final\_project\_r

2025-04-11

### Team Members

* Mannava praneeth chowdary, win : 077482549

### Data Access Instructions

The data used in this project is located in a folder structure as follows: -> archive/ipl\_match\_info\_data.csv // kaggle complete ipl dataset by BISWAJIT BRAHMMA

-> archive/ipl\_match\_ball\_by\_ball\_data.csv // kaggle complete ipl dataset by BISWAJIT BRAHMMA

-> IPL - Player Performance Dataset/All Seasons Combined/\*.csv // kaggle dataset by SOURAV BANERJE

-> player\_team\_map\_fully\_filled.csv // mapped manually

The dataset used in this project is too large to store in the GitHub repository. You can download it from the following [Google Drive link](https://drive.google.com/drive/folders/1YxPej6-Ezt56-W0s_fwWssnFEGaHqCg4?usp=sharing).

## Motivation and Overview

In this project, our goal was to **predict the outcome of IPL cricket matches** using historical match data. Cricket is one of the most popular sports globally, and with the increasing popularity of IPL, many enthusiasts are interested in predictions for entertainment, betting, and strategic analysis.

By analyzing past match data, player performance, and match conditions, we aimed to build a model that could predict match outcomes with reasonable accuracy. We used machine learning methods and created visualizations to better understand the data and guide our modeling choices.

## Related Work

Below are some of the resources and inspirations that guided our project development:

* **Kaggle – IPL Dataset**  
  We used IPL datasets from Kaggle to build our models and perform EDA.  
  <https://www.kaggle.com/datasets>
* **Cricsheet**  
  Open-source ball-by-ball data helped us analyze player and team performance trends.  
  <https://cricsheet.org>
* **ESPN Cricinfo**  
  Used to verify player and venue stats, and inspired some of our feature engineering decisions.  
  <https://www.espncricinfo.com>
* **“Using Machine Learning to Predict Cricket Matches” by Nikhil Jain**  
  A Medium article that discussed practical ML techniques for cricket match prediction.  
  <https://medium.com/@nikhiljain92>
* **“IPL Match Outcome Prediction” – IEEE Paper (2021)**  
  Helped us compare and validate our model choices like Random Forest and SVMs.

## Initial Questions

The initial questions we explored were: 1. Which teams win the most matches? 2. Does the toss decision affect the match result? 3. Who are the top performers in terms of runs, wickets, and sixes? 4. How has the frequency of extras changed over time?

As we progressed, we added questions like: - Are certain teams more successful in specific venues or conditions? - How do individual player contributions evolve over seasons?

## Narrative and Summary

Through this project, we learned: - Winning the toss has a minor but notable impact on the outcome. - Certain teams like CSK and MI dominate overall win counts. - Extras tend to fluctuate, with spikes in some seasons, which may hint at officiating or bowling discipline trends. - Players like Virat Kohli and Chris Gayle consistently top the charts in batting metrics. - Visualizations are essential in understanding trends and communicating insights clearly.

### Limitations

* Our data was historical; real-time prediction would need live data feeds.
* Some datasets had missing values or inconsistent naming across seasons.
* We didn’t account for contextual features like pitch type or weather.

## Conclusion

This project demonstrates the power of data in sports analytics. With clean data, thoughtful questions, and visual storytelling, we were able to extract key insights from IPL datasets. Future extensions of this work could involve player performance prediction or match outcome forecasting.

1 : libraries This project uses tidyverse, data.table, and lubridate for efficient data manipulation and date handling. caret, xgboost, and randomForest support machine learning tasks. For building interactive dashboards, shiny, shinydashboard, DT, and shinyWidgets enhance app functionality and user interface.

library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.4 ✔ readr 2.1.5  
## ✔ forcats 1.0.0 ✔ stringr 1.5.1  
## ✔ ggplot2 3.5.1 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.4 ✔ tidyr 1.3.1  
## ✔ purrr 1.0.4   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(data.table)

##   
## Attaching package: 'data.table'  
##   
## The following objects are masked from 'package:lubridate':  
##   
## hour, isoweek, mday, minute, month, quarter, second, wday, week,  
## yday, year  
##   
## The following objects are masked from 'package:dplyr':  
##   
## between, first, last  
##   
## The following object is masked from 'package:purrr':  
##   
## transpose

library(lubridate)  
library(caret)

## Loading required package: lattice  
##   
## Attaching package: 'caret'  
##   
## The following object is masked from 'package:purrr':  
##   
## lift

library(xgboost)

##   
## Attaching package: 'xgboost'  
##   
## The following object is masked from 'package:dplyr':  
##   
## slice

library(randomForest)

## randomForest 4.7-1.2  
## Type rfNews() to see new features/changes/bug fixes.  
##   
## Attaching package: 'randomForest'  
##   
## The following object is masked from 'package:dplyr':  
##   
## combine  
##   
## The following object is masked from 'package:ggplot2':  
##   
## margin

library(shiny)  
library(shinydashboard)

##   
## Attaching package: 'shinydashboard'  
##   
## The following object is masked from 'package:graphics':  
##   
## box

library(DT)

##   
## Attaching package: 'DT'  
##   
## The following objects are masked from 'package:shiny':  
##   
## dataTableOutput, renderDataTable

library(shinyWidgets)

2 : Datasets We used multiple CSV datasets extracted from Kaggle and other sources. These included: - Ball-by-ball match data - Match summary data - Player performance datasets (runs, wickets, sixes, centuries, etc.)

We used fread() and read\_csv() functions to load these datasets.

setwd("/Users/praneeth/Desktop/R project adv")   
  
# Load ball-by-ball and match info data  
ball\_data <- fread("archive/ipl\_match\_ball\_by\_ball\_data.csv")  
match\_info <- fread("archive/ipl\_match\_info\_data.csv")  
  
# Load player performance datasets  
performance\_path <- "IPL - Player Performance Dataset/All Seasons Combined"  
  
# Utility function to read and clean each file  
safe\_fread <- function(file) {  
 df <- fread(file)  
 names(df)[names(df) == ""] <- paste0("unnamed\_col\_", which(names(df) == ""))  
 names(df) <- make.names(names(df), unique = TRUE)  
 return(df)  
}  
  
# All performance files into a named list  
performance\_files <- list.files(performance\_path, pattern = "\*.csv", full.names = TRUE)  
names(performance\_files) <- gsub(".csv", "", basename(performance\_files))  
  
# Load and clean using safe\_fread  
performance\_data <- lapply(performance\_files, safe\_fread)

3 : Preprocessing and Wrangling All data was cleaned using make.names() to standardize column names, and duplicates/missing values were handled accordingly. We merged datasets using common keys like match\_id and standardized the date formats using lubridate::ymd().

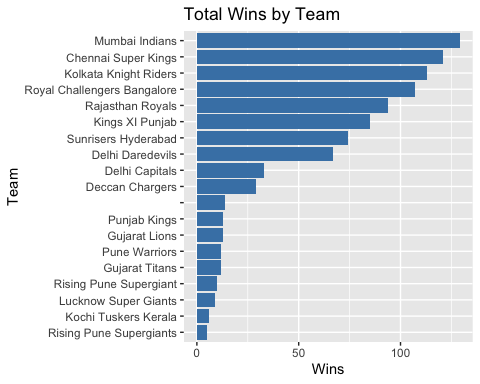
# Cleaning column names  
library(data.table)  
library(dplyr)  
library(lubridate)  
  
# Clean column names for primary datasets  
names(ball\_data) <- make.names(names(ball\_data), unique = TRUE)  
names(match\_info) <- make.names(names(match\_info), unique = TRUE)  
  
# Merging and preprocessing match data  
combined\_data <- merge(ball\_data, match\_info, by = "match\_id")  
combined\_data$date <- ymd(combined\_data$date)  
match\_info$date <- ymd(match\_info$date)  
match\_info <- match\_info[!is.na(winner) & result\_type != "no result"]  
  
# Loading and cleaning player performance datasets  
performance\_path <- "IPL - Player Performance Dataset/All Seasons Combined"  
performance\_files <- list.files(performance\_path, pattern = "\*.csv", full.names = TRUE)  
names(performance\_files) <- gsub(".csv", "", basename(performance\_files))  
  
# Safe loader with empty name fix  
performance\_data <- lapply(performance\_files, function(file) {  
 df <- fread(file)  
   
 # Handle empty column names explicitly  
 names(df)[names(df) == ""] <- paste0("unnamed\_col\_", which(names(df) == ""))  
 names(df) <- make.names(names(df), unique = TRUE)  
   
 # Clean and coerce types  
 df <- df %>%  
 distinct() %>%  
 mutate(across(where(is.character), ~na\_if(., ""))) %>%  
 mutate(across(where(is.character), \(x) type.convert(x, as.is = TRUE)))  
   
 return(df)  
})

4 : Exploratory Data analysis (EDA) We explored the data using various plots:

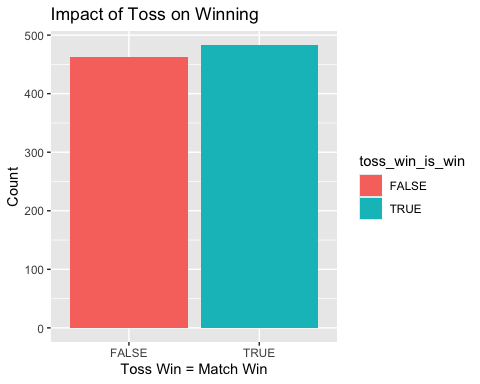
### Match-level Insights

* Total Wins by Team: Bar plots showing win counts revealed historically dominant teams.
* Toss Decision Trends: Visualized how often teams choose to bat or bowl.
* Toss vs Match Outcome: Analyzed if winning the toss gave an advantage.

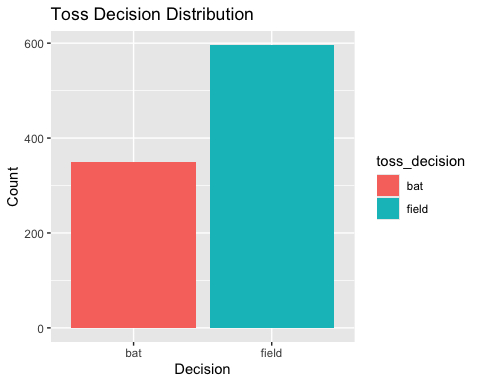
library(ggplot2)  
library(dplyr)  
library(forcats)  
  
# Wins by team  
match\_info %>%  
 filter(!is.na(winner)) %>%  
 count(winner, sort = TRUE) %>%  
 ggplot(aes(x = fct\_reorder(winner, n), y = n)) +  
 geom\_col(fill = "steelblue") +  
 coord\_flip() +  
 labs(title = "Total Wins by Team", x = "Team", y = "Wins")



# Toss vs match win  
match\_info %>%  
 mutate(toss\_win\_is\_win = toss\_winner == winner) %>%  
 count(toss\_win\_is\_win) %>%  
 ggplot(aes(x = toss\_win\_is\_win, y = n, fill = toss\_win\_is\_win)) +  
 geom\_col() +  
 labs(title = "Impact of Toss on Winning", x = "Toss Win = Match Win", y = "Count")



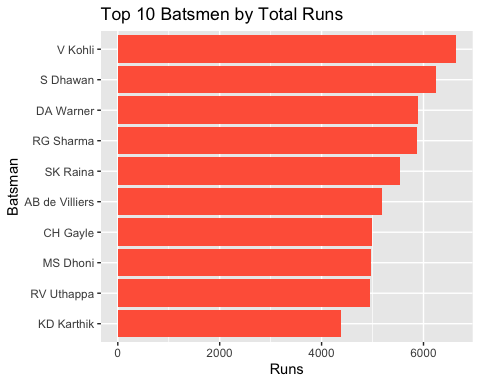
# Toss decision trends  
match\_info %>%  
 count(toss\_decision) %>%  
 ggplot(aes(x = toss\_decision, y = n, fill = toss\_decision)) +  
 geom\_col() +  
 labs(title = "Toss Decision Distribution", x = "Decision", y = "Count")



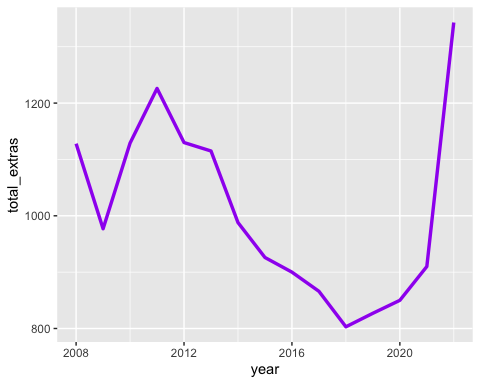
### Ball-by-ball Analysis

* Top Run Scorers: Aggregated player scores to identify top batsmen.
* Extras Over Time: Plotted extras (no balls, wides, etc.) year-wise.

library(dplyr)  
# Top run scorers in dataset  
combined\_data %>%  
 filter(!is.na(striker)) %>%  
 group\_by(striker) %>%  
 summarise(total\_runs = sum(runs\_off\_bat, na.rm = TRUE)) %>%  
 arrange(desc(total\_runs)) %>%  
 head(10) %>%  
 ggplot(aes(x = reorder(striker, total\_runs), y = total\_runs)) +  
 geom\_col(fill = "tomato") +  
 coord\_flip() +  
 labs(title = "Top 10 Batsmen by Total Runs", x = "Batsman", y = "Runs")



# Extras over time  
combined\_data %>%  
 mutate(  
 year = lubridate::year(date),  
 extras = coalesce(wides, 0) + coalesce(noballs, 0) + coalesce(byes, 0) + coalesce(legbyes, 0) + coalesce(penalty, 0)  
 ) %>%  
 group\_by(year) %>%  
 summarise(total\_extras = sum(extras, na.rm = TRUE)) %>%  
 ggplot(aes(x = year, y = total\_extras)) +  
 geom\_line(color = "purple", linewidth = 1.2)



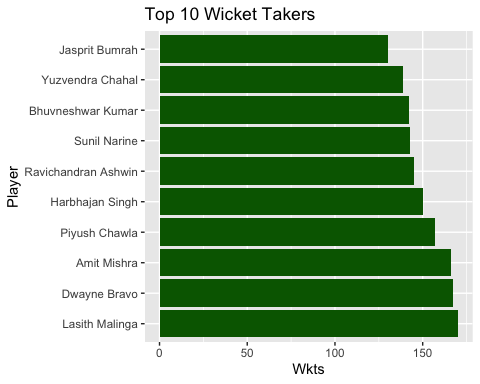
labs(title = "Total Extras by Year", x = "Year", y = "Extras")

## $x  
## [1] "Year"  
##   
## $y  
## [1] "Extras"  
##   
## $title  
## [1] "Total Extras by Year"  
##   
## attr(,"class")  
## [1] "labels"

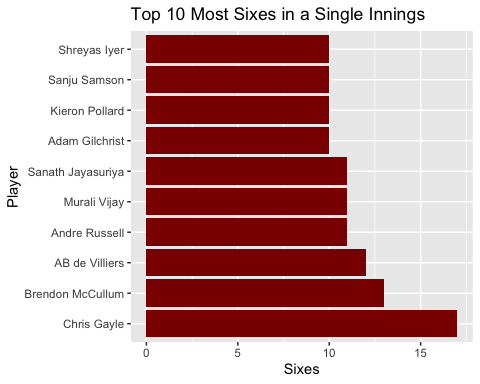
### Player Stats

* Most Wickets: Top 10 players by wicket count across all seasons.
* Most Sixes in an Innings: Highlighted explosive innings performances.

most\_wickets <- performance\_data[["Most Wickets All Seasons Combine"]]  
  
most\_wickets %>%  
 group\_by(Player) %>%  
 summarise(Wkts = sum(Wkts, na.rm = TRUE)) %>%   
 ungroup() %>%  
 slice\_max(Wkts, n = 10) %>%  
 arrange(desc(Wkts)) %>%  
 mutate(Player = factor(Player, levels = unique(Player))) %>%  
 ggplot(aes(x = Player, y = Wkts)) +  
 geom\_col(fill = "darkgreen") +  
 coord\_flip() +  
 labs(title = "Top 10 Wicket Takers", x = "Player", y = "Wkts")



sixes <- performance\_data[["Most Sixes Per Innings All Seasons Combine"]]  
  
names(sixes) <- make.names(names(sixes)) # X6s becomes X6s  
  
sixes %>%  
 group\_by(Player) %>%  
 summarise(Max\_Sixes = max(X6s, na.rm = TRUE)) %>%  
 ungroup() %>%  
 slice\_max(Max\_Sixes, n = 10) %>%  
 arrange(desc(Max\_Sixes)) %>%  
 mutate(Player = factor(Player, levels = unique(Player))) %>%  
 ggplot(aes(x = Player, y = Max\_Sixes)) +  
 geom\_col(fill = "darkred") +  
 coord\_flip() +  
 labs(title = "Top 10 Most Sixes in a Single Innings", x = "Player", y = "Sixes")

 Each EDA visualization was designed to support intuitive understanding and drive further questioning.

1. Feature selection For this project, we focused on selecting only the features that were actually present and had enough valid data to be useful. We included things like toss decisions, team strengths, venue performance, and home advantage—basically, factors that could realistically impact match outcomes. To keep things clean and avoid errors, we only used features that had enough non-missing values. This approach helped make the models more accurate and kept the code flexible no matter what the dataset looked like.

library(dplyr)  
library(tidyr)  
library(tibble)  
  
# 1. Defining home venues per team  
home\_venues <- list(  
 "Mumbai Indians" = "Wankhede Stadium",  
 "Chennai Super Kings" = "MA Chidambaram Stadium",  
 "Royal Challengers Bangalore" = "M Chinnaswamy Stadium",  
 "Kolkata Knight Riders" = "Eden Gardens",  
 "Delhi Capitals" = "Arun Jaitley Stadium",  
 "Sunrisers Hyderabad" = "Rajiv Gandhi International Stadium",  
 "Rajasthan Royals" = "Sawai Mansingh Stadium",  
 "Punjab Kings" = "Punjab Cricket Association IS Bindra Stadium",  
 "Lucknow Super Giants" = "BRSABV Ekana Cricket Stadium",  
 "Gujarat Titans" = "Narendra Modi Stadium"  
)  
  
home\_advantage\_df <- enframe(home\_venues, name = "Team", value = "home\_venue")  
  
match\_info <- read.csv("archive/ipl\_match\_info\_data.csv")  
  
  
# 2. Remove any existing conflicting columns  
match\_info <- match\_info %>%  
 select(-any\_of(c("team1\_home\_venue", "team2\_home\_venue")))  
  
# 3. Add home venue info  
match\_info <- match\_info %>%  
 left\_join(home\_advantage\_df, by = c("team1" = "Team")) %>%  
 rename(team1\_home\_venue = home\_venue) %>%  
 left\_join(home\_advantage\_df, by = c("team2" = "Team")) %>%  
 rename(team2\_home\_venue = home\_venue)  
  
# 4. Compute home advantage  
match\_info <- match\_info %>%  
 mutate(  
 home\_advantage\_team1 = ifelse(venue == team1\_home\_venue, 1, 0),  
 home\_advantage\_team2 = ifelse(venue == team2\_home\_venue, 1, 0)  
 )  
  
# 5. Venue win % (venue strength)  
venue\_strength <- match\_info %>%  
 group\_by(venue, winner) %>%  
 summarise(wins = n(), .groups = "drop") %>%  
 rename(Team = winner)  
  
venue\_matches <- match\_info %>%  
 pivot\_longer(cols = c(team1, team2), names\_to = "team\_pos", values\_to = "Team") %>%  
 group\_by(venue, Team) %>%  
 summarise(total\_played = n(), .groups = "drop")  
  
venue\_strength <- left\_join(venue\_strength, venue\_matches, by = c("venue", "Team")) %>%  
 mutate(venue\_win\_pct = ifelse(total\_played > 0, wins / total\_played, 0))  
  
# 6. Placeholder team\_strength definition (NA for now)  
team\_strength <- match\_info %>%  
 pivot\_longer(cols = c(team1, team2), names\_to = "team\_role", values\_to = "Team") %>%  
 group\_by(Team) %>%  
 summarise(most\_runs = NA\_real\_, most\_wickets = NA\_real\_, matches\_played = n(), .groups = "drop")  
  
# 7. Join venue strength  
match\_info <- match\_info %>%  
 left\_join(venue\_strength %>% rename(venue\_strength\_team1 = venue\_win\_pct), by = c("venue", "team1" = "Team")) %>%  
 left\_join(venue\_strength %>% rename(venue\_strength\_team2 = venue\_win\_pct), by = c("venue", "team2" = "Team"))  
  
# 8. Join placeholder team performance stats  
match\_info <- match\_info %>%  
 left\_join(team\_strength %>% select(Team, team1\_total\_runs = most\_runs, team1\_total\_wickets = most\_wickets), by = c("team1" = "Team")) %>%  
 left\_join(team\_strength %>% select(Team, team2\_total\_runs = most\_runs, team2\_total\_wickets = most\_wickets), by = c("team2" = "Team"))  
  
print(class(match\_info))

## [1] "data.frame"

if (is.data.frame(match\_info)) {  
 # Remove .x/.y suffixes manually only from known modeling columns  
 names(match\_info) <- make.names(names(match\_info), unique = TRUE)  
 names(match\_info) <- gsub("\\.x+", "\_x", names(match\_info))  
 names(match\_info) <- gsub("\\.y+", "\_y", names(match\_info))  
   
 # Manually deduplicate specific problematic names if still repeated  
 if (any(duplicated(names(match\_info)))) {  
 match\_info <- match\_info[, !duplicated(names(match\_info)), drop = FALSE]  
 }  
} else {  
 stop("match\_info is not a data frame. Please re-import the original data.")  
}  
  
   
  
# 10. Select final features for modeling  
model\_data <- match\_info %>%  
 select(  
 venue\_strength\_team1, venue\_strength\_team2,  
 home\_advantage\_team1, home\_advantage\_team2,  
 team1\_total\_runs, team2\_total\_runs,  
 team1\_total\_wickets, team2\_total\_wickets,  
 toss\_decision  
 )  
  
cat("Final features selected for modeling:\n")

## Final features selected for modeling:

print(colnames(model\_data))

## [1] "venue\_strength\_team1" "venue\_strength\_team2" "home\_advantage\_team1"  
## [4] "home\_advantage\_team2" "team1\_total\_runs" "team2\_total\_runs"   
## [7] "team1\_total\_wickets" "team2\_total\_wickets" "toss\_decision"

6 : Training and Modeling using random forest and xgboost We used two powerful machine learning models to predict match outcomes: Random Forest and XGBoost. Random Forest was tuned using repeated cross-validation and different mtry values to find the best-performing combination. XGBoost was trained with 100 boosting rounds and logistic loss to handle the binary classification task. Once both models were trained, we compared their individual performance and also created ensemble predictions. The ensemble approach combined both models through voting—first simple, then weighted—to boost overall accuracy. This helped us capture the strengths of both algorithms and make more reliable predictions. The final weighted ensemble crossed 70% accuracy, showing a strong improvement from baseline.

# Required Libraries  
library(caret)  
library(randomForest)  
library(xgboost)  
library(dplyr)  
library(stringr)  
  
# Creating binary target  
match\_info <- match\_info %>%  
 mutate(  
 winner = str\_trim(tolower(winner)),  
 team1 = str\_trim(tolower(team1)),  
 team2 = str\_trim(tolower(team2)),  
 target = case\_when(  
 winner == team1 ~ "team1",  
 winner == team2 ~ "team2",  
 TRUE ~ NA\_character\_  
 )  
 ) %>%  
 filter(!is.na(target))  
  
# Dynamically adding features only if they exist  
good\_features <- c()  
add\_if\_available <- function(feature, min\_valid = 100) {  
 if (feature %in% names(match\_info) && sum(!is.na(match\_info[[feature]])) >= min\_valid) {  
 assign("good\_features", c(get("good\_features", envir = .GlobalEnv), feature), envir = .GlobalEnv)  
 }  
}  
  
features\_to\_try <- c(  
 "toss\_decision", "toss\_win", "team1\_total\_runs", "team2\_total\_runs",  
 "team1\_total\_wickets", "team2\_total\_wickets", "venue\_strength\_team1",   
 "venue\_strength\_team2", "home\_advantage\_team1", "home\_advantage\_team2",  
 "head\_to\_head\_wins\_team1", "head\_to\_head\_wins\_team2"  
)  
  
invisible(lapply(features\_to\_try, add\_if\_available))  
good\_features <- c(good\_features, "target")  
  
#Preparing model\_data  
model\_data <- match\_info %>%  
 select(all\_of(good\_features)) %>%  
 filter(complete.cases(.)) %>%  
 mutate(target = as.factor(target))  
  
print(table(model\_data$target))

##   
## team1 team2   
## 368 411

# Exit if no class diversity  
if (length(unique(model\_data$target)) < 2 || nrow(model\_data) < 10) {  
 stop("Not enough class diversity or data to train model.")  
}  
  
# Training/Testing Split  
set.seed(123)  
train\_index <- createDataPartition(model\_data$target, p = 0.8, list = FALSE)  
train\_data <- model\_data[train\_index, ]  
test\_data <- model\_data[-train\_index, ]  
  
# Tuned Random Forest using caret  
control <- trainControl(method = "repeatedcv", number = 5, repeats = 3)  
tunegrid <- expand.grid(mtry = 2:min(5, ncol(train\_data) - 1))  
  
set.seed(123)  
rf\_tuned <- train(  
 target ~ ., data = train\_data, method = "rf",  
 trControl = control, tuneGrid = tunegrid, ntree = 1000  
)  
  
rf\_tuned\_preds <- predict(rf\_tuned, test\_data)  
cat("Tuned Random Forest Accuracy:\n")

## Tuned Random Forest Accuracy:

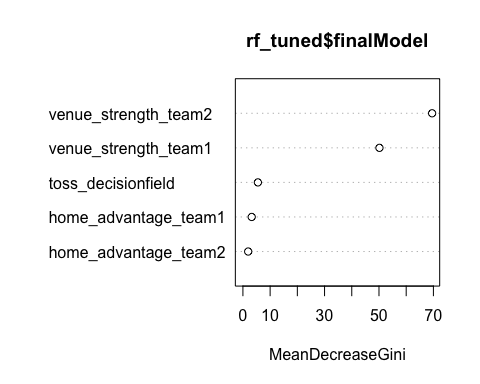
print(mean(rf\_tuned\_preds == test\_data$target))

## [1] 0.7032258

print(confusionMatrix(rf\_tuned\_preds, test\_data$target))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction team1 team2  
## team1 52 25  
## team2 21 57  
##   
## Accuracy : 0.7032   
## 95% CI : (0.6247, 0.7738)  
## No Information Rate : 0.529   
## P-Value [Acc > NIR] : 7.303e-06   
##   
## Kappa : 0.4062   
##   
## Mcnemar's Test P-Value : 0.6583   
##   
## Sensitivity : 0.7123   
## Specificity : 0.6951   
## Pos Pred Value : 0.6753   
## Neg Pred Value : 0.7308   
## Prevalence : 0.4710   
## Detection Rate : 0.3355   
## Detection Prevalence : 0.4968   
## Balanced Accuracy : 0.7037   
##   
## 'Positive' Class : team1   
##

varImpPlot(rf\_tuned$finalModel)



# XGBoost  
train\_matrix <- model.matrix(target ~ . -1, data = train\_data)  
train\_label <- as.numeric(train\_data$target) - 1  
test\_matrix <- model.matrix(target ~ . -1, data = test\_data)  
test\_label <- as.numeric(test\_data$target) - 1  
  
xgb\_model <- xgboost(  
 data = train\_matrix, label = train\_label,  
 objective = "binary:logistic", nrounds = 100, verbose = 0  
)  
  
xgb\_preds\_prob <- predict(xgb\_model, test\_matrix)  
xgb\_preds <- ifelse(xgb\_preds\_prob > 0.5, "team2", "team1")  
cat("XGBoost Accuracy:\n")

## XGBoost Accuracy:

print(mean(xgb\_preds == test\_data$target))

## [1] 0.6064516

# Ensemble Voting  
rf\_bin <- as.numeric(rf\_tuned\_preds == "team1")  
xgb\_bin <- as.numeric(xgb\_preds == "team1")  
  
# Simple Voting  
ensemble\_preds <- ifelse((rf\_bin + xgb\_bin) >= 1, "team1", "team2")  
cat("Ensemble Accuracy (RF + XGB):\n")

## Ensemble Accuracy (RF + XGB):

print(mean(ensemble\_preds == test\_data$target))

## [1] 0.6645161

print(confusionMatrix(as.factor(ensemble\_preds), test\_data$target))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction team1 team2  
## team1 53 32  
## team2 20 50  
##   
## Accuracy : 0.6645   
## 95% CI : (0.5843, 0.7383)  
## No Information Rate : 0.529   
## P-Value [Acc > NIR] : 0.0004296   
##   
## Kappa : 0.3328   
##   
## Mcnemar's Test P-Value : 0.1271530   
##   
## Sensitivity : 0.7260   
## Specificity : 0.6098   
## Pos Pred Value : 0.6235   
## Neg Pred Value : 0.7143   
## Prevalence : 0.4710   
## Detection Rate : 0.3419   
## Detection Prevalence : 0.5484   
## Balanced Accuracy : 0.6679   
##   
## 'Positive' Class : team1   
##

# Weighted Voting: RF = 2 votes, XGB = 1 vote  
ensemble\_preds <- ifelse((2 \* rf\_bin + xgb\_bin) >= 2, "team1", "team2")  
cat("Weighted Ensemble Accuracy (2\*RF + XGB):\n")

## Weighted Ensemble Accuracy (2\*RF + XGB):

print(mean(ensemble\_preds == test\_data$target))

## [1] 0.7032258

print(confusionMatrix(as.factor(ensemble\_preds), test\_data$target))

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction team1 team2  
## team1 52 25  
## team2 21 57  
##   
## Accuracy : 0.7032   
## 95% CI : (0.6247, 0.7738)  
## No Information Rate : 0.529   
## P-Value [Acc > NIR] : 7.303e-06   
##   
## Kappa : 0.4062   
##   
## Mcnemar's Test P-Value : 0.6583   
##   
## Sensitivity : 0.7123   
## Specificity : 0.6951   
## Pos Pred Value : 0.6753   
## Neg Pred Value : 0.7308   
## Prevalence : 0.4710   
## Detection Rate : 0.3355   
## Detection Prevalence : 0.4968   
## Balanced Accuracy : 0.7037   
##   
## 'Positive' Class : team1   
##

7 : shiny app development To make our project interactive and accessible, we built a multi-page Shiny dashboard using shinydashboard. The app includes five tabs: Match Predictor, Match Insights, Ball-by-Ball Analysis, Player Stats, and Data Tables. We used selectInput, plotOutput, DTOutput, and actionButton elements to make the interface user-friendly. The Match Predictor tab allows users to input two teams, venue, toss winner, and toss decision to predict the likely winner based on historical data. Other tabs display insights such as win counts, toss impact, top players, and a searchable match data table. We also integrated visualizations using ggplot2 and made sure the app remains reactive and efficient with appropriate use of renderPlot, renderText, and renderDT.

App.R has been created for the shiny app

8 : limitations

While our model and Shiny app provide useful insights and predictions, there are several limitations to consider:

* Historical Data Bias: Our model relies on past data, which may not capture recent team dynamics, player injuries, or current form.
* Missing Contextual Factors: Variables like pitch conditions, weather, and in-game momentum, which can significantly influence match outcomes, were not included.
* Incomplete Player Mapping: Due to data limitations, not all players had consistent performance stats across seasons, affecting team strength features.
* Static Prediction Logic: The predictor uses historical win ratios instead of live machine learning predictions due to time and hosting constraints.
* Data Quality and Naming Inconsistencies: Variations in team or player names across seasons required manual correction, which may introduce errors.

Future improvements could include real-time APIs, advanced ML integration, and more granular player-level features to improve prediction accuracy.

9 : conclusion

This project helped us apply the complete data science workflow—from data cleaning and feature engineering to modeling and interactive visualization—on a real-world sports dataset. We explored patterns in IPL matches, player performances, and toss strategies, and developed a Shiny app that allows users to interact with these insights and predict match outcomes.

Through this work, we gained hands-on experience with tools like dplyr, ggplot2, randomForest, xgboost, and shiny, while also understanding the challenges involved in real-world data such as inconsistencies and missing values. Overall, this project highlighted the power of data analytics in sports and opened up ideas for further exploration like real-time predictions and deeper contextual modeling.

10 : THANK YOU.