**IPL Dataset Analysis and**

**Winner Prediction Using Machine Learning**

**1.Context**This project focuses on predicting the winner of an Indian Premier League (IPL) match using machine learning techniques. The dataset includes detailed match information such as teams, toss outcomes, venue, and results, making it suitable for exploring classification models to forecast match winners.

**2.Scope**  
The goal is to analyze the dataset, preprocess the data, and evaluate multiple classification algorithms to determine the best-performing model. The project covers data cleaning, exploratory analysis, sampling strategies, feature encoding, model training and evaluation, and hyperparameter tuning to improve predictive accuracy.

**3.Audience**  
This documentation is intended for data analysts, data scientists, and technical stakeholders interested in understanding the model process and results. It is also suitable for business users seeking insights into predictive model for sports analytics.

**5.Techniques Used**

The project applied comprehensive data preprocessing, including handling missing values and encoding categorical variables. Sampling strategies were tested to address class imbalance in multiclass classification. Various machine learning models were trained and evaluated, such as Logistic Regression, SVM, KNN, Decision Trees, Random Forest, AdaBoost, Gradient Boosting, and XGBoost. Cross-validation, accuracy metrics, confusion matrices, and hyperparameter tuning (grid or random search) were used to compare and optimize models for best performance.

**6.Dataset Overview**

The IPL dataset contains detailed records of matches played in the Indian Premier League. Key features include the season, city, date, teams playing, toss winner and decision, match result, victory margin (by runs or wickets), player of the match, venue, and umpire information. This structured, tabular dataset enables exploration of factors that influence match outcomes and supports building machine learning models to predict the winning team based on these match-related attributes.

**7. Data Cleaning**

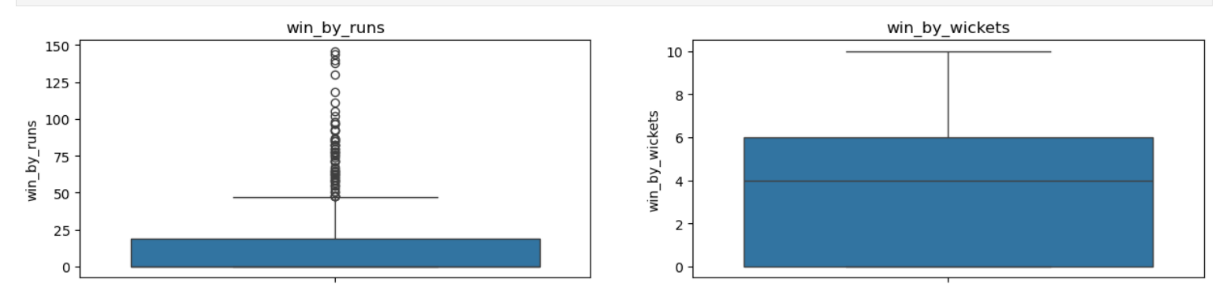
* Identified categorical columns and reviewed unique values to spot typos or inconsistencies.
* Filled missing values in the city column by cross-referencing with venue for logical consistency.
* Removed rows where result was 'no result' since these matches have no winner and can't be used for prediction.
* Standardised team and venue names to correct inconsistencies and ensure uniformity across records.
* Extracted the year from the season column to simplify analysis and improve model features.
* Dropped the date column due to formatting issues and redundancy with season.
* Removed columns with too many missing values, like umpire3, which lacked sufficient data.
* Excluded data for Kochi Tuskers Kerala since they played only in 2011, providing too little data for meaningful model.

**8. Exploratory Data Analysis (EDA)**

**Win by Runs**

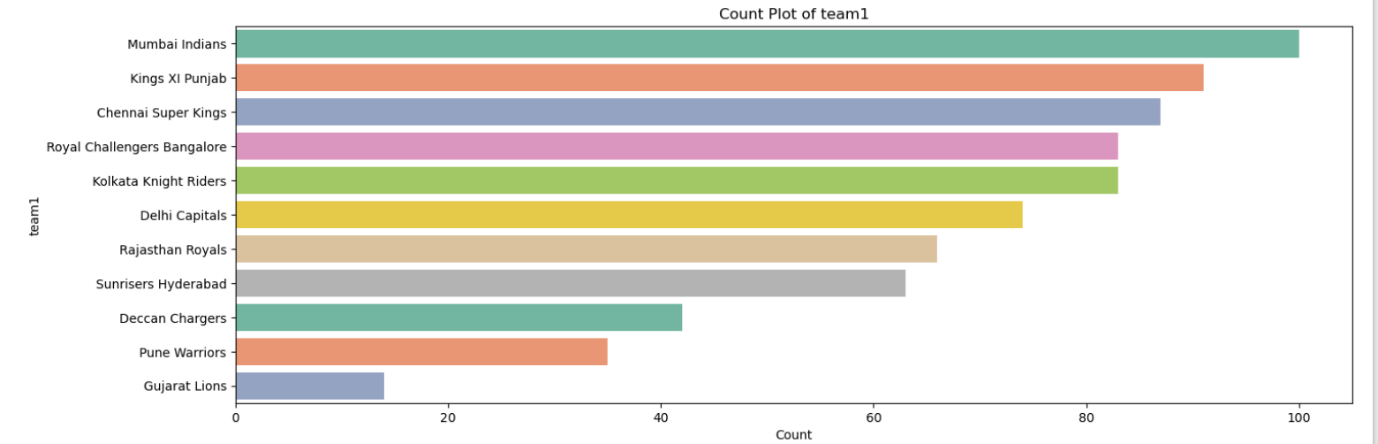
Most matches are won by fewer than 50 runs, showing close competition. However, some teams won by up to 150 runs, indicating occasional one-sided, dominant performances.

**Win by Wickets**

Wins usually happen with 4 to 6 wickets remaining, suggesting balanced contests. A few matches end with 10-wicket wins, showing complete dominance by the chasing team.

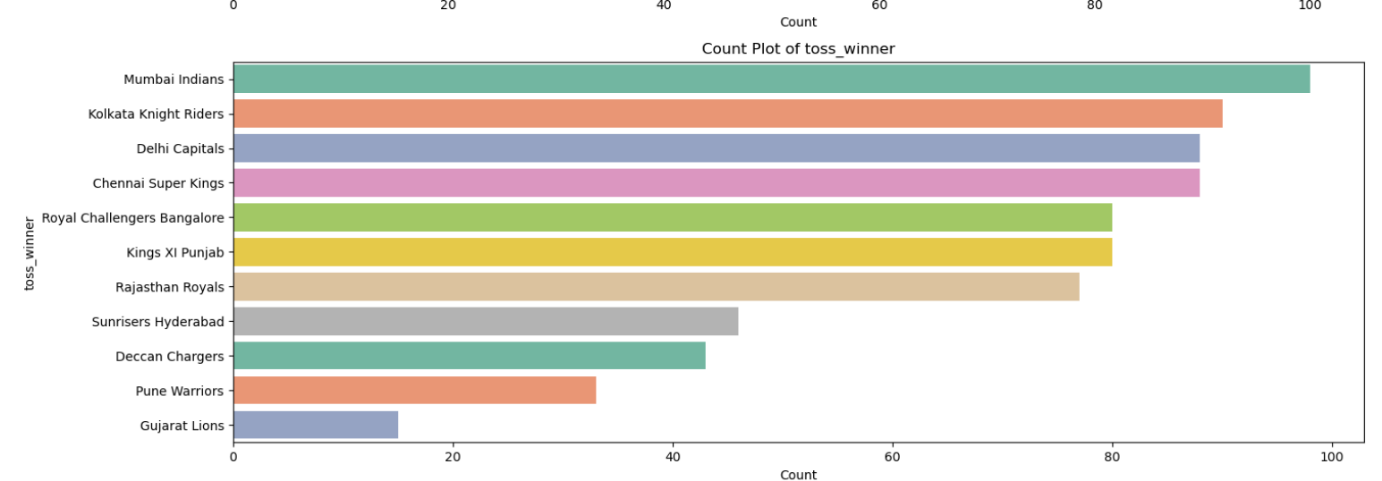
**Team Participation**

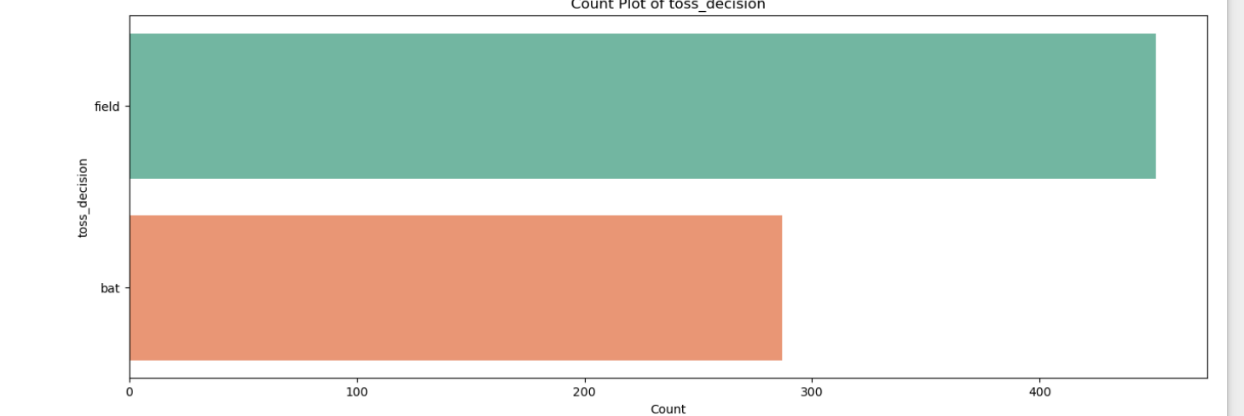
Mumbai Indians, Kings XI Punjab, and Chennai Super Kings have played the most matches, reflecting their long-term participation. Teams like Gujarat Lions and Pune Warriors played fewer seasons, so their match counts are lower.



**Toss Winners**

Mumbai Indians, Kolkata Knight Riders, and Delhi Capitals have won the toss most often. This pattern closely follows their overall match participation, indicating consistent opportunities to choose strategies.

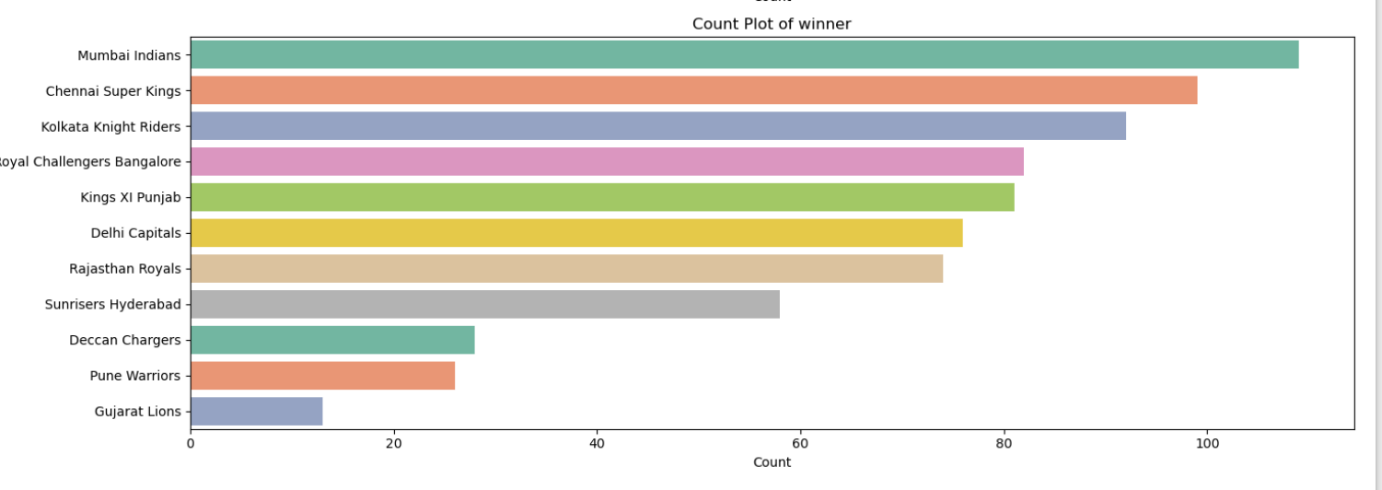


**Toss Decision**

Teams usually choose to field first after winning the toss. This shows a clear preference for chasing, likely because setting a target is often riskier while chasing offers better control.

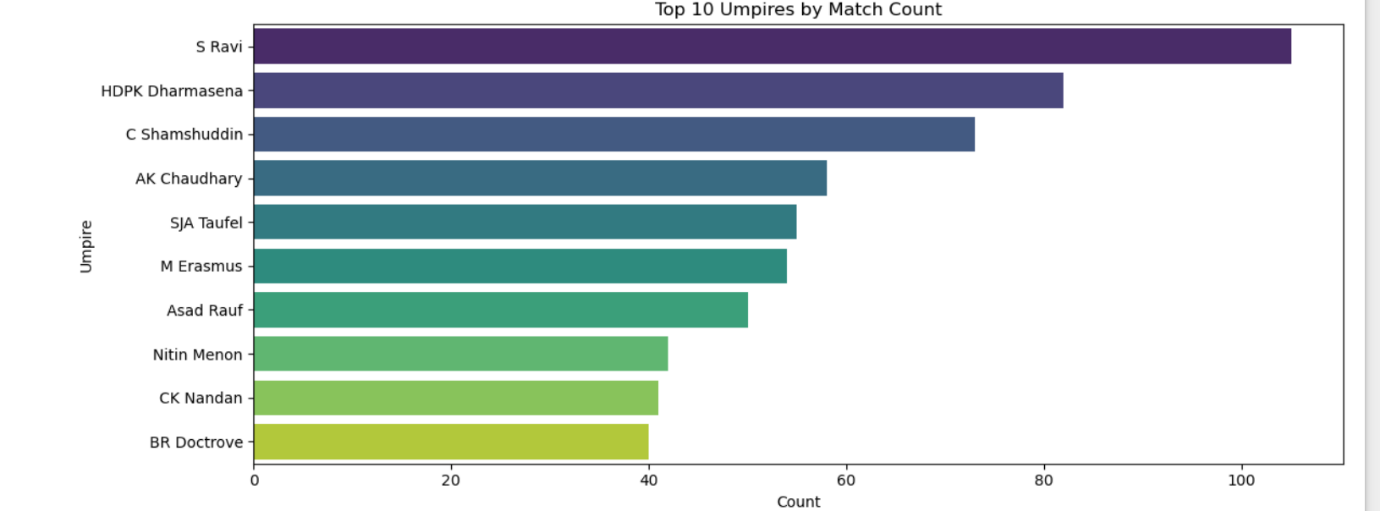
**Match Winners**

Mumbai Indians have the highest number of wins, followed by Chennai Super Kings and Kolkata Knight Riders. Teams like Deccan Chargers, Pune Warriors, and Gujarat Lions have fewer wins due to playing fewer seasons.



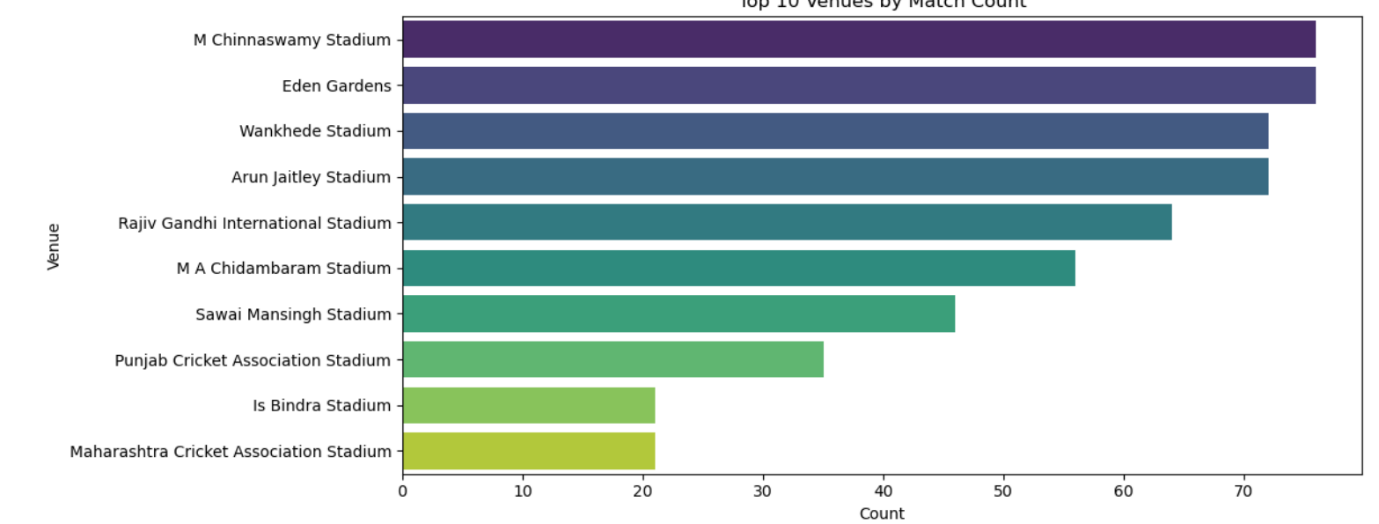
**Top Umpires**

S Ravi, HDPK Dharmasena, and C Shamsuddin have officiated the most matches. This highlights their experience and the trust placed in them to oversee high-profile games.



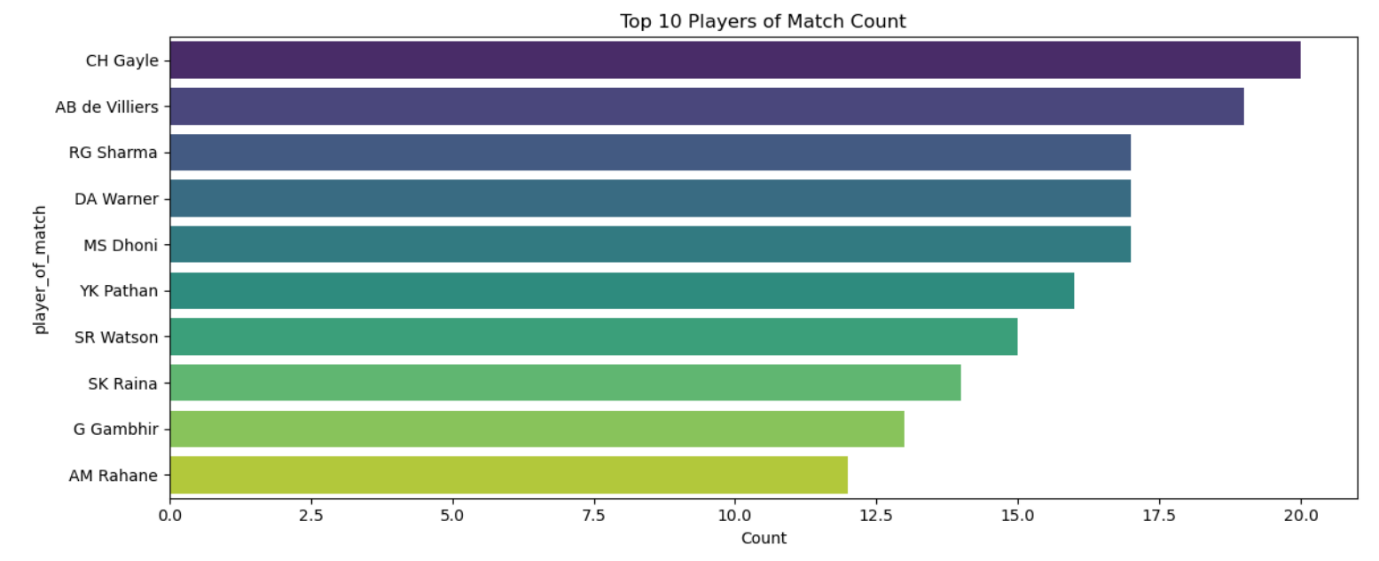
**Top Venues**

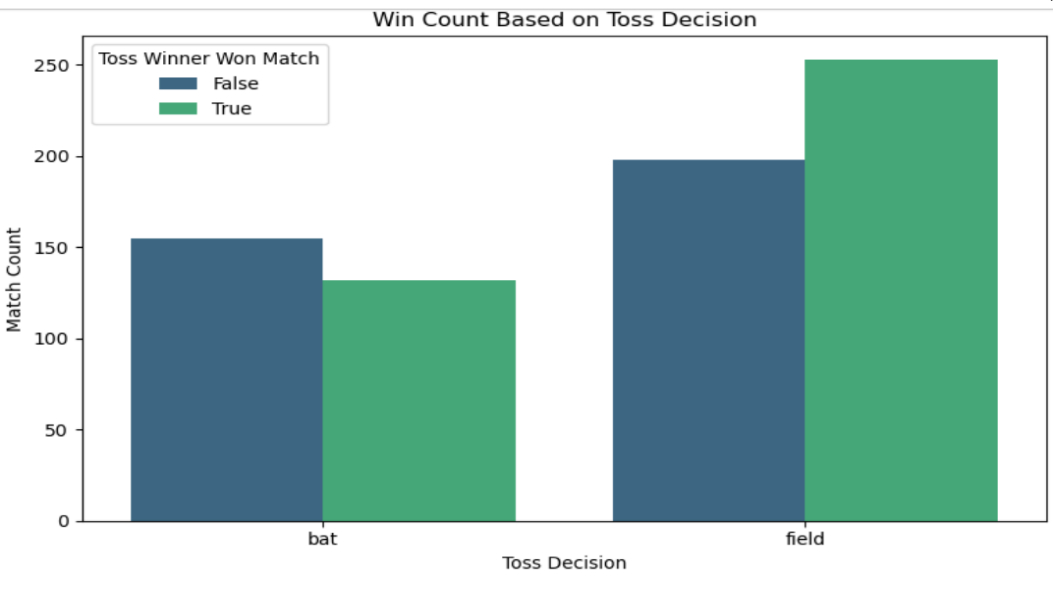
M Chinnaswamy Stadium, Eden Gardens, and Wankhede Stadium host the most matches. These popular, high-capacity venues are key locations for IPL events in India**.**

****

**Player of the Match (Top 10)**

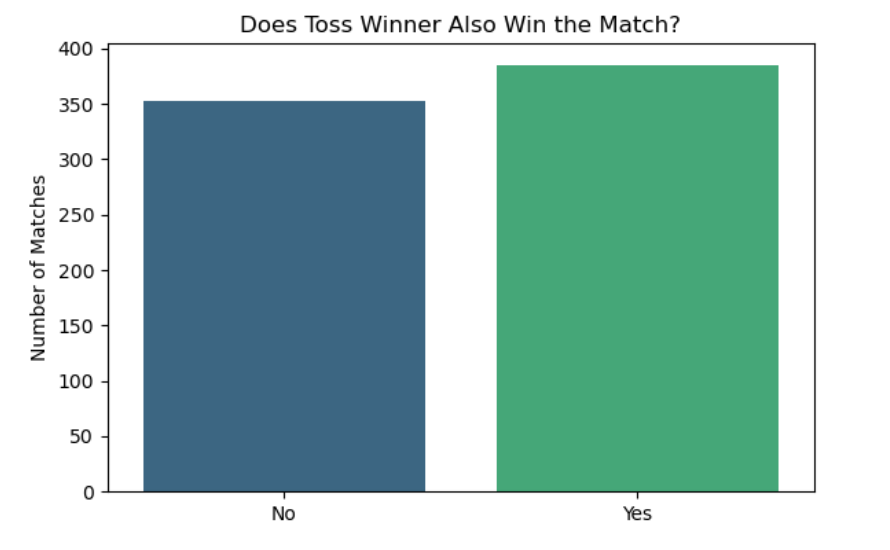
Players like CH Gayle, AB de Villiers, and RG Sharma frequently win Player of the Match awards, showing their consistent, match-winning performances and star power in batting.



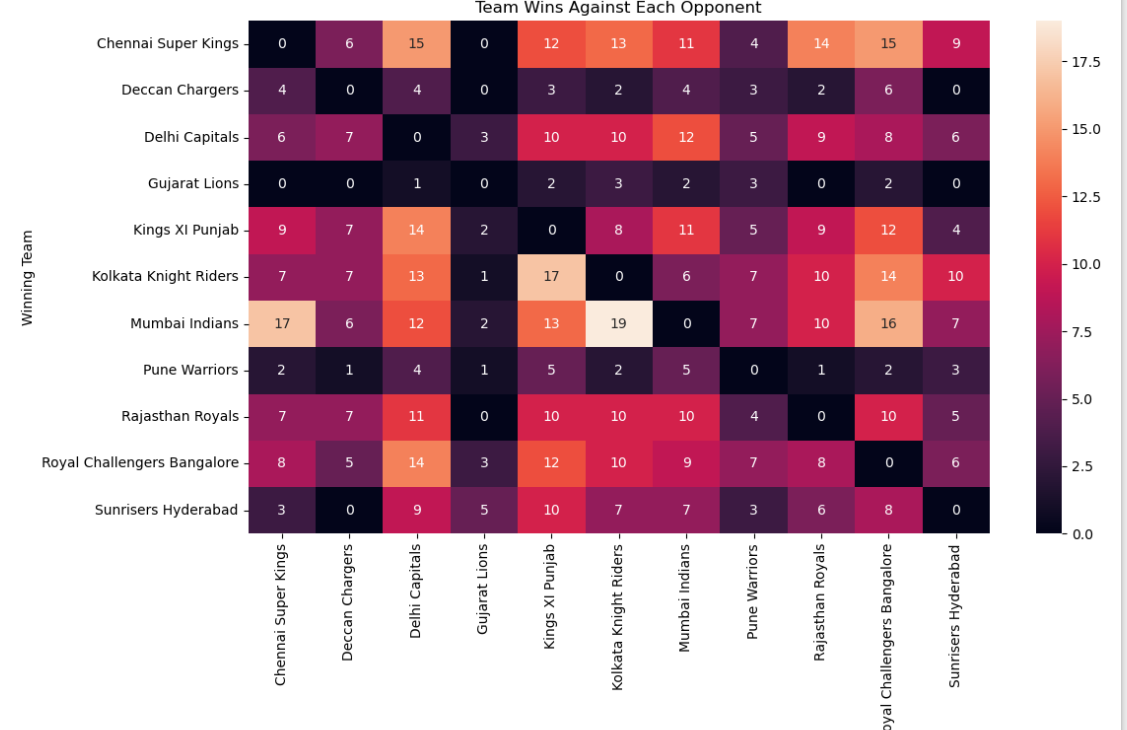
**Win Count Based on Toss Decision**

Teams that chose to field after winning the toss won significantly more matches than those who chose to bat. This suggests that fielding first gives a clear advantage, as chasing with a known target is easier and more successful.

**Team Wins When Batting First vs Chasing**

Most teams have more wins when chasing. Teams like KKR, RCB, and MI show strong chasing records. Chennai Super Kings is an exception with balanced success both ways, showing flexibility, but overall chasing is the preferred and effective strategy

**Team Wins Against Each Opponent (Heatmap Insight)**

The heatmap shows that Mumbai Indians and Chennai Super Kings consistently win against many teams, highlighting their dominance. Teams with fewer seasons, like Gujarat Lions and Pune Warriors, have predictably lower win counts, showing how team longevity affects overall success.

**9. Feature Engineering**

**Feature Selection**

Feature selection was performed using Recursive Feature Elimination (RFE) to identify the most important predictors for the model. RFE systematically removed less important features based on model performance. After running RFE, the top 5 selected features were team1, team2, toss\_winner, toss\_decision, and venue, with winner as the target variable. These features were chosen because they directly influence match outcomes and capture key game conditions.

**Feature Transforming**

Initially, label encoding was applied to all categorical features and evaluated using logistic regression. However, this approach led to underfitting, as the model struggled to capture the complexity in team and venue relationships. To improve representation, one-hot encoding was then applied to the input features (X) to create binary columns for each category, while label encoding was retained for the target variable (y). This transformation allowed the model to better learn categorical distinctions and improved its predictive performance.

**Feature Scaling**

Feature scaling was not applied in this project because the selected features were entirely categorical and transformed into encoded vectors. Since there were no continuous numerical variables in the chosen set, scaling was unnecessary.

**10. Model Implementation and Evaluation**

For every model in this project, the following consistent approach was applied:

* Sampling: Class imbalance was addressed using oversampling techniques. Data was resampled to increase representation of minority classes, particularly important for multiclass prediction.
* Train-Test Split: Data was split into training and testing sets, typically using an 80/20 split to evaluate model generalisation.
* Cross-Validation: K-fold cross-validation was used on the training set to estimate performance robustness and reduce variance in evaluation metrics.
* Hyperparameter Tuning: Grid search or random search was employed to find optimal hyperparameters for better model performance.
* Evaluation Metrics: Accuracy was the primary metric, alongside monitoring for overfitting by comparing train, test, and cross-validation results.

**Logistic Regression**

Logistic Regression was implemented with and without sampling to address class imbalance. The data was split into training and testing sets using an 80/20 split, and cross-validation was performed to assess stability. Without sampling, the model showed consistent but limited performance, with train accuracy around 56%, test accuracy 53%, and mean cross-validation accuracy 53%, indicating reasonable generalisation without overfitting. With sampling, train accuracy increased to 66%, but test accuracy dropped to 49%, suggesting overfitting to the resampled data. Hyperparameter tuning had minimal impact, reflecting the model's limited ability to handle multiclass complexity in this dataset.

**K-Nearest Neighbors (KNN)**

The KNN model was trained both with and without sampling to improve class balance. An 80/20 train-test split and cross-validation were used for evaluation. Without sampling, the model achieved train accuracy of 82%, test accuracy of 72%, and mean CV accuracy of 68.8%, indicating mild overfitting but good generalisation overall. Sampling further improved performance, with train accuracy rising to 86% and test accuracy to 79%, showing a small and acceptable train-test gap. Hyperparameter tuning focused on selecting the optimal number of neighbors and distance metrics, contributing to improved results after sampling.

**Bernoulli Naive Bayes**

Bernoulli Naive Bayes was implemented with consistent sampling, splitting, and cross-validation steps. Without sampling, the model achieved train accuracy of 62%, test accuracy of 53%, and CV accuracy around 52%, indicating stable generalisation with no major overfitting. When sampling was applied, train accuracy increased to 70%, but test accuracy remained at 53%, revealing overfitting to the resampled training data. Hyperparameter tuning was limited mainly to the smoothing parameter, with minimal impact on improving test performance, making the unsampled version more reliable.

**Support Vector Machine (SVM)**

The SVM model was implemented with sampling, data splitting, and cross-validation for evaluation. Initially, with basic settings, the model showed severe underfitting, achieving train accuracy of 24% and test accuracy between 10–18%. After hyperparameter tuning (optimising kernel type, regularisation parameter C, and gamma), train accuracy improved dramatically to 93–95%, but test accuracy only reached 64–65%, highlighting overfitting to training data. Label encoding and high-dimensional sparse features contributed to this behaviour, making SVM less effective for this multiclass problem with the available data.

**Decision Tree**

Decision Tree models were implemented with sampling to address class imbalance, using an 80/20 train-test split and cross-validation to evaluate performance. The model achieved 100% accuracy on the training set, demonstrating its tendency to memorise the data, but test accuracy remained high at around 93–94%, with cross-validation accuracy around 87%, showing reasonable generalisation. Hyperparameter tuning focused on controlling tree depth and minimum samples for splits to mitigate overfitting, resulting in stable and strong performance for this dataset size.

**Random Forest**

Random Forest was trained with sampling to balance classes, using train-test splitting and cross-validation for evaluation. The model consistently achieved 100% train accuracy and around 90–91% test accuracy, with cross-validation accuracy close to 89%, showing a small train-test gap and strong generalisation. Hyperparameter tuning (number of trees, max depth, min samples split) further improved model stability, with a slight boost in test accuracy after tuning. This approach confirmed Random Forest as a robust choice for the given dataset.

**AdaBoost**

AdaBoost was implemented with sampling and the standard train-test split, along with cross-validation. Initially, before tuning, the model showed low accuracy (between 23% and 38%), clearly underfitting the data. After hyperparameter tuning, especially by using a Decision Tree as the base estimator, the model’s train accuracy improved to 100%, test accuracy reached 93%, and cross-validation accuracy was around 86%, indicating much better generalisation and effectiveness. This demonstrated the value of tuning and choosing a strong base learner.

**Gradient Boosting**

Gradient Boosting was applied with sampling, splitting the data for training and testing, and validating performance through cross-validation. The model achieved excellent results with 100% train accuracy, 99% test accuracy, and cross-validation accuracy around 97%, showing minimal overfitting and strong generalisation. Hyperparameter tuning further improved consistency, making this one of the best-performing models on the dataset without requiring extensive adjustments.

**XGBoost**

XGBoost was trained with sampling to handle class imbalance, along with train-test splitting and cross-validation. It delivered exceptional performance, with 100% train accuracy, 98% test accuracy, and cross-validation accuracy around 96%, showing tiny gaps and excellent generalisation. Even without extensive hyperparameter tuning, XGBoost performed strongly, and further tuning (learning rate, max depth, number of estimators) confirmed it as one of the most effective and reliable models for this multiclass prediction task.



**11. Conclusion**

Feature transformation, selection, and encoding improve performance. Sampling can balance classes but may add noise in multiclass problems. Hyperparameter tuning reduces overfitting and improves generalisation. Simpler models are easier to interpret, while ensembles perform better. Gradient Boosting and XGBoost delivered the best, with high accuracy and consistent results. Use strong feature engineering, careful sampling, and tuning for best outcomes.

**12. Future Scope**

This project can be expanded and improved in several meaningful ways. Collecting more historical IPL match data or incorporating newer seasons will help increase sample size and improve model learning and generalisation. Additional features such as player statistics, team form, weather conditions, and pitch reports can be integrated to capture more predictive signals. Advanced feature engineering, including interaction terms and domain-specific insights, could boost accuracy further. Exploring more sophisticated sampling techniques (like SMOTE or ADASYN for multiclass) may address imbalance without introducing as much noise. Experimenting with deep learning models or ensemble stacking could also enhance performance, especially with richer data. Finally, deploying the trained model as a web app or API would make it usable for fans, analysts, or betting agencies, bringing practical value to the predictive system.

**Date : 03-07-2025 by,**

**Yahavarshini E**

**Top of Form**

**Bottom of Form**