A

PROJECT REPORT

ON

IPL Win Prediction

IN

THE PARTIAL FULFILLMENT OF REQUIREMENT FOR THE AWARD OF THE DEGREE

OF

MASTER OF COMPUTER APPLICATION

Under the supervision of:

Mr. Sumit Singha
Assistant Professor
Department of Computer Applications
Asansol Engineering College, Asansol-713305

Submitted by:

Rishabh Agarwal
Akash Sen
Aman Kumar Gupta
Shaibal Patra
Raj Kumar Jha

DEPARTMENT OF COMPUTER APPLICATIONS
ASANSOL ENGINEERING COLLEGE, ASANSOL (AFFILIATED TO MAKAUT UNIVERSITY) WEST BENGAL KOLKATA – 700064

CANDIDATE'S DECLARATION

We, "**Rishabh Agarwal**", "**Akash Sen**", "**Aman Kumar Gupta**", "**Shaibal Patra**" and "**Raj Kumar Jha**", hereby declare that the work presented in the project report entitled "**IPL win Prediction**" submitted to Department of Computer Applications, Asansol Engineering College, affiliated to **Maulana Abul Kalam Azad University of Technology, West Bengal** for the partial fulfillment of the award of degree of "**Master of Computer Application**" is an authentic record of our work carried out during **the MCA 4th Semester 2024,** under the supervision of **Dr. Arnab Chakraborty**, Trainer (as External Guide) and **Mr. Sumit Singha**, Department of Computer Applications, **Asansol Engineering College** (as Internal Guide).

The matter embodied in this project report has not been submitted elsewhere by anybody for the award of any other degree.

Group Members:

Rishabh Agarwal
Maulana Abul Kalam Azad University of Technology
10871023036

Akash Sen
Maulana Abul Kalam Azad University of Technology
10871023009

Aman Kumar Gupta
Maulana Abul Kalam Azad University of Technology
10871023010

Shaibal Patra
Maulana Abul Kalam Azad University of Technology
10871023044

Raj Kumar Jha
Maulana Abul Kalam Azad University of Technology
10871023035

ASANSOL ENGINEERING COLLEGE

Kanyapur, Vivekananda Sarani, Asansol, Paschim Bardhaman, West Bengal-713305

Phone: 225-3057, 225-2108 Telefax: (0341) 225-6334

Email: principal.aecwb@gmail.com Website: www.aecwb.edu.in

Reference No.: CA-MCA/2025/ **Date: …….**

CERTIFICATE

To whom it may concern

This is to certify that the project titled "**IPL Win Prediction**" is a bonafide work carried out by **Rishabh Agarwal (10871023036)**, **Akash Sen (10871023009)**, **Aman Kumar Gupta (10871023010)**, **Shaibal Patra (10871023044)**, **Raj Kumar Jha (10871023035)** in the partial fulfillment of the awards of the degree of **Master of Computer Application** from Asasnol Engineering College, Asansol, under the supervision of **Mr. Sumit Singha**.

Mr. Sumit Singha
Assistant Professor
Dept. of Computer Application

Dr. Anup Mukhopadhyay
Head of Department
Dept. of Computer Application

Dr. G.S.Panda
Principal In-Charge
Asansol Engineering College

Contents

Acknowledgement

We would like to thank people who were part of this work in numerous ways. In particular, we wish to thank **Dr. Arnab Chakraborty** (Trainer, External Guide), our project guides for their suggestions and improvements in this project and providing continuous guidance at each and every stage of the project.

Thanks are extended especially to our guide **Ms. Sumit Singha** (Asst. Professor, CA Department, Asansol Engineering College, Asansol and to **Dr. Anup Mukhopadhyay** (HOD, CA Department, Asansol Engineering College). We also thank **Ms. Debika Chatterjee** (Training and Placement Officer, Asansol Engineering College) for her valuable support. We must also be thankful to classmates and friends for their continuous co-operations and help in completing this project.

Last but not the least, We want to express our thanks to our parents and family members for their support at every step of life.

_____
Rishabh Agarwal

_____
Akash Sen

_____
Aman Kumar Gupta

_____
Shaibal Patra

_____
Raj Kumar Jha

Date: **2nd Year**

Place: Asansol **Dept. of Computer Application**

System Study

The **System Study** forms a foundational part of this project report, aiming to evaluate the problem domain, examine existing alternatives, and present a thorough analysis of the proposed IPL Win Predictor model. It includes understanding user requirements, analyzing feasibility, and designing the architecture to meet the outlined objectives.

1. Problem Identification

Cricket, particularly the Indian Premier League (IPL), is a highly dynamic and unpredictable sport. Match outcomes can change dramatically within a few deliveries, making it difficult for viewers, analysts, and even professional commentators to accurately assess which team has the upper hand during a live match.

Traditionally, match predictions have relied on expert opinions or simplistic heuristics based on runs required, balls remaining, and wickets lost. However, such approaches:

- Fail to quantify win/loss probabilities.
- Do not adapt to real-time inputs.
- Lack data-driven insights.

There was a need for an intelligent system capable of leveraging historical match data and real-time statistics to predict the probability of a team winning a game. This gap motivated the development of a **machine learning-based IPL Win Predictor**.

2. Existing Systems

A few platforms such as CricBuzz, ESPNcricinfo, and official IPL broadcasters provide live match coverage with basic win probability meters. However:

- These systems are typically proprietary and lack transparency in how predictions are made.
- They do not allow user interaction or scenario simulation.
- They are not designed as open platforms for experimentation, learning, or community-based analysis.

There was a lack of an **open-source**, **interactive**, and **data-driven** application where users could enter live match data and receive win/loss probabilities through a visually intuitive interface.

3. Proposed System

The proposed system addresses these limitations through a machine learning-based web application that provides win probability predictions during an IPL match. Key highlights include:

a. User Input Interface

Users can input:

- Batting and bowling teams
- Match city (venue)
- Target score
- Current score
- Overs completed

- Wickets fallen

b. Data Processing

The system computes:

- Runs left
- Balls remaining
- Current run rate (CRR)
- Required run rate (RRR)
- Remaining wickets

c. Machine Learning Model

The backend model is trained on historical IPL data using several machine learning algorithms:

- **Logistic Regression**: For linear decision boundaries and baseline benchmarking.
- **Random Forest**: For ensemble learning with decision trees.
- **AdaBoost & XGB**: For improving accuracy via boosting.
- **SVM**: For optimized decision boundaries in high-dimensional space.

The final model is selected based on performance metrics such as **accuracy**, **F1-score**, and **ROC-AUC**.

d. Interactive Front-End

A web application is built using:

- **Streamlit** for UI components.
- **Plotly** for dynamic visualizations such as pie charts.
- **Pickle** for model serialization.

The app presents users with the predicted win and loss probabilities along with real-time visual feedback.

4. Feasibility Study

a. Technical Feasibility

- Python is used as the primary language with supporting libraries like `scikit-learn`, `pandas`, `plotly`, and `streamlit`.
- The model is trained and deployed using open-source tools, making the system highly feasible for development and maintenance.

b. Economic Feasibility

- No proprietary software or commercial tools are used.
- Deployment on platforms like Streamlit Cloud or localhost ensures zero hosting cost in development and early deployment phases.

c. Operational Feasibility

- The system is highly user-friendly and requires no technical expertise from end-users.

- Visual elements and real-time updates make it accessible for cricket fans and casual users alike.

d. Schedule Feasibility

The system was developed in structured phases:

- Data collection and preprocessing.
- Model training and evaluation.
- UI development and integration.
- Testing and deployment.

All stages were completed within a reasonable timeline using agile development practices.

5. System Architecture Overview

The architecture of the IPL Win Predictor system follows a modular and scalable structure:

a. Data Layer

- Cleaned and structured historical IPL match data.
- Features include team names, venue, score details, overs, wickets, and match outcomes.

b. Model Layer

- Trained machine learning model (`pipe.pkl`) created using a pipeline approach for preprocessing and prediction.
- Models were evaluated for optimal performance and interpretability.

c. Logic Layer

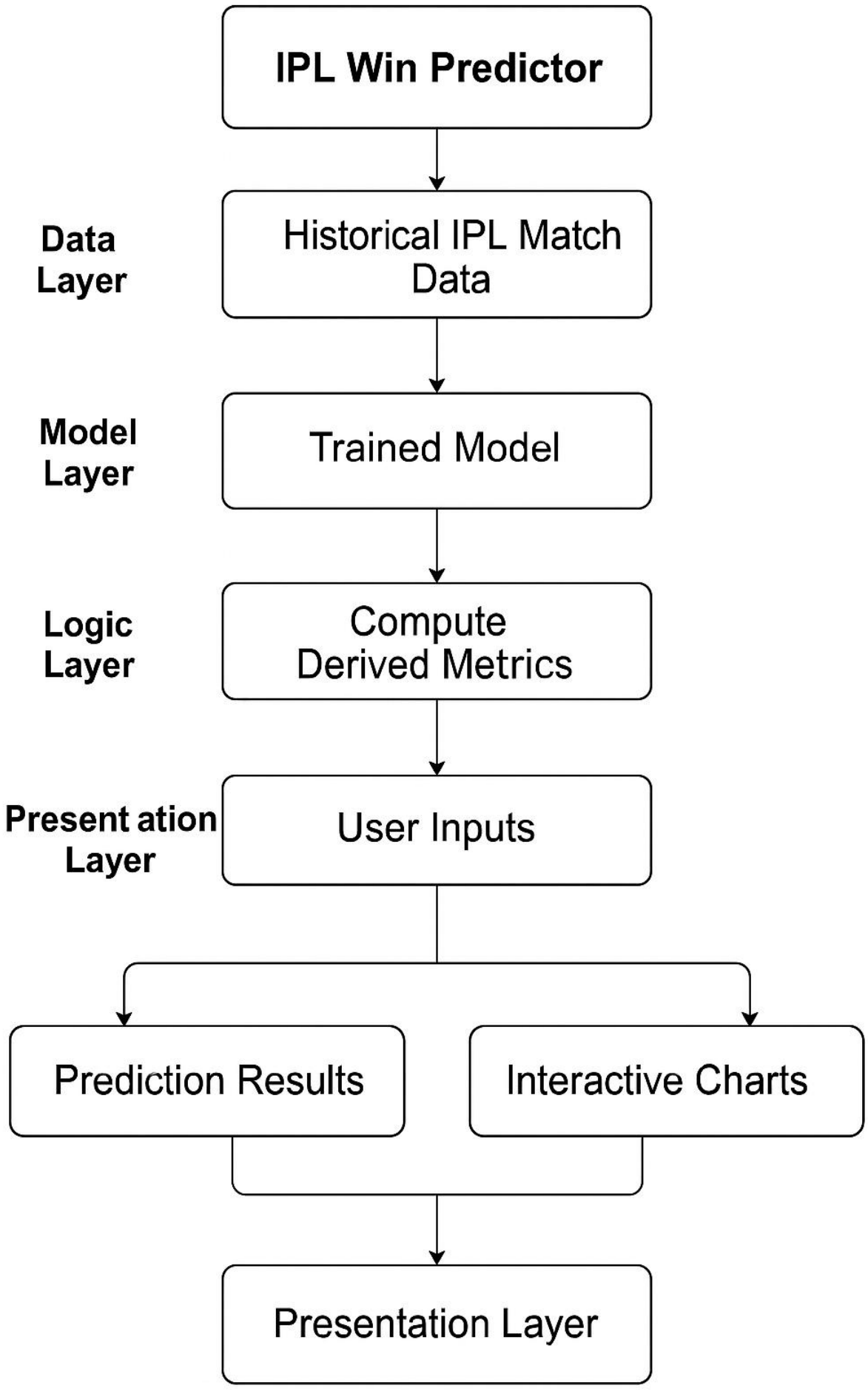
- Computes derived metrics: runs left, balls left, CRR, RRR, and wickets remaining.
- Packages inputs into a format compatible with the model.

d. Presentation Layer

- Built using Streamlit for input forms and interactive charts.
- Displays prediction results and win probabilities in textual and graphical formats.

6. Benefits of the Proposed System

- **Real-Time Insights**: Users receive instant feedback on win/loss chances.
- **Interactive Learning Tool**: Useful for understanding match dynamics and model behavior.
- **Open and Extendable**: The system can be easily extended with more data or new models.
- **Engaging Visuals**: Enhances user experience through charts and stats.


IPL Win Predictor
Data Layer
Historical IPL Match Data
Model Layer
Trained Model
Logic Layer
Compute Derived Metrics
Present ation Layer
User Inputs
Prediction Results
Interactive Charts
Presentation Layer

Project Objective

The main objective of this project is to build a data-driven predictive model that forecasts the probable winner of an IPL (Indian Premier League) match based on historical match data and performance metrics. The project leverages data preprocessing, feature engineering, and machine learning to model real-world scenarios for T20 cricket matches.

Key Goals:

- **Data Collection & Cleaning:**
  - Load and process two key datasets: matches.csv and deliveries.csv.
  - Standardize team names to ensure consistency (e.g., "Delhi Daredevils" → "Delhi Capitals").
  - Filter out irrelevant or outdated teams to focus only on current franchises.
- **Feature Engineering:**
  - Calculate first-inning scores using ball-by-ball data.
  - Merge score data with match-level data to derive useful features like total first-inning runs.
- **Visualization & Insights:**
  - Use libraries like Matplotlib and Seaborn for EDA (Exploratory Data Analysis).
  - Generate insights about team performance trends and scoring behavior over the years.
- **Predictive Modeling:**
  - Use historical performance metrics such as runs scored, toss decisions, and venue to predict outcomes.
  - Apply classification models (likely Logistic Regression or Decision Tree based on standard practice) to predict win probability for chasing teams.
- **Real-time Input:**
  - Enable users to input match scenarios (like team names, target score, overs, etc.) and get win probabilities as output.

Outcome:

- A functional prediction system that estimates the likelihood of a team winning an IPL match, particularly while chasing a target.
- A deeper understanding of how various factors like toss result, venue, and first-inning score affect match outcomes.

Project Scope

This project is scoped to analyze historical IPL data to build a predictive model that estimates match outcomes under certain input scenarios. It covers a full data science pipeline from raw data to deployment-ready insights.

In Scope:

- **Data Sources:**
  - Match-level data (`matches.csv`)
  - Ball-by-ball data (`deliveries.csv`)
  - Filtering based on current IPL teams (8 franchises as of recent seasons).
- **Data Preprocessing:**
  - Handling null values, renaming outdated teams, dropping irrelevant columns.
  - Merging datasets on match ID to link innings scores with match metadata.
- **Feature Selection:**
  - Batting team and bowling team
  - Venue of the match
  - First-inning total runs
  - Current Run Rate
  - Required Run Rate
  - Wickets Fallen
- **Modeling & Evaluation:**
  - Building a classification model (Logistic Regression, Random Forest, AdaBoost, XGBoost, SVM) to predict match results.
  - Training the model using features like team names, runs, overs, and wickets.
  - Testing the model against known outcomes to validate accuracy.
- **Visualization:**
  - Trends in scoring
  - Team-wise win ratios
  - Impact of toss and venue on results
- **Application Use Case:**
  - Ideal for sports analysts, IPL fans, or betting enthusiasts looking for data-backed predictions.

Out of Scope:

- Player-specific performance prediction (e.g., batsman scoring prediction).
- Real-time score prediction during ongoing matches.
- Live API integration or deployment as a web/mobile app.

Data Description

Source of the data:

Kaggle. The datasets used are adapted and filtered versions of the original IPL datasets available on Kaggle.

Dataset 1: matches.csv

- **Rows**: ~756
- **Columns**: 18

Attribute Name	Type	Description	Target Attribute
id	Non-categorical	Unique match identifier	No
season	Categorical	IPL season/year	No
city	Categorical	City where the match was played	No
date	Non-categorical	Date of the match	No
team1	Categorical	First team	No
team2	Categorical	Second team	No
toss_winner	Categorical	Team that won the toss	No
toss_decision	Categorical	Decision taken after winning the toss (bat/field)	No
result	Categorical	Result type (normal/tie/no result)	No
winner	Categorical	Match winner	Yes

Table 1: Data Description (matches.csv)

Dataset 2: deliveries.csv

- **Rows**: ~1,798,000
- **Columns**: 21

Columns	Attribute Name	Type	Description	Target Attribute
match_id	match_id	Non-categorical	Match identifier	No
inning	inning	Categorical	1st or 2nd innings	No
over	over	Non-categorical	Over number	No
ball	ball	Non-categorical	Ball number within an over	No
batsman	batsman	Categorical	Name of the batsman on strike	No
bowler	bowler	Categorical	Name of the bowler	No
total_runs	total_runs	Non-categorical	Runs scored off the delivery including extras	No
player_dismissed	player_dismissed	Categorical	Name of the player dismissed	No

Table 2: Data Description (deliveries.csv)

Dataset 3: final_dataset.csv

- **Rows:** ~760
- **Columns**: 12

Columns	Attribute Name	Type	Description	Target Attribute
batting_team	batting_team	Categorical	Team currently batting	No
bowling_team	bowling_team	Categorical	Opponent team bowling	No
city	city	Categorical	City where the match is being played	No
runs_left	runs_left	Non-categorical	Runs left to win	No
balls_left	balls_left	Non-categorical	Balls remaining	No
wickets_left	wickets_left	Non-categorical	Wickets remaining	No
total_runs_x	total_runs_x	Non-categorical	Target score	No
current_run_rate	current_run_rate	Non-categorical	Current run rate	No
required_run_rate	required_run_rate	Non-categorical	Required run rate	No
result	result	Categorical	Win (1) or loss (0)	Yes

Table 3: Data Description (final_dataset.csv)

Now we will pre-process the data. The methodology followed is given below:

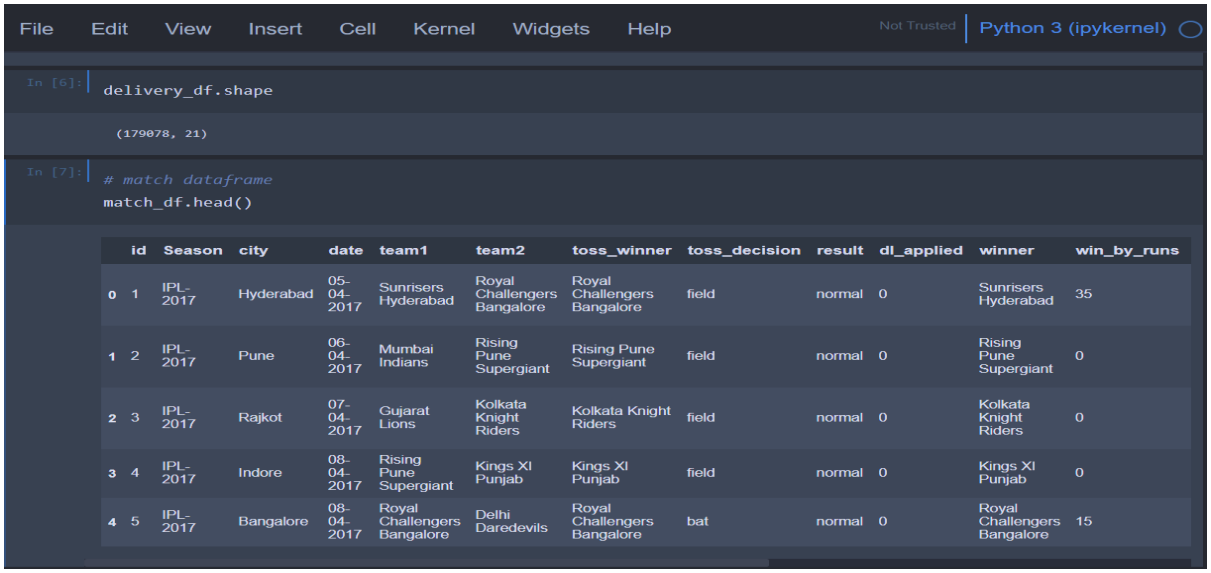
- ❒ Checking for null values.
  - ➢ If null values are present, we will fill them or drop the row containing the null value based on the dataset.
- ❒ Check for duplicate values.
  - ➢ If duplicate values are present, we will remove them.

Data Preprocessing

This section prepares the raw IPL data into a structured form usable for modeling.

1.1 Loading Datasets

- `matches.csv` and `deliveries.csv` are loaded using `pandas.read_csv()`.
- Shapes of dataframes are displayed to understand their size.



File Edit View Insert Cell Kernel Widgets Help — Not Trusted | Python 3 (ipykernel)

In [6]:
```
delivery_df.shape
```

```
(179078, 21)
```

In [7]:
```
# match dataframe
match_df.head()
```

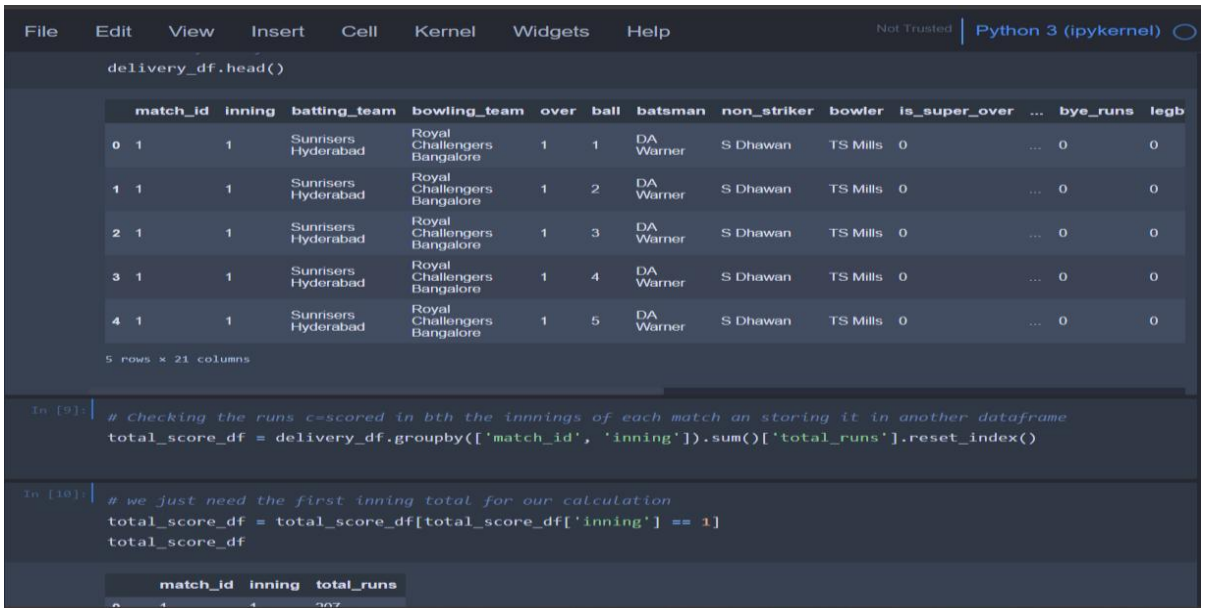
	id	Season	city	date	team1	team2	toss_winner	toss_decision	result	dl_applied	winner	win_by_runs
0	1	IPL-2017	Hyderabad	05-04-2017	Sunrisers Hyderabad	Royal Challengers Bangalore	Royal Challengers Bangalore	field	normal	0	Sunrisers Hyderabad	35
1	2	IPL-2017	Pune	06-04-2017	Mumbai Indians	Rising Pune Supergiant	Rising Pune Supergiant	field	normal	0	Rising Pune Supergiant	0
2	3	IPL-2017	Rajkot	07-04-2017	Gujarat Lions	Kolkata Knight Riders	Kolkata Knight Riders	field	normal	0	Kolkata Knight Riders	0
3	4	IPL-2017	Indore	08-04-2017	Rising Pune Supergiant	Kings XI Punjab	Kings XI Punjab	field	normal	0	Kings XI Punjab	0
4	5	IPL-2017	Bangalore	08-04-2017	Royal Challengers Bangalore	Delhi Daredevils	Royal Challengers Bangalore	bat	normal	0	Royal Challengers Bangalore	15

1.2 Inspecting DataFrames

- `.head()`, `.info()`, and `.describe()` are used.
- Focus is on match-level and ball-level data.

1.3 Calculating First Innings Totals

- `delivery_df` is grouped by `match_id` and `inning` to compute `total_runs`.
- Only `inning == 1` is retained as it represents the target score.



File Edit View Insert Cell Kernel Widgets Help — Not Trusted | Python 3 (ipykernel)

```
delivery_df.head()
```

	match_id	inning	batting_team	bowling_team	over	ball	batsman	non_striker	bowler	is_super_over	...	bye_runs	legb
0	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	1	DA Warner	S Dhawan	TS Mills	0	...	0	0
1	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	2	DA Warner	S Dhawan	TS Mills	0	...	0	0
2	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	3	DA Warner	S Dhawan	TS Mills	0	...	0	0
3	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	4	DA Warner	S Dhawan	TS Mills	0	...	0	0
4	1	1	Sunrisers Hyderabad	Royal Challengers Bangalore	1	5	DA Warner	S Dhawan	TS Mills	0	...	0	0

5 rows × 21 columns

In [9]:
```
# checking the runs c=scored in bth the innnings of each match an storing it in another dataframe
total_score_df = delivery_df.groupby(['match_id', 'inning']).sum()['total_runs'].reset_index()
```

In [10]:
```
# we just need the first inning total for our calculation
total_score_df = total_score_df[total_score_df['inning'] == 1]
total_score_df
```

	match_id	inning	total_runs

1.4 Merging with Match Data

- `match_df` is merged with `total_score_df` using `match_id`.
- Columns like `team1`, `team2`, and `total_runs` are retained.

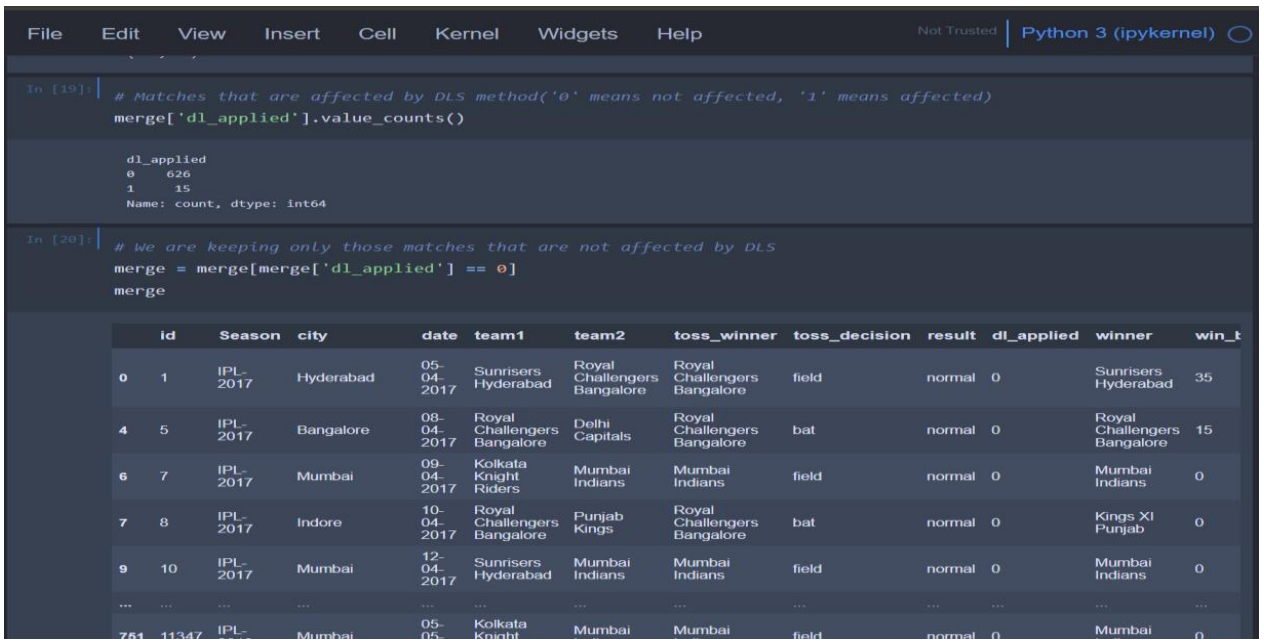
1.5 Team Name Standardization

- Legacy team names (`Kings XI Punjab`, `Delhi Daredevils`) are replaced with current ones using `str.replace()`.
- Only currently active teams are retained using a whitelist.

```
merge['team1'] = merge['team1'].str.replace('Delhi Daredevils', 'Delhi Capitals')
merge['team2'] = merge['team2'].str.replace('Delhi Daredevils', 'Delhi Capitals')

merge['team1'] = merge['team1'].str.replace('Deccan Chargers', 'Sunrisers
Hyderabad')
merge['team2'] = merge['team2'].str.replace('Deccan Chargers', 'Sunrisers
Hyderabad')

merge['team1'] = merge['team1'].str.replace('Kings XI Punjab', 'Punjab Kings')
merge['team2'] = merge['team2'].str.replace('Kings XI Punjab', 'Punjab Kings')
```

1.6 Filtering for Valid Matches

- DLS-affected matches (`dl_applied == 1`) are dropped.
- This ensures uniformity in target-setting logic.



1.7 Merging Ball-by-Ball Data

- `merge` is again joined with `delivery_df` on `match_id`.
- Only 2nd innings deliveries are retained (`inning == 2`) to analyze chases.

1.8 Handling Missing Values

- `city` is filled with mode.
- `winner` is filled with "No Result" where applicable.

```
merge_df = merge_df.assign(
    city = merge_df['city'].fillna(merge_df['city'].mode()[0]),
    winner = merge_df['winner'].fillna('No Result')
)
```

1.9 Final Feature Cleanup

- Final features prepared: `batting_team`, `bowling_team`, `city`, `total_runs_x`, `winner`, etc.
- String columns are stripped of extra spaces and checked for consistent casing.

```
final_df =
merge_df[['batting_team','bowling_team','city','runs_left','balls_left','wickets','t
otal_runs_x','crr','rrr','result']]
final_df
```

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a crucial phase in our fake news detection project, involving the initial investigation of the dataset to uncover patterns, spot anomalies, test hypotheses, and check assumptions through summary statistics and graphical representations. EDA helps us understand the underlying structure of the data, identify relationships between variables, and guide further preprocessing and modelling steps.

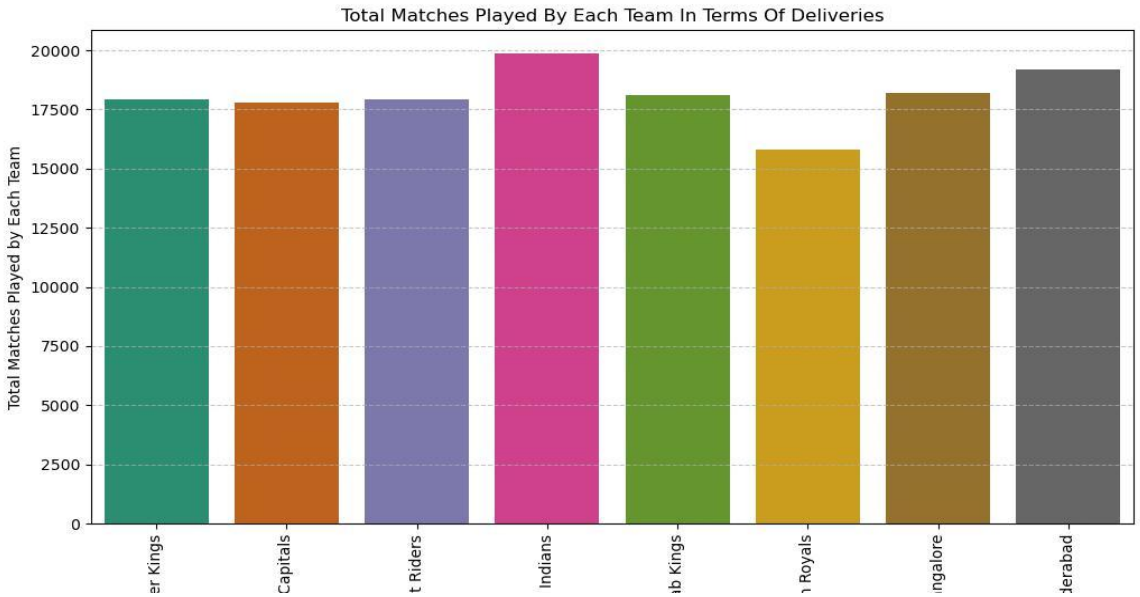


Fig 1: *The above graph showing Total matches played by each team in terms of deliveries*

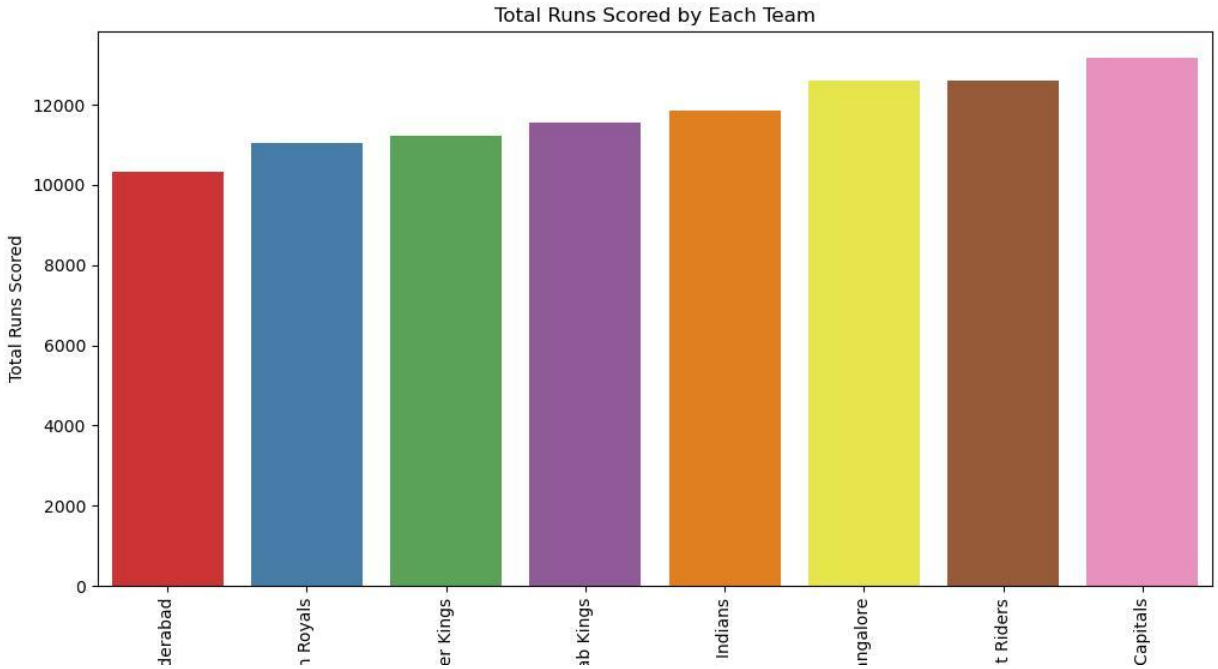


Fig 2: *Delhi Capitals scored the highest number of runs over the years up to 2019 and Sunrisers Hyderabad scored the less amount of runs compared to other teams.*

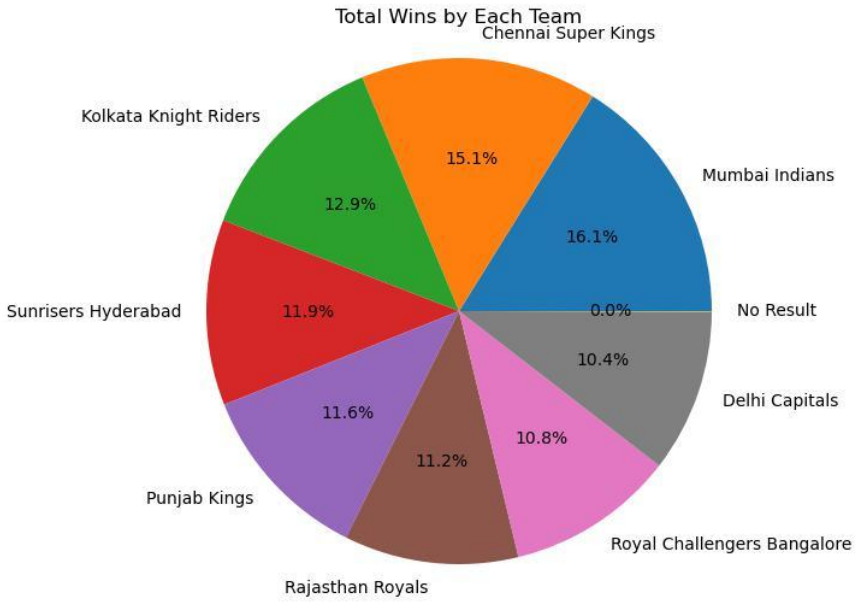


Fig 3:
Pie chart showcasing win percentage by each team

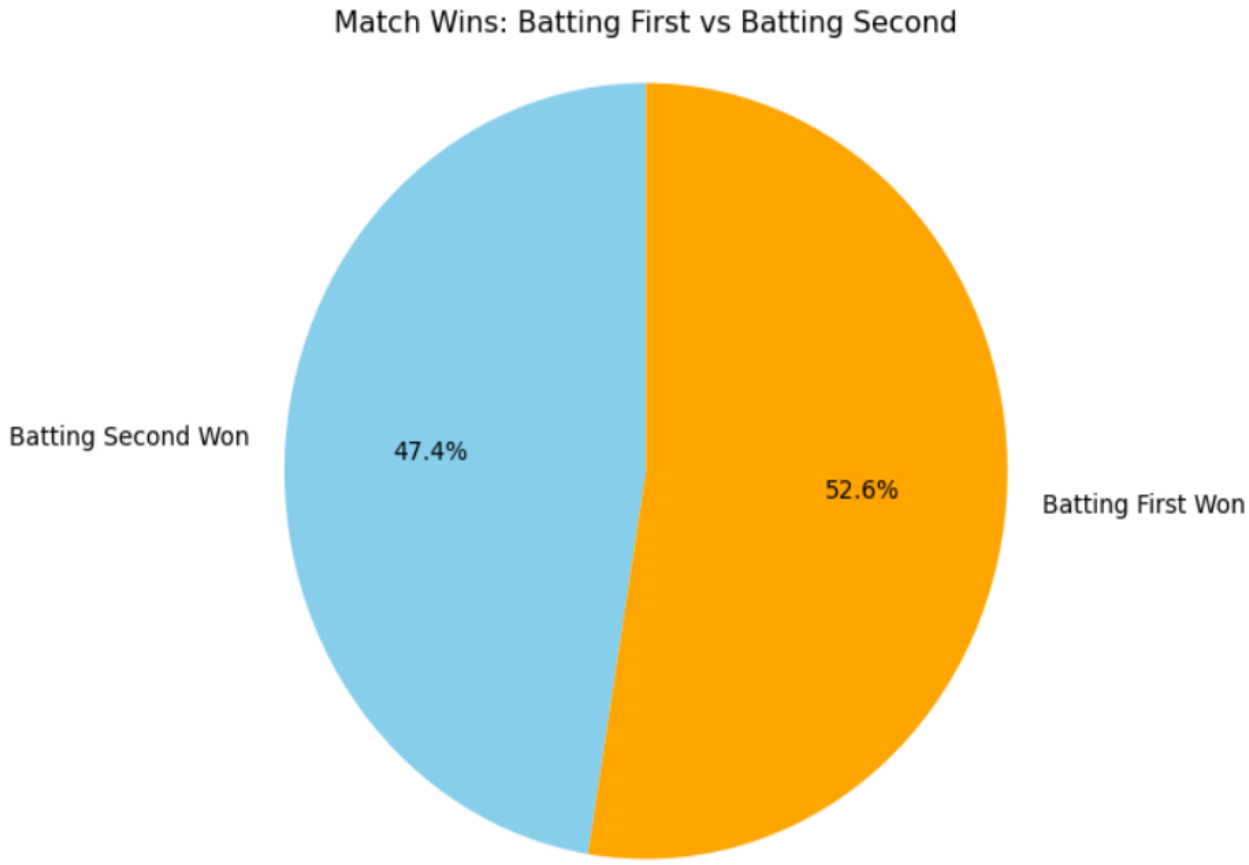


Fig 4:
Pie chart showcasing winning percentage of teams batting first vs batting second

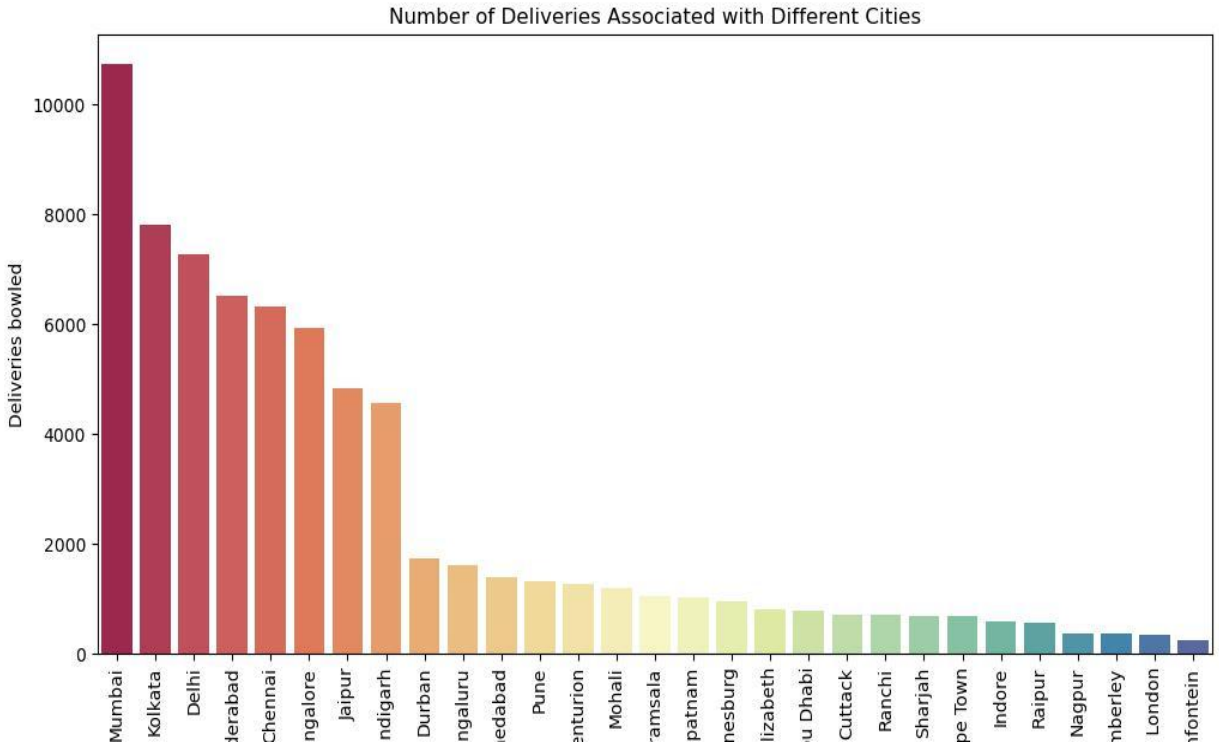


Fig:5

Diagram showcasing Number of Deliveries Associated with Different Cities

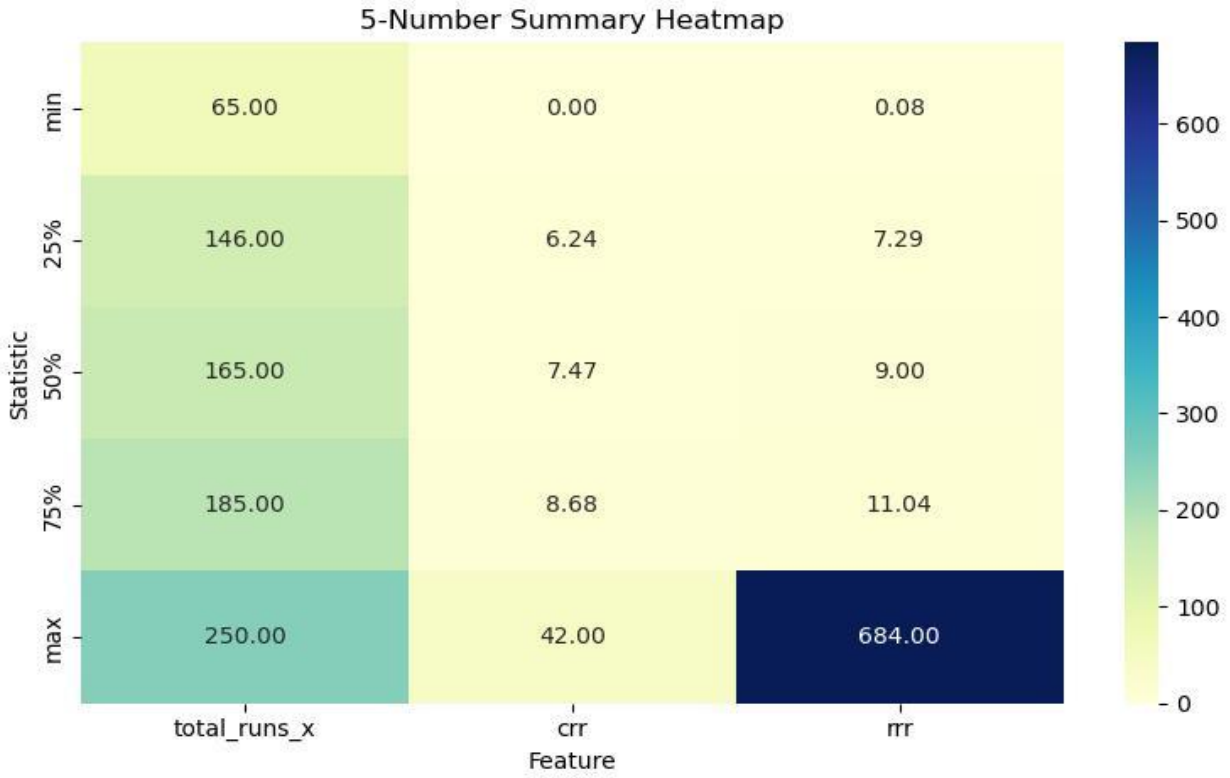


Fig 6: *Heatmap of current run rate, net run rate and total runs*

In the context of our ‘IPL win predictor’ project, the heatmap can be used to visualize the relationships between different features in your dataset. For example, you might see that certain features are highly correlated with each other, while others are not. This information can be used to select features for our model, as well as to understand how different features interact with each other.

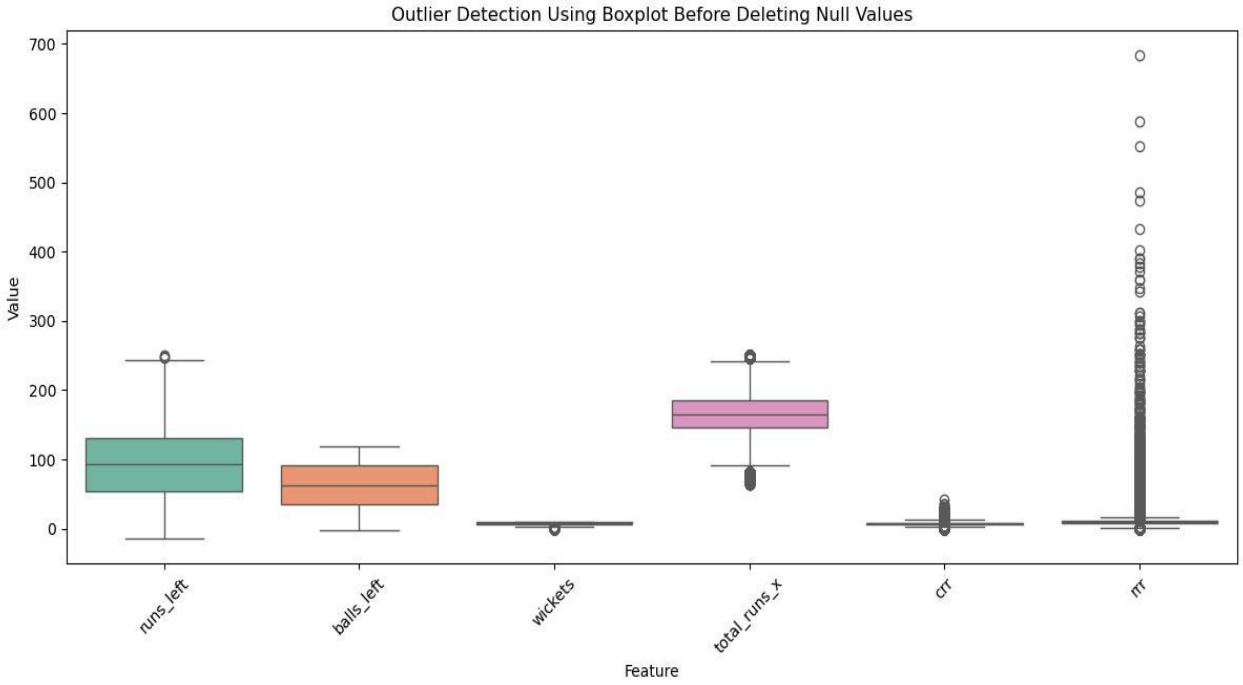


Fig 7:
Boxplot showcasing the outliers

Description

Figure 1: Total Matches Played by Each Team (via Deliveries)

- This bar chart visualizes the total number of deliveries faced by each team, serving as a proxy for the number of matches played. It helps understand team participation and consistency across seasons.

Figure 2: Total Runs Scored by Each Team

- This graph compares cumulative runs scored by teams across all IPL seasons up to 2019. It highlights Delhi Capitals as the highest-scoring team and Sunrisers Hyderabad with the lowest aggregate runs, reflecting overall batting performance**.**

Figure 3: Win Percentage by Each Team (Pie Chart)

- A pie chart representing the proportion of matches won by each team. It provides a quick visual overview of team dominance and competitive balance in the IPL.

Figure 4: Win Percentage – Batting First vs Batting Second

- This pie chart contrasts win ratios of teams batting first versus chasing. It helps assess whether chasing or defending totals offers a higher chance of winning in IPL matches.

Figure 5: Number of Deliveries Associated with Each City

- A bar graph showing the number of deliveries bowled in each city, which indicates how frequently each city has hosted IPL matches. Cities with higher counts are primary venues.

Figure 6: Heatmap of Current Run Rate, Net Run Rate & Total Runs

- A correlation heatmap that shows the relationships among continuous features such as current run rate, required run rate, and total runs. This helps in identifying multicollinearity and selecting key predictors for modeling.

Figure 7: Boxplot Before Data Cleaning

- This boxplot displays feature distributions and highlights outliers before handling null values and correcting negative entries. It visually justifies the need for preprocessing.
- **Outlier:** An outlier is a data point that significantly differs from other observations in a dataset. It lies far outside the overall pattern of the distribution and can be unusually high or low compared to the rest of the data.

Therefore from the EDA we can determine that Teams batting first have a higher win percentage compared to teams batting second(chasing).
This insight is supported by the pie chart in **Figure 4**, which compares win percentages based on batting order. It suggests that **setting a target is often more successful in IPL matches**. This finding can be **used as a feature in our model** (e.g., "batting first" flag), and it supports the intuition that **toss decision (bat or field)** may significantly influence match outcome

Model Building

Step-by-Step Elaboration on Model Training:

1. Data Preparation:

Extracting Features and Labels

- **Explanation:** The features (X) consist of the cleaned text data, and the labels (y) represent the class of results (win or loss). This separation is essential for training and evaluating the machine learning model.

2. Splitting the Data:

- **Explanation:** The dataset is split into training and testing sets using an 80-20 split. The random_state ensures reproducibility. This step is crucial for evaluating the model's performance on unseen data.

3. Division of Input Features :

Explanation: The input features(Batting Team, Bowling Team, City, runs_left, balls_left, wickets, total_runs_x, current run rate, required run rate are divided into two types i.e**.** **Categorical** and **Numeric** features.

4. Column Transformation:

- **Explanation: ColumnTransformer** class from the sklearn.compose module. ColumnTransformer is used to apply different preprocessing steps to different columns of a dataset, which is useful when you have a mix of numerical and categorical data that requires different transformations. **OneHotEncoder** is used to convert categorical variables into binary vectors (dummy variables).

5. Standardizing and Normalizing Input Features:

Explanation: We have used **StandardScaler** that scales numerical features. **StandardScaler** standardizes features by removing the mean and scaling to unit variance.

6. Pipeline Creation:

- **Explanation:** We have used **Pipeline** for the following reasons:
    - **Cleaner code**: You don't have to manually transform and then fit the model.
    - **Less error-prone**: Ensures that the same transformations are always applied.

In our IPL Win Prediction project, the data preparation and model training steps involve extracting and cleaning noisy data, splitting it, and transforming it into numerical data using a ColumnTransformer. By standardizing the numerical features, we ensure they are suitable for machine learning algorithms. These steps form the foundation for building a robust and accurate IPL Win Prediction Model.

Below are the ML Algorithms we have implemented in our project:

1. Logistic Regression:

- **Introduction: Logistic Regression** is a type of supervised learning algorithm used for classification tasks. Despite its name containing **"regression"** it is actually used for classification problems. It is called logistic regression because it employs the logistic function to model the probability of a binary outcome given a set of independent variables.

- **Working Principle:**

  Model Representation: In logistic regression, the output (dependent variable) is a binary value, typically encoded as 0 or 1. Given a set of input features (independent variables), LR models the probability that a given input belongs to a particular class.

  Logistic Function (Sigmoid): The logistic function is a key component of logistic regression. It maps any real-valued number into a range of values between 0 and 1, making it suitable for modelling probabilities.

  Sigmoid Function:

$$g(z) = \frac{1}{1 + e^{-z}}$$

where $z = \theta^T x$, θ is the parameter vector, and x is the input feature vector.

  Decision Boundary: The decision boundary separates the classes in the feature space. In logistic regression with two classes, this boundary is a line. For more than two classes, it becomes a hyperplane.

- **Learning/Training Process:**

  Cost Function (Log Loss): Logistic regression uses a cost function called log loss (or cross-entropy loss) to measure its performance. The goal is to minimise this cost function.

  Log Loss Function:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

where $h_\theta(x)$ is the predicted probability that $y = 1$ given x, and y is the actual label.

Gradient Descent: To minimise the cost function, logistic regression typically uses optimization algorithms like gradient descent. The gradient descent algorithm adjusts the parameters θ iteratively to reach the optimal values that minimise the cost function.

In summary, Logistic Regression is a fundamental and widely used classification algorithm due to its simplicity, interpretability, and effectiveness for linearly separable data. However, it's important to consider its assumptions and limitations when applying it to real world scenarios.

The Way we have implemented in our Project:

Importing Necessary Libraries:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.pipeline import Pipeline
```

This imports the LogisticRegression class from sklearn's linear_model module, as well as various evaluation metrics from the metrics module.

Instantiate the Logistic Regression Model in a pipeline:

```
log_pipe = Pipeline([
   ('preprocessor', preprocessor),
   ('classifier', LogisticRegression())
])
```

This creates an instance of the LogisticRegression class.

Training the Model:

```
log_pipe.fit(X_train, y_train)
```

This trains the logistic regression model on the training data, where X_train represents the input features (presumably preprocessed and transformed into numerical vectors) and y_train represents the corresponding target labels.

Making Predictions:

```
y_pred_log = log_pipe.predict(X_test)
```

Here, predict method computes probabilities of the classes.Then, we convert these probabilities into binary predictions by thresholding at 0.5.

Model Evaluation:

```
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_random))
print("\nClassification Report:\n", classification_report(y_test, y_pred_random))
```

print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_random))

Here, we evaluate the model's performance using various metrics like accuracy and make the classification report as well as confusion matrix based on the results.

Results:

Logistic Regression Accuracy: 0.8042258399722896

Classification Report:

	precision	recall	f1-score	support
0	**0.80**	**0.78**	**0.79**	**6759**
1	**0.81**	**0.82**	**0.82**	**7676**
accuracy			**0.80**	**14435**
macro avg	**0.80**	**0.80**	**0.80**	**14435**
weighted av	**0.80**	**0.80**	**0.80**	**14435**

Confusion Matrix:

[[5285 1474]

[1352 6324]]

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

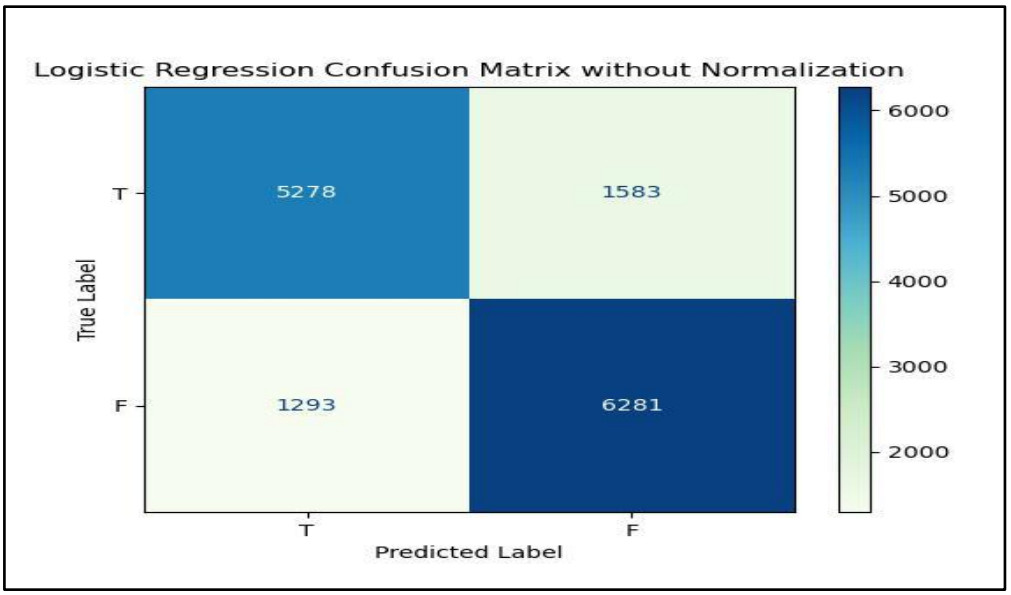


Fig 10: *confusion matrix*

So, we have computed that

- True Positives (**TP**) = 5278
- False Negatives (**FN**) = 1293
- False Positives (**FP**) = 1583
- True Negatives (**TN**) = 6281

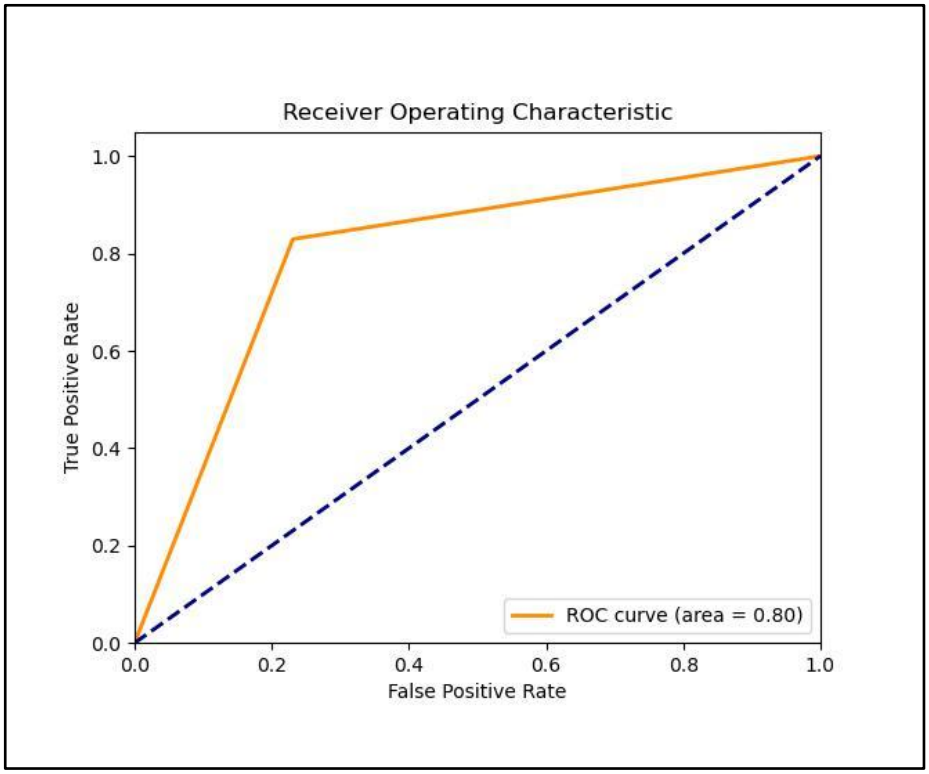


Fig 11: *ROC-AUC of logistic regression*

Now we will be preparing the classification report of our Logistic Regression Classifier model.

Classification Report of Logistic Regression model				
Accuracy			0.804	
	precision	**recall**	**f1-score**	**support**
0	0.80	0.77	0.79	6861
1	0.80	0.83	0.81	7574

Table 22: *classification report*

2. Random Forest Classifier:

- **Introduction:** Random Forest is an ensemble machine learning algorithm that builds multiple decision trees and merges their predictions to achieve more accurate and stable results. It is particularly effective for classification problems with structured tabular data — such as predicting the outcome of an IPL match based on game statistics. In this project, we use the Random Forest Classifier to estimate the winning probability of a team based on features like runs left, wickets, current run rate (CRR), required run rate (RRR), and other match-related metrics.

- **Working Principle:**

The Random Forest Classifier operates using the following principle:

1. It creates **multiple decision trees** during training time using random subsets of the data and features.
2. Each tree gives a prediction (i.e., which team will win).
3. The **majority vote** (in classification tasks) from all the trees becomes the final prediction.

- **Key techniques used:**

- **Bootstrap Aggregation (Bagging)**: Random subsets of the training data are sampled with replacement.
- **Feature Randomness**: A random subset of features is used at each split, helping decorrelate the trees and reduce overfitting.

- **Model Representation:** A Random Forest is not represented by a single formula but by an ensemble of decision trees. Each decision tree is structured as a set of nodes:

- **Root Node**: Represents the first feature to split on.
- **Internal Nodes**: Each makes a decision based on a feature's value.
- **Leaf Nodes**: Represent the final prediction.

For an IPL Win Predictor, each tree may learn patterns such as:

- "If runs_left is low and wickets are high, then team likely wins."
- "If RRR is very high and only 3 wickets left, then team likely loses."

- **Splitting Criteria:** Each decision tree in the forest uses a **splitting criterion** to decide how to split data at each node. For classification tasks, the most common criteria are:

$$\text{Gini Impurity} = 1 - \sum_{i=1}^{C} p_i^2$$

$$\text{Entropy} = -\sum_{i=1}^{C} p_i \log(p_i)$$

The goal is to split the data such that the child nodes are as "pure" as possible — meaning they contain mostly samples of one class (e.g., win or lose).

❖ **Evaluation**

To evaluate the performance of the Random Forest Classifier for predicting match outcomes, we used the following metrics:

1. **Accuracy**: The ratio of correct predictions to total predictions.
2. **Confusion Matrix**: Gives a breakdown of true positives, false positives, true negatives, and false negatives.
3. **Precision, Recall, and F1 Score**: These give more insight, especially in the case of imbalanced data.
4. **ROC Curve & AUC**: Helps visualize model performance across different classification thresholds.

The Way we have implemented in our Project:

Importing Necessary Libraries:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.pipeline import Pipeline
```

This imports the Random Forest Classifier class from sklearn's ensemble module, as well as various evaluation metrics from the metrics module.

Instantiate the Decision Tree Model:

```
random_pipe = Pipeline(steps=[
   ('step1',preprocessor),
   ('step2',RandomForestClassifier())
])
```

This creates an instance of the RandomForestClassifier class within the pipeline named as random_pipe.

Training the Model:

```
random_pipe.fit(X_train,y_train)
```

This trains the random forest model on the training data, where X_train represents the input features (preprocessed and transformed into numerical vectors) and y_train represents the corresponding target labels.

Making Predictions:

```
y_pred_random = random_pipe.predict(X_test)
```

Here, the predict() method is used to generate predictions on the test data.

Model Evaluation:

```
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_random))

print("\nClassification Report:\n", classification_report(y_test, y_pred_random))

print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_random))
```

Here, we evaluate the model's performance using various metrics like accuracy, and generate the classification report and confusion matrix.

Results:

Random Forest Accuracy: 0.9989608590232075

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	6861
1	1.00	1.00	1.00	7574
accuracy			1.00	14435
macro avg	1.00	1.00	1.00	14435
weighted avg	1.00	1.00	1.00	14435

Confusion Matrix:

```
[[6853    8]
[   7 7567]]
```

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

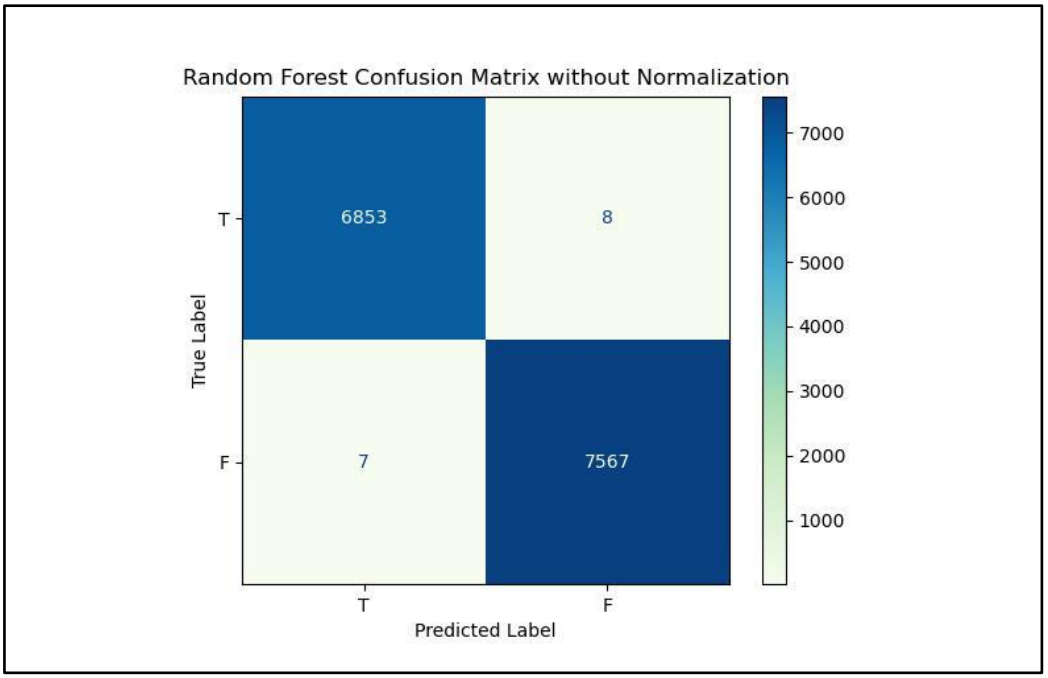
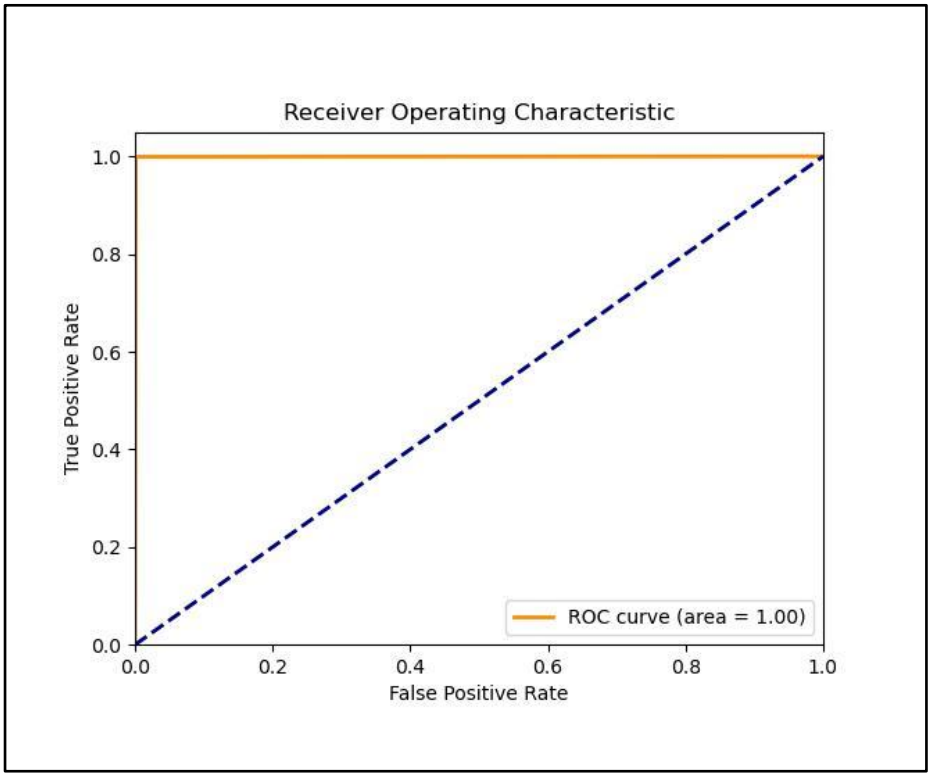


Fig 11: *confusion matrix*

Building so, we have computed that

- True Positives (**TP**) = 6853
- False Positives (**FP**) = 8
- False Negatives(**FN**) = 7



- True Negatives (**TN**) = 7567

Now we will be preparing the classification report of our Random Forest Classifier model.

Classification Report of Random Forest model				
Accuracy			0.99	
	precision	**recall**	**f1-score**	**support**
0	1.00	1.00	1.00	6861
1	1.00	1.00	1.00	7574

Table 13: *classification report*

3. XGBoost model:

- **Introduction: XGBoost** is a powerful and scalable machine learning algorithm that is an implementation of gradient boosted decision trees designed for speed and performance. It is widely used for both classification and regression tasks.

- **Key Features:**

  - **Gradient Boosting**: XGBoost builds an ensemble of trees sequentially, where each new tree corrects errors made by the previous ones. It optimises a differentiable loss function by adding new models.

  - **Regularisation**: XGBoost includes regularisation terms (L1 and L2) to control overfitting, which helps improve generalisation and model performance.

  - **Handling Missing Data**: XGBoost can handle missing data within the dataset by learning the best direction to split.

  - **Parallelization**: It supports parallel processing, which speeds up training on large datasets.

  - **Tree Pruning**: XGBoost uses a technique called "max_depth" to limit the depth of the trees, and it performs tree pruning to remove branches that do not contribute significantly to the final prediction.

- **Working Mechanism:**

  - Boosting: Boosting is an ensemble technique that combines the predictions of several base estimators to improve robustness over a single estimator.

  - Additive Training: Models are trained sequentially, and each new model focuses on reducing the residual errors of the previous models.

  - Objective Function: XGBoost uses a combination of the loss function and a regularisation term to ensure the model does not overfit.

The Way we have implemented in our Project:

Importing Necessary Libraries:

```
from sklearn.pipeline import Pipeline
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

This imports the XGBClassifier class from xgboost module, as well as various evaluation metrics from the metrics module.

Instantiate the XGBoost Model:

```
xgb_pipe = Pipeline(steps=[

   ('step1', preprocessor),
```

```
    ('step2', XGBClassifier(use_label_encoder=False,
eval_metric='logloss'))
])
```

This creates an instance of the XGBClassifier class, suitable for classification, particularly good for structured/tabular data. Setting the label_encoder to **False** tells XGBoost not to use its deprecated internal label encoder. Eval_metric specifies the **evaluation metric** used during training and 'logloss' (logarithmic loss or cross-entropy loss) is appropriate for **classification** problems.

Training the Model:

```
xgb_pipe.fit(X_train, y_train)
```

This trains the XGBoost model on the training data, where X_train represents the input features (preprocessed and transformed into numerical vectors) and y_train represents the corresponding target labels.

Making Predictions:

```
y_pred_xgb = xgb_pipe.predict(X_test)
```

Here, the predict() method is used to generate predictions on the test data.

Model Evaluation:

```
print("XGBoost Accuracy:", accuracy_score(y_test, y_pred_xgb))
print("\nClassification Report:\n", classification_report(y_test, y_pred_xgb))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_xgb))
```

Here, we evaluate the model's performance using various metrics like accuracy and generate classification report and confusion matrix.

Results:

XGBoost Accuracy: 0.997783165916176

Classification Report:

	precision	**recall**	**f1-score**	**support**
0	**1.00**	**1.00**	**1.00**	**6861**
1	**1.00**	**1.00**	**1.00**	**7574**
accuracy			**1.00**	**14435**
macro avg	**1.00**	**1.00**	**1.00**	**14435**

weighted avg 1.00 1.00 1.00 14435

Confusion Matrix:

[[6839 22]

[10 7564]]

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

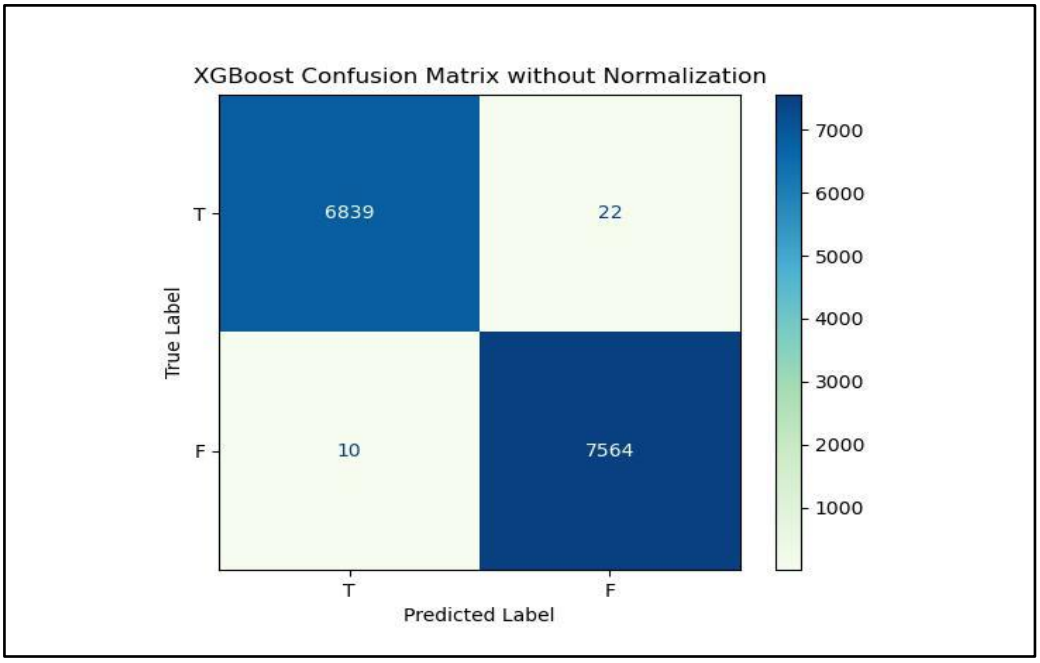
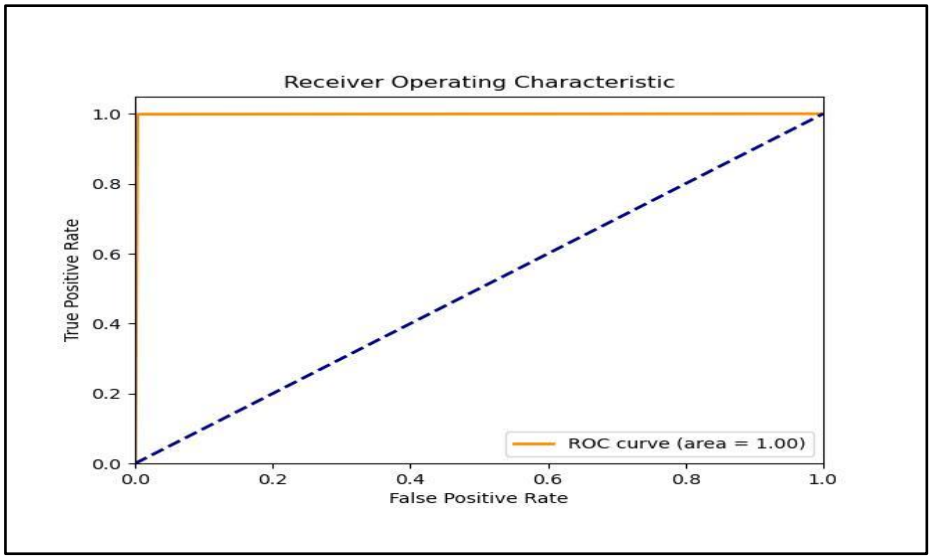


Fig 12: *confusion matri*

So, we have computed that

- True Positives (**TP**) = 6839
- False Positives (**FP**) = 22
- False Negatives (**FN**) = 10
- True Negatives (**TN**) = 7564



Now we will be preparing the classification report of our XGBoost model.

Classification Report of XGBoost model				
Accuracy			0.997	
	precision	**recall**	**f1-score**	**support**
0	1.00	1.00	1.00	6861
1	1.00	1.00	1.00	7574

Table 10: *classification report*

The metrics used in classification report are:

1. Accuracy- the number of correctly classified data instances over the total number of data instances.

Accuracy= (TP+TN)/ (TN+TP+FN+FP)

2. Precision- measures how many of the samples predicted as positive are actually positive.

Precision= TP/ (TP + FP)

3. Recall- measures how many of the positive samples are captured by the positive predictions.

Recall= TP/ (TP + FN)

4. F1-score or f-measure- which is equal to the harmonic mean of precision and recall. *F1-score= 2 * (precision * recall)/ (precision + recall)*

5. Support- the number of instances in a dataset that belong to a particular class. It is also known as the frequency of a class.

4. Support Vector Machine (SVM) Classifier:

- **Introduction: Support Vector Machine or SVM** is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

  The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n- dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

  SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence the algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

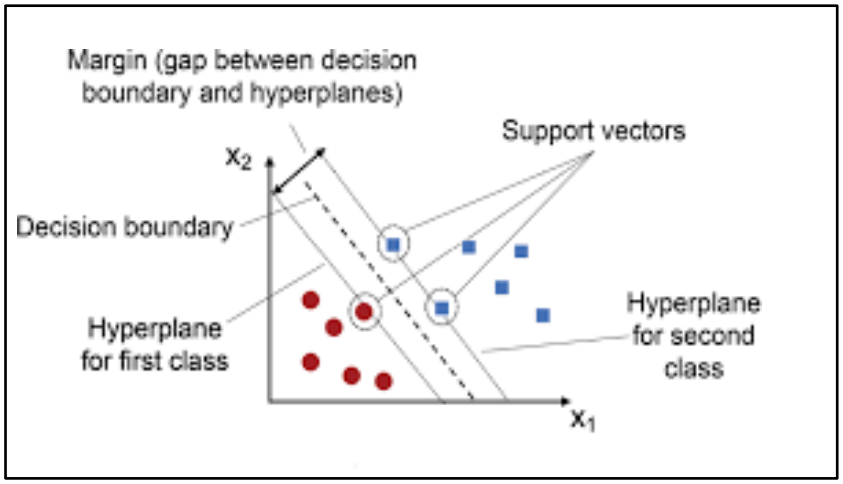


Fig 13: *SVM Diagra*

- **Terminologies:**

  Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

  Support Vectors: The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. These vectors support the hyperplane, hence called a Support vector.

- **Types of SVM:**

  Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

  Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

❖ **Working Principle:**

Model Representation: SVM represents the data in a high-dimensional space and finds the hyperplane that maximises the margin between the classes. The data points closest to the hyperplane are called support vectors.

Hyperplane: A hyperplane in an n-dimensional space is a flat affine subspace of dimension n−1 that separates the data into different classes.

Margin: The margin is the distance between the hyperplane and the nearest data points from each class. SVM aims to maximise this margin to ensure the best separation between classes.

Kernels: SVM can efficiently handle non-linearly separable data by using kernel functions to transform the data into a higher-dimensional space. Common kernel functions include:

- Linear Kernel: $\langle x, x' \rangle$
- Polynomial Kernel: $(\gamma \langle x, x' \rangle + r)^d$
- Radial Basis Function (RBF) Kernel: $\exp(-\gamma \|x - x'\|^2)$

❖ **Evaluation:**

Metrics: Common evaluation metrics for SVM include accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC).

Cross-Validation: SVM models are often evaluated using techniques like k-fold cross-validation to assess their generalisation performance on unseen data.

In summary, SVM is a powerful and flexible classification algorithm that can handle both linear and non- linear data. Its ability to create complex decision boundaries makes it suitable for a wide range of applications, although it may require careful tuning and significant computational resources for large datasets.

The Way we have implemented in our Project:

Importing Necessary Libraries:

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

This imports the SVC class from sklearn's svm module, as well as various evaluation metrics from the metrics module.

Instantiate the SVM Model:

```
svc_pipe = Pipeline([
   ('preprocessor', preprocessor),
   ('classifier', SVC(kernel='rbf', probability=True, random_state=42))
])
```

This creates an instance of the SVC class with a ‘rbf’ kernel, probability = True means we need the the results in terms of probability , and a random state for reproducibility.

Training the Model:

```
svc_pipe.fit(X_train, y_train)
```

Making Predictions:

```
_pred_svc = svc_pipe.predict(X_test)
```

Here, the predict() method is used to generate predictions on the test data.

Model Evaluation:

```
print("Accuracy:", accuracy_score(y_test, y_pred_svc))
print("SVC Classifier Report:\n")
print(classification_report(y_test, y_pred_svc))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svc))
```

Here, we evaluate the model's performance using various metrics like accuracy and generate classification report as well as the confusion matrix for the model.

Results:

Accuracy: 0.9590578455143748

SVC Classifier Report:

	precision	recall	f1-score	support
0	**0.96**	**0.95**	**0.96**	**6861**
1	**0.96**	**0.97**	**0.96**	**7574**
accuracy			**0.96**	**14435**
macro avg	**0.96**	**0.96**	**0.96**	**14435**
weighted avg	**0.96**	**0.96**	**0.96**	**14435**

Confusion Matrix:

```
[[6518  343]
 [ 248 7326]]
```

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

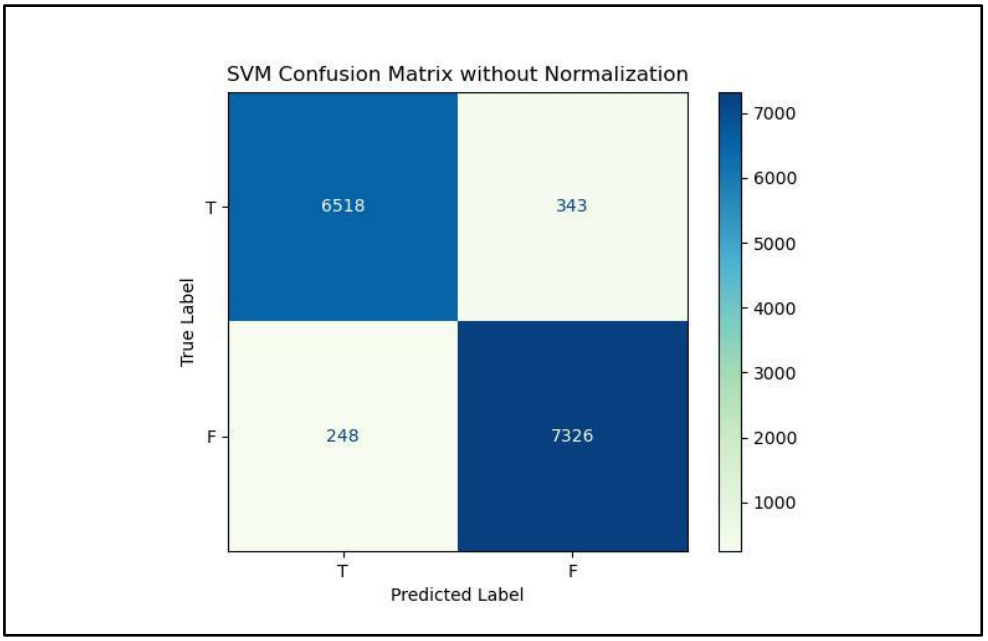
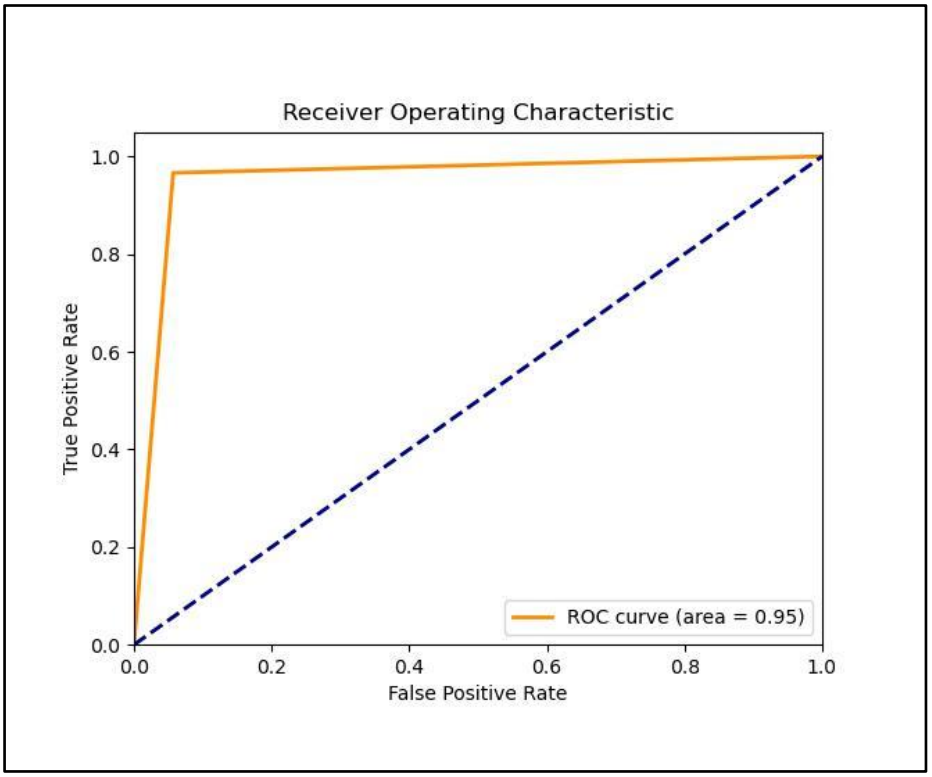


Fig 14: *confusion matrix*

So, we have computed that

- True Positives (**TP**) = 6518
- False Positives (**FP**) = 343
- False Negatives (**FN**) = 248
- True Negatives (**TN**) = 7326



CLASIFICATION REPORT OF SVM

Classification Report of SVM model				
Accuracy			0.959	
	precision	**recall**	**f1-score**	**support**
0	0.96	0.95	0.96	6861
1	0.96	0.97	0.96	7574

Table 16: *classification report*

5. AdaBoost(Adaptive Boosting) Classifier:

❖ **Introduction: AdaBoost (Adaptive Boosting)** is an ensemble learning algorithm that combines multiple **weak learners** (typically decision stumps) to create a **strong classifier**. It focuses on improving prediction accuracy by giving more weight to incorrectly classified samples at each iteration. In the context of this project, AdaBoost is used to predict the outcome of IPL matches (win/loss) based on match-related features like current run rate (CRR), required run rate (RRR), balls left, wickets, etc.

❖ **Working Principle:**

AdaBoost works in the following way:

1. **Initialize weights** for all training instances equally.
2. For each iteration:
   - Train a weak learner (e.g., decision stump).
   - Evaluate errors: Misclassified samples are given **higher weights**.
   - Train the next weak learner on the **re-weighted** data.
3. Combine all weak learners into a single strong model using **weighted voting**, where more accurate learners have higher influence.

❖ **Model Representation:**

- The final AdaBoost model is a **weighted sum of weak classifiers**:

$$H(x) = \text{sign}(\Sigma \text{ from } t = 1 \text{ to } T \text{ of } \alpha_t \cdot h_t(x))$$

Where:

- $h_t(x)$ is the prediction of the t-th weak classifier
- α_t is the weight (importance) of that classifier
- T is the total number of boosting rounds

In an IPL match scenario, weak classifiers might individually make simple decisions like:

- "If runs left < 30, predict win"
- "If wickets < 3 and RRR > 10, predict loss"

These simple rules, when combined and weighted properly, lead to strong performance.

Splitting Criteria:

The **base learner** in AdaBoost is often a **decision stump** — a tree with only one split. The splitting criterion for these trees is usually:

- **Gini Impurity** or **Entropy**, similar to full decision trees.

However, AdaBoost itself does **not use a new splitting criterion**. Instead, it **modifies the sample weights**:

- Samples that are misclassified are given higher importance in the next round.
- The base learners use the **weighted training data** to perform splits that better focus on hard examples.

❖ **Evaluation:**

Metrics: To evaluate the performance of the AdaBoost classifier in the IPL Win Predictor model, we use:

1. **Accuracy**: Percentage of correctly predicted match outcomes.
2. **Confusion Matrix**: Shows how many wins and losses were correctly/incorrectly predicted.
3. **Precision, Recall, and F1 Score**: Especially useful if win/loss classes are imbalanced.
4. **ROC-AUC Score**: Measures model's ability to discriminate between winning and losing cases.
5. **Cross-validation**: Ensures the model performs well on unseen data and avoids overfitting.

AdaBoost offers a powerful alternative to traditional classifiers, especially when simpler models like decision trees are prone to underfitting. By focusing on mistakes and iteratively improving weak models, AdaBoost achieved strong predictive performance in the IPL Win Predictor. It successfully captured match dynamics and delivered consistent results in predicting the outcome based on key features.

The Way we have implemented in our Project:

Importing Necessary Libraries:

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

This imports the AdaBoostClassifier class from sklearn's ensemble module, as well as various evaluation metrics from the metrics module.

Instantiate the AdaBoost Model:

```
ada_model = AdaBoostClassifier(
    estimator=DecisionTreeClassifier(max_depth=1),
    n_estimators=50,
    learning_rate=1.0,
    random_state=42
)
```

This creates an instance of the AdaBoostClassifier class with estimator as a Decision Tree with max depth = 1, n_estimators=50 means AdaBoost will build **50 decision stumps**, one at a time, each correcting the mistakes of the previous one, learning rate = 1 is the default and means full weight is given to each learner. and random state = 42 sets the **random seed** to make results **reproducible**.

```
ada_pipe = Pipeline(steps=[
    ('step1', preprocessor),
    ('step2', ada_model)
])
```

This is pipelining step for the AdaBoost model with preprocessing of the data.

Training the Model:

```
ada_pipe.fit(X_train, y_train)
```

This trains the AdaBoost model on the training data, where X_train represents the input features (preprocessed and transformed into numerical vectors) and y_train represents the corresponding target labels.

Making Predictions:

```
y_pred_ada = ada_pipe.predict(X_test)
```

Here, the predict() method is used to generate predictions on the test data.

Model Evaluation:

```
print("Accuracy:", accuracy_score(y_test, y_pred_ada))
print("\nClassification Report:\n", classification_report(y_test, y_pred_ada))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_ada))
```

Here, we evaluate the model's performance using various metrics like accuracy, classification report and confusion matrix.

Results:

Accuracy: 0.7564253550398338

Classification Report:

	precision	**recall**	**f1-score**	**support**
0	**0.73**	**0.78**	**0.75**	**6861**
1	**0.79**	**0.73**	**0.76**	**7574**
accuracy			**0.76**	**14435**
macro avg	**0.76**	**0.76**	**0.76**	**14435**
weighted avg	**0.76**	**0.76**	**0.76**	**14435**

Confusion Matrix:

```
[[5366  1495]
 [2021  5553]]
```

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

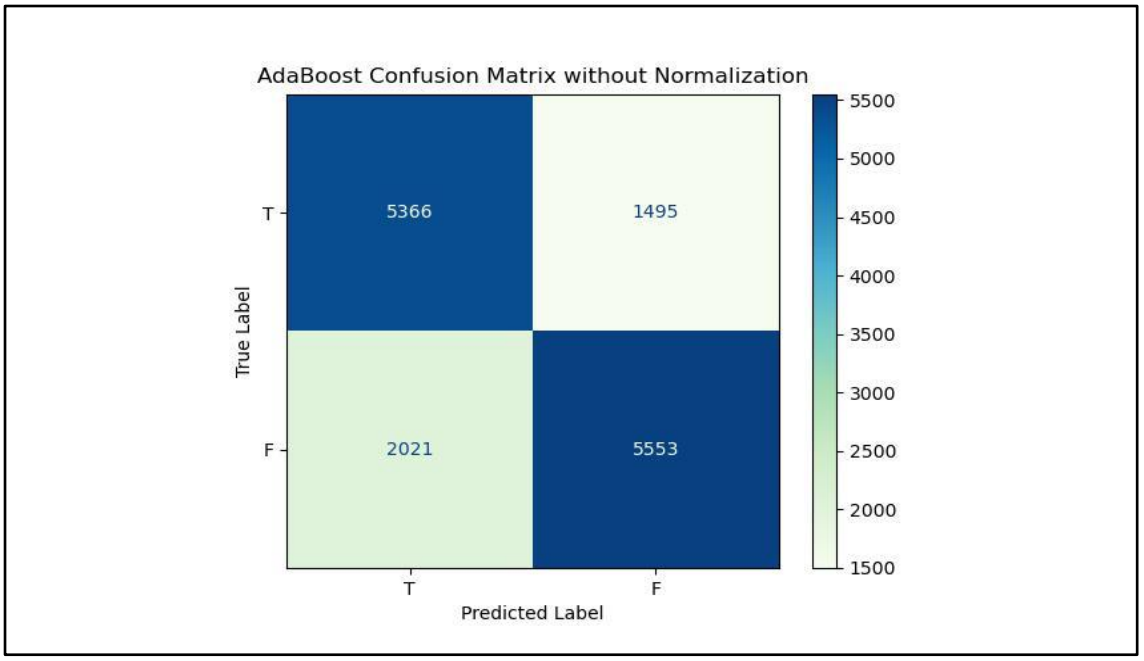
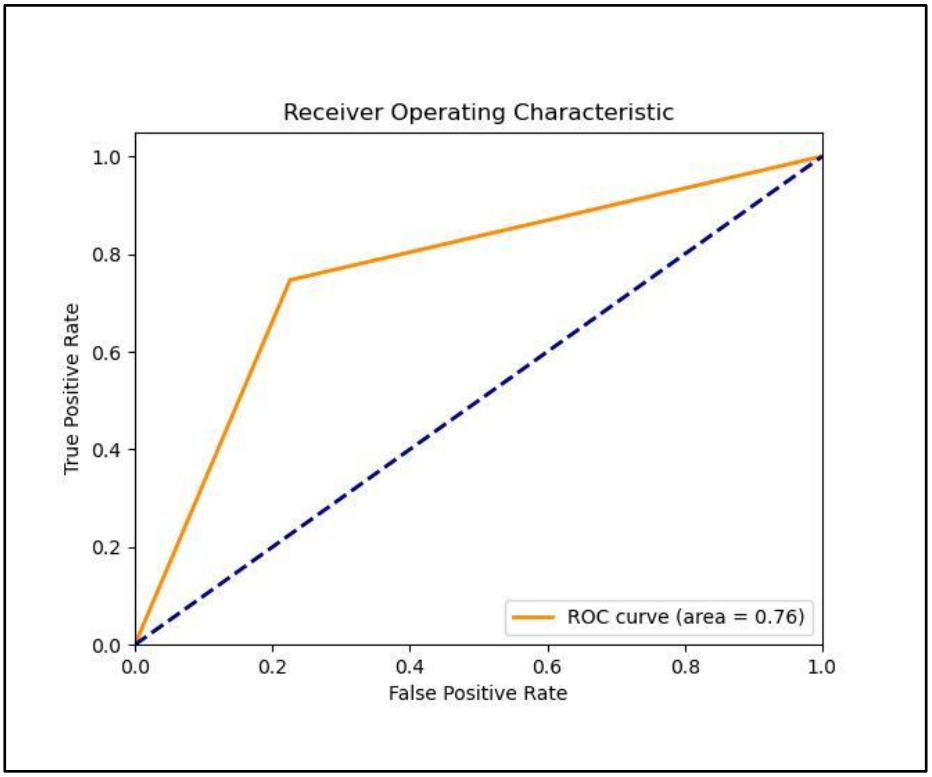


Fig 16: *confusion matrix*

So, we have computed that

- ✔ True Positives (**TP**) = 5366
- ✔ False Positives (**FP**) = 1495
- ✔ False Negatives (**FN**) = 2021
- ✔ True Negatives (**TN**) = 5533



Now we will be preparing the classification report of our AdaBoost model.

Classification Report of AdaBoost model				
Accuracy			0.756	
	precision	**recall**	**f1-score**	**support**
0	0.73	0.77	0.75	6743
1	0.79	0.75	0.77	7692

Table 19: *classification report*

Comparison of the Models trained

We trained 6 models using the 6 algorithms viz.

1. Logistic Regression
2. Random Forest classifier model
3. XGBoost classifier model
4. Support vector classifier model
5. AdaBoost classifier model

The 5 models had different accuracy. The comparisons of the accuracies of the models are given below:

Model	Accuracy (in %)
Logistic Regression	80
Random Forest classifier model	99.83
XGBoost classifier model	99.67
Support vector classifier model	95.71
AdaBoost model	75.96

Table 24: *Comparison of trained model's accuracy*

COMPARISON BETWEEN VARIOUS MODELS

Model	Logistic Regression	Random Forest	AdaBoost	XGBoost	Support Vector Machine
Accuracy	**0.802**	**0.998**	**0.768**	**0.997**	**0.956**
Precision	**0.804**	**0.998**	**0.792**	**0.997**	**0.954**
Recall	**0.826**	**0.999**	**0.759**	**0.998**	**0.967**
F1-Score	**0.815**	**0.998**	**0.775**	**0.998**	**0.960**

The following bar graph shows the accuracy comparison in graphical way:

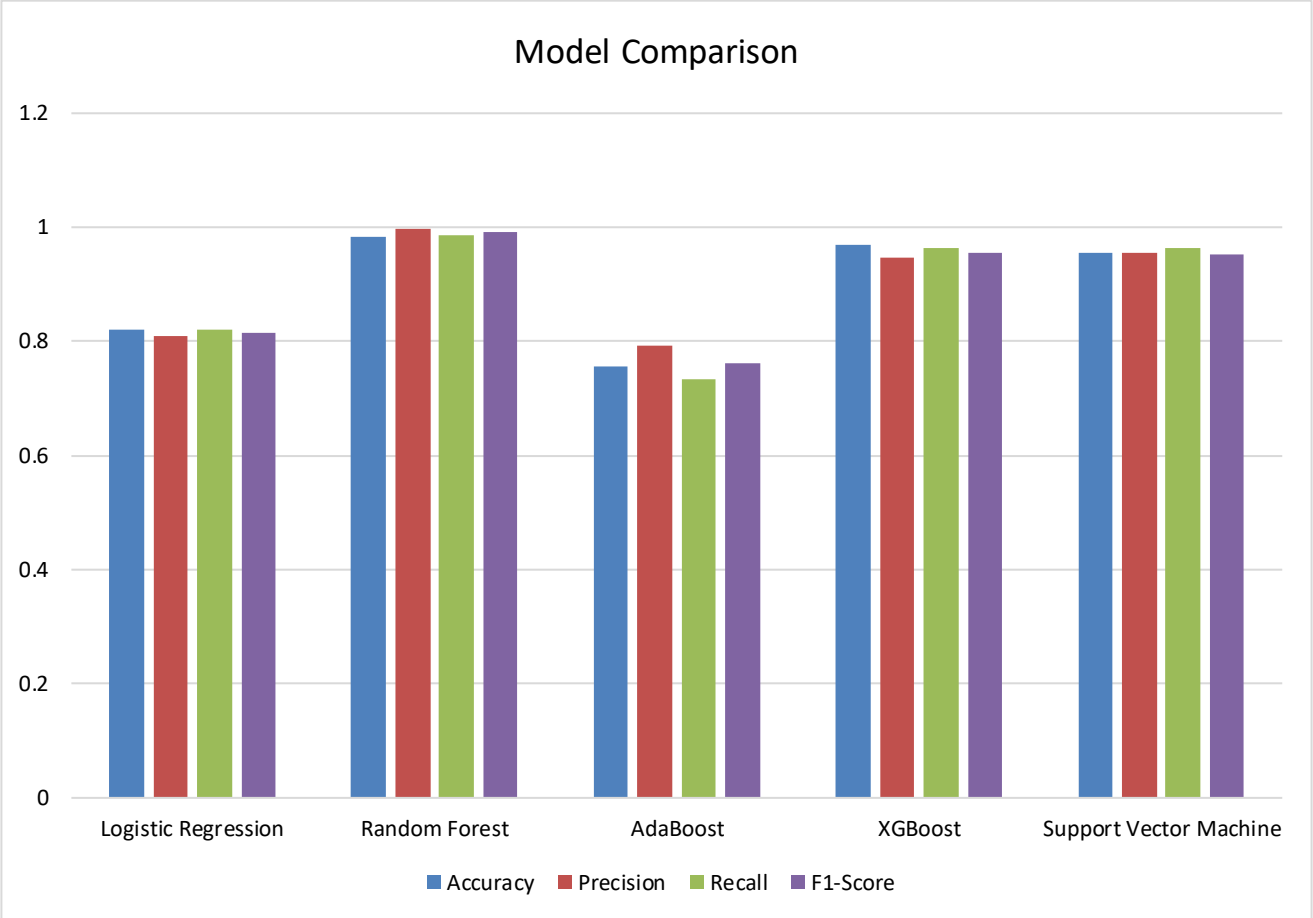


Fig 15: *Comparison of accuracy of 6 trained models*

Given the superior performance metrics of the Random Forest model, proceeding with it for our application is a well- founded decision. The Random Forest model's high accuracy, precision, recall, and F1 score indicate that it is the most effective at correctly classifying the data, minimising both false positives and false negatives.

Test Dataset

After pre-processing of the test dataset had been done, we trained the models with 100% of the train dataset given and used those trained models to predict the outcome for each input in the test dataset. The outcomes obtained are tabulated as below:

Classification Models	Outcomes	
	0	1
Logistic Regression	6586	7849
Random Forest	6734	7701
AdaBoost	7169	7266
XGBoost	6721	7714
SVM	6619	7816

USER INTERFACE DESIGN

The user interface of the IPL Win Predictor project is designed to provide an intuitive and interactive experience for users to predict the outcome of an ongoing cricket match based on various input parameters. Here's a breakdown of the UI components:

1. **Title and Team Selection**:
   - The title " 🏏 IPL Win Predictor" sets the context for the application.
   - Users can select the batting and bowling teams from a dropdown list, ensuring ease of team selection.
2. **City Selection**:
   - A dropdown menu allows users to choose the city where the match is being hosted, influencing match conditions.
3. **Target Score and Current Match Situation**:
   - Users input the target score and current score, with validation to ensure the current score is less than the target score to proceed.
   - They also input the overs completed and the number of wickets fallen, crucial factors in determining match dynamics.
4. **Prediction and Probability Calculation**:
   - Upon clicking the "Predict Probability" button, the application calculates various match parameters:
     - Runs left to score and balls left to play.
     - Wickets remaining and current run rates (CRR) and required run rates (RRR).
   - Using a pre-trained machine learning model (`pipe.pkl`), the application predicts the probability of winning and losing for the selected teams.
5. **Result Display**:
   - Results are displayed in a clear and visually appealing manner:
     - **Win Probability**: Shows the predicted win probability for both the batting and bowling teams in percentage format.
     - **Pie Chart**: A graphical representation using a pie chart illustrates the win and loss probabilities, enhancing the visualization of outcomes.
6. **Feedback and Interaction**:
   - Interactive elements like toast messages alert users to input errors or necessary corrections.
   - The interface is designed to be responsive and user-friendly, ensuring seamless interaction and real-time feedback.

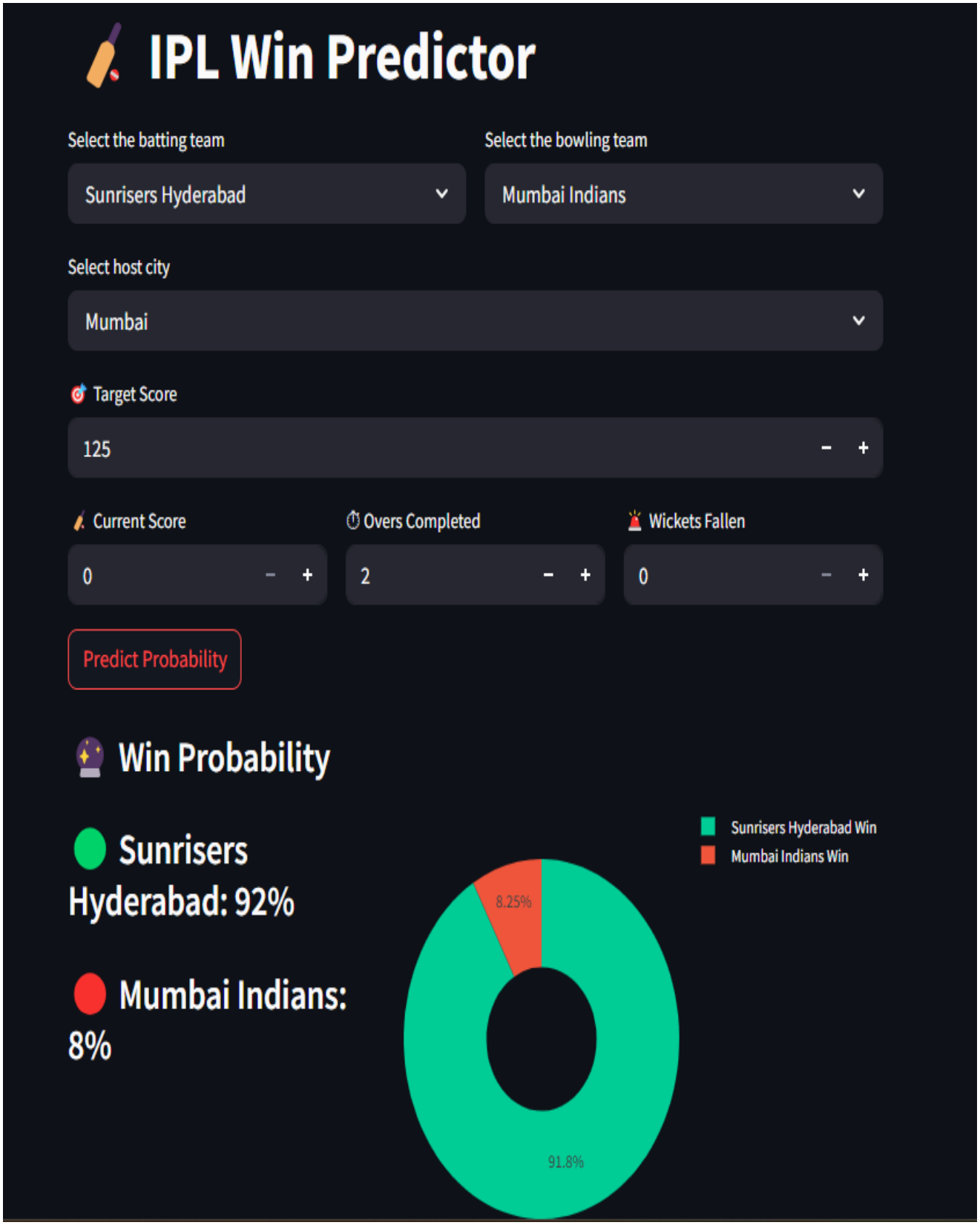
This UI design ensures that cricket enthusiasts and analysts can efficiently predict match outcomes based on real-time match data and historical statistics, making it a valuable tool for cricket fans and professionals alike.

The Way we have implemented in our Project:

```
import streamlit as st
import pickle
import sklearn
import pandas as pd
import plotly.express as px

teams = [
    'Sunrisers Hyderabad',
    'Mumbai Indians',
    'Royal Challengers Bangalore',
    'Kolkata Knight Riders',
    'Punjab Kings',
    'Chennai Super Kings',
    'Rajasthan Royals',
    'Delhi Capitals'
]

cities = ['Hyderabad', 'Bangalore', 'Mumbai', 'Indore', 'Kolkata', 'Delhi',
       'Chandigarh', 'Jaipur', 'Chennai', 'Cape Town', 'Port Elizabeth',
       'Durban', 'Centurion', 'East London', 'Johannesburg', 'Kimberley',
       'Bloemfontein', 'Ahmedabad', 'Cuttack', 'Nagpur', 'Dharamsala',
       'Visakhapatnam', 'Pune', 'Raipur', 'Ranchi', 'Abu Dhabi',
       'Sharjah', 'Mohali', 'Bengaluru']

pipe = pickle.load(open('pipe.pkl','rb'))
st.title('🏏 IPL Win Predictor')

col1, col2 = st.columns(2)

with col1:
    batting_team = st.selectbox('Select the batting team',sorted(teams))

team2 = [team for team in teams if team != batting_team]

with col2:
    bowling_team = st.selectbox('Select the bowling team',sorted(team2))

selected_city = st.selectbox('Select host city',sorted(cities))

target = st.number_input("🎯 Target Score", min_value=1, max_value=350, step=1,
format="%d")

col3,col4,col5 = st.columns(3)

with col3:
```

```
    score = st.number_input("🏏 Current Score", min_value=0, max_value=target,
step=1, format="%d")
    if( score >= target):
        st.toast("Score should be less than target", icon="⚠️")
        st.stop()

with col4:
    overs = st.number_input("⏱️ Overs Completed", min_value=1, max_value=20, step=1,
format="%d")


with col5:
    wickets = st.number_input("🚨 Wickets Fallen", min_value=0, max_value=10,
step=1, format="%d")

if st.button('Predict Probability'):
    runs_left = target - score
    balls_left = 120 - (overs*6)
    wickets = 10 - wickets
    crr = score/overs
    rrr = (runs_left*6)/balls_left

    input_df =
pd.DataFrame({'batting_team':[batting_team],'bowling_team':[bowling_team],'city':[se
lected_city],'runs_left':[runs_left],'balls_left':[balls_left],'wickets':[wickets],'
total_runs_x':[target],'crr':[crr],'rrr':[rrr]})

    result = pipe.predict_proba(input_df)
    loss = result[0][0]
    win = result[0][1]



    col_result, col_chart = st.columns([1, 1.5])

    with col_result:
        st.subheader("🔮 Win Probability")
        st.write(f"### 🟢 {batting_team}: **{round(win * 100)}%**")
        st.write(f"### 🟢 {bowling_team}: **{round(loss * 100)}%**")

    with col_chart:
        # Pie Chart
        pie_df = pd.DataFrame({
            'Outcome': [f'{batting_team} Win', f'{bowling_team} Win'],
            'Probability': [win, loss]
        })

        fig_pie = px.pie(
            pie_df, values='Probability', names='Outcome',
            color_discrete_sequence=['#00cc96', '#EF553B'],
            title='',
```

```
        hole=0.4
    )

    st.plotly_chart(fig_pie, use_container_width=True)
```



Codes

```
# Importing required modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Reading the datsets
match_df = pd.read_csv("Datasets/matches.csv")
delivery_df = pd.read_csv("Datasets/deliveries.csv")
```

```
match_df.shape
```

```
(756, 18)
```

```
delivery_df.shape
```

```
(179078, 21)
```

```
# match dataframe
match_df.head()
```

```
# delivery dataframe
delivery_df.head()
```

```
# Checking the runs c=scored in bth the innnings of each match an storing it in
another dataframe
total_score_df = delivery_df.groupby(['match_id',
'inning']).sum()['total_runs'].reset_index()
```

```
# we just need the first inning total for our calculation
total_score_df = total_score_df[total_score_df['inning'] == 1]
total_score_df
# Joining the dataframes
merge = match_df.merge(total_score_df[['match_id', 'total_runs']], left_on = 'id',
right_on= 'match_id')
merge
merge['team1'].unique()
```

```
array(['Sunrisers Hyderabad', 'Mumbai Indians', 'Gujarat Lions',
       'Rising Pune Supergiant', 'Royal Challengers Bangalore',
       'Kolkata Knight Riders', 'Delhi Daredevils', 'Kings XI Punjab',
       'Chennai Super Kings', 'Rajasthan Royals', 'Deccan Chargers',
       'Kochi Tuskers Kerala', 'Pune Warriors', 'Rising Pune Supergiants',
       'Delhi Capitals'], dtype=object)
```

```
# Name of the teams fro the datasets that are playing till today
teams = [
    'Sunrisers Hyderabad',
    'Mumbai Indians',
    'Royal Challengers Bangalore',
```

```
    'Kolkata Knight Riders',
    'Punjab Kings',
    'Chennai Super Kings',
    'Rajasthan Royals',
    'Delhi Capitals'
]
# Changing the name of the playing teams to their current names
merge['team1'] = merge['team1'].str.replace('Delhi Daredevils', 'Delhi Capitals')
merge['team2'] = merge['team2'].str.replace('Delhi Daredevils', 'Delhi Capitals')

merge['team1'] = merge['team1'].str.replace('Deccan Chargers', 'Sunrisers
Hyderabad')
merge['team2'] = merge['team2'].str.replace('Deccan Chargers', 'Sunrisers
Hyderabad')

merge['team1'] = merge['team1'].str.replace('Kings XI Punjab', 'Punjab Kings')
merge['team2'] = merge['team2'].str.replace('Kings XI Punjab', 'Punjab Kings')
# We only need the teams that are playing till current date, so we are dropping
others
merge = merge[merge['team1'].isin(teams)]
merge = merge[merge['team2'].isin(teams)]
merge.shape
```

```
(641, 20)
```

```
# Matches that are affected by DLS method('0' means not affected, '1' means
affected)
merge['dl_applied'].value_counts()
```

```
0    626
1     15
Name: dl_applied, dtype: int64
```

```
# We are keeping only those matches that are not affected by DLS
merge = merge[merge['dl_applied'] == 0]
merge
# Taking only the required columns
merge = merge[['match_id', 'city', 'winner', 'total_runs']]
merge_df = merge.merge(delivery_df, on = 'match_id')
merge_df
merge_df = merge_df[merge_df['inning'] == 2]
merge_df.shape
```

```
(72413, 24)
```

```
# Checking the null values from datset
print(merge_df.isnull().sum())
```

```
match_id              0
city                832
winner               27
total_runs_x          0
inning                0
batting_team          0
bowling_team          0
over                  0
ball                  0
batsman               0
non_striker           0
bowler                0
is_super_over         0
wide_runs             0
bye_runs              0
legbye_runs           0
noball_runs           0
penalty_runs          0
batsman_runs          0
extra_runs            0
total_runs_y          0
player_dismissed    68860
dismissal_kind      68860
fielder             69855
dtype: int64
```

```
# Filling the missing values in the city column with the mode of their records and
the winner column with 'No result
merge_df = merge_df.assign(
    city = merge_df['city'].fillna(merge_df['city'].mode()[0]),
    winner = merge_df['winner'].fillna('No Result')
)
# Checking after filling the missing values
print(merge_df.isnull().sum())
```

```
match_id              0
city                  0
winner                0
total_runs_x          0
inning                0
batting_team          0
bowling_team          0
over                  0
ball                  0
batsman               0
non_striker           0
bowler                0
is_super_over         0
wide_runs             0
bye_runs              0
legbye_runs           0
noball_runs           0
penalty_runs          0
batsman_runs          0
extra_runs            0
total_runs_y          0
player_dismissed    68860
dismissal_kind      68860
fielder             69855
dtype: int64
```

```
# Check Data Types of each column
print(merge_df.dtypes)
```

```
match_id            int64
city               object
winner             object
total_runs_x        int64
inning              int64
batting_team       object
bowling_team       object
over                int64
ball                int64
batsman            object
non_striker        object
bowler             object
is_super_over       int64
wide_runs           int64
bye_runs            int64
legbye_runs         int64
noball_runs         int64
penalty_runs        int64
batsman_runs        int64
extra_runs          int64
total_runs_y        int64
player_dismissed   object
dismissal_kind     object
fielder            object
dtype: object
```

```
# Check Unique Values in Key Columns
print(merge_df['winner'].unique())
print(merge_df['batting_team'].unique())
print(merge_df['bowling_team'].unique())
```

```
['Sunrisers Hyderabad' 'Royal Challengers Bangalore' 'Mumbai Indians'
 'Kings XI Punjab' 'Kolkata Knight Riders' 'Delhi Daredevils'
 'Chennai Super Kings' 'Rajasthan Royals' 'Deccan Chargers' 'No Result'
 'Delhi Capitals']
['Royal Challengers Bangalore' 'Delhi Daredevils' 'Mumbai Indians'
 'Kings XI Punjab' 'Kolkata Knight Riders' 'Sunrisers Hyderabad'
 'Rajasthan Royals' 'Chennai Super Kings' 'Deccan Chargers'
 'Delhi Capitals']
['Sunrisers Hyderabad' 'Royal Challengers Bangalore'
 'Kolkata Knight Riders' 'Kings XI Punjab' 'Delhi Daredevils'
 'Mumbai Indians' 'Chennai Super Kings' 'Rajasthan Royals'
 'Deccan Chargers' 'Delhi Capitals']
```

```
# Replacing the name of the teams with their current names in the merged dataframe
merge_df['batting_team'] = merge_df['batting_team'].str.replace('Deccan Chargers',
'Sunrisers Hyderabad')
merge_df['bowling_team'] = merge_df['bowling_team'].str.replace('Deccan Chargers',
'Sunrisers Hyderabad')
merge_df['winner'] = merge_df['winner'].str.replace('Deccan Chargers', 'Sunrisers
Hyderabad')

merge_df['batting_team'] = merge_df['batting_team'].str.replace('Delhi Daredevils',
'Delhi Capitals')
```

```
merge_df['bowling_team'] = merge_df['bowling_team'].str.replace('Delhi Daredevils',
'Delhi Capitals')
merge_df['winner'] = merge_df['winner'].str.replace('Delhi Daredevils', 'Delhi
Capitals')
# After replacing checking Unique Values in Key Columns
print(merge_df['winner'].unique())
print(merge_df['batting_team'].unique())
print(merge_df['bowling_team'].unique())
```

```
['Sunrisers Hyderabad' 'Royal Challengers Bangalore' 'Mumbai Indians'
 'Kings XI Punjab' 'Kolkata Knight Riders' 'Delhi Capitals'
 'Chennai Super Kings' 'Rajasthan Royals' 'No Result']
['Royal Challengers Bangalore' 'Delhi Capitals' 'Mumbai Indians'
 'Kings XI Punjab' 'Kolkata Knight Riders' 'Sunrisers Hyderabad'
 'Rajasthan Royals' 'Chennai Super Kings']
['Sunrisers Hyderabad' 'Royal Challengers Bangalore'
 'Kolkata Knight Riders' 'Kings XI Punjab' 'Delhi Capitals'
 'Mumbai Indians' 'Chennai Super Kings' 'Rajasthan Royals']
```

```
# One record was left in the previous changing i.e. 'Kingx XI Punjab' -> 'Punjab
Kings'
merge_df['batting_team'] = merge_df['batting_team'].str.replace('King XI Punjab',
'Punjab Kings')
merge_df['bowling_team'] = merge_df['bowling_team'].str.replace('Kings XI Punjab',
'Punjab Kings')
merge_df['winner'] = merge_df['winner'].str.replace('Kings XI Punjab', 'Punjab
Kings')
print(merge_df['batting_team'].unique())
```

```
['Royal Challengers Bangalore' 'Delhi Capitals' 'Mumbai Indians'
 'Kings XI Punjab' 'Kolkata Knight Riders' 'Sunrisers Hyderabad'
 'Rajasthan Royals' 'Chennai Super Kings']
```

```
# One of them giving errors due to spacing problems, that's why we used strip()
merge_df['batting_team'] = merge_df['batting_team'].str.strip()

merge_df['batting_team'] = merge_df['batting_team'].str.replace('Kings XI Punjab',
'Punjab Kings')
print(merge_df['batting_team'].unique())
```

```
['Royal Challengers Bangalore' 'Delhi Capitals' 'Mumbai Indians'
 'Punjab Kings' 'Kolkata Knight Riders' 'Sunrisers Hyderabad'
 'Rajasthan Royals' 'Chennai Super Kings']
```

```
# We take the help of cumulative sum to calculate the total runs ball by ball
merge_df['current_score'] = merge_df.groupby(['match_id'])['total_runs_y'].cumsum()
# Since the target is 1 run more than the actual runs scored in the first innings
merge_df['runs_left'] = (merge_df['total_runs_x'] - merge_df['current_score']) + 1
merge_df
# Calculating balls left after each ball
merge_df['balls_left'] = 126 - (merge_df['over']*6 + merge_df['ball'])
merge_df
# Filling the null values in the player dismised column with 0 and calculating the
newly addded wickets columm using it
merge_df['player_dismissed'] = merge_df['player_dismissed'].fillna("0")
```

```
merge_df['player_dismissed'] = merge_df['player_dismissed'].apply(lambda x:x if x == "0" else "1")
merge_df['player_dismissed'] = merge_df['player_dismissed'].astype('int')
wickets = merge_df.groupby('match_id')['player_dismissed'].cumsum()
merge_df['wickets'] = 10 - wickets
merge_df.head()
# Current run rate(CRR)
merge_df['crr'] = (merge_df['current_score']*6)/(120 - merge_df['balls_left'])
#Required run rate(RRR)
merge_df['rrr'] = (merge_df['runs_left']*6)/merge_df['balls_left']
merge_df
# Result declare in form of 0 & 1
def result(row):
    return 1 if row['batting_team'] == row['winner'] else 0
# Deleting the unnecessary columns
merge_df  = merge_df.drop(columns = [ 'dismissal_kind', 'fielder'])
# Applying modified result column-wise to the dataframe
merge_df['result'] = merge_df.apply(result,axis=1)
# Again checking if there are some null values or not
print(merge_df.isnull().sum())
```

```
match_id          0
city              0
winner            0
total_runs_x      0
inning            0
batting_team      0
bowling_team      0
over              0
ball              0
batsman           0
non_striker       0
bowler            0
is_super_over     0
wide_runs         0
bye_runs          0
legbye_runs       0
noball_runs       0
penalty_runs      0
batsman_runs      0
extra_runs        0
total_runs_y      0
player_dismissed  0
current_score     0
runs_left         0
balls_left        0
...
crr               0
rrr               5
```

```
# Checking the first 5 rows
print(merge_df.head())
```

```
     match_id       city              winner  total_runs_x  inning  \
125         1  Hyderabad  Sunrisers Hyderabad           207       2
126         1  Hyderabad  Sunrisers Hyderabad           207       2
127         1  Hyderabad  Sunrisers Hyderabad           207       2
128         1  Hyderabad  Sunrisers Hyderabad           207       2
129         1  Hyderabad  Sunrisers Hyderabad           207       2

                  batting_team         bowling_team  over  ball  \
125  Royal Challengers Bangalore  Sunrisers Hyderabad     1     1
126  Royal Challengers Bangalore  Sunrisers Hyderabad     1     2
127  Royal Challengers Bangalore  Sunrisers Hyderabad     1     3
128  Royal Challengers Bangalore  Sunrisers Hyderabad     1     4
129  Royal Challengers Bangalore  Sunrisers Hyderabad     1     5

           batsman  ... extra_runs total_runs_y  player_dismissed  \
125      CH Gayle  ...          0            1                 0
126  Mandeep Singh  ...          0            0                 0
127  Mandeep Singh  ...          0            0                 0
128  Mandeep Singh  ...          0            2                 0
129  Mandeep Singh  ...          0            4                 0

     current_score  runs_left  balls_left  wickets  crr       rrr  result
125              1        207         119       10  6.0  10.436975       0
126              1        207         118       10  3.0  10.525424       0
127              1        207         117       10  2.0  10.615385       0
128              3        205         116       10  4.5  10.603448       0
129              7        201         115       10  8.4  10.486957       0

[5 rows x 29 columns]
```

```
# Saving the cleaned datset into a new dataset
merge_df.to_csv('Datasets/final_dataset.csv', index=False)
# Creating a new dataframe with all the necessary features extracted from the
previous dataframe
final_df =
merge_df[['batting_team','bowling_team','city','runs_left','balls_left','wickets','t
otal_runs_x','crr','rrr','result']]
final_df
# Suffling to check if everything is fine
final_df = final_df.sample(final_df.shape[0])
final_df
final_df['batting_team'].unique()
```

```
array(['Mumbai Indians', 'Punjab Kings', 'Royal Challengers Bangalore',
       'Chennai Super Kings', 'Kolkata Knight Riders', 'Delhi Capitals',
       'Sunrisers Hyderabad', 'Rajasthan Royals'], dtype=object)
```

```
final_df['batting_team'] = final_df['batting_team'].str.replace('King XI Punjab',
'Punjab Kings')
final_df['bowling_team'].unique()
```

```
array(['Chennai Super Kings', 'Royal Challengers Bangalore',
       'Kolkata Knight Riders', 'Delhi Capitals', 'Punjab Kings',
       'Mumbai Indians', 'Rajasthan Royals', 'Sunrisers Hyderabad'],
      dtype=object)
```

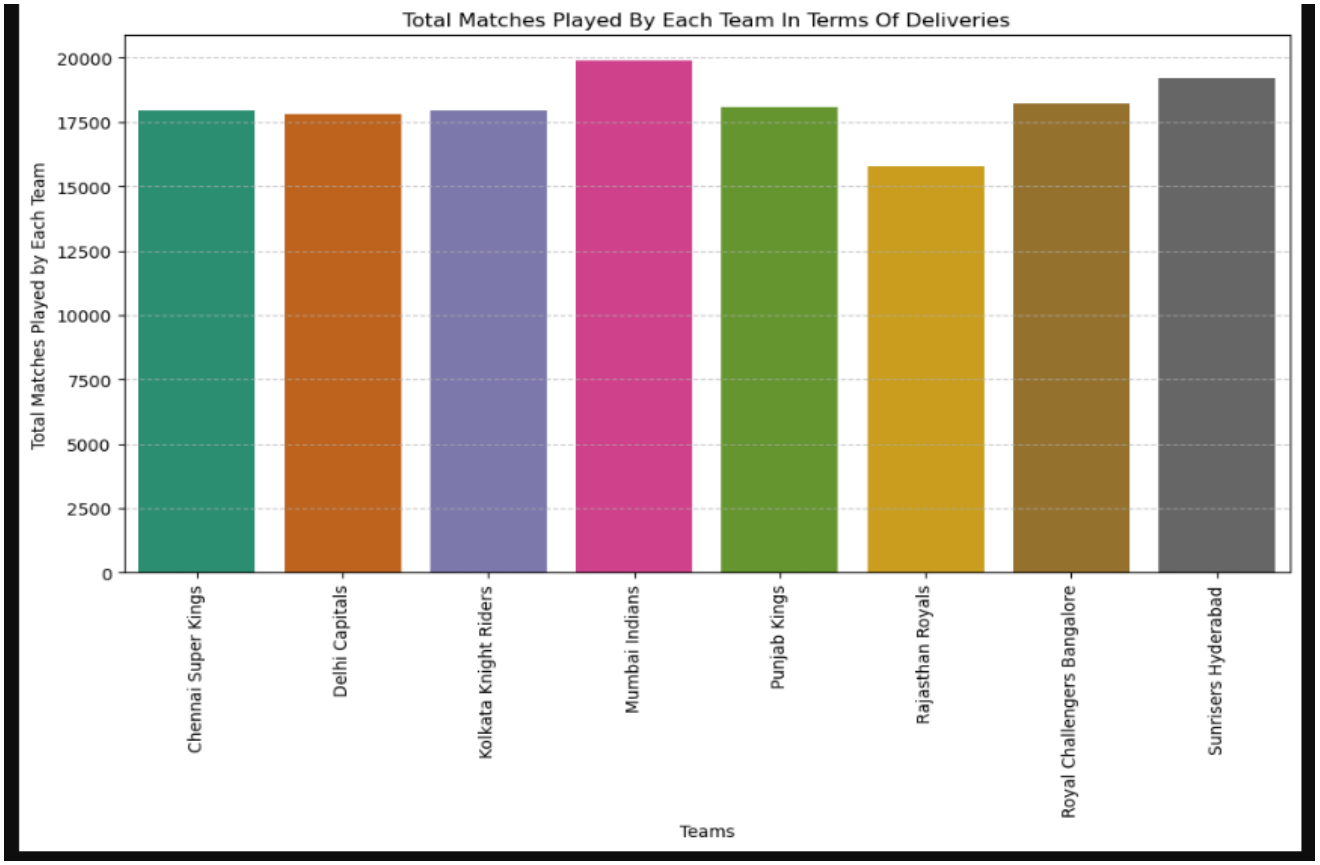
```
final_df.sample()
final_df['batting_team'] = final_df['batting_team'].str.replace('King XI Punjab',
'Punjab Kings')
final_df['batting_team'].unique()
```

```
array(['Mumbai Indians', 'Punjab Kings', 'Royal Challengers Bangalore',
       'Chennai Super Kings', 'Kolkata Knight Riders', 'Delhi Capitals',
       'Sunrisers Hyderabad', 'Rajasthan Royals'], dtype=object)
```

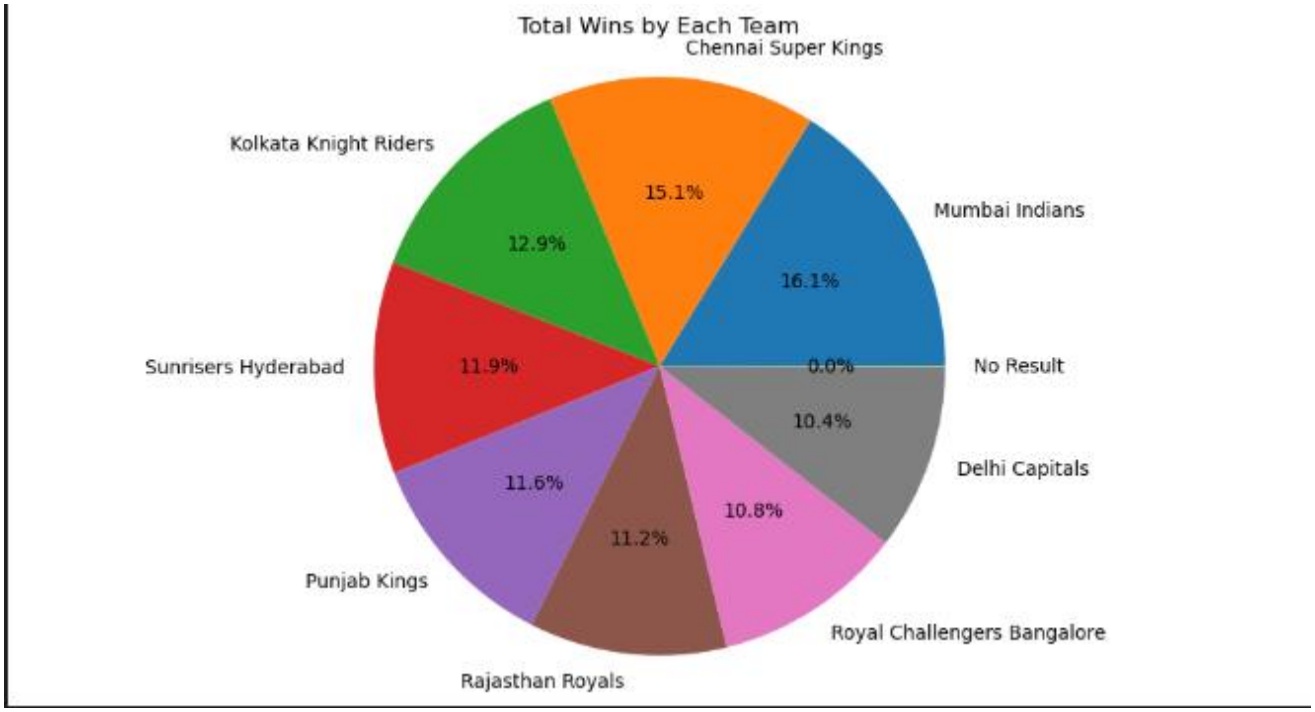
```
final_df['bowling_team'].unique()
```

```
array(['Chennai Super Kings', 'Royal Challengers Bangalore',
       'Kolkata Knight Riders', 'Delhi Capitals', 'Punjab Kings',
       'Mumbai Indians', 'Rajasthan Royals', 'Sunrisers Hyderabad'],
      dtype=object)
```

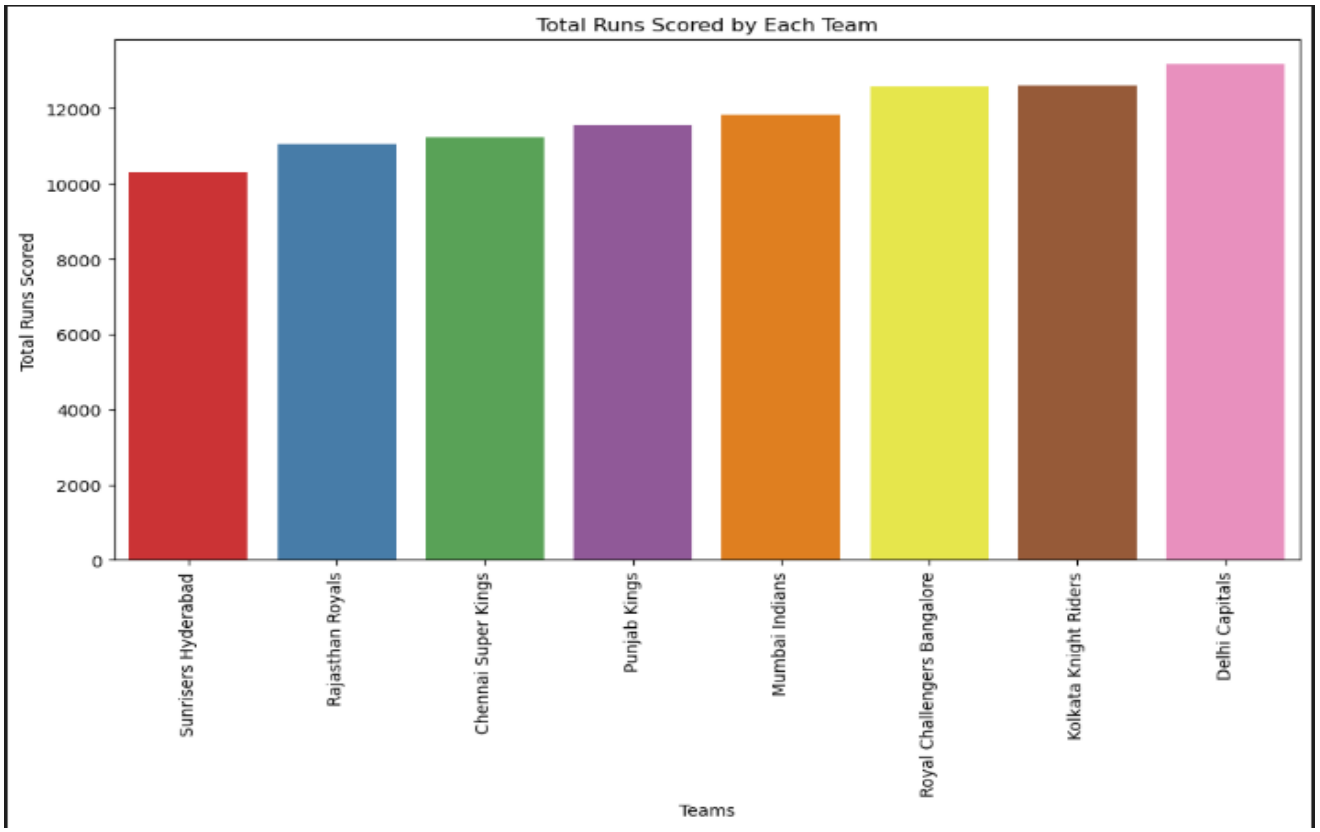
```
# We are using matplotlib and seaborn library for data visualization with different graphs and charts
# Total Matches Played by Each Team

team_matches = final_df['batting_team'].value_counts() +
final_df['bowling_team'].value_counts()
plt.figure(figsize = (12,6))
sns.barplot(x = team_matches.index, y = team_matches.values, palette = 'Dark2')
plt.xticks(rotation = 90)
plt.xlabel("Teams")
plt.ylabel("Total Matches Played")
plt.title("Total Matches Played by Each Team")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



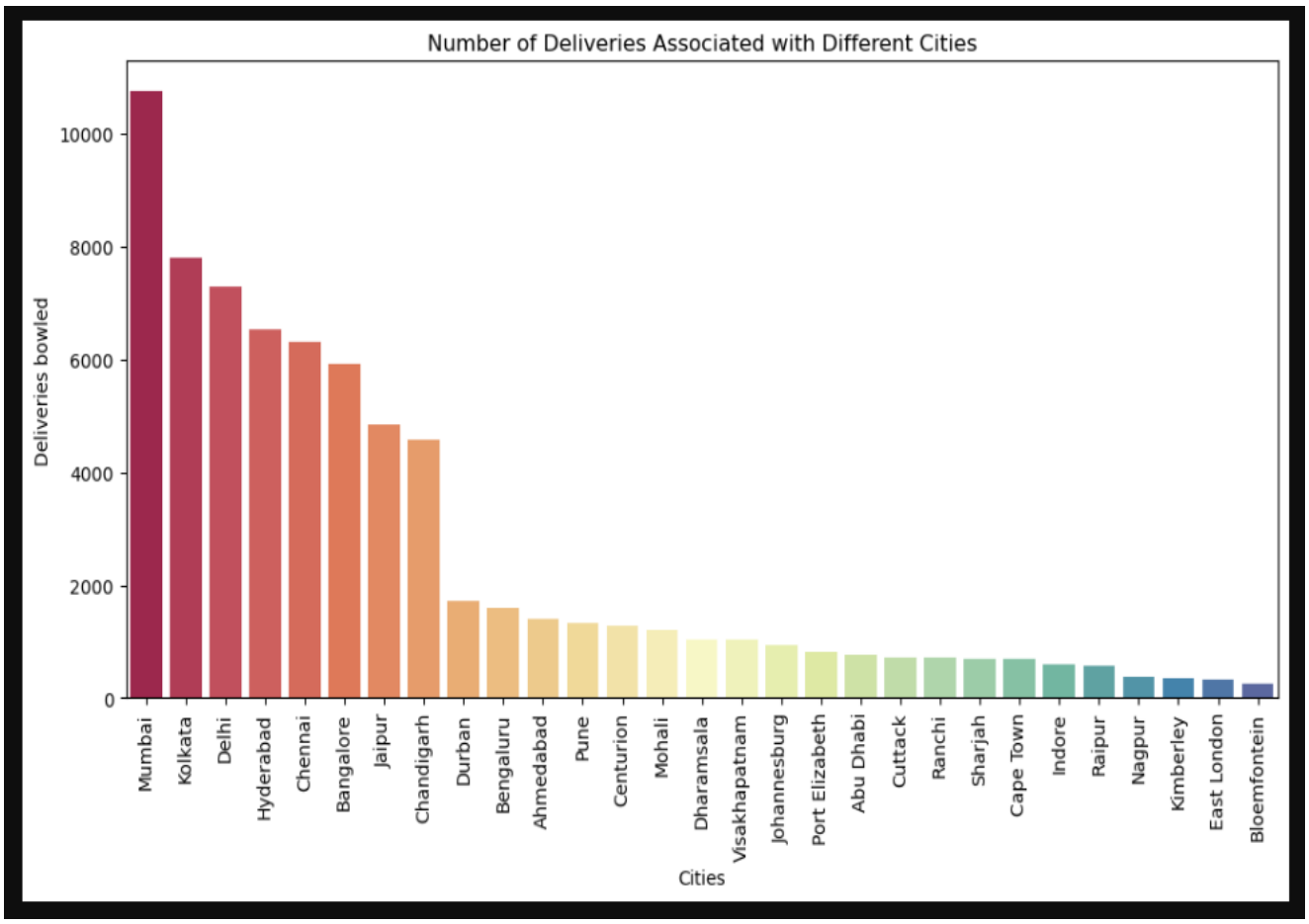

```
# Total number of wins by each team
winner_counts = merge_df['winner'].value_counts()
plt.figure(figsize = (12,6))
plt.pie(winner_counts, labels = winner_counts.index, autopct='%1.1f%%')
plt.title("Total Wins by Each Team")
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```




```
# Total runs scored by each team
team_total_runs =
merge_df.groupby('batting_team')['total_runs_y'].sum().sort_values()
plt.figure(figsize=(12,6))
sns.barplot(x = team_total_runs.index, y = team_total_runs.values, palette = 'Set1')
plt.xticks(rotation=90)
plt.xlabel("Teams")
plt.ylabel("Total Runs Scored")
plt.title("Total Runs Scored by Each Team")
plt.show()
```



```
# Matches played in different cities
city_matches = merge_df['city'].value_counts()
plt.figure(figsize = (12,6))
sns.barplot(x = city_matches.index, y=city_matches.values, palette = 'Spectral')
plt.xticks(rotation = 90)
plt.xlabel("Cities")
plt.ylabel("Matches Hosted")
plt.title("Number of Matches Played in Different Cities")
plt.show()
```




```
final_df.dropna(inplace=True)
# If 0 number of balls is left in a match, then it means the winner is already
decide. So we drop the records where no balls left in a match
final_df = final_df[final_df['balls_left'] != 0]
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Define features and target
X = final_df.drop('result', axis=1) # features
y = final_df['result'] # target

# Train/test split (80% training data and 20% testing data)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Columns
```

```
categorical_features = ['batting_team', 'bowling_team', 'city']
numeric_features = ['runs_left', 'balls_left', 'wickets', 'total_runs_x', 'crr',
'rrr']
# Train dataset
X_train
# Column transformer
from sklearn.compose import ColumnTransformer

preprocessor = ColumnTransformer([
    ('trf', OneHotEncoder(drop='first'), categorical_features),
    ('scaler', StandardScaler(), numeric_features)
])
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
# Create a pipeline with logistic regression
log_pipe = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression())
])

log_pipe.fit(X_train, y_train)

y_pred_log = log_pipe.predict(X_test)
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_log))

print("Classification Report:\n")
print(classification_report(y_test, y_pred_log))

cm = confusion_matrix(y_test, y_pred_log)
print("Confusion Matrix:\n", cm)
```

```
Logistic Regression Accuracy: 0.8029095947350191
Classification Report:

              precision    recall  f1-score   support

           0       0.79      0.78      0.79      6779
           1       0.81      0.82      0.82      7656

    accuracy                           0.80     14435
   macro avg       0.80      0.80      0.80     14435
weighted avg       0.80      0.80      0.80     14435

Confusion Matrix:
 [[5307 1472]
 [1373 6283]]
```

```
random_pipe = Pipeline(steps=[
    ('step1',preprocessor),
    ('step2',RandomForestClassifier())
])

random_pipe.fit(X_train,y_train)
```

```python
y_pred_random = random_pipe.predict(X_test)
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_random))
print("\nClassification Report:\n", classification_report(y_test, y_pred_random))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_random))
```

```
Random Forest Accuracy: 0.9984759265673709

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6779
           1       1.00      1.00      1.00      7656

    accuracy                           1.00     14435
   macro avg       1.00      1.00      1.00     14435
weighted avg       1.00      1.00      1.00     14435


Confusion Matrix:
 [[6770    9]
 [  13 7643]]
```

```python
!pip install xgboost # Installing xgboost module in the system
from sklearn.pipeline import Pipeline
from xgboost import XGBClassifier

xgb_pipe = Pipeline(steps=[
    ('step1', preprocessor),
    ('step2', XGBClassifier(use_label_encoder=False, eval_metric='logloss'))
])
xgb_pipe.fit(X_train, y_train)

y_pred_xgb = xgb_pipe.predict(X_test)
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

print("XGBoost Accuracy:", accuracy_score(y_test, y_pred_xgb))
print("\nClassification Report:\n", classification_report(y_test, y_pred_xgb))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_xgb))
```

```
XGBoost Accuracy: 0.9977138898510565

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6779
           1       1.00      1.00      1.00      7656

    accuracy                           1.00     14435
   macro avg       1.00      1.00      1.00     14435
weighted avg       1.00      1.00      1.00     14435


Confusion Matrix:
 [[6767   12]
 [  21 7635]]
```

```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Create AdaBoost model with DecisionTreeClassifier as the base estimator
ada_model = AdaBoostClassifier(
    estimator=DecisionTreeClassifier(max_depth=1),
    n_estimators=50,
    learning_rate=1.0,
    random_state=42
)
```

```
ada_pipe = Pipeline(steps=[
    ('step1', preprocessor),
    ('step2', ada_model)
])

ada_pipe.fit(X_train, y_train)
y_pred_ada = ada_pipe.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred_ada))
print("\nClassification Report:\n", classification_report(y_test, y_pred_ada))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_ada))
```

```
Accuracy: 0.7559404225839972

Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.78      0.75      6779
           1       0.79      0.73      0.76      7656

    accuracy                           0.76     14435
   macro avg       0.76      0.76      0.76     14435
weighted avg       0.76      0.76      0.76     14435


Confusion Matrix:
 [[5320 1459]
 [2064 5592]]
```

```
rom sklearn.svm import SVC

# Pipeline
svc_pipe = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', SVC(kernel='rbf', probability=True, random_state=42))
])

# Train
svc_pipe.fit(X_train, y_train)

# Predict and evaluate
y_pred_svc = svc_pipe.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred_svc))
print("SVC Classifier Report:\n")
print(classification_report(y_test, y_pred_svc))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svc))
```

```
Accuracy: 0.9567024593003117
SVC Classifier Report:

              precision    recall  f1-score   support

           0       0.96      0.95      0.95      6779
           1       0.95      0.96      0.96      7656

    accuracy                           0.96     14435
   macro avg       0.96      0.96      0.96     14435
weighted avg       0.96      0.96      0.96     14435

Confusion Matrix:
 [[6427  352]
 [ 273 7383]]
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
# Accuracy
print(f"Logistic Regression: {accuracy_score(y_test,y_pred_log)}")
print(f"Random Forest Classifier: {accuracy_score(y_test,y_pred_random)}")
```

```
print(f"AdaBoost Claasifer: {accuracy_score(y_test,y_pred_ada)}")
print(f"XGBoost Classifer: {accuracy_score(y_test,y_pred_xgb)}")
print(f"Support Vector Machine: {accuracy_score(y_test,y_pred_svc)}")
```

```
Logistic Regression: 0.8029095947350191
Random Forest Classifier: 0.9984759265673709
AdaBoost Claasifer: 0.7559404225839972
XGBoost Classifer: 0.9977138898510565
Support Vector Machine: 0.9567024593003117
```

```
# Precision
print(f"Logistic Regression: {precision_score(y_test, y_pred_log)}")
print(f"Random Forest Classifer: {precision_score(y_test, y_pred_random)}")
print(f"AdaBoost Classifier: {precision_score(y_test, y_pred_ada)}")
print(f"XGBoost Classifier: {precision_score(y_test, y_pred_xgb)}")
print(f"Support Vector Machine: {precision_score(y_test, y_pred_svc)}")
```

```
Logistic Regression: 0.8101869761444229
Random Forest Classifer: 0.9988238369053842
AdaBoost Classifier: 0.7930789958871082
XGBoost Classifier: 0.9984307571596704
Support Vector Machine: 0.9544925662572722
```

```
# Recall
print(f"Logistic Regression: {recall_score(y_test, y_pred_log)}")
print(f"Random Forest Classifer: {recall_score(y_test, y_pred_random)}")
print(f"AdaBoost Classifier: {recall_score(y_test, y_pred_ada)}")
print(f"XGBoost Classifier: {recall_score(y_test, y_pred_xgb)}")
print(f"Support Vector Machine: {recall_score(y_test, y_pred_svc)}")
```

```
Logistic Regression: 0.8206635318704284
Random Forest Classifer: 0.9983019853709509
AdaBoost Classifier: 0.7304075235109718
XGBoost Classifier: 0.997257053291536
Support Vector Machine: 0.9643416927899686
```

```
# F1 Score
print(f"Logistic Regression: {f1_score(y_test, y_pred_log)}")
print(f"Random Forest Classifer: {f1_score(y_test, y_pred_random)}")
print(f"AdaBoost Classifier: {f1_score(y_test, y_pred_ada)}")
print(f"XGBoost Classifier: {f1_score(y_test, y_pred_xgb)}")
print(f"Support Vector Machine: {f1_score(y_test, y_pred_svc)}")
```

```
Logistic Regression: 0.8153916034001687
Random Forest Classifer: 0.9985628429579305
AdaBoost Classifier: 0.7604542054803834
XGBoost Classifier: 0.9978435600862576
Support Vector Machine: 0.9593918523812618
```

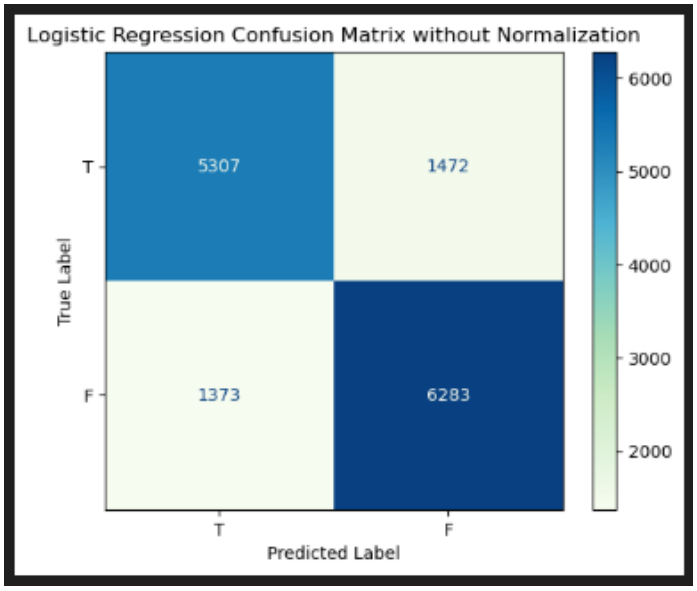
```
# Confusion Matrix
log_reg_cm = confusion_matrix(y_test, y_pred_log)
print(f"Logistic Regression:\n {confusion_matrix(y_test, y_pred_log)}")

random_cm = confusion_matrix(y_test, y_pred_random)
print(f"Random Forest Classifer:\n {confusion_matrix(y_test, y_pred_random)}")
```

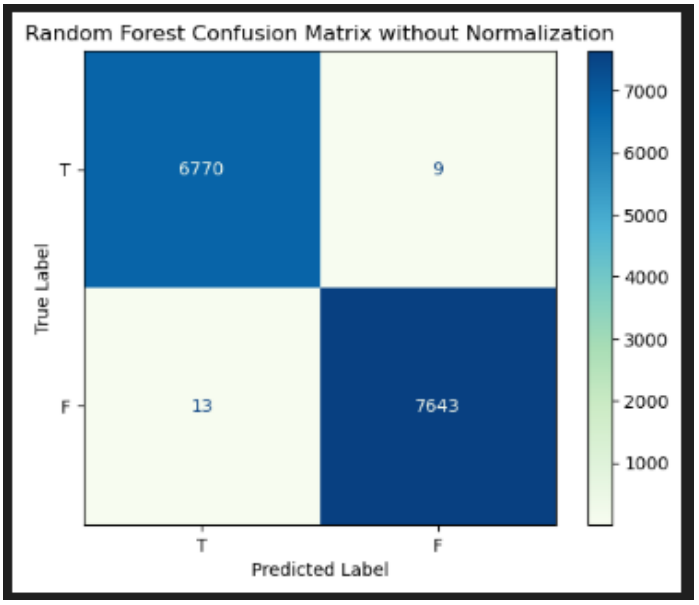
```
ada_cm = confusion_matrix(y_test, y_pred_ada)
print(f"AdaBoost Classifier:\n {confusion_matrix(y_test, y_pred_ada)}")

xgb_cm = confusion_matrix(y_test, y_pred_xgb)
print(f"XGBoost Classifier:\n {confusion_matrix(y_test, y_pred_xgb)}")

svc_cm = confusion_matrix(y_test, y_pred_svc)
print(f"Support Vector Machine:\n {confusion_matrix(y_test, y_pred_svc)}")
```

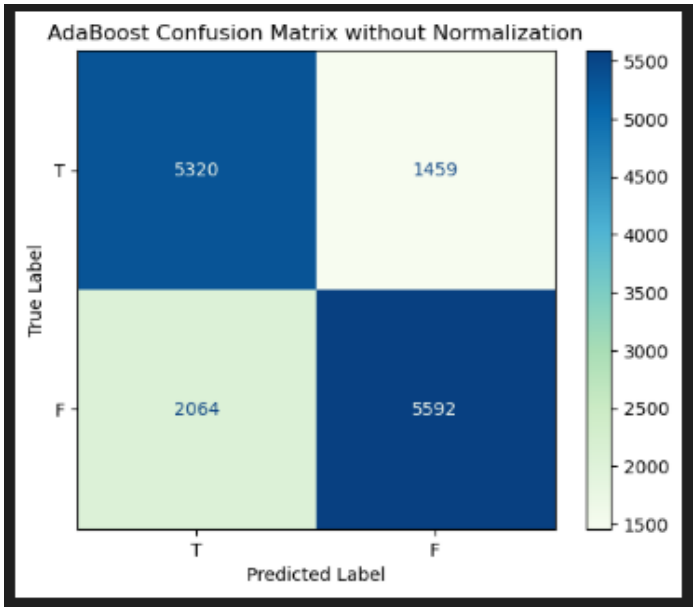
```
Logistic Regression:
 [[5307 1472]
 [1373 6283]]
Random Forest Classifer:
 [[6770    9]
 [  13 7643]]
AdaBoost Classifier:
 [[5320 1459]
 [2064 5592]]
XGBoost Classifier:
 [[6767   12]
 [  21 7635]]
Support Vector Machine:
 [[6427  352]
 [ 273 7383]]
```

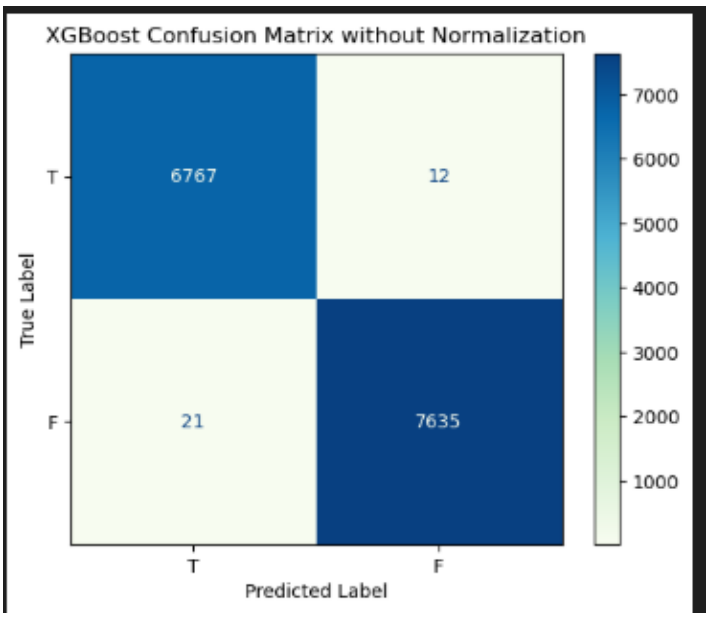
```
# Visualize the confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay

disp1 = ConfusionMatrixDisplay(
    confusion_matrix=log_reg_cm,
    display_labels= ['T', 'F']
)
disp1.plot(cmap='GnBu')
plt.title("Logistic Regression Confusion Matrix without Normalization")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

disp2 = ConfusionMatrixDisplay(
    confusion_matrix=random_cm,
    display_labels= ['T', 'F']
)
disp2.plot(cmap='GnBu')
plt.title("Random Forest Confusion Matrix without Normalization")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

disp3 = ConfusionMatrixDisplay(
    confusion_matrix=ada_cm,
    display_labels= ['T', 'F']
)
disp3.plot(cmap='GnBu')
plt.title("AdaBoost Confusion Matrix without Normalization")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

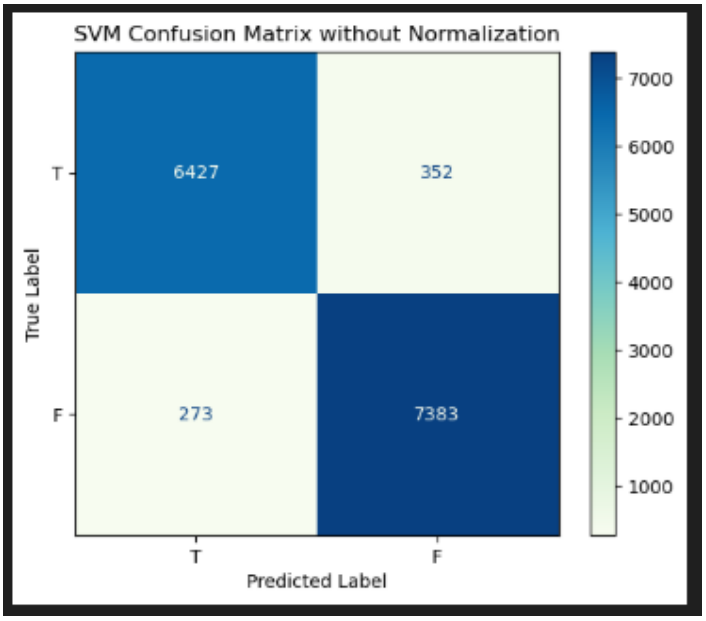
```
disp4 = ConfusionMatrixDisplay(
    confusion_matrix=xgb_cm,
    display_labels= ['T', 'F']
)
disp4.plot(cmap='GnBu')
plt.title("XGBoost Confusion Matrix without Normalization")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

disp5 = ConfusionMatrixDisplay(
    confusion_matrix=svc_cm,
    display_labels= ['T', 'F']
)
disp5.plot(cmap='GnBu')
plt.title("SVM Confusion Matrix without Normalization")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```




Random Forest Confusion Matrix without Normalization
6770
9
13
7643
T
F
True Label
Predicted Label
7000
6000
5000
4000
3000
2000
1000

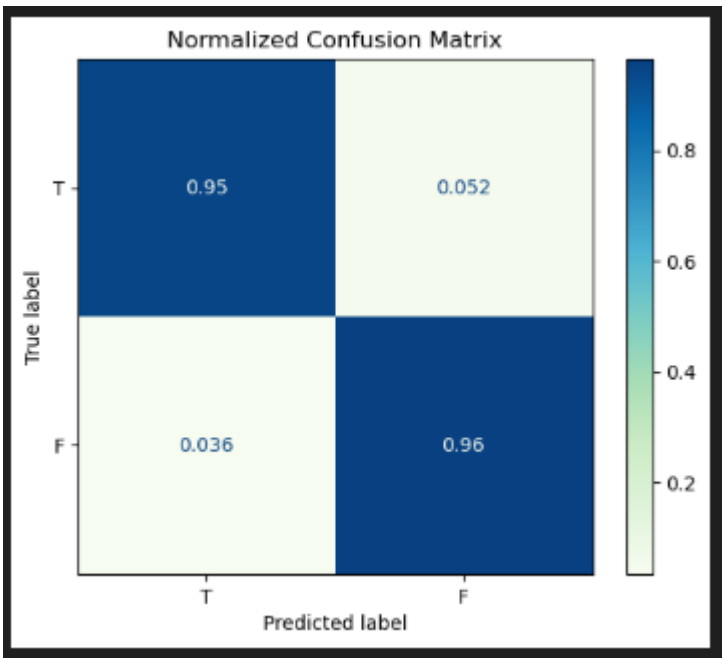

AdaBoost Confusion Matrix without Normalization
5320
1459
2064
5592
T
F
True Label
Predicted Label
5500
5000
4500
4000
3500
3000
2500
2000
1500


XGBoost Confusion Matrix without Normalization
6767
12
21
7635
T
F
True Label
Predicted Label
7000
6000
5000
4000
3000
2000
1000




```
# Normalization of Confusion Matrix (We only show here one normalised confusion
matrix, similarly we can show the normalised confusion matrix for other four models)
cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
print("Normalized Confusion Matrix")
np.set_printoptions(precision=4, suppress=True)
print(cm_normalized)

disp = ConfusionMatrixDisplay.from_predictions(y_test, y_pred_svc, normalize='true',
display_labels=["T", "F"], cmap='GnBu')
disp.ax_.set_title("Normalized Confusion Matrix")
plt.show()
```

```
Normalized Confusion Matrix
[[0.7829 0.2171]
 [0.1793 0.8207]]
```

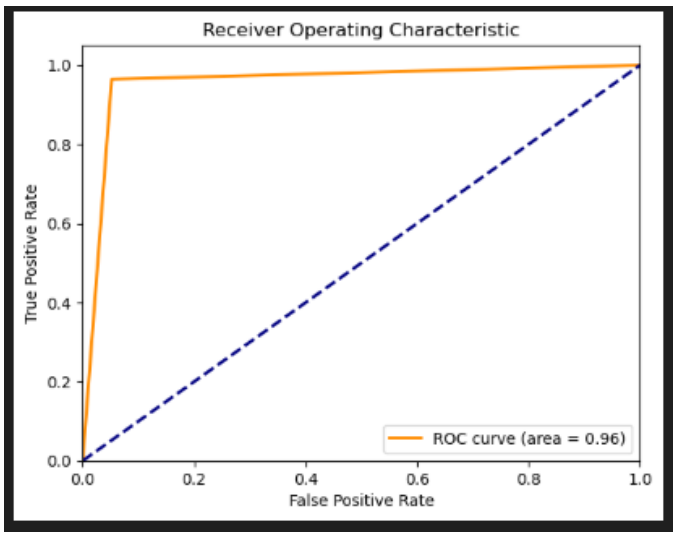


AUC and ROC for Different Models

```python
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred_svc)

# Compute Area Under the Curve (AUC) using the trapezoidal rule (we only show for
SVM model, we can also show for the other four models in the similar way)
roc_auc = auc(fpr, tpr)
print(f"Y: {y_test}")
print(f"Y_HAT: {y_pred_svc}")
print(f"FPR: {fpr}")
print(f"TPR: {tpr}")
print (F"Optimal threshold index: {np.argmax(tpr - fpr)}")
print (F"Optimal threshold value: {thresholds[np.argmax(tpr - fpr)]}")
print(f"AUC: {roc_auc}")
```

```
Y: 115156   1
31286    1
48936    1
112639   0
94897    1
        ..
58537    1
143193   1
18776    0
4329     0
138402   0
Name: result, Length: 14435, dtype: int64
Y_HAT: [1 1 1 ... 0 0 0]
FPR: [0.     0.0519 1.   ]
TPR: [0.     0.9643 1.   ]
Optimal threshold index: 1
Optimal threshold value: 1.0
AUC: 0.9562083150481779
```

```python
# ROC Curve
plt.figure()
lw = 2
plt.plot(fpr, tpr, color = 'darkorange',
         lw = lw, label = 'ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color = 'navy', lw = lw, linestyle = '--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc = "lower right")
plt.show()
```



Test results on Different Models

```
print(f"Logistic regression testing: {log_pipe.predict_proba(X_test)[10]}")
print(f"Random Forest testing: {random_pipe.predict_proba(X_test)[10]}")
print(f"AdaBoost testing: {ada_pipe.predict_proba(X_test)[10]}")
print(f"XGBoost testing: {xgb_pipe.predict_proba(X_test)[10]}")
print(f"SVM testing: {svc_pipe.predict_proba(X_test)[10]}")
```

```
Logistic regression testing: [0.5506 0.4494]
Random Forest testing: [0.06 0.94]
AdaBoost testing: [0.5082 0.4918]
XGBoost testing: [0.0251 0.9749]
SVM testing: [0.0264 0.9736]
```

```
import pickle
pickle.dump(log_pipe,open('log_pipe.pkl','wb'))
pickle.dump(random_pipe,open('random_pipe.pkl','wb'))
pickle.dump(ada_pipe,open('ada_pipe.pkl','wb'))
pickle.dump(xgb_pipe,open('xgb_pipe.pkl','wb'))
pickle.dump(svc_pipe,open('pipe.pkl','wb'))
```

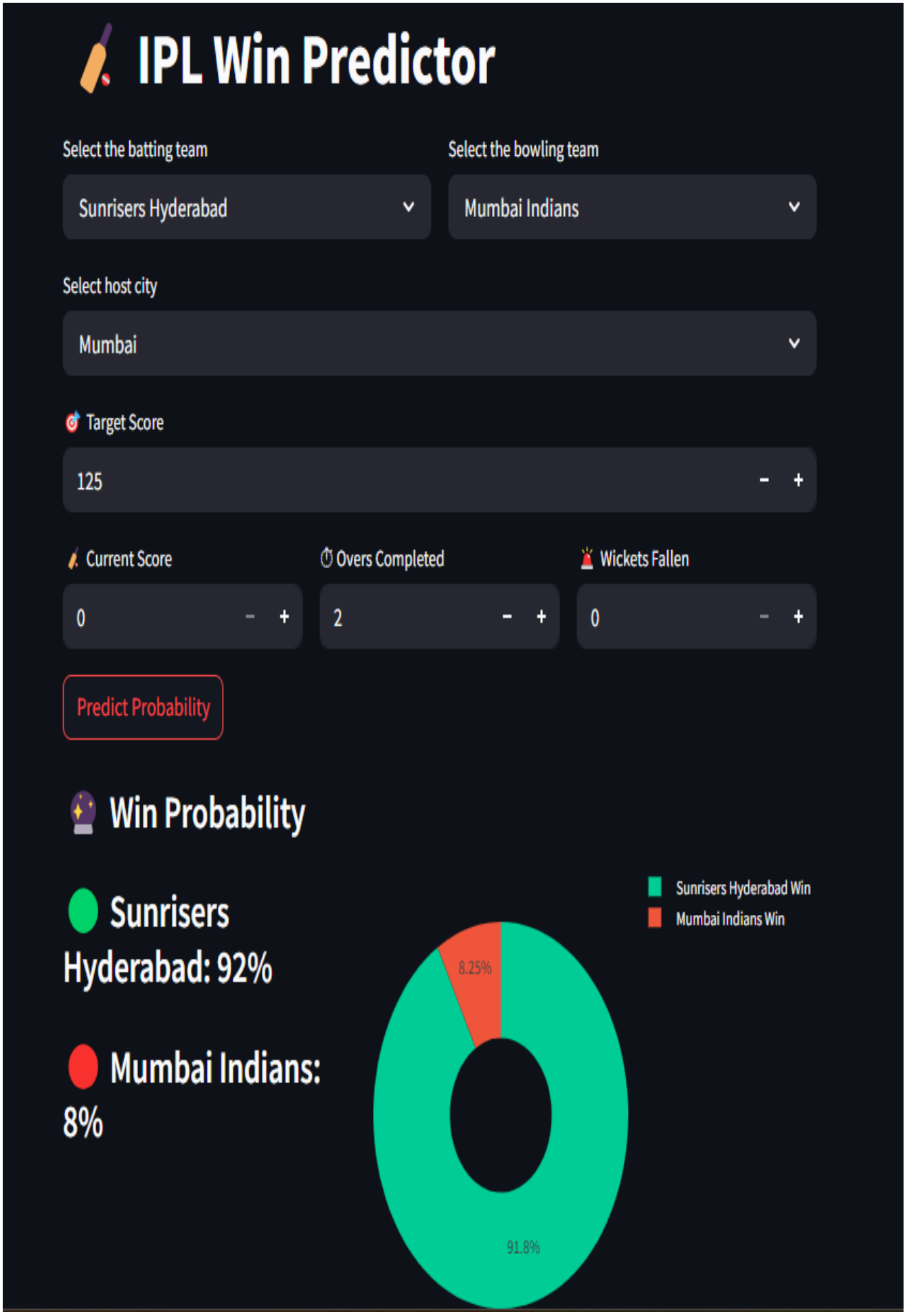
USER INTERFACE BUILDING (app.py)

```
import streamlit as st
import pickle
import sklearn
import pandas as pd
import plotly.express as px

teams = [
    'Sunrisers Hyderabad',
    'Mumbai Indians',
    'Royal Challengers Bangalore',
    'Kolkata Knight Riders',
    'Punjab Kings',
    'Chennai Super Kings',
    'Rajasthan Royals',
```

```
    'Delhi Capitals'
]

cities = ['Hyderabad', 'Bangalore', 'Mumbai', 'Indore', 'Kolkata', 'Delhi',
       'Chandigarh', 'Jaipur', 'Chennai', 'Cape Town', 'Port Elizabeth',
       'Durban', 'Centurion', 'East London', 'Johannesburg', 'Kimberley',
       'Bloemfontein', 'Ahmedabad', 'Cuttack', 'Nagpur', 'Dharamsala',
       'Visakhapatnam', 'Pune', 'Raipur', 'Ranchi', 'Abu Dhabi',
       'Sharjah', 'Mohali', 'Bengaluru']

pipe = pickle.load(open('pipe.pkl','rb'))
st.title('🏏 IPL Win Predictor')

col1, col2 = st.columns(2)

with col1:
    batting_team = st.selectbox('Select the batting team',sorted(teams))

team2 = [team for team in teams if team != batting_team]

with col2:
    bowling_team = st.selectbox('Select the bowling team',sorted(team2))

selected_city = st.selectbox('Select host city',sorted(cities))

target = st.number_input("🎯 Target Score", min_value=1, max_value=350, step=1,
format="%d")

col3,col4,col5 = st.columns(3)

with col3:
    score = st.number_input("🏏 Current Score", min_value=0, max_value=target,
step=1, format="%d")
    if( score >= target):
        st.toast("Score should be less than target", icon="⚠️")
        st.stop()

with col4:
    overs = st.number_input("⏱️ Overs Completed", min_value=1, max_value=20, step=1,
format="%d")


with col5:
    wickets = st.number_input("🚨 Wickets Fallen", min_value=0, max_value=10,
step=1, format="%d")

if st.button('Predict Probability'):
    runs_left = target - score
    balls_left = 120 - (overs*6)
    wickets = 10 - wickets
    crr = score/overs
    rrr = (runs_left*6)/balls_left
```

```
    input_df =
pd.DataFrame({'batting_team':[batting_team],'bowling_team':[bowling_team],'city':[se
lected_city],'runs_left':[runs_left],'balls_left':[balls_left],'wickets':[wickets],'
total_runs_x':[target],'crr':[crr],'rrr':[rrr]})

    result = pipe.predict_proba(input_df)
    loss = result[0][0]
    win = result[0][1]



    col_result, col_chart = st.columns([1, 1.5])

    with col_result:
        st.subheader("🏏 Win Probability")
        st.write(f"### 🟢 {batting_team}: **{round(win * 100)}%**")
        st.write(f"### 🟢 {bowling_team}: **{round(loss * 100)}%**")

    with col_chart:
        # Pie Chart
        pie_df = pd.DataFrame({
            'Outcome': [f'{batting_team} Win', f'{bowling_team} Win'],
            'Probability': [win, loss]
        })

        fig_pie = px.pie(
            pie_df, values='Probability', names='Outcome',
            color_discrete_sequence=['#00cc96', '#EF553B'],
            title='',
            hole=0.4
        )

        st.plotly_chart(fig_pie, use_container_width=True)
```


IPL Win Predictor
Select the batting team
Chennai Super Kings
Select the bowling team
Delhi Capitals
Select host city
Abu Dhabi
Target Score
1
Current Score
0
Overs Completed
1
Wickets Fallen
0
Predict Probability


IPL Win Predictor
Select the batting team
Sunrisers Hyderabad
Select the bowling team
Mumbai Indians
Select host city
Mumbai
Target Score
125
Current Score
0
Overs Completed
2
Wickets Fallen
0
Predict Probability
Win Probability
Sunrisers Hyderabad: 92%
Mumbai Indians: 8%
Sunrisers Hyderabad Win
Mumbai Indians Win
8.25%
91.8%

Future Scope of Improvement

While the IPL Win Predictor demonstrates strong potential and delivers accurate predictions based on key match parameters, several enhancements can be made to improve the model's accuracy, usability, and adaptability. Below are possible future improvements and directions for expansion:

1. Incorporation of Real-Time Ball-by-Ball Data

Currently, the model predicts outcomes based on summary statistics like runs, overs, and wickets. Integrating real-time ball-by-ball feeds from APIs (such as CricAPI or ESPNcricinfo) could:

- Enable live updating predictions after every ball.
- Incorporate player-specific performance per delivery (e.g., dot balls, boundaries).
- Add predictive trends based on recent overs.

2. Player Form and Line-up Details

The model does not currently account for individual player performance, injuries, or recent form. Future versions could include:

- Batter and bowler form (average, strike rate, economy).
- Playing XI impact (star players vs. bench strength).
- Player-vs-player matchup insights.

By adding this layer, predictions would be more dynamic and tailored to real match conditions.

3. Venue and Pitch Conditions

Although city information is considered, detailed pitch reports and weather forecasts (e.g., dew factor, pitch type—batting or bowling friendly) could significantly enhance prediction accuracy. These could be fetched using:

- Weather APIs for live updates.
- Historical data per stadium (first/second innings success rate, average score, etc.).

4. Model Optimization and Alternatives

Currently, the model uses a machine learning pipeline (likely based on logistic regression or random forest). Future work can explore:

- Advanced ensemble methods like XGBoost or CatBoost for better feature handling.
- Deep learning approaches like RNNs or transformers for sequence modeling of match progression.
- Time-series forecasting methods for predicting match direction over time.

Comparison of model performances (e.g., accuracy, AUC-ROC) could validate the best-fit algorithm.

5. Interactive Dashboard Enhancements

The current Streamlit UI is user-friendly and functional, but future iterations can include:

- **Match Timeline Visuals**: A line chart showing how win probabilities change over time.
- **Score Projection Tools**: Predict projected score based on current run rate and batting aggression.
- **Scenario Simulation**: Allow users to simulate "what-if" cases (e.g., "What if Kohli scores 20 runs in the next over?").

These features would make the tool more engaging for analysts and fans.

6. Multilingual and Mobile-Friendly Versions

To reach a broader audience across India and globally:

- Offer multilingual UI (e.g., Hindi, Tamil, Bengali).
- Deploy a responsive mobile application version using Flutter or React Native.
- Add voice input for hands-free interaction.

7. Historical Analysis and Post-Match Review

The model can be extended to perform:

- Match result retrospectives with prediction accuracy reviews.
- Historical performance trends per team, venue, or captain.
- Use as a training and analysis tool for teams and commentators.

8. Gamification and Community Features

To encourage user interaction and retention:

- Users could guess match outcomes and earn points.
- Leaderboards and prediction contests.
- Social sharing of predictions to generate engagement.

Conclusion

The IPL Win Predictor project demonstrates how machine learning can be applied effectively to the domain of sports analytics, particularly in the fast-paced and dynamic environment of T20 cricket. The central objective of the project was to build a robust model that could predict the winning probability of a team based on match-specific features such as the batting and bowling teams, city, current score, overs completed, wickets fallen, and the target score.

To achieve this, multiple machine learning models were implemented, tested, and compared, including:

- **Logistic Regression**: Used as a baseline due to its interpretability and suitability for binary classification problems like win/loss.
- **Random Forest**: Leveraged for its ensemble nature and ability to capture non-linear relationships in the dataset.
- **AdaBoost (Adaptive Boosting)**: Used to reduce bias and variance by combining multiple weak learners into a strong learner.
- **Extreme Gradient Boosting (XGB)**: Applied for its superior performance in handling structured data and reducing overfitting.
- **Support Vector Machine (SVM)**: Evaluated for its effectiveness in high-dimensional spaces and ability to define optimal decision boundaries.

After training and evaluating these models using performance metrics like accuracy, precision, recall, and ROC-AUC score, the best-performing model was selected and serialized using `pickle` for deployment in a real-time prediction environment.

The chosen model was integrated into an interactive web application built with **Streamlit** for a clean and responsive front-end interface. **Plotly** was used to provide engaging visualizations, such as dynamic pie charts representing win probabilities. The UI allows users to input live match data and immediately receive predictions, providing a seamless experience for cricket fans, analysts, and enthusiasts.

Although the current model performs well, it operates within the limitations of its feature set and historical data scope. It simplifies the complexity of a cricket match into numerical inputs and does not yet factor in player form, pitch behavior, or real-time external factors such as weather. These simplifications represent opportunities for enhancement rather than shortcomings.

In conclusion, this project not only fulfills its objective of predicting IPL match outcomes but also lays a strong foundation for the application of machine learning in sports. With future improvements like real-time data integration, advanced player analytics, and enhanced modeling techniques, the IPL Win Predictor can evolve into a comprehensive decision-support and engagement tool for the cricketing ecosystem—serving fans, commentators, analysts, and even team strategists.

References

- https://www.kaggle.com/datasets
- https://numpy.org/doc/
- https://pandas.pydata.org/
- https://scikit-learn.org/
- https://matplotlib.org/stable/index.html/
- https://seaborn.pydata.org/
- https://bitbucket.org/toarnabtrainer/aec_ml_mca_feb_2025/
- https://docs.streamlit.io/