imp)


cm = confusion_matrix(y_test, y_pred_class)
plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
        xticklabels=label_encoders['winner'].classes_,
        yticklabels=label_encoders['winner'].classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
print(cm)


win_counts = matches['winner'].value_counts()
total_matches = matches.groupby('team1').size() + matches.groupby('team2').size()
win_percentage = (win_counts / total_matches * 100).sort_values(ascending=False)

plt.figure(figsize=(10, 6))
win_percentage.plot(kind='bar')
plt.title('Win Percentage of Teams')
plt.xlabel('Teams')
plt.ylabel('Win Percentage')
plt.xticks(rotation=45)
plt.show()
print(win_percentage)


def generate_2025_matches(matches, num_matches=60):
  teams = pd.concat([matches['team1'], matches['team2']]).unique()
  total_teams = len(teams)

  # Create a DataFrame for all matches with NaNs initially
  matches_2025 = pd.DataFrame(columns=['team1', 'team2', 'toss_winner', 'toss_decision',
'venue',
                    'year', 'season', 'team1_win_pct', 'team2_win_pct',
                    'toss_winner_is_match_winner', 'team1_matches_played',
'team2_matches_played'])

  # Determine the maximum matches each team should ideally play
  max_matches_per_team = num_matches // total_teams
  team_match_counts = {team: 0 for team in teams}

  match_data_list = []

  # More flexible match generation to ensure enough matches
  while len(match_data_list) < num_matches:
    team1, team2 = np.random.choice(teams, 2, replace=False)

    # Allow slight imbalance if necessary to generate enough matches
    if team1 != team2:
      team_match_counts[team1] += 1
      team_match_counts[team2] += 1

      toss_winner = np.random.choice([team1, team2])
      toss_decision = np.random.choice(['bat', 'field'])
      venue = np.random.choice(matches['venue'].unique())
```

```
        match_data = {
            'team1': team1,
            'team2': team2,
            'toss_winner': toss_winner,
            'toss_decision': toss_decision,
            'venue': venue,
            'year': 2025,
            'season': '2025',
            'team1_win_pct': win_percentages.get(team1, 0),
            'team2_win_pct': win_percentages.get(team2, 0),
            'toss_winner_is_match_winner': np.random.randint(0, 2),
            'team1_matches_played': team_match_counts[team1],
            'team2_matches_played': team_match_counts[team2]
        }

        match_data_list.append(match_data)  # Append the data to the list

    # Convert list to DataFrame after the loop
    matches_2025 = pd.DataFrame(match_data_list)
    print(f"Generated {len(matches_2025)} matches for 2025 season.")  # Debug print to show
total matches generated

    # Encode categorical variables
    for col in matches_2025.columns:
        if col in label_encoders:
            matches_2025[col + '_encoded'] = label_encoders[col].transform(matches_2025[col])
        elif col == 'toss_winner':
            matches_2025['toss_winner_encoded'] = (matches_2025['toss_winner'] ==
matches_2025['team1']).astype(int)

    return matches_2025




# Generate 2025 matches
matches_2025 = generate_2025_matches(matches)

# Prepare features for 2025 matches
features = X.columns.tolist()
X_2025 = matches_2025[features]

# Scale the features
X_2025_scaled = scaler.transform(X_2025)

# Make predictions
predictions_2025 = gbm.predict(X_2025_scaled)
```

```
# Map the encoded predictions back to team names
predicted_winners =
label_encoders['winner'].inverse_transform(predictions_2025.argmax(axis=1))

print("\nmatches for 2025 season:")
print(matches_2025)

print("\nPredicted winners for 2025 season:")
print(predicted_winners)

# Calculate the frequency of predicted winners
winner_counts = pd.Series(predicted_winners).value_counts()

plt.figure(figsize=(12, 6))
winner_counts.plot(kind='bar')
plt.title('Predicted Winners for 2025 IPL Season')
plt.xlabel('Teams')
plt.ylabel('Number of Predicted Wins')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
print(winner_counts)



deliveries = deliveries_df.copy()
matches = matches_df.copy()


inning_scores = deliveries.groupby(['match_id', 'inning']).sum()['total_runs'].reset_index()
inning_scores = inning_scores[inning_scores['inning']==1]
inning_scores.head()

inning_scores['target'] = inning_scores['total_runs'] + 1
inning_scores.head()

matches.rename(columns={'id': 'match_id'}, inplace=True)
matches.head()

matches = matches.merge(inning_scores[['match_id','target']], on='match_id')
matches.head()

matches['team1'].unique()


# Mapping dictionary for old names to standardized names
```

```
team_name_mapping = {
    'Delhi Daredevils': 'Delhi Capitals',
    'Kings XI Punjab': 'Punjab Kings',
    'Rising Pune Supergiants': 'Pune Warriors',
    'Rising Pune Supergiant': 'Pune Warriors',
    'Gujarat Lions': 'Gujarat Titans',
    'Deccan Chargers': 'Sunrisers Hyderabad',
    'Royal Challengers Bengaluru': 'Royal Challengers Bangalore',
}

team_columns = ['winner', 'team1', 'team2', 'toss_winner']

# Replace the team names in team columns
matches[team_columns] = matches[team_columns].replace(team_name_mapping)


# Teams that will play in 2025 season
teams_2025 = [
    'Chennai Super Kings',
    'Delhi Capitals',
    'Gujarat Titans',
    'Kolkata Knight Riders',
    'Lucknow Super Giants',
    'Mumbai Indians',
    'Punjab Kings',
    'Rajasthan Royals',
    'Royal Challengers Bangalore',
    'Sunrisers Hyderabad',
]

# Extracting only the selected teams
matches = matches[matches['team1'].isin(teams_2025)]
matches = matches[matches['team2'].isin(teams_2025)]

matches.head()


matches['city'].unique()


# Rplace city names
map_city_names = {'Bengaluru': 'Bangalore'}
matches['city'] = matches['city'].replace(map_city_names)

matches['city'].unique()
```

```
matches.isnull().sum()


# Extracting only selected columns
matches = matches[['match_id', 'season', 'city', 'team1', 'team2', 'winner', 'target']].dropna()
matches.head()


matches.isnull().sum()

# Replace the team names in the 'batting_team' column
deliveries['batting_team'] = deliveries['batting_team'].replace(team_name_mapping)
deliveries['bowling_team'] = deliveries['bowling_team'].replace(team_name_mapping)

deliveries = deliveries[deliveries['batting_team'].isin(teams_2025)]
deliveries = deliveries[deliveries['bowling_team'].isin(teams_2025)]
deliveries.head()


# Merge to the final dataset
final = matches.merge(deliveries, on='match_id')
final.head()


# Get only second inning matches
final = final[final['inning'] == 2]
final.head()


# Add 'current_score'
final = final.copy()
final['current_score'] = final.groupby('match_id')['total_runs'].cumsum()
final.head()


# Add 'runs_left'
final.loc[:, 'runs_left'] = np.where(final['target']-final['current_score']>=0, final['target']-
final['current_score'], 0)
final.head()


# Add 'balls_left'
final.loc[:, 'balls_left'] = np.where(120 - final['over']*6 - final['ball']>=0, 120 - final['over']*6 -
final['ball'], 0)
final.head()
```

```
# Add 'wickets_left'
final.loc[:, 'wickets_left'] = 10 - final.groupby('match_id')['is_wicket'].cumsum()
final.head()


# Calculate 'current_run_rate'
final['current_run_rate'] = np.where(
    120 - final['balls_left'] > 0,
    (final['current_score'] * 6) / (120 - final['balls_left']),
    0
)

# Calculate 'required_run_rate'
final['required_run_rate'] = np.where(
    final['balls_left'] > 0,
    (final['runs_left'] * 6) / final['balls_left'],
    np.where(final['runs_left'] > 0, 1e6, 0)
)

final.head()


def result(row):
    return 1 if row['batting_team'] == row['winner'] else 0

final['result'] = final.apply(result, axis=1)
final.head()


winning_pred = final[['season', 'batting_team', 'bowling_team', 'city', 'runs_left', 'balls_left',
'wickets_left', 'current_run_rate', 'required_run_rate', 'target', 'result']]
winning_pred.head()


winning_pred['batting_team'].unique()


season_2024 = winning_pred[winning_pred['season'] == '2024']
other_seasons = winning_pred[winning_pred['season'] != '2024']

# Define X and y for both datasets
X_train_23 = other_seasons.drop(['result', 'season'], axis=1)
y_train_23 = other_seasons['result']

X_test_24 = season_2024.drop(['result', 'season'], axis=1)
y_test_24 = season_2024['result']
```

```
# Verify the splits
print("Training set size:", X_train_23.shape, y_train_23.shape)
print("Test set size:", X_test_24.shape, y_test_24.shape)


X = winning_pred.drop(['result', 'season'], axis=1)
y = winning_pred['result']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)


# Column transformer
trf = ColumnTransformer([
   ('cat', OneHotEncoder(sparse_output=False, drop='first', handle_unknown='ignore'),
['batting_team', 'bowling_team', 'city']),
   ('num', StandardScaler(), ['runs_left', 'balls_left', 'wickets_left', 'current_run_rate',
'required_run_rate', 'target'])
], remainder='passthrough')

pipe = Pipeline(steps=[
   ('preprocessor', trf),
   ('classifier', RandomForestClassifier())
])

# Hyperparameter tuning
param_grid = {
   'classifier__n_estimators': [100, 200],
   'classifier__max_depth': [10, 20],
   'classifier__min_samples_split': [2, 5]
}

grid_search = GridSearchCV(pipe, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Best model
best_model_1 = grid_search.best_estimator_
best_params_1 = grid_search.best_params_
best_score_1 = grid_search.best_score_

# Print the results
print(f"Best parameters: {best_params_1}")
print(f"Best cross-validation score: {best_score_1}")


print(X_train_23.dtypes)
```

```
# Encode the target variable
le_winner = LabelEncoder()
y_encoded = le_winner.fit_transform(pd.concat([y_train_23, y_test_24]))
y_train_23_encoded = y_encoded[:len(y_train_23)]
y_test_24_encoded = y_encoded[len(y_train_23):]

# Create a copy of X_train_23 and X_test_24 to avoid modifying the original data
X_train_23_encoded = X_train_23.copy()
X_test_24_encoded = X_test_24.copy()

# Identify categorical columns
categorical_cols = X_train_23.select_dtypes(include=['object']).columns

# Encode categorical columns
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    # Combine unique values from both train and test sets
    unique_values = pd.concat([X_train_23[col], X_test_24[col]]).unique()
    le.fit(unique_values)
    X_train_23_encoded[col] = le.transform(X_train_23[col])
    X_test_24_encoded[col] = le.transform(X_test_24[col])
    label_encoders[col] = le

# Encode the target variable
le_winner = LabelEncoder()
y_encoded = le_winner.fit_transform(pd.concat([y_train_23, y_test_24]))
y_train_23_encoded = y_encoded[:len(y_train_23)]
y_test_24_encoded = y_encoded[len(y_train_23):]

# Create a second model (for example, a Decision Tree)
dt_model = DecisionTreeClassifier(random_state=0)
dt_model.fit(X_train_23_encoded, y_train_23_encoded)

# Retrain best_model_1 if it's not a pipeline that includes preprocessing
if not isinstance(best_model_1, Pipeline):
    best_model_1.fit(X_train_23_encoded, y_train_23_encoded)

# Create a voting classifier combining these models
final_model = VotingClassifier(estimators=[
    ('rnd_model_1', best_model_1),
    ('dt_model', dt_model),
], voting='soft')

# Train the voting classifier on the train dataset
final_model.fit(X_train_23_encoded, y_train_23_encoded)
```

```
# Evaluate the performance on a test set
y_pred_encoded = final_model.predict(X_test_24_encoded)
print("Accuracy on test set:", accuracy_score(y_test_24_encoded, y_pred_encoded))
print("precision on test set:", precision_score(y_test_24_encoded, y_pred_encoded))
print("recall on test set:", recall_score(y_test_24_encoded, y_pred_encoded))
print("f1-score on test set:", f1_score(y_test_24_encoded, y_pred_encoded))

# If you want to see the actual team names
y_test_24_names = le_winner.inverse_transform(y_test_24_encoded)
y_pred_names = le_winner.inverse_transform(y_pred_encoded)

# Print some predictions
print("\npredictions data:")
for true, pred in zip(y_test_24_names[:10], y_pred_names[:10]):
    print(f"True: {true}, Predicted: {pred}")


# Create a DataFrame with actual and predicted values
results_df = pd.DataFrame({
    'Actual': y_test_24_names,
    'Predicted': y_pred_names
})

# Plot the distribution of predictions for each actual class
plt.figure(figsize=(12, 6))
for actual_class in le_winner.classes_:
    class_predictions = results_df[results_df['Actual'] ==
actual_class]['Predicted'].value_counts()
    plt.bar(class_predictions.index, class_predictions.values, alpha=0.5, label=actual_class)

plt.title('Distribution of Predictions for Each Actual Class')
plt.xlabel('Predicted Class')
plt.ylabel('Count')
plt.legend(title='Actual Class', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```