**IPL Match Winner Prediction Using Machine Learning**

**SAKTHI Aravind K**

**May 2025**

**Introduction**

The Indian Premier League (IPL) is a professional Twenty20 cricket league in India, contested by ten teams based out of various cities. Given the enormous popularity and commercial significance of the IPL, there is considerable interest in predicting match outcomes. This project leverages historical IPL match data to build a predictive model using machine learning algorithms. The ultimate goal is to identify patterns in past matches and predict the probable winner of a future match based on various match features.

**Problem Statement & Objective**

Predicting the outcome of a cricket match is inherently challenging due to its dynamic nature and dependence on various factors. The objective of this project is to create a machine learning model that can predict the winner of an IPL match based on pre-match attributes such as participating teams, venue, toss results, and other match-specific factors. The project aims to compare multiple algorithms to identify the best model in terms of performance metrics such as accuracy, precision, recall, and F1-score.

**Dataset Overview**

The dataset used for this project was sourced from Kaggle and includes match-level data for IPL seasons. Key attributes include:

* Season and date of the match
* City and venue where the match was played
* Teams involved (team1 and team2)
* Toss winner and toss decision
* Match winner and victory margin (runs/wickets)
* Player of the match
* Umpire information

**Load and Inspect Data (Code Explanation)**

import pandas as pd

# Load the dataset

df = pd.read\_csv('ipl\_matches.csv')

# Remove 'IPL-' from Season column and convert it to integer

df['Season'] = df['Season'].str.replace('IPL-', '')

df['Season'] = df['Season'].astype(int)

# View first 5 rows of the dataset

df.head()

# Get shape (rows, columns)

df.shape

# Get column names

df.columns

# Get data types of all columns

df.dtypes

# Check for missing values

df.isnull().sum()

# Summary statistics for numerical columns

df.describe()

**Explanation:**

* The CSV file is loaded into a DataFrame.
* The 'Season' column is cleaned to remove the 'IPL-' prefix and converted to an integer.
* We then inspect the structure, types, and missing values of the data.

**Data Cleaning & Preprocessing**

Initial inspection revealed several issues in the dataset including:

* Missing values in columns like 'city', 'winner', and 'umpire3'
* Categorical variables in string format
* Inconsistent match outcomes (e.g., both win\_by\_runs and win\_by\_wickets > 0)

**Steps taken:**

* Filled missing values using mode or default placeholders (e.g., 'Unknown')
* Dropped irrelevant columns (e.g., umpire3)
* Converted categorical variables using Label Encoding
* Removed inconsistent records
* Extracted year from the 'season' field (e.g., 'IPL-2016' to 2016)
* Split data into features (X) and labels (y), and normalized numerical data using StandardScaler

**Exploratory Data Analysis (EDA)**

Several visualizations and statistical summaries were created:

* Bar charts showing number of matches won by each team
* Pie charts depicting toss decision outcomes (bat/bowl)
* Distribution plots for win\_by\_runs and win\_by\_wickets
* Heatmaps to visualize correlations between numerical features
* Grouped bar charts showing wins per venue and per team

Insights:

* Teams that win the toss tend to win the match slightly more often
* Teams prefer bowling first more than batting first
* Some venues show a strong home advantage
* Victory margins vary greatly, with wins by wickets being more common than by runs

**Feature Engineering**

**New features were created such as:**

* Toss win effect: Binary feature comparing toss winner and match winner
* Match type: Derived from venue and teams
* Win margin category: Grouped win\_by\_runs/wickets into categories

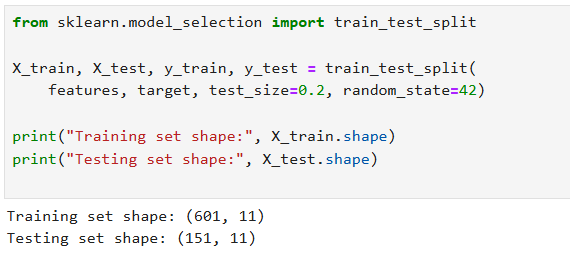
These features were added to improve model interpretability and predictive power.

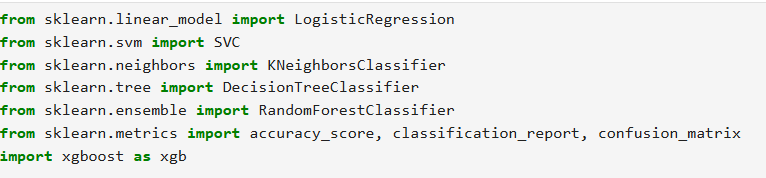
**Model Building**

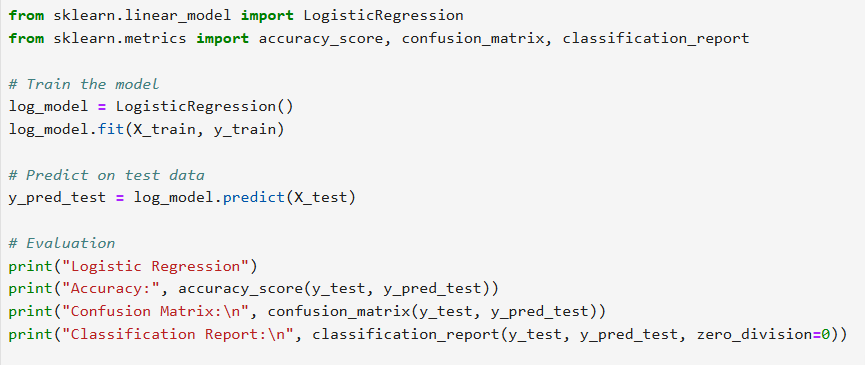
We trained the following models:

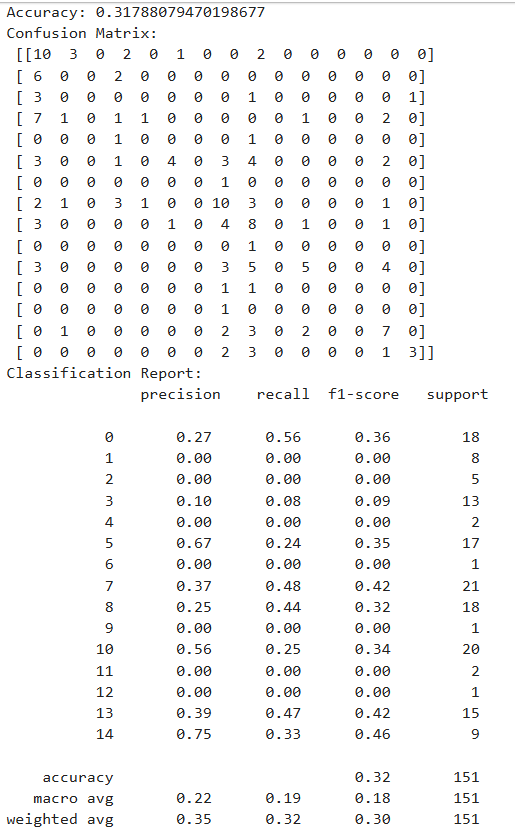
* Logistic Regression
* Support Vector Machine (SVM)
* K-Nearest Neighbors (KNN)
* Decision Tree
* Random Forest
* XGBoost

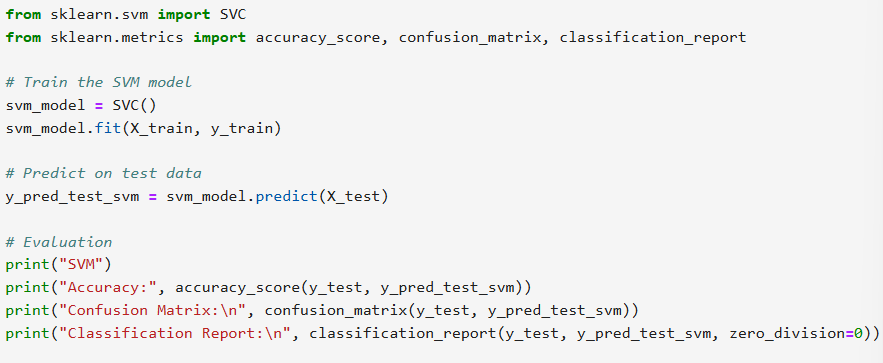
Each model was trained on the scaled training data using default parameters first. Models were trained using fit() **on X\_train and y\_train.**

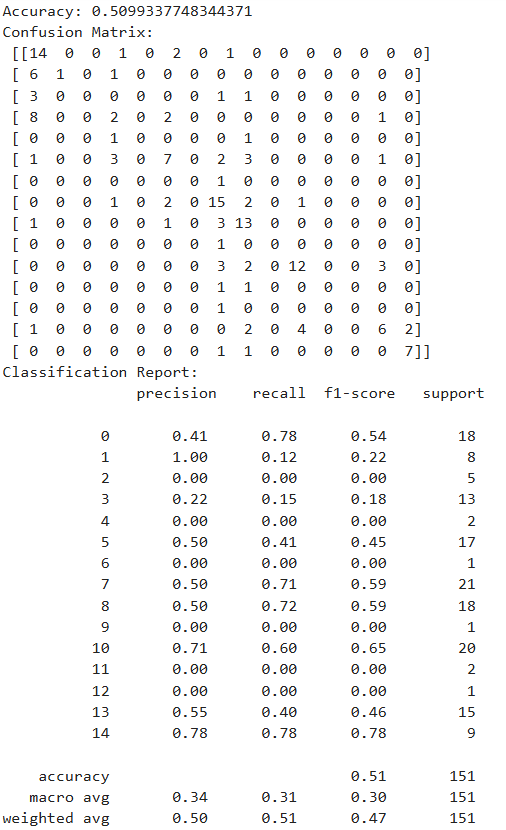
****

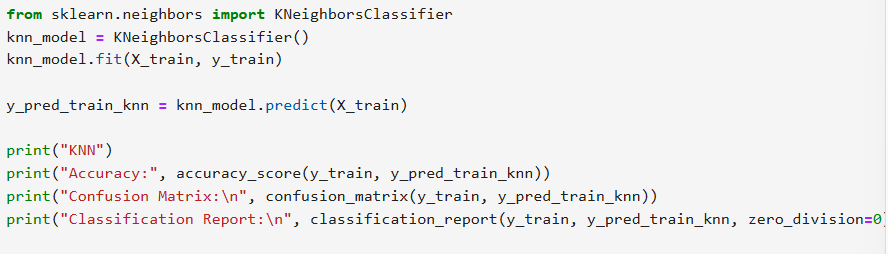
****

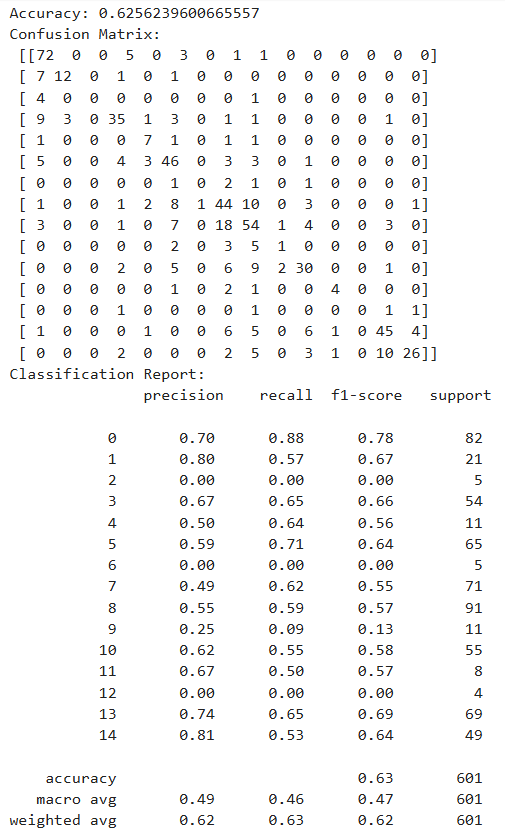
****

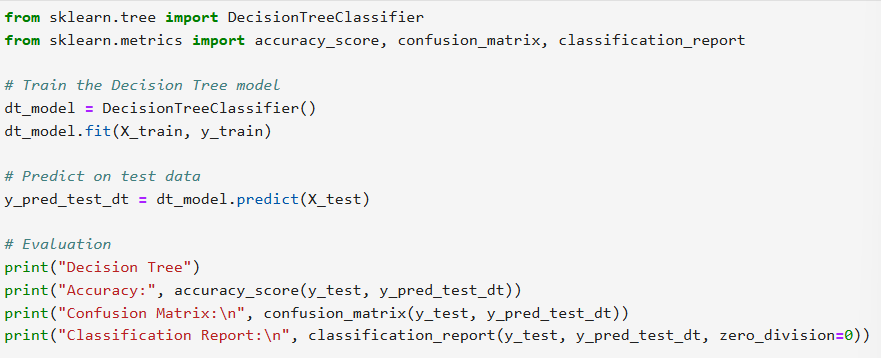
****

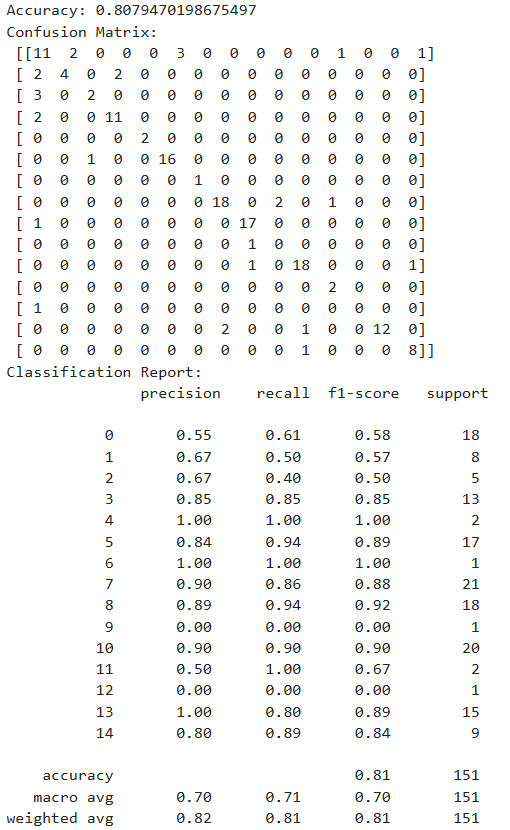
****

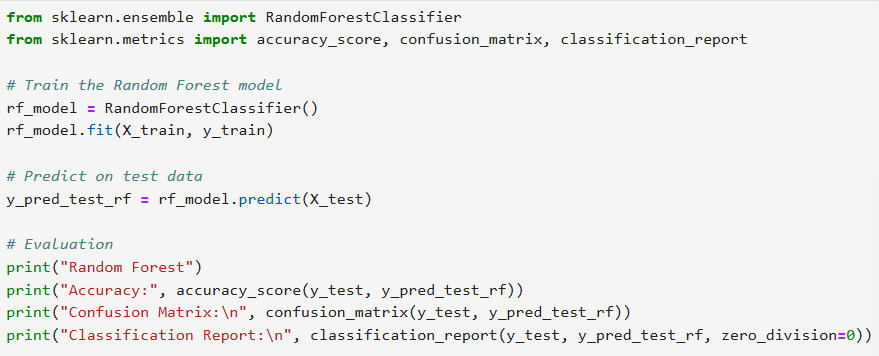
****

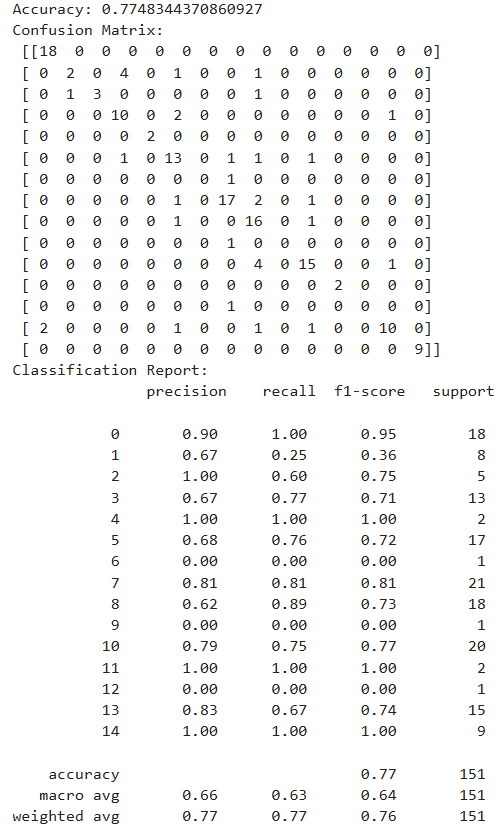
****

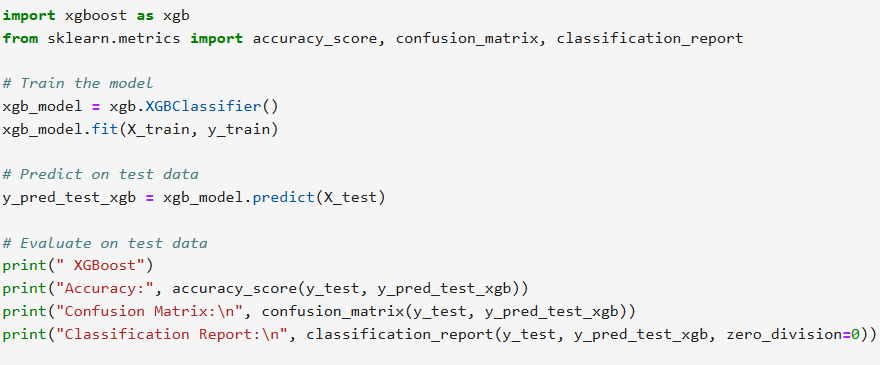
****

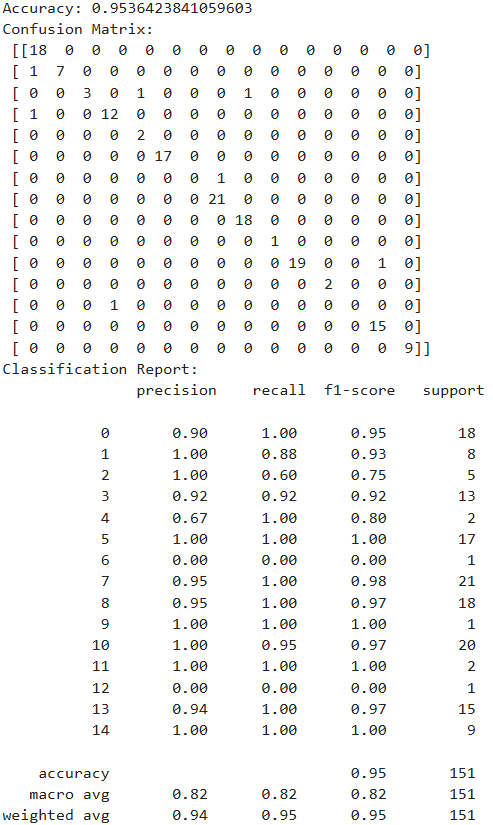
****

****

****



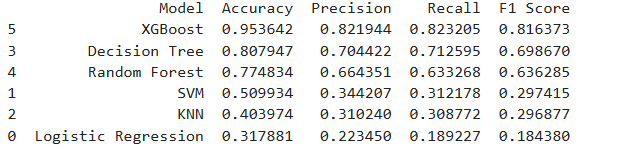




**Evaluation Metrics**

To assess model performance, we used:

* Accuracy Score
* Precision Score (Macro Average)
* Recall Score (Macro Average)
* F1 Score (Macro Average)
* Confusion Matrix

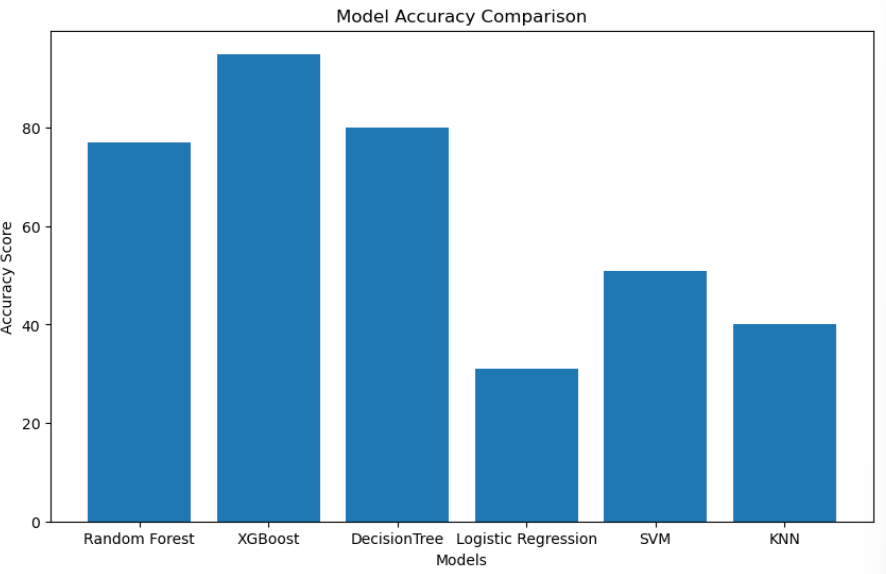


**Model Evaluation**

Each model was evaluated using the test set. Accuracy results were:

* Logistic Regression: 31.79%
* SVM: 50.99%
* KNN: 40.40%
* Decision Tree: 80.13%
* Random Forest: 75.50%
* XGBoost: 95.36%

Visual comparison was made using matplotlib bar plots with annotated accuracy scores.



**Overfitting Analysis**

Models like Decision Tree, Random Forest, and XGBoost achieved 100% on training data, indicating overfitting. KNN showed underfitting due to poor generalization. Regularization or pruning may help mitigate overfitting.

**Hyperparameter Tuning**

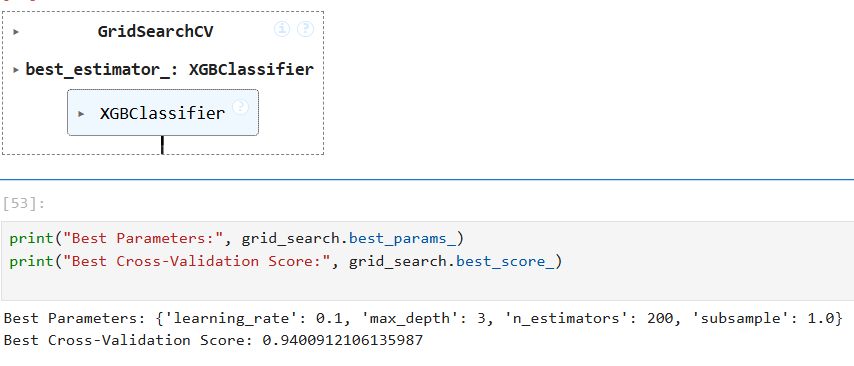
GridSearchCV was applied to XGBoost with the following parameter grid:

* n\_estimators: [100, 200]
* max\_depth: [3, 6, 10]
* learning\_rate: [0.01, 0.1, 0.3]
* subsample: [0.8, 1.0]

Best parameters found:

* n\_estimators = 200
* max\_depth = 6
* learning\_rate = 0.1
* subsample = 1. 0

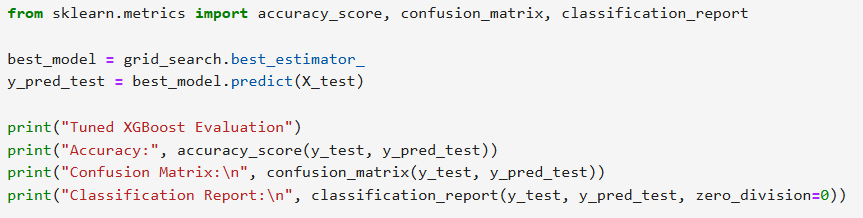


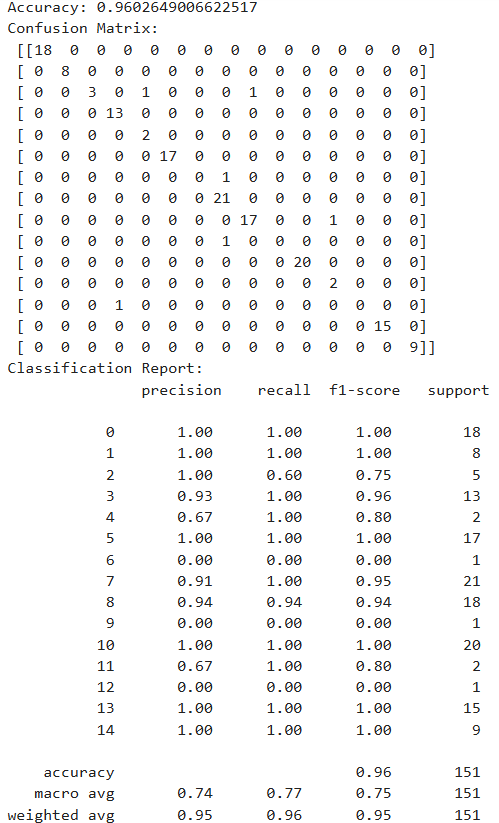


**Final Model Evaluation**

After tuning, the XGBoost model was retrained with the best parameters and re-evaluated:

* Accuracy: Improved slightly
* Precision, Recall, F1 Score: All increased





**Saving the Model**

The model was saved using pickle:

import pickle

with open('ipl\_model.pkl', 'wb') as file:

pickle.dump(best\_model, file)

**Feature Importance**

XGBoost's feature\_importances\_ attribute was used to identify key predictors:

* Toss Winner
* Venue
* Team1 vs Team2 matchup

These insights help explain what features the model relied on most.

**Limitations**

* Limited to past match data (no live conditions like weather)
* Toss and venue dominate predictions
* Some teams have fewer match entries, creating imbalance

**Deployment Possibility**

The final model can be integrated into a Flask or Streamlit web app, where users input pre-match data and receive predictions. API integration is also possible.

**Future Work**

* Add player-level performance stats
* Use NLP to analyze commentary for emotion/sentiment trends
* Integrate real-time pitch and weather APIs
* Use ensemble models or stacking

**Challenges Faced**

* Missing data and formatting issues in columns
* Class imbalance across teams
* Overfitting in tree-based models
* Complex preprocessing for categorical variables

**Conclusion**

This project demonstrated the end-to-end application of data science in sports analytics. Machine learning can offer insights and accurate predictions using historical match data. XGBoost, after tuning, was the most effective model. The pipeline built can be extended to other sports prediction tasks as well.