SA Win Prediction Project

Collin

2025-02-18

packages

```r
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 4.4.2
```

```r
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.4.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.4.2
```

```r
library(tidyr)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v readr     2.1.5
## v lubridate 1.9.3     v stringr   1.5.1
## v purrr     1.0.2     v tibble    3.2.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(caTools)
```

```
## Warning: package 'caTools' was built under R version 4.4.2
```

```r
library(caret)
```

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

```r
library(rpart)
library(httr)
```

```
## Warning: package 'httr' was built under R version 4.4.2
```

```
##
## Attaching package: 'httr'
##
## The following object is masked from 'package:caret':
##
##     progress
```

```r
library(rvest)
```

```
## Warning: package 'rvest' was built under R version 4.4.2
```

```
##
## Attaching package: 'rvest'
##
## The following object is masked from 'package:readr':
##
##     guess_encoding
```

```r
library(jsonlite)
```

```
##
## Attaching package: 'jsonlite'
##
## The following object is masked from 'package:purrr':
##
##     flatten
```

```r
library(scales)
```

```
##
## Attaching package: 'scales'
##
## The following object is masked from 'package:purrr':
##
##     discard
##
## The following object is masked from 'package:readr':
##
##     col_factor
```

```r
library(xgboost)
```

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

```r
library(Metrics)
```

```
##
## Attaching package: 'Metrics'
##
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```r
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:ggplot2':
##
##     margin
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
library(DMwR)
```

```
## Loading required package: grid
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```r
library(dplyr)
library(caTools)
library(caret)
```

load in team data

```r
T16 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2015-2016_NBA_Box_Score_Team-Stats
```

```
## New names:
## * `` -> `...39`
## * `` -> `...40`
## * `` -> `...41`
## * `` -> `...42`
```

```r
T17 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2016-2017_NBA_Box_Score_Team-Stats
```

```
## New names:
## * `` -> `...39`
## * `` -> `...40`
## * `` -> `...41`
## * `` -> `...42`
```

```r
T18 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2017-2018_NBA_Box_Score_Team-Stats
```

```
## New names:
## * `` -> `...39`
## * `` -> `...40`
## * `` -> `...41`
## * `` -> `...42`
```

```r
T19 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2018-2019_NBA_Box_Score_Team-Stats
```

```
## New names:
## * `` -> `...39`
## * `` -> `...40`
## * `` -> `...41`
## * `` -> `...42`
```

```r
T20 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2019-2020_NBA_Box_Score_Team-Stats
```

```
## New names:
## * `` -> `...39`
## * `` -> `...40`
## * `` -> `...41`
## * `` -> `...42`
```

```r
T21 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2020-2021_NBA_Box_Score_Team-Stats
```

```
## New names:
## * `` -> `...39`
## * `` -> `...40`
## * `` -> `...41`
## * `` -> `...42`
```

```
T22 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2021-2022_NBA_Box_Score_Team-Stats
```

```
## New names:
## * `` -> `...39`
## * `` -> `...40`
## * `` -> `...41`
## * `` -> `...42`
```

```
T23 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2022-2023_NBA_Box_Score_Team-Stats
```

```
## New names:
## * `` -> `...39`
## * `` -> `...40`
## * `` -> `...41`
## * `` -> `...42`
```

```
T24 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2023-2024_NBA_Box_Score_Team-Stats
```

```
## New names:
## * `` -> `...39`
## * `` -> `...40`
## * `` -> `...41`
## * `` -> `...42`
```

load in player data

```
P16 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2015-2016-NBA-Player-BoxScore-Data
P17 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2016-2017-NBA-Player-BoxScore-Data
P18 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2017-2018-NBA-Player-BoxScore-Data
P19 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2018-2019_NBA_Player-BoxScore-Data
P20 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2019-2020_NBA_Player-BoxScore-Data
P21 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2020-2021_NBA_Player-BoxScore-Data
P22 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2021-2022_NBA_Player-BoxScore-Data
P23 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2022-2023_NBA_Player-BoxScore-Data
P24 <- read_excel("C:/Users/cstra/projects/NBASingleGameStatWinModel/2023-2024-NBA-Player-BoxScore-Data
```

rename

```
#T16 <- T16 %>%
  #rename(DATASET = `BIGDATABALL\r\nDATASET`)
#T17 <- T17 %>%
  #rename(DATASET = `BIGDATABALL\r\nDATASET`)
#T18 <- T18 %>%
  #rename(DATASET = `BIGDATABALL\r\nDATASET`)
```

```r
#T19 <- T19 %>%
  #rename(DATASET = `BIGDATABALL\r\nDATASET`)
T20 <- T20 %>%
  rename(DATASET = `BIGDATABALL\r\nDATASET`)
T21 <- T21 %>%
  rename(DATASET = `BIGDATABALL\r\nDATASET`)
T22 <- T22 %>%
  rename(DATASET = `BIGDATABALL\r\nDATASET`)
T23 <- T23 %>%
  rename(DATASET = `BIGDATABALL\r\nDATASET`)
T24 <- T24 %>%
  rename(DATASET = `BIGDATABALL\r\nDATASET`)


P16 <- P16 %>%
  rename(DATASET = `BIGDATABALL\r\nDATASET`)
P17 <- P17 %>%
  rename(DATASET = `BIGDATABALL\r\nDATASET`)
P18 <- P18 %>%
  rename(DATASET = `BIGDATABALL\r\nDATASET`)
#P19 <- P19 %>%
  #rename(DATASET = `BIGDATABALL\r\nDATASET`)
P20 <- P20 %>%
  rename(DATASET = `BIGDATABALL\r\nDATASET`)
P21 <- P21 %>%
  rename(DATASET = `BIGDATABALL\r\nDATASET`)
P22 <- P22 %>%
  rename(DATASET = `BIGDATABALL\r\nDATASET`)
P23 <- P23 %>%
  rename(DATASET = `BIGDATABALL\r\nDATASET`)
P24 <- P24 %>%
  rename(DATASET = `BIGDATABALL\r\nDATASET`)
```

combine datasets

```r
TeamData <- bind_rows(T16, T17, T18, T19, T20, T21, T22, T23, T24)

PlayerData <- bind_rows(P16, P17, P18, P19, P20, P21, P22, P23, P24)
```

create a win loss column

```r
TeamData <- TeamData %>% select(1:15)

TeamData <- TeamData %>%
  group_by(`GAME-ID`) %>%
  mutate(RESULT = ifelse(F == max(F), "Win", "Loss")) %>%
  ungroup()
```

combine team and game id

```r
TeamData <- TeamData %>%
  mutate(
```

```
    TeamGameID = paste(`GAME-ID`, TEAM, sep = " ")
  )

PlayerData <- PlayerData %>%
  mutate(
    TeamGameID = paste(`GAME-ID`, `OWN \r\nTEAM`, sep = " ")
  )
```

join data

```
Data <- merge(TeamData, PlayerData, by = "TeamGameID", all = FALSE)
```

clean

```
Data <- Data %>%
  rename(
    StarterYN = `STARTER\r\n(Y/N)`,
    PlayerFullName = `PLAYER \r\nFULL NAME`,
    OwnTeam = `OWN \r\nTEAM`,
    OpponentTeam = `OPPONENT \r\nTEAM`,
    UsageRate = `USAGE \r\nRATE (%)`,
    DaysRest = `DAYS\r\nREST`
  )


Data <- Data %>%
  mutate(
    GameID = coalesce(`GAME-ID.x`, `GAME-ID.y`),
    Date = coalesce(`DATE.x`, `DATE.y`),
    Dataset = coalesce(DATASET.x, DATASET.y)
  ) %>%
  select(-c(`GAME-ID.x`, `GAME-ID.y`, `DATE.x`, `DATE.y`, DATASET.x, DATASET.y))

Data <- Data %>%
  select(-c(`VENUE\r\n(R/H/N)`, `VENUE\r\n(R/H)`))

Data <- Data %>%
  rename(
    Player = PlayerFullName,
  )

Data <- Data %>%
  mutate(POSITION = substr(POSITION, 1, 1)
  )


Data$VENUE[is.na(Data$VENUE)] <- "N"
```

remove blowouts

```
Data <- Data %>%
  group_by(GameID) %>%
```

```
  mutate(ScoreDifference = abs(F - lag(F))) %>%
  filter(is.na(ScoreDifference) | ScoreDifference <= 25) %>%
  ungroup() %>%
  select(-ScoreDifference)
```

Only Starters

```
Data <- Data %>%
  filter(StarterYN == "Y" & MIN > 12)
```

Only All Stars - separate data set - maybe use

```
all_stars <- c(
  "LeBron James", "Kareem Abdul-Jabbar", "Kobe Bryant", "Julius Erving", "Tim Duncan",
  "Kevin Garnett", "Shaquille O'Neal", "Kevin Durant", "Michael Jordan", "Karl Malone",
  "Dirk Nowitzki", "Jerry West", "Wilt Chamberlain", "Bob Cousy", "John Havlicek",
  "Moses Malone", "Dwyane Wade", "Rick Barry", "Larry Bird", "George Gervin",
  "Elvin Hayes", "Magic Johnson", "Hakeem Olajuwon", "Chris Paul", "Oscar Robertson",
  "Bill Russell", "Dolph Schayes", "Isiah Thomas", "Charles Barkley", "Elgin Baylor",
  "Chris Bosh", "Patrick Ewing", "Artis Gilmore", "Allen Iverson", "Bob Pettit",
  "Ray Allen", "Carmelo Anthony", "Paul Arizin", "Stephen Curry", "Clyde Drexler",
  "Hal Greer", "James Harden", "Jason Kidd", "Paul Pierce", "David Robinson",
  "John Stockton", "Anthony Davis", "Paul George", "Robert Parish", "Gary Payton",
  "Russell Westbrook", "Lenny Wilkens", "Dominique Wilkins", "Giannis Antetokounmpo",
  "Vince Carter", "Dave Cowens", "Dave DeBusschere", "Alex English", "Larry Foust",
  "Dwight Howard",
  "Kyrie Irving", "Bob Lanier", "Damian Lillard", "Yao Ming", "Dikembe Mutombo",
  "Steve Nash", "Bill Sharman", "LaMarcus Aldridge", "Dave Bing", "Louie Dampier",
  "Mel Daniels", "Joel Embiid", "Walt Frazier", "Harry Gallatin", "Grant Hill",
  "Dan Issel", "Joe Johnson", "Jerry Lucas", "Ed Macauley", "Slater Martin",
  "Tracy McGrady", "Dick McGuire", "Kevin McHale", "Alonzo Mourning", "Scottie Pippen",
  "Willis Reed", "Jack Sikma", "Nate Thurmond", "Chet Walker", "Jo Jo White",
  "James Worthy", "Tiny Archibald", "Jimmy Butler", "Larry Costello", "Adrian Dantley",
  "Walter Davis", "DeMar DeRozan", "Joe Dumars", "Pau Gasol", "Blake Griffin",
  "Richie Guerin", "Cliff Hagan", "Connie Hawkins", "Tom Heinsohn", "Bailey Howell",
  "Lou Hudson", "Neil Johnston", "Nikola Jokić", "Jimmy Jones", "Shawn Kemp",
  "Kawhi Leonard", "Kyle Lowry", "George McGinnis", "Vern Mikkelsen", "Jermaine O'Neal",
  "Tony Parker", "Mitch Richmond", "Amar'e Stoudemire", "Jack Twyman", "George Yardley",
  "Zelmo Beaty", "Chauncey Billups", "Carl Braun", "Mack Calvin", "Billy Cunningham",
  "Brad Daugherty", "Luka Dončić", "Wayne Embry", "Donnie Freeman", "Tom Gola",
  "Gail Goodrich", "Tim Hardaway", "Spencer Haywood", "Al Horford", "Dennis Johnson",
  "Gus Johnson", "Marques Johnson", "Bobby Jones", "Sam Jones", "Larry Kenon",
  "Rudy LaRusso", "Kevin Love", "Maurice Lucas", "Pete Maravich", "Bob McAdoo",
  "Reggie Miller", "Donovan Mitchell", "Sidney Moncrief", "Chris Mullin", "Don Ohl",
  "Andy Phillip", "Charlie Scott", "Gene Shue", "Ralph Simpson", "Jayson Tatum",
  "David Thompson", "Klay Thompson", "Rudy Tomjanovich", "Wes Unseld", "John Wall",
  "Bobby Wanzer", "Chris Webber", "Paul Westphal", "Vin Baker", "Walt Bellamy",
  "Otis Birdsong", "Rolando Blackman", "Devin Booker", "Ron Boone", "Roger Brown",
  "Joe Caldwell", "Tom Chambers", "Maurice Cheeks", "Doug Collins", "DeMarcus Cousins",
  "Bob Dandridge", "Bob Davies", "Dick Garmaker", "Draymond Green", "Johnny Green",
  "Anfernee Hardaway", "Mel Hutchins", "Warren Jabali", "Larry Jones", "Bernard King",
  "Bill Laimbeer", "Clyde Lovellette", "Shawn Marion", "George Mikan", "Paul Millsap",
  "Earl Monroe", "Willie Naulls", "Bob Netolicky", "Billy Paultz", "Jim Pollard",
```

```
"Micheal Ray Richardson", "Arnie Risen", "Red Robbins", "Alvin Robertson", "Guy Rodgers",
"Rajon Rondo", "Ralph Sampson", "Latrell Sprewell", "Karl-Anthony Towns", "Kemba Walker",
"Ben Wallace", "Rasheed Wallace", "Sidney Wicks",
"Bam Adebayo", "Mark Aguirre", "Gilbert Arenas", "Bradley Beal", "John Beasley",
"Bill Bridges", "Jaylen Brown", "Larry Brown", "Darel Carrier", "Phil Chenier",
"Glen Combs", "Terry Dischinger", "Steve Francis", "Marc Gasol", "Rudy Gobert",
"Richard Hamilton", "Kevin Johnson", "Stew Johnson", "Eddie Jones", "Steve Jones",
"Bob Kauffman", "Red Kerr", "Billy Knight", "Freddie Lewis", "Bob Love",
"Dan Majerle", "Bill Melchionni", "Khris Middleton", "Doug Moe", "Jeff Mullins",
"Larry Nance", "Julius Randle", "Glen Rice", "Derrick Rose", "Dan Roundfield",
"Brandon Roy", "Domantas Sabonis", "Detlef Schrempf", "Paul Seymour", "Ben Simmons",
"Peja Stojaković", "Maurice Stokes", "George Thompson", "Dick Van Arsdale",
"Tom Van   Arsdale",
"Norm Van Lier", "Antoine Walker", "Jamaal Wilkes", "Buck Williams", "Deron Williams",
"Willie Wise", "Trae Young", "Marvin Barnes", "Leo Barnhorst", "Byron Beck",
"Art Becker", "Carlos Boozer", "Elton Brand", "Terrell Brandon", "Frankie Brian",
"John Brisker", "Don Buse", "Caron Butler", "Archie Clark", "Terry Cummings",
"Baron Davis", "Warren Davis", "Luol Deng", "John Drew", "Andre Drummond",
"Kevin Duckworth", "Walter Dukes", "Dike Eddleman", "Anthony Edwards", "Sean Elliott",
"Michael Finley", "Joe Fulks", "Jack George", "Shai Gilgeous-Alexander", "Manu Ginóbili",
"Tyrese Haliburton", "Roy Hibbert", "Jrue Holiday", "Allan Houston", "Hot Rod Hundley",
"Les Hunter", "Zydrunas Ilgauskas", "Antawn Jamison", "Eddie Johnson", "John Johnson",
"Larry Johnson", "Rich Jones", "Don Kojis", "Wendell Ladner", "Zach LaVine",
"David Lee", "Fat Lever", "Mike Lewis", "Rashard Lewis", "Jeff Malone",
"Danny Manning", "Stephon Marbury", "Jack Marin", "Brad Miller", "Ja Morant",
"Swen Nater", "Norm Nixon", "Joakim Noah", "Victor Oladipo", "Jim Paxson",
"Geoff Petrie", "Terry Porter", "Cincy Powell", "Zach Randolph", "Glenn Robinson",
"Truck Robinson", "Red Rocha", "Dennis Rodman", "Jeff Ruland", "Fred Scolari",
"Kenny Sears", "Frank Selvy", "Pascal Siakam", "James Silas", "Paul Silas",
"Jerry Sloan", "Phil Smith", "Randy Smith", "Jerry Stackhouse", "Levern Tart",
"Brian Taylor", "Reggie Theus", "Isaiah Thomas", "Andrew Toney", "Kelly Tripucka",
"Kiki Vandeweghe", "Bob Verga", "Nikola Vučević", "Nikola Vucevic", "Jimmy Walker",
"Bill Walton",
"Scott Wedman", "David West", "Charlie Williams", "Chuck Williams", "Gus Williams",
"Zion Williamson", "Brian Winters",
"Shareef Abdur-Rahim", "Alvan Adams", "Michael Adams", "Danny Ainge", "Jarrett Allen",
"Kenny Anderson", "B.J. Armstrong", "LaMelo Ball", "Paolo Banchero", "Don Barksdale",
"Scottie Barnes", "Dick Barnett", "Dana Barros", "Butch Beard", "Ralph Beard",
"Mookie Blaylock", "John Block", "Bob Boozer", "Vince Boryla", "Bill Bradley",
"Fred Brown", "Roger Brown", "Jalen Brunson", "Larry Bunce", "Andrew Bynum",
"Austin Carr", "Joe Barry Carroll", "George Carter", "Bill Cartwright", "Sam Cassell",
"Cedric Ceballos", "Tyson Chandler", "Len Chappell", "Nat Clifton", "Derrick Coleman",
"Jack Coleman", "Mike Conley", "Antonio Davis", "Dale Davis", "Vlade Divac",
"James Donaldson", "Goran Dragić", "Jim Eakins", "Mark Eaton", "Dale Ellis",
"Ray Felix", "Sleepy Floyd", "Jimmy Foster", "De'Aaron Fox", "World B. Free",
"Bill Gabor", "Darius Garland", "Chris Gatling", "Gus Gerard", "Gerald Govan",
"Danny Granger", "Horace Grant", "A.C. Green", "Mike Green", "Rickey Green",
"Alex Groza", "Tom Gugliotta", "Devin Harris", "Bob Harrison", "Hersey Hawkins",
"Gordon Hayward", "Walt Hazzard", "Art Heyman", "Wayne Hightower", "Tyrone Hill",
"Lionel Hollins", "Jeff Hornacek", "Josh Howard", "Juwan Howard", "Andre Iguodala",
"Darrall Imhoff", "Brandon Ingram", "Jaren Jackson Jr.", "Luke Jackson", "Mark Jackson",
"Merv Jackson", "Tony Jackson", "Neil Johnson", "Steve Johnson", "Caldwell Jones",
"Wil Jones", "DeAndre Jordan", "Chris Kaman", "Julius Keye", "Jim King",
```

```
  "Andrei Kirilenko", "Kyle Korver", "Sam Lacey", "Christian Laettner", "Clyde Lee",
  "Reggie Lewis", "Goose Ligon", "Brook Lopez", "Jamaal Magloire", "Randy Mahaffey",
  "Lauri Markkanen", "Kenyon Martin", "Jamal Mashburn", "Anthony Mason", "Tyrese Maxey",
  "Ted McClain", "Xavier McDaniel", "Jim McDaniels", "Antonio McDyess", "Jon McGlocklin",
  "Dewitt Menyard", "Tom Meschery", "Eddie Miles", "Mike Mitchell", "Steve Mix",
  "Jack Molinas", "Gene Moore", "Calvin Murphy", "Dejounte Murray", "Calvin Natt",
  "Jameer Nelson", "Chuck Noble", "Charles Oakley", "Mehmet Okur", "Ricky Pierce",
  "Kristaps Porziņģis", "Jim Price", "Theo Ratliff", "Michael Redd", "Richie Regan",
  "Doc Rivers", "Clifford Robinson", "Flynn Robinson", "Curtis Rowe", "Bob Rule",
  "Campy Russell", "Cazzie Russell", "D'Angelo Russell", "Woody Sauldsberry", "Fred Schaus",
  "Lee Shaffer", "Lonnie Shelton", "Walt Simon", "Adrian Smith", "Steve Smith",
  "Rik Smits", "Willie Somerset", "John Starks", "Don Sunderlage", "Wally Szczerbiak",
  "Jeff Teague", "Claude Terry", "Skip Thoren", "Otis Thorpe", "Monte Towe",
  "Dave Twardzik", "Nick Van Exel", "Fred VanVleet", "Chico Vaughn", "Gerald Wallace",
  "Paul Walther", "Ben Warley", "Kermit Washington", "Trooper Washington", "Andrew Wiggins",
  "Jayson Williams", "Mo Williams", "Kevin Willis", "Metta World Peace", "Max Zaslofsky"
)


DataAS <- Data %>%
  filter(Player %in% all_stars)
```

model time

rf - all players

```
# Split the data into training and test sets
set.seed(123) # For reproducibility
train_index_rf <- createDataPartition(Data$RESULT, p = 0.8, list = FALSE)
train_data_rf <- Data[train_index_rf, ]
test_data_rf <- Data[-train_index_rf, ]

# Define the control method
control <- trainControl(method = "cv", number = 5, search = "grid")

# Define the grid of hyperparameters to search
grid <- expand.grid(mtry = 1)

# Train the random forest model
rf_grid <- train(RESULT ~ PTS + TOT + A, data = train_data_rf,
                 method = "rf",
                 trControl = control,
                 tuneGrid = grid,
                 ntree = 5)  # Added ntree parameter here

# Make predictions on the test data
pred <- predict(rf_grid, test_data_rf)

# Evaluate the model using a confusion matrix
conf_matrix <- confusionMatrix(pred, as.factor(test_data_rf$RESULT))
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction Loss  Win
##       Loss 7306 6307
##       Win  3948 5001
##
##                Accuracy : 0.5455
##                  95% CI : (0.5389, 0.552)
##     No Information Rate : 0.5012
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.0914
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6492
##             Specificity : 0.4423
##          Pos Pred Value : 0.5367
##          Neg Pred Value : 0.5588
##              Prevalence : 0.4988
##          Detection Rate : 0.3238
##    Detection Prevalence : 0.6034
##       Balanced Accuracy : 0.5457
##
##        'Positive' Class : Loss
##
```

log reg - Lebron

```r
# Filter the dataset for the specific player (e.g., LeBron James)
player_name <- "LeBron James"
player_data <- Data %>%
  filter(Player == player_name)

# Select relevant features and ensure the RESULT column is a factor
player_data <- player_data %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT) %>%
  mutate(RESULT = as.factor(RESULT))

# Split the data into training and test sets
set.seed(123) # For reproducibility
train_index <- createDataPartition(player_data$RESULT, p = 0.8, list = FALSE)
train_data <- player_data[train_index, ]
test_data <- player_data[-train_index, ]

# Train a logistic regression model
log_model <- train(RESULT ~ ., data = train_data, method = "glm", family = "binomial")

# Make predictions on the test data
pred <- predict(log_model, test_data)

# Evaluate the model using a confusion matrix
conf_matrix <- confusionMatrix(pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Loss Win
##       Loss   15  13
##       Win    36  71
##
##                Accuracy : 0.637
##                  95% CI : (0.5499, 0.718)
##     No Information Rate : 0.6222
##     P-Value [Acc > NIR] : 0.397677
##
##                   Kappa : 0.1529
##
##  Mcnemar's Test P-Value : 0.001673
##
##             Sensitivity : 0.2941
##             Specificity : 0.8452
##          Pos Pred Value : 0.5357
##          Neg Pred Value : 0.6636
##              Prevalence : 0.3778
##          Detection Rate : 0.1111
##    Detection Prevalence : 0.2074
##       Balanced Accuracy : 0.5697
##
##        'Positive' Class : Loss
##
```

gradient boosting - Lebron

```r
# Load necessary packages
library(gbm)
```

```
## Loaded gbm 2.2.2
```

```
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.c
```

```r
library(caret)
library(dplyr)

# Filter the dataset for the specific player (e.g., LeBron James)
player_name <- "LeBron James"
player_data <- Data %>%
  filter(Player == player_name)

# Select relevant features and ensure the RESULT column is a factor
player_data <- player_data %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT) %>%
  mutate(RESULT = as.factor(RESULT))

# Split the data into training and test sets
set.seed(123) # For reproducibility
train_index <- createDataPartition(player_data$RESULT, p = 0.8, list = FALSE)
```

```r
train_data <- player_data[train_index, ]
test_data <- player_data[-train_index, ]

# Train a GBM model
gbm_model <- train(RESULT ~ ., data = train_data, method = "gbm",
                   trControl = trainControl(method = "cv", number = 5),
                   tuneLength = 5)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3200             nan     0.1000   -0.0009
##      2        1.3158             nan     0.1000    0.0007
##      3        1.3103             nan     0.1000    0.0016
##      4        1.3063             nan     0.1000    0.0011
##      5        1.3010             nan     0.1000    0.0004
##      6        1.2968             nan     0.1000    0.0009
##      7        1.2933             nan     0.1000    0.0007
##      8        1.2895             nan     0.1000   -0.0003
##      9        1.2859             nan     0.1000   -0.0006
##     10        1.2868             nan     0.1000   -0.0046
##     20        1.2589             nan     0.1000   -0.0014
##     40        1.2108             nan     0.1000   -0.0011
##     60        1.1781             nan     0.1000   -0.0005
##     80        1.1551             nan     0.1000   -0.0006
##    100        1.1362             nan     0.1000   -0.0015
##    120        1.1180             nan     0.1000   -0.0014
##    140        1.1042             nan     0.1000   -0.0020
##    160        1.0931             nan     0.1000   -0.0009
##    180        1.0874             nan     0.1000   -0.0029
##    200        1.0759             nan     0.1000   -0.0009
##    220        1.0654             nan     0.1000   -0.0016
##    240        1.0552             nan     0.1000   -0.0010
##    250        1.0511             nan     0.1000   -0.0017
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3133             nan     0.1000    0.0023
##      2        1.3064             nan     0.1000    0.0002
##      3        1.2952             nan     0.1000    0.0013
##      4        1.2897             nan     0.1000   -0.0012
##      5        1.2869             nan     0.1000   -0.0019
##      6        1.2788             nan     0.1000    0.0007
##      7        1.2719             nan     0.1000   -0.0017
##      8        1.2636             nan     0.1000    0.0011
##      9        1.2590             nan     0.1000   -0.0010
##     10        1.2537             nan     0.1000   -0.0004
##     20        1.1982             nan     0.1000   -0.0015
##     40        1.1242             nan     0.1000   -0.0009
##     60        1.0810             nan     0.1000   -0.0010
##     80        1.0478             nan     0.1000   -0.0026
##    100        1.0138             nan     0.1000   -0.0025
##    120        0.9826             nan     0.1000    0.0005
##    140        0.9580             nan     0.1000   -0.0061
##    160        0.9322             nan     0.1000   -0.0037
##    180        0.9094             nan     0.1000   -0.0029
```

```
##    200        0.8861             nan     0.1000   -0.0025
##    220        0.8625             nan     0.1000   -0.0022
##    240        0.8440             nan     0.1000   -0.0008
##    250        0.8369             nan     0.1000   -0.0019
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3079             nan     0.1000    0.0047
##      2        1.3000             nan     0.1000   -0.0018
##      3        1.2917             nan     0.1000   -0.0017
##      4        1.2821             nan     0.1000    0.0015
##      5        1.2704             nan     0.1000    0.0014
##      6        1.2591             nan     0.1000    0.0004
##      7        1.2493             nan     0.1000   -0.0011
##      8        1.2372             nan     0.1000    0.0031
##      9        1.2283             nan     0.1000    0.0016
##     10        1.2209             nan     0.1000   -0.0020
##     20        1.1531             nan     0.1000    0.0001
##     40        1.0644             nan     0.1000   -0.0036
##     60        1.0100             nan     0.1000   -0.0022
##     80        0.9565             nan     0.1000   -0.0034
##    100        0.9110             nan     0.1000   -0.0021
##    120        0.8652             nan     0.1000   -0.0019
##    140        0.8265             nan     0.1000   -0.0016
##    160        0.7925             nan     0.1000   -0.0037
##    180        0.7541             nan     0.1000   -0.0014
##    200        0.7251             nan     0.1000   -0.0026
##    220        0.6998             nan     0.1000   -0.0030
##    240        0.6743             nan     0.1000   -0.0013
##    250        0.6603             nan     0.1000   -0.0019
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3119             nan     0.1000   -0.0047
##      2        1.2943             nan     0.1000    0.0037
##      3        1.2826             nan     0.1000   -0.0005
##      4        1.2682             nan     0.1000   -0.0003
##      5        1.2578             nan     0.1000   -0.0026
##      6        1.2427             nan     0.1000    0.0030
##      7        1.2291             nan     0.1000    0.0011
##      8        1.2219             nan     0.1000   -0.0029
##      9        1.2088             nan     0.1000    0.0005
##     10        1.2001             nan     0.1000   -0.0019
##     20        1.1143             nan     0.1000   -0.0006
##     40        0.9982             nan     0.1000   -0.0036
##     60        0.9297             nan     0.1000   -0.0016
##     80        0.8707             nan     0.1000   -0.0043
##    100        0.8209             nan     0.1000   -0.0037
##    120        0.7688             nan     0.1000   -0.0014
##    140        0.7233             nan     0.1000   -0.0035
##    160        0.6873             nan     0.1000   -0.0023
##    180        0.6526             nan     0.1000   -0.0024
##    200        0.6143             nan     0.1000   -0.0037
##    220        0.5841             nan     0.1000   -0.0022
##    240        0.5527             nan     0.1000   -0.0015
##    250        0.5350             nan     0.1000   -0.0018
```

```
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3057             nan     0.1000    0.0005
##      2        1.2883             nan     0.1000    0.0006
##      3        1.2715             nan     0.1000    0.0036
##      4        1.2542             nan     0.1000    0.0029
##      5        1.2467             nan     0.1000   -0.0046
##      6        1.2314             nan     0.1000    0.0027
##      7        1.2130             nan     0.1000    0.0062
##      8        1.2033             nan     0.1000   -0.0031
##      9        1.1944             nan     0.1000   -0.0014
##     10        1.1832             nan     0.1000   -0.0008
##     20        1.0961             nan     0.1000   -0.0027
##     40        0.9663             nan     0.1000   -0.0022
##     60        0.8784             nan     0.1000   -0.0020
##     80        0.7918             nan     0.1000   -0.0028
##    100        0.7314             nan     0.1000   -0.0036
##    120        0.6672             nan     0.1000   -0.0015
##    140        0.6179             nan     0.1000   -0.0020
##    160        0.5737             nan     0.1000   -0.0029
##    180        0.5311             nan     0.1000   -0.0022
##    200        0.4955             nan     0.1000   -0.0030
##    220        0.4607             nan     0.1000   -0.0013
##    240        0.4281             nan     0.1000   -0.0024
##    250        0.4113             nan     0.1000   -0.0024
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3191             nan     0.1000   -0.0004
##      2        1.3132             nan     0.1000    0.0020
##      3        1.3110             nan     0.1000   -0.0004
##      4        1.3081             nan     0.1000   -0.0001
##      5        1.3062             nan     0.1000   -0.0014
##      6        1.3012             nan     0.1000    0.0016
##      7        1.2989             nan     0.1000   -0.0008
##      8        1.2949             nan     0.1000    0.0002
##      9        1.2935             nan     0.1000   -0.0015
##     10        1.2895             nan     0.1000   -0.0009
##     20        1.2587             nan     0.1000   -0.0012
##     40        1.2134             nan     0.1000   -0.0007
##     60        1.1771             nan     0.1000   -0.0001
##     80        1.1514             nan     0.1000   -0.0018
##    100        1.1315             nan     0.1000   -0.0015
##    120        1.1123             nan     0.1000   -0.0004
##    140        1.1025             nan     0.1000   -0.0010
##    160        1.0926             nan     0.1000   -0.0023
##    180        1.0846             nan     0.1000   -0.0014
##    200        1.0777             nan     0.1000   -0.0015
##    220        1.0668             nan     0.1000   -0.0009
##    240        1.0587             nan     0.1000   -0.0017
##    250        1.0560             nan     0.1000   -0.0008
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3127             nan     0.1000    0.0030
##      2        1.3041             nan     0.1000    0.0032
```

```
##      3        1.3004             nan     0.1000   -0.0028
##      4        1.2910             nan     0.1000    0.0003
##      5        1.2829             nan     0.1000    0.0021
##      6        1.2775             nan     0.1000   -0.0029
##      7        1.2710             nan     0.1000   -0.0011
##      8        1.2672             nan     0.1000   -0.0010
##      9        1.2632             nan     0.1000   -0.0020
##     10        1.2565             nan     0.1000   -0.0001
##     20        1.2037             nan     0.1000    0.0008
##     40        1.1383             nan     0.1000   -0.0028
##     60        1.0899             nan     0.1000   -0.0027
##     80        1.0494             nan     0.1000   -0.0013
##    100        1.0203             nan     0.1000   -0.0028
##    120        0.9983             nan     0.1000   -0.0025
##    140        0.9594             nan     0.1000   -0.0037
##    160        0.9383             nan     0.1000   -0.0010
##    180        0.9163             nan     0.1000   -0.0034
##    200        0.8927             nan     0.1000   -0.0015
##    220        0.8735             nan     0.1000   -0.0023
##    240        0.8462             nan     0.1000   -0.0032
##    250        0.8365             nan     0.1000   -0.0030
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3113             nan     0.1000    0.0023
##      2        1.3015             nan     0.1000   -0.0008
##      3        1.2957             nan     0.1000   -0.0035
##      4        1.2820             nan     0.1000    0.0038
##      5        1.2723             nan     0.1000   -0.0006
##      6        1.2632             nan     0.1000   -0.0009
##      7        1.2544             nan     0.1000   -0.0006
##      8        1.2499             nan     0.1000   -0.0013
##      9        1.2382             nan     0.1000   -0.0024
##     10        1.2321             nan     0.1000   -0.0014
##     20        1.1739             nan     0.1000   -0.0031
##     40        1.0718             nan     0.1000   -0.0014
##     60        1.0031             nan     0.1000   -0.0041
##     80        0.9483             nan     0.1000   -0.0024
##    100        0.9012             nan     0.1000   -0.0019
##    120        0.8626             nan     0.1000   -0.0023
##    140        0.8251             nan     0.1000   -0.0027
##    160        0.7873             nan     0.1000   -0.0030
##    180        0.7534             nan     0.1000   -0.0037
##    200        0.7234             nan     0.1000   -0.0023
##    220        0.6943             nan     0.1000   -0.0005
##    240        0.6700             nan     0.1000   -0.0021
##    250        0.6554             nan     0.1000   -0.0013
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3025             nan     0.1000    0.0002
##      2        1.2879             nan     0.1000    0.0012
##      3        1.2718             nan     0.1000   -0.0022
##      4        1.2613             nan     0.1000    0.0010
##      5        1.2413             nan     0.1000    0.0027
##      6        1.2297             nan     0.1000    0.0013
```

```
##      7        1.2162             nan     0.1000    0.0020
##      8        1.2052             nan     0.1000   -0.0003
##      9        1.1943             nan     0.1000   -0.0018
##     10        1.1861             nan     0.1000   -0.0025
##     20        1.1066             nan     0.1000   -0.0014
##     40        1.0005             nan     0.1000   -0.0011
##     60        0.9158             nan     0.1000   -0.0029
##     80        0.8645             nan     0.1000   -0.0032
##    100        0.8014             nan     0.1000   -0.0036
##    120        0.7617             nan     0.1000   -0.0025
##    140        0.7210             nan     0.1000   -0.0023
##    160        0.6726             nan     0.1000   -0.0016
##    180        0.6355             nan     0.1000   -0.0031
##    200        0.5978             nan     0.1000   -0.0032
##    220        0.5675             nan     0.1000   -0.0027
##    240        0.5327             nan     0.1000   -0.0005
##    250        0.5203             nan     0.1000   -0.0017
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3023             nan     0.1000    0.0060
##      2        1.2835             nan     0.1000    0.0010
##      3        1.2641             nan     0.1000    0.0051
##      4        1.2459             nan     0.1000    0.0049
##      5        1.2309             nan     0.1000   -0.0004
##      6        1.2182             nan     0.1000    0.0009
##      7        1.2035             nan     0.1000   -0.0000
##      8        1.1933             nan     0.1000   -0.0008
##      9        1.1858             nan     0.1000   -0.0020
##     10        1.1738             nan     0.1000    0.0001
##     20        1.0813             nan     0.1000   -0.0044
##     40        0.9633             nan     0.1000   -0.0039
##     60        0.8745             nan     0.1000   -0.0034
##     80        0.7965             nan     0.1000   -0.0022
##    100        0.7255             nan     0.1000   -0.0041
##    120        0.6659             nan     0.1000   -0.0052
##    140        0.6156             nan     0.1000   -0.0044
##    160        0.5654             nan     0.1000   -0.0031
##    180        0.5205             nan     0.1000   -0.0036
##    200        0.4868             nan     0.1000   -0.0012
##    220        0.4452             nan     0.1000   -0.0014
##    240        0.4115             nan     0.1000   -0.0023
##    250        0.3951             nan     0.1000   -0.0020
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3170             nan     0.1000    0.0014
##      2        1.3126             nan     0.1000    0.0011
##      3        1.3115             nan     0.1000   -0.0021
##      4        1.3077             nan     0.1000    0.0014
##      5        1.3005             nan     0.1000   -0.0007
##      6        1.2972             nan     0.1000    0.0002
##      7        1.2938             nan     0.1000    0.0001
##      8        1.2911             nan     0.1000   -0.0009
##      9        1.2876             nan     0.1000    0.0010
##     10        1.2826             nan     0.1000    0.0008
```

```
##     20        1.2456             nan     0.1000   -0.0010
##     40        1.1959             nan     0.1000    0.0009
##     60        1.1595             nan     0.1000   -0.0004
##     80        1.1289             nan     0.1000   -0.0010
##    100        1.1049             nan     0.1000   -0.0008
##    120        1.0919             nan     0.1000   -0.0017
##    140        1.0749             nan     0.1000   -0.0013
##    160        1.0626             nan     0.1000   -0.0009
##    180        1.0498             nan     0.1000   -0.0011
##    200        1.0444             nan     0.1000   -0.0011
##    220        1.0363             nan     0.1000   -0.0012
##    240        1.0303             nan     0.1000   -0.0013
##    250        1.0274             nan     0.1000   -0.0009
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3104             nan     0.1000    0.0033
##      2        1.3018             nan     0.1000   -0.0006
##      3        1.2938             nan     0.1000   -0.0023
##      4        1.2867             nan     0.1000    0.0006
##      5        1.2815             nan     0.1000   -0.0000
##      6        1.2739             nan     0.1000    0.0019
##      7        1.2650             nan     0.1000    0.0014
##      8        1.2563             nan     0.1000    0.0008
##      9        1.2510             nan     0.1000   -0.0010
##     10        1.2460             nan     0.1000   -0.0012
##     20        1.1882             nan     0.1000   -0.0008
##     40        1.1158             nan     0.1000   -0.0014
##     60        1.0599             nan     0.1000   -0.0005
##     80        1.0145             nan     0.1000   -0.0025
##    100        0.9835             nan     0.1000   -0.0010
##    120        0.9516             nan     0.1000   -0.0023
##    140        0.9234             nan     0.1000   -0.0024
##    160        0.8991             nan     0.1000   -0.0032
##    180        0.8687             nan     0.1000   -0.0028
##    200        0.8471             nan     0.1000   -0.0020
##    220        0.8250             nan     0.1000   -0.0014
##    240        0.8054             nan     0.1000   -0.0014
##    250        0.7966             nan     0.1000   -0.0005
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3124             nan     0.1000   -0.0020
##      2        1.2944             nan     0.1000    0.0025
##      3        1.2819             nan     0.1000    0.0031
##      4        1.2708             nan     0.1000    0.0014
##      5        1.2593             nan     0.1000    0.0023
##      6        1.2502             nan     0.1000   -0.0011
##      7        1.2373             nan     0.1000    0.0034
##      8        1.2289             nan     0.1000   -0.0001
##      9        1.2175             nan     0.1000    0.0004
##     10        1.2062             nan     0.1000   -0.0023
##     20        1.1286             nan     0.1000   -0.0006
##     40        1.0387             nan     0.1000   -0.0025
##     60        0.9716             nan     0.1000   -0.0037
##     80        0.9150             nan     0.1000   -0.0021
```

```
##    100        0.8622             nan     0.1000   -0.0032
##    120        0.8006             nan     0.1000   -0.0012
##    140        0.7586             nan     0.1000   -0.0031
##    160        0.7242             nan     0.1000   -0.0036
##    180        0.6907             nan     0.1000   -0.0031
##    200        0.6553             nan     0.1000   -0.0013
##    220        0.6368             nan     0.1000   -0.0021
##    240        0.6056             nan     0.1000   -0.0011
##    250        0.5966             nan     0.1000   -0.0019
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3114             nan     0.1000   -0.0016
##      2        1.2966             nan     0.1000    0.0005
##      3        1.2795             nan     0.1000    0.0022
##      4        1.2655             nan     0.1000    0.0028
##      5        1.2549             nan     0.1000   -0.0031
##      6        1.2422             nan     0.1000    0.0000
##      7        1.2350             nan     0.1000   -0.0050
##      8        1.2220             nan     0.1000    0.0010
##      9        1.2130             nan     0.1000    0.0018
##     10        1.1981             nan     0.1000   -0.0002
##     20        1.1182             nan     0.1000   -0.0037
##     40        0.9913             nan     0.1000   -0.0020
##     60        0.9115             nan     0.1000   -0.0022
##     80        0.8334             nan     0.1000   -0.0035
##    100        0.7750             nan     0.1000   -0.0021
##    120        0.7215             nan     0.1000   -0.0012
##    140        0.6693             nan     0.1000   -0.0017
##    160        0.6247             nan     0.1000   -0.0026
##    180        0.5961             nan     0.1000   -0.0028
##    200        0.5580             nan     0.1000   -0.0022
##    220        0.5293             nan     0.1000   -0.0026
##    240        0.4939             nan     0.1000   -0.0019
##    250        0.4788             nan     0.1000   -0.0005
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2979             nan     0.1000    0.0058
##      2        1.2762             nan     0.1000    0.0017
##      3        1.2584             nan     0.1000    0.0007
##      4        1.2430             nan     0.1000   -0.0018
##      5        1.2263             nan     0.1000    0.0023
##      6        1.2142             nan     0.1000   -0.0021
##      7        1.1982             nan     0.1000    0.0019
##      8        1.1885             nan     0.1000   -0.0014
##      9        1.1788             nan     0.1000   -0.0014
##     10        1.1670             nan     0.1000   -0.0009
##     20        1.0732             nan     0.1000   -0.0021
##     40        0.9346             nan     0.1000   -0.0022
##     60        0.8334             nan     0.1000   -0.0014
##     80        0.7605             nan     0.1000   -0.0026
##    100        0.6934             nan     0.1000   -0.0036
##    120        0.6387             nan     0.1000   -0.0016
##    140        0.5869             nan     0.1000   -0.0022
##    160        0.5353             nan     0.1000   -0.0017
```

```
##    180        0.4870             nan     0.1000   -0.0028
##    200        0.4524             nan     0.1000   -0.0027
##    220        0.4131             nan     0.1000   -0.0023
##    240        0.3813             nan     0.1000   -0.0026
##    250        0.3651             nan     0.1000   -0.0017
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3209             nan     0.1000    0.0007
##      2        1.3150             nan     0.1000    0.0011
##      3        1.3100             nan     0.1000    0.0009
##      4        1.3051             nan     0.1000    0.0015
##      5        1.3003             nan     0.1000    0.0007
##      6        1.2952             nan     0.1000    0.0003
##      7        1.2912             nan     0.1000    0.0003
##      8        1.2875             nan     0.1000   -0.0023
##      9        1.2840             nan     0.1000   -0.0014
##     10        1.2820             nan     0.1000   -0.0012
##     20        1.2550             nan     0.1000   -0.0022
##     40        1.2058             nan     0.1000   -0.0003
##     60        1.1687             nan     0.1000   -0.0004
##     80        1.1440             nan     0.1000   -0.0002
##    100        1.1214             nan     0.1000   -0.0014
##    120        1.1025             nan     0.1000   -0.0005
##    140        1.0875             nan     0.1000   -0.0015
##    160        1.0757             nan     0.1000   -0.0014
##    180        1.0666             nan     0.1000   -0.0007
##    200        1.0522             nan     0.1000   -0.0017
##    220        1.0438             nan     0.1000   -0.0017
##    240        1.0338             nan     0.1000   -0.0004
##    250        1.0305             nan     0.1000   -0.0013
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3179             nan     0.1000    0.0001
##      2        1.3076             nan     0.1000    0.0001
##      3        1.2957             nan     0.1000    0.0032
##      4        1.2886             nan     0.1000   -0.0015
##      5        1.2782             nan     0.1000    0.0012
##      6        1.2680             nan     0.1000   -0.0001
##      7        1.2592             nan     0.1000    0.0020
##      8        1.2525             nan     0.1000    0.0009
##      9        1.2486             nan     0.1000   -0.0013
##     10        1.2442             nan     0.1000   -0.0003
##     20        1.1889             nan     0.1000    0.0004
##     40        1.1290             nan     0.1000   -0.0024
##     60        1.0764             nan     0.1000   -0.0007
##     80        1.0382             nan     0.1000   -0.0039
##    100        1.0096             nan     0.1000   -0.0022
##    120        0.9802             nan     0.1000   -0.0010
##    140        0.9477             nan     0.1000   -0.0016
##    160        0.9271             nan     0.1000   -0.0009
##    180        0.8965             nan     0.1000   -0.0024
##    200        0.8776             nan     0.1000   -0.0031
##    220        0.8552             nan     0.1000   -0.0022
##    240        0.8376             nan     0.1000   -0.0028
```

```
##    250        0.8271             nan     0.1000   -0.0021
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3105             nan     0.1000    0.0010
##      2        1.2987             nan     0.1000    0.0035
##      3        1.2866             nan     0.1000    0.0024
##      4        1.2749             nan     0.1000    0.0019
##      5        1.2632             nan     0.1000    0.0018
##      6        1.2532             nan     0.1000    0.0004
##      7        1.2413             nan     0.1000   -0.0001
##      8        1.2367             nan     0.1000   -0.0039
##      9        1.2277             nan     0.1000    0.0002
##     10        1.2167             nan     0.1000    0.0005
##     20        1.1457             nan     0.1000   -0.0003
##     40        1.0492             nan     0.1000   -0.0012
##     60        0.9832             nan     0.1000   -0.0036
##     80        0.9336             nan     0.1000   -0.0016
##    100        0.8840             nan     0.1000   -0.0004
##    120        0.8415             nan     0.1000   -0.0032
##    140        0.8088             nan     0.1000   -0.0026
##    160        0.7648             nan     0.1000   -0.0018
##    180        0.7383             nan     0.1000   -0.0017
##    200        0.7125             nan     0.1000   -0.0020
##    220        0.6834             nan     0.1000   -0.0038
##    240        0.6564             nan     0.1000   -0.0022
##    250        0.6471             nan     0.1000   -0.0019
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3098             nan     0.1000    0.0024
##      2        1.2970             nan     0.1000    0.0018
##      3        1.2825             nan     0.1000    0.0044
##      4        1.2690             nan     0.1000    0.0017
##      5        1.2580             nan     0.1000   -0.0002
##      6        1.2448             nan     0.1000   -0.0030
##      7        1.2344             nan     0.1000   -0.0011
##      8        1.2201             nan     0.1000    0.0021
##      9        1.2102             nan     0.1000   -0.0003
##     10        1.1969             nan     0.1000    0.0021
##     20        1.1209             nan     0.1000   -0.0017
##     40        1.0096             nan     0.1000   -0.0027
##     60        0.9157             nan     0.1000   -0.0037
##     80        0.8372             nan     0.1000    0.0008
##    100        0.7819             nan     0.1000   -0.0036
##    120        0.7352             nan     0.1000   -0.0019
##    140        0.6946             nan     0.1000   -0.0019
##    160        0.6500             nan     0.1000   -0.0014
##    180        0.6111             nan     0.1000   -0.0007
##    200        0.5722             nan     0.1000   -0.0019
##    220        0.5393             nan     0.1000   -0.0008
##    240        0.5059             nan     0.1000   -0.0022
##    250        0.4937             nan     0.1000   -0.0016
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2991             nan     0.1000    0.0072
```

```
##      2        1.2838             nan     0.1000    0.0020
##      3        1.2694             nan     0.1000    0.0036
##      4        1.2545             nan     0.1000    0.0019
##      5        1.2473             nan     0.1000   -0.0042
##      6        1.2374             nan     0.1000   -0.0053
##      7        1.2221             nan     0.1000    0.0033
##      8        1.2099             nan     0.1000   -0.0001
##      9        1.2005             nan     0.1000   -0.0022
##     10        1.1847             nan     0.1000    0.0007
##     20        1.0930             nan     0.1000   -0.0019
##     40        0.9469             nan     0.1000   -0.0032
##     60        0.8547             nan     0.1000   -0.0024
##     80        0.7913             nan     0.1000   -0.0026
##    100        0.7262             nan     0.1000   -0.0037
##    120        0.6731             nan     0.1000   -0.0018
##    140        0.6243             nan     0.1000   -0.0018
##    160        0.5684             nan     0.1000   -0.0012
##    180        0.5286             nan     0.1000   -0.0024
##    200        0.4932             nan     0.1000   -0.0020
##    220        0.4556             nan     0.1000   -0.0025
##    240        0.4282             nan     0.1000   -0.0009
##    250        0.4093             nan     0.1000   -0.0010
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3177             nan     0.1000    0.0014
##      2        1.3130             nan     0.1000    0.0008
##      3        1.3069             nan     0.1000    0.0000
##      4        1.3031             nan     0.1000    0.0009
##      5        1.3001             nan     0.1000   -0.0013
##      6        1.2956             nan     0.1000    0.0010
##      7        1.2918             nan     0.1000   -0.0001
##      8        1.2873             nan     0.1000    0.0002
##      9        1.2832             nan     0.1000   -0.0005
##     10        1.2806             nan     0.1000   -0.0007
##     20        1.2504             nan     0.1000   -0.0014
##     40        1.2063             nan     0.1000   -0.0013
##     60        1.1742             nan     0.1000   -0.0018
##     80        1.1454             nan     0.1000   -0.0009
##    100        1.1218             nan     0.1000   -0.0030
##    120        1.1097             nan     0.1000   -0.0012
##    140        1.0968             nan     0.1000   -0.0010
##    160        1.0877             nan     0.1000   -0.0018
##    180        1.0761             nan     0.1000   -0.0032
##    200        1.0665             nan     0.1000   -0.0019
##    220        1.0606             nan     0.1000   -0.0015
##    240        1.0541             nan     0.1000   -0.0016
##    250        1.0509             nan     0.1000   -0.0008
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3156             nan     0.1000   -0.0004
##      2        1.3066             nan     0.1000    0.0024
##      3        1.2962             nan     0.1000    0.0007
##      4        1.2893             nan     0.1000   -0.0007
##      5        1.2796             nan     0.1000    0.0018
```

```
##      6        1.2725             nan     0.1000    0.0005
##      7        1.2646             nan     0.1000    0.0001
##      8        1.2568             nan     0.1000   -0.0003
##      9        1.2487             nan     0.1000    0.0011
##     10        1.2442             nan     0.1000   -0.0010
##     20        1.1961             nan     0.1000   -0.0026
##     40        1.1252             nan     0.1000   -0.0051
##     60        1.0730             nan     0.1000   -0.0016
##     80        1.0384             nan     0.1000   -0.0031
##    100        1.0090             nan     0.1000   -0.0033
##    120        0.9812             nan     0.1000   -0.0020
##    140        0.9534             nan     0.1000   -0.0022
##    160        0.9319             nan     0.1000   -0.0017
##    180        0.9087             nan     0.1000   -0.0022
##    200        0.8894             nan     0.1000   -0.0030
##    220        0.8632             nan     0.1000   -0.0015
##    240        0.8371             nan     0.1000   -0.0005
##    250        0.8273             nan     0.1000   -0.0040
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3109             nan     0.1000   -0.0001
##      2        1.2989             nan     0.1000    0.0007
##      3        1.2904             nan     0.1000   -0.0029
##      4        1.2812             nan     0.1000    0.0004
##      5        1.2710             nan     0.1000    0.0008
##      6        1.2623             nan     0.1000    0.0010
##      7        1.2507             nan     0.1000   -0.0009
##      8        1.2411             nan     0.1000    0.0008
##      9        1.2308             nan     0.1000    0.0028
##     10        1.2228             nan     0.1000   -0.0010
##     20        1.1473             nan     0.1000   -0.0020
##     40        1.0639             nan     0.1000   -0.0014
##     60        0.9942             nan     0.1000   -0.0020
##     80        0.9327             nan     0.1000   -0.0024
##    100        0.8874             nan     0.1000   -0.0019
##    120        0.8386             nan     0.1000   -0.0018
##    140        0.7985             nan     0.1000   -0.0026
##    160        0.7653             nan     0.1000   -0.0013
##    180        0.7320             nan     0.1000   -0.0028
##    200        0.7032             nan     0.1000   -0.0039
##    220        0.6772             nan     0.1000   -0.0016
##    240        0.6481             nan     0.1000   -0.0029
##    250        0.6327             nan     0.1000   -0.0033
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3109             nan     0.1000   -0.0014
##      2        1.2945             nan     0.1000    0.0019
##      3        1.2812             nan     0.1000    0.0006
##      4        1.2745             nan     0.1000    0.0002
##      5        1.2602             nan     0.1000   -0.0002
##      6        1.2491             nan     0.1000    0.0007
##      7        1.2341             nan     0.1000   -0.0008
##      8        1.2189             nan     0.1000   -0.0012
##      9        1.2094             nan     0.1000   -0.0022
```

```
##     10        1.1984             nan     0.1000    0.0007
##     20        1.1232             nan     0.1000   -0.0022
##     40        1.0077             nan     0.1000   -0.0014
##     60        0.9117             nan     0.1000   -0.0027
##     80        0.8455             nan     0.1000   -0.0027
##    100        0.7861             nan     0.1000   -0.0036
##    120        0.7342             nan     0.1000   -0.0029
##    140        0.6811             nan     0.1000   -0.0020
##    160        0.6474             nan     0.1000   -0.0020
##    180        0.6129             nan     0.1000   -0.0015
##    200        0.5727             nan     0.1000   -0.0032
##    220        0.5372             nan     0.1000   -0.0027
##    240        0.5103             nan     0.1000   -0.0025
##    250        0.4937             nan     0.1000   -0.0033
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2992             nan     0.1000    0.0055
##      2        1.2821             nan     0.1000   -0.0014
##      3        1.2678             nan     0.1000    0.0005
##      4        1.2530             nan     0.1000   -0.0012
##      5        1.2360             nan     0.1000    0.0010
##      6        1.2190             nan     0.1000    0.0001
##      7        1.2062             nan     0.1000   -0.0016
##      8        1.1920             nan     0.1000    0.0002
##      9        1.1757             nan     0.1000    0.0019
##     10        1.1647             nan     0.1000    0.0010
##     20        1.0723             nan     0.1000   -0.0013
##     40        0.9274             nan     0.1000    0.0003
##     60        0.8338             nan     0.1000   -0.0041
##     80        0.7569             nan     0.1000   -0.0031
##    100        0.6893             nan     0.1000   -0.0016
##    120        0.6276             nan     0.1000   -0.0035
##    140        0.5739             nan     0.1000   -0.0040
##    160        0.5276             nan     0.1000   -0.0024
##    180        0.4859             nan     0.1000   -0.0018
##    200        0.4519             nan     0.1000   -0.0015
##    220        0.4179             nan     0.1000   -0.0017
##    240        0.3871             nan     0.1000   -0.0013
##    250        0.3750             nan     0.1000   -0.0010
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.3095             nan     0.1000    0.0008
##      2        1.2921             nan     0.1000    0.0054
##      3        1.2773             nan     0.1000    0.0010
##      4        1.2666             nan     0.1000   -0.0016
##      5        1.2554             nan     0.1000   -0.0003
##      6        1.2477             nan     0.1000   -0.0003
##      7        1.2382             nan     0.1000   -0.0004
##      8        1.2297             nan     0.1000   -0.0004
##      9        1.2194             nan     0.1000   -0.0009
##     10        1.2118             nan     0.1000   -0.0015
##     20        1.1393             nan     0.1000   -0.0019
##     40        1.0388             nan     0.1000   -0.0026
##     50        1.0029             nan     0.1000   -0.0023
```

```r
# Make predictions on the test data
gbm_pred <- predict(gbm_model, test_data)

# Evaluate the model using a confusion matrix
gbm_conf_matrix <- confusionMatrix(gbm_pred, test_data$RESULT)
print(gbm_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Loss Win
##       Loss   16  14
##       Win    35  70
##
##                Accuracy : 0.637
##                  95% CI : (0.5499, 0.718)
##     No Information Rate : 0.6222
##     P-Value [Acc > NIR] : 0.397677
##
##                   Kappa : 0.16
##
##  Mcnemar's Test P-Value : 0.004275
##
##             Sensitivity : 0.3137
##             Specificity : 0.8333
##          Pos Pred Value : 0.5333
##          Neg Pred Value : 0.6667
##              Prevalence : 0.3778
##          Detection Rate : 0.1185
##    Detection Prevalence : 0.2222
##       Balanced Accuracy : 0.5735
##
##        'Positive' Class : Loss
##
```

KNN - Lebron

```r
# Load necessary packages
library(class)
library(caret)
library(dplyr)

# Filter the dataset for the specific player (e.g., LeBron James)
player_name <- "LeBron James"
player_data <- Data %>%
  filter(Player == player_name)

# Select relevant features and ensure the RESULT column is a factor
player_data <- player_data %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT) %>%
  mutate(RESULT = as.factor(RESULT))

# Split the data into training and test sets
```

```r
set.seed(123) # For reproducibility
train_index <- createDataPartition(player_data$RESULT, p = 0.8, list = FALSE)
train_data <- player_data[train_index, ]
test_data <- player_data[-train_index, ]

# Scale the features
train_features <- train_data %>%
  select(-RESULT) %>%
  scale()
train_labels <- train_data$RESULT

test_features <- test_data %>%
  select(-RESULT) %>%
  scale()
test_labels <- test_data$RESULT

# Train and predict using KNN
k <- 5  # You can tune this value
knn_pred <- knn(train = train_features, test = test_features, cl = train_labels, k = k)

# Evaluate the model using a confusion matrix
knn_conf_matrix <- confusionMatrix(knn_pred, test_labels)
print(knn_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Loss Win
##       Loss   21  25
##       Win    30  59
##
##                Accuracy : 0.5926
##                  95% CI : (0.5047, 0.6763)
##     No Information Rate : 0.6222
##     P-Value [Acc > NIR] : 0.7885
##
##                   Kappa : 0.1164
##
##  Mcnemar's Test P-Value : 0.5896
##
##             Sensitivity : 0.4118
##             Specificity : 0.7024
##          Pos Pred Value : 0.4565
##          Neg Pred Value : 0.6629
##              Prevalence : 0.3778
##          Detection Rate : 0.1556
##    Detection Prevalence : 0.3407
##       Balanced Accuracy : 0.5571
##
##        'Positive' Class : Loss
##
```

Here's the complete code to estimate the team's expected win percentage for their next game based on a player's average stats

Lebron Rf

```
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games LeBron Played
player_name <- "LeBron James"
lebron_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(lebron_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A"

# Ensure RESULT is a factor with two levels
lebron_data$RESULT <- ifelse(lebron_data$RESULT == "Win", 1, 0)
lebron_data$RESULT <- factor(lebron_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
lebron_data <- lebron_data %>% na.omit()

# Step 4: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(lebron_data$RESULT, p = 0.8, list = FALSE)
train_data <- lebron_data[train_index, ]
test_data <- lebron_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 15 11
##          1 36 73
##
##                Accuracy : 0.6519
##                  95% CI : (0.5651, 0.7317)
##     No Information Rate : 0.6222
##     P-Value [Acc > NIR] : 0.2687482
##
```

```
##                   Kappa : 0.1806
##
##  Mcnemar's Test P-Value : 0.0004639
##
##             Sensitivity : 0.2941
##             Specificity : 0.8690
##          Pos Pred Value : 0.5769
##          Neg Pred Value : 0.6697
##              Prevalence : 0.3778
##          Detection Rate : 0.1111
##    Detection Prevalence : 0.1926
##       Balanced Accuracy : 0.5816
##
##        'Positive' Class : 0
##
```

```
# Step 5: Predict Team's Expected Win Percentage for the Next Game
# Calculate player's average stats for use in next game prediction
player_avg_stats <- lebron_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

# Transpose the player's average stats and ensure it is a single row with multiple columns
next_game_data <- as.data.frame(t(player_avg_stats))  # Transpose data and convert to data frame
next_game_data <- t(next_game_data)  # Transpose again to get single row format

# Ensure column names are properly set for prediction
colnames(next_game_data) <- colnames(train_data)[-16]  # Align the column names for prediction

# Predict win probability for the next game
win_prob <- predict(rf_model, next_game_data, type = "prob")[,2]
print(win_prob)
```

```
## [1] 0.42
```

steph avg

```
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games Stephen Curry Played
player_name <- "Stephen Curry"
curry_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(curry_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A",

# Ensure RESULT is a factor with two levels
curry_data$RESULT <- ifelse(curry_data$RESULT == "Win", 1, 0)
curry_data$RESULT <- factor(curry_data$RESULT, levels = c(0, 1))
```

```r
# Step 3: Remove rows with missing values
curry_data <- curry_data %>% na.omit()

# Step 4: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(curry_data$RESULT, p = 0.8, list = FALSE)
train_data <- curry_data[train_index, ]
test_data <- curry_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  9  2
##          1 29 88
##
##                Accuracy : 0.7578
##                  95% CI : (0.6742, 0.8291)
##     No Information Rate : 0.7031
##     P-Value [Acc > NIR] : 0.1027
##
##                   Kappa : 0.2701
##
##  Mcnemar's Test P-Value : 3.016e-06
##
##             Sensitivity : 0.23684
##             Specificity : 0.97778
##          Pos Pred Value : 0.81818
##          Neg Pred Value : 0.75214
##              Prevalence : 0.29688
##          Detection Rate : 0.07031
##    Detection Prevalence : 0.08594
##       Balanced Accuracy : 0.60731
##
##        'Positive' Class : 0
##
```

```r
# Step 5: Predict Team's Expected Win Percentage for the Next Game
# Calculate player's average stats for use in next game prediction
player_avg_stats <- curry_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))
```

```r
# Transpose the player's average stats and ensure it is a single row with multiple columns
next_game_data <- as.data.frame(t(player_avg_stats))  # Transpose data and convert to data frame
next_game_data <- t(next_game_data)  # Transpose again to get single row format

# Ensure column names are properly set for prediction
colnames(next_game_data) <- colnames(train_data)[-16]  # Align the column names for prediction

# Predict win probability for the next game
win_prob <- predict(rf_model, next_game_data, type = "prob")[,2]
print(win_prob)
```

```
## [1] 0.662
```

steph high

```r
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games Stephen Curry Played
player_name <- "Stephen Curry"
curry_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(curry_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A",

# Ensure RESULT is a factor with two levels
curry_data$RESULT <- ifelse(curry_data$RESULT == "Win", 1, 0)
curry_data$RESULT <- factor(curry_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
curry_data <- curry_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
curry_stats_mean <- curry_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

curry_stats_sd <- curry_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one standard deviation above the mean
curry_stats_high <- curry_stats_mean + curry_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(curry_data$RESULT, p = 0.8, list = FALSE)
train_data <- curry_data[train_index, ]
test_data <- curry_data[-train_index, ]
```

```r
# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  9  2
##          1 29 88
##
##                Accuracy : 0.7578
##                  95% CI : (0.6742, 0.8291)
##     No Information Rate : 0.7031
##     P-Value [Acc > NIR] : 0.1027
##
##                   Kappa : 0.2701
##
##  Mcnemar's Test P-Value : 3.016e-06
##
##             Sensitivity : 0.23684
##             Specificity : 0.97778
##          Pos Pred Value : 0.81818
##          Neg Pred Value : 0.75214
##              Prevalence : 0.29688
##          Detection Rate : 0.07031
##    Detection Prevalence : 0.08594
##       Balanced Accuracy : 0.60731
##
##        'Positive' Class : 0
##
```

```r
# Step 6: Predict Team's Expected Win Percentage for the Next Game
# Transpose the player's high stats to ensure it is a single row with multiple columns
next_game_data <- as.data.frame(t(curry_stats_high))  # Transpose data and convert to data frame
next_game_data <- as.data.frame(t(next_game_data))  # Ensure it is a single row format

# Ensure column names are properly set for prediction
colnames(next_game_data) <- colnames(train_data)[-16]  # Align the column names for prediction

# Predict win probability for the next game
win_prob <- predict(rf_model, next_game_data, type = "prob")[,2]
print(win_prob)
```

```
## [1] 0.866
```

steph low

```r
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games Stephen Curry Played
player_name <- "Stephen Curry"
curry_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(curry_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A",

# Ensure RESULT is a factor with two levels
curry_data$RESULT <- ifelse(curry_data$RESULT == "Win", 1, 0)
curry_data$RESULT <- factor(curry_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
curry_data <- curry_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
curry_stats_mean <- curry_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

curry_stats_sd <- curry_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one standard deviation below the mean
curry_stats_low <- curry_stats_mean - curry_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(curry_data$RESULT, p = 0.8, list = FALSE)
train_data <- curry_data[train_index, ]
test_data <- curry_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  9  2
```

```
##          1 29 88
##
##                Accuracy : 0.7578
##                  95% CI : (0.6742, 0.8291)
##     No Information Rate : 0.7031
##     P-Value [Acc > NIR] : 0.1027
##
##                   Kappa : 0.2701
##
##  Mcnemar's Test P-Value : 3.016e-06
##
##             Sensitivity : 0.23684
##             Specificity : 0.97778
##          Pos Pred Value : 0.81818
##          Neg Pred Value : 0.75214
##              Prevalence : 0.29688
##          Detection Rate : 0.07031
##    Detection Prevalence : 0.08594
##       Balanced Accuracy : 0.60731
##
##        'Positive' Class : 0
##
```

```r
# Step 6: Predict Team's Expected Win Percentage for the Next Game
# Transpose the player's low stats to ensure it is a single row with multiple columns
next_game_data <- as.data.frame(t(curry_stats_low))  # Transpose data and convert to data frame
next_game_data <- as.data.frame(t(next_game_data))  # Ensure it is a single row format

# Ensure column names are properly set for prediction
colnames(next_game_data) <- colnames(train_data)[-16]  # Align the column names for prediction

# Predict win probability for the next game
win_prob <- predict(rf_model, next_game_data, type = "prob")[,2]
print(win_prob)
```

```
## [1] 0.65
```

All 5 Lebron

```r
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games LeBron James Played
player_name <- "LeBron James"
lebron_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(lebron_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A"
```

```r
# Ensure RESULT is a factor with two levels
lebron_data$RESULT <- ifelse(lebron_data$RESULT == "Win", 1, 0)
lebron_data$RESULT <- factor(lebron_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
lebron_data <- lebron_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
lebron_stats_mean <- lebron_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

lebron_stats_sd <- lebron_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one and two standard deviations above and below the mean
lebron_stats_high <- lebron_stats_mean + lebron_stats_sd
lebron_stats_very_high <- lebron_stats_mean + 2 * lebron_stats_sd
lebron_stats_low <- lebron_stats_mean - lebron_stats_sd
lebron_stats_very_low <- lebron_stats_mean - 2 * lebron_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(lebron_data$RESULT, p = 0.8, list = FALSE)
train_data <- lebron_data[train_index, ]
test_data <- lebron_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 15 11
##          1 36 73
##
##                Accuracy : 0.6519
##                  95% CI : (0.5651, 0.7317)
##     No Information Rate : 0.6222
##     P-Value [Acc > NIR] : 0.2687482
##
##                   Kappa : 0.1806
##
##  Mcnemar's Test P-Value : 0.0004639
##
```

```
##             Sensitivity : 0.2941
##             Specificity : 0.8690
##          Pos Pred Value : 0.5769
##          Neg Pred Value : 0.6697
##              Prevalence : 0.3778
##          Detection Rate : 0.1111
##    Detection Prevalence : 0.1926
##       Balanced Accuracy : 0.5816
##
##        'Positive' Class : 0
##
```

```
# Get model accuracy
accuracy <- conf_matrix$overall['Accuracy']

# Step 6: Predict Team's Expected Win Percentage for the Next Game

# 6.1 Using Very High Stats (two standard deviations above average)
next_game_data_very_high <- as.data.frame(t(lebron_stats_very_high))
next_game_data_very_high <- as.data.frame(t(next_game_data_very_high))
colnames(next_game_data_very_high) <- colnames(train_data)[-16]
win_prob_very_high <- predict(rf_model, next_game_data_very_high, type = "prob")[,2]

# 6.2 Using High Stats (one standard deviation above average)
next_game_data_high <- as.data.frame(t(lebron_stats_high))
next_game_data_high <- as.data.frame(t(next_game_data_high))
colnames(next_game_data_high) <- colnames(train_data)[-16]
win_prob_high <- predict(rf_model, next_game_data_high, type = "prob")[,2]

# 6.3 Using Average Stats
next_game_data_avg <- as.data.frame(t(lebron_stats_mean))
next_game_data_avg <- as.data.frame(t(next_game_data_avg))
colnames(next_game_data_avg) <- colnames(train_data)[-16]
win_prob_avg <- predict(rf_model, next_game_data_avg, type = "prob")[,2]

# 6.4 Using Low Stats (one standard deviation below average)
next_game_data_low <- as.data.frame(t(lebron_stats_low))
next_game_data_low <- as.data.frame(t(next_game_data_low))
colnames(next_game_data_low) <- colnames(train_data)[-16]
win_prob_low <- predict(rf_model, next_game_data_low, type = "prob")[,2]

# 6.5 Using Very Low Stats (two standard deviations below average)
next_game_data_very_low <- as.data.frame(t(lebron_stats_very_low))
next_game_data_very_low <- as.data.frame(t(next_game_data_very_low))
colnames(next_game_data_very_low) <- colnames(train_data)[-16]
win_prob_very_low <- predict(rf_model, next_game_data_very_low, type = "prob")[,2]

# Print combined output
print(paste("We can predict with", round(accuracy * 100, 2),
            "% accuracy that the win probability for the team when LeBron James has an excellent game i
            "%, a good game is:", round(win_prob_high * 100, 1),
            "%, an average game is:", round(win_prob_avg * 100, 2),
            "%, a bad game is:", round(win_prob_low * 100, 2),
            "%, and a terrible game is:", round(win_prob_very_low * 100, 2), "%"))
```

```
## [1] "We can predict with 65.19 % accuracy that the win probability for the team when LeBron James ha
```

Steph

```r
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games Stephen Curry Played
player_name <- "Stephen Curry"
curry_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(curry_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A",

# Ensure RESULT is a factor with two levels
curry_data$RESULT <- ifelse(curry_data$RESULT == "Win", 1, 0)
curry_data$RESULT <- factor(curry_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
curry_data <- curry_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
curry_stats_mean <- curry_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

curry_stats_sd <- curry_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one and two standard deviations above and below the mean
curry_stats_high <- curry_stats_mean + curry_stats_sd
curry_stats_very_high <- curry_stats_mean + 2 * curry_stats_sd
curry_stats_low <- curry_stats_mean - curry_stats_sd
curry_stats_very_low <- curry_stats_mean - 2 * curry_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(curry_data$RESULT, p = 0.8, list = FALSE)
train_data <- curry_data[train_index, ]
test_data <- curry_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  9  2
##          1 29 88
##
##                Accuracy : 0.7578
##                  95% CI : (0.6742, 0.8291)
##     No Information Rate : 0.7031
##     P-Value [Acc > NIR] : 0.1027
##
##                   Kappa : 0.2701
##
##  Mcnemar's Test P-Value : 3.016e-06
##
##             Sensitivity : 0.23684
##             Specificity : 0.97778
##          Pos Pred Value : 0.81818
##          Neg Pred Value : 0.75214
##              Prevalence : 0.29688
##          Detection Rate : 0.07031
##    Detection Prevalence : 0.08594
##       Balanced Accuracy : 0.60731
##
##        'Positive' Class : 0
##
```

```r
# Get model accuracy
accuracy <- conf_matrix$overall['Accuracy']

# Step 6: Predict Team's Expected Win Percentage for the Next Game

# 6.1 Using Very High Stats (two standard deviations above average)
next_game_data_very_high <- as.data.frame(t(curry_stats_very_high))
next_game_data_very_high <- as.data.frame(t(next_game_data_very_high))
colnames(next_game_data_very_high) <- colnames(train_data)[-16]
win_prob_very_high <- predict(rf_model, next_game_data_very_high, type = "prob")[,2]

# 6.2 Using High Stats (one standard deviation above average)
next_game_data_high <- as.data.frame(t(curry_stats_high))
next_game_data_high <- as.data.frame(t(next_game_data_high))
colnames(next_game_data_high) <- colnames(train_data)[-16]
win_prob_high <- predict(rf_model, next_game_data_high, type = "prob")[,2]

# 6.3 Using Average Stats
next_game_data_avg <- as.data.frame(t(curry_stats_mean))
next_game_data_avg <- as.data.frame(t(next_game_data_avg))
colnames(next_game_data_avg) <- colnames(train_data)[-16]
win_prob_avg <- predict(rf_model, next_game_data_avg, type = "prob")[,2]

# 6.4 Using Low Stats (one standard deviation below average)
next_game_data_low <- as.data.frame(t(curry_stats_low))
next_game_data_low <- as.data.frame(t(next_game_data_low))
```

```
colnames(next_game_data_low) <- colnames(train_data)[-16]
win_prob_low <- predict(rf_model, next_game_data_low, type = "prob")[,2]

# 6.5 Using Very Low Stats (two standard deviations below average)
next_game_data_very_low <- as.data.frame(t(curry_stats_very_low))
next_game_data_very_low <- as.data.frame(t(next_game_data_very_low))
colnames(next_game_data_very_low) <- colnames(train_data)[-16]
win_prob_very_low <- predict(rf_model, next_game_data_very_low, type = "prob")[,2]

# Print combined output
print(paste("We can predict with", round(accuracy * 100, 2),
            "% accuracy that the win probability for the team when Stephen Curry has an excellent game i
            "%, a good game is:", round(win_prob_high * 100, 1),
            "%, an average game is:", round(win_prob_avg * 100, 2),
            "%, a bad game is:", round(win_prob_low * 100, 2),
            "%, and a terrible game is:", round(win_prob_very_low * 100, 2), "%"))
```

```
## [1] "We can predict with 75.78 % accuracy that the win probability for the team when Stephen Curry ha
```

Klay

```
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games Klay Thompson Played
player_name <- "Klay Thompson"
klay_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(klay_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A",

# Ensure RESULT is a factor with two levels
klay_data$RESULT <- ifelse(klay_data$RESULT == "Win", 1, 0)
klay_data$RESULT <- factor(klay_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
klay_data <- klay_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
klay_stats_mean <- klay_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

klay_stats_sd <- klay_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one and two standard deviations above and below the mean
klay_stats_high <- klay_stats_mean + klay_stats_sd
klay_stats_very_high <- klay_stats_mean + 2 * klay_stats_sd
klay_stats_low <- klay_stats_mean - klay_stats_sd
```

```r
klay_stats_very_low <- klay_stats_mean - 2 * klay_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(klay_data$RESULT, p = 0.8, list = FALSE)
train_data <- klay_data[train_index, ]
test_data <- klay_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 10  6
##          1 24 77
##
##                Accuracy : 0.7436
##                  95% CI : (0.6546, 0.8198)
##     No Information Rate : 0.7094
##     P-Value [Acc > NIR] : 0.240288
##
##                   Kappa : 0.2629
##
##  Mcnemar's Test P-Value : 0.001911
##
##             Sensitivity : 0.29412
##             Specificity : 0.92771
##          Pos Pred Value : 0.62500
##          Neg Pred Value : 0.76238
##              Prevalence : 0.29060
##          Detection Rate : 0.08547
##    Detection Prevalence : 0.13675
##       Balanced Accuracy : 0.61091
##
##        'Positive' Class : 0
##
```

```r
# Get model accuracy
accuracy <- conf_matrix$overall['Accuracy']

# Step 6: Predict Team's Expected Win Percentage for the Next Game

# 6.1 Using Very High Stats (two standard deviations above average)
```

```r
next_game_data_very_high <- as.data.frame(t(klay_stats_very_high))
next_game_data_very_high <- as.data.frame(t(next_game_data_very_high))
colnames(next_game_data_very_high) <- colnames(train_data)[-16]
win_prob_very_high <- predict(rf_model, next_game_data_very_high, type = "prob")[,2]

# 6.2 Using High Stats (one standard deviation above average)
next_game_data_high <- as.data.frame(t(klay_stats_high))
next_game_data_high <- as.data.frame(t(next_game_data_high))
colnames(next_game_data_high) <- colnames(train_data)[-16]
win_prob_high <- predict(rf_model, next_game_data_high, type = "prob")[,2]

# 6.3 Using Average Stats
next_game_data_avg <- as.data.frame(t(klay_stats_mean))
next_game_data_avg <- as.data.frame(t(next_game_data_avg))
colnames(next_game_data_avg) <- colnames(train_data)[-16]
win_prob_avg <- predict(rf_model, next_game_data_avg, type = "prob")[,2]

# 6.4 Using Low Stats (one standard deviation below average)
next_game_data_low <- as.data.frame(t(klay_stats_low))
next_game_data_low <- as.data.frame(t(next_game_data_low))
colnames(next_game_data_low) <- colnames(train_data)[-16]
win_prob_low <- predict(rf_model, next_game_data_low, type = "prob")[,2]

# 6.5 Using Very Low Stats (two standard deviations below average)
next_game_data_very_low <- as.data.frame(t(klay_stats_very_low))
next_game_data_very_low <- as.data.frame(t(next_game_data_very_low))
colnames(next_game_data_very_low) <- colnames(train_data)[-16]
win_prob_very_low <- predict(rf_model, next_game_data_very_low, type = "prob")[,2]

# Print combined output
print(paste("We can predict with", round(accuracy * 100, 2),
            "% accuracy that the win probability for the team when Klay Thompson has an excellent game i
            "%, a good game is:", round(win_prob_high * 100, 1),
            "%, an average game is:", round(win_prob_avg * 100, 2),
            "%, a bad game is:", round(win_prob_low * 100, 2),
            "%, and a terrible game is:", round(win_prob_very_low * 100, 2), "%"))
```

```
## [1] "We can predict with 74.36 % accuracy that the win probability for the team when Klay Thompson ha
```

Draymond

```r
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games Draymond Green Played
player_name <- "Draymond Green"
green_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
```

```r
colnames(green_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A",

# Ensure RESULT is a factor with two levels
green_data$RESULT <- ifelse(green_data$RESULT == "Win", 1, 0)
green_data$RESULT <- factor(green_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
green_data <- green_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
green_stats_mean <- green_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

green_stats_sd <- green_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one and two standard deviations above and below the mean
green_stats_high <- green_stats_mean + green_stats_sd
green_stats_very_high <- green_stats_mean + 2 * green_stats_sd
green_stats_low <- green_stats_mean - green_stats_sd
green_stats_very_low <- green_stats_mean - 2 * green_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(green_data$RESULT, p = 0.8, list = FALSE)
train_data <- green_data[train_index, ]
test_data <- green_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  6 10
##          1 38 81
##
##                Accuracy : 0.6444
##                  95% CI : (0.5575, 0.7249)
##     No Information Rate : 0.6741
##     P-Value [Acc > NIR] : 0.7966
##
##                   Kappa : 0.0317
##
```

```
##  Mcnemar's Test P-Value : 9.735e-05
##
##             Sensitivity : 0.13636
##             Specificity : 0.89011
##          Pos Pred Value : 0.37500
##          Neg Pred Value : 0.68067
##              Prevalence : 0.32593
##          Detection Rate : 0.04444
##    Detection Prevalence : 0.11852
##       Balanced Accuracy : 0.51324
##
##        'Positive' Class : 0
##
```

```
# Get model accuracy
accuracy <- conf_matrix$overall['Accuracy']

# Step 6: Predict Team's Expected Win Percentage for the Next Game

# 6.1 Using Very High Stats (two standard deviations above average)
next_game_data_very_high <- as.data.frame(t(green_stats_very_high))
next_game_data_very_high <- as.data.frame(t(next_game_data_very_high))
colnames(next_game_data_very_high) <- colnames(train_data)[-16]
win_prob_very_high <- predict(rf_model, next_game_data_very_high, type = "prob")[,2]

# 6.2 Using High Stats (one standard deviation above average)
next_game_data_high <- as.data.frame(t(green_stats_high))
next_game_data_high <- as.data.frame(t(next_game_data_high))
colnames(next_game_data_high) <- colnames(train_data)[-16]
win_prob_high <- predict(rf_model, next_game_data_high, type = "prob")[,2]

# 6.3 Using Average Stats
next_game_data_avg <- as.data.frame(t(green_stats_mean))
next_game_data_avg <- as.data.frame(t(next_game_data_avg))
colnames(next_game_data_avg) <- colnames(train_data)[-16]
win_prob_avg <- predict(rf_model, next_game_data_avg, type = "prob")[,2]

# 6.4 Using Low Stats (one standard deviation below average)
next_game_data_low <- as.data.frame(t(green_stats_low))
next_game_data_low <- as.data.frame(t(next_game_data_low))
colnames(next_game_data_low) <- colnames(train_data)[-16]
win_prob_low <- predict(rf_model, next_game_data_low, type = "prob")[,2]

# 6.5 Using Very Low Stats (two standard deviations below average)
next_game_data_very_low <- as.data.frame(t(green_stats_very_low))
next_game_data_very_low <- as.data.frame(t(next_game_data_very_low))
colnames(next_game_data_very_low) <- colnames(train_data)[-16]
win_prob_very_low <- predict(rf_model, next_game_data_very_low, type = "prob")[,2]

# Print combined output
print(paste("We can predict with", round(accuracy * 100, 2),
            "% accuracy that the win probability for the team when Draymond Green has an excellent game
            "%, a good game is:", round(win_prob_high * 100, 1),
            "%, an average game is:", round(win_prob_avg * 100, 2),
```

```
            "%, a bad game is:", round(win_prob_low * 100, 2),
            "%, and a terrible game is:", round(win_prob_very_low * 100, 2), "%"))
```

```
## [1] "We can predict with 64.44 % accuracy that the win probability for the team when Draymond Green I
```

Players requested by the class Josh Hart

```
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games Josh Hart Played
player_name <- "Josh Hart"
hart_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(hart_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A",

# Ensure RESULT is a factor with two levels
hart_data$RESULT <- ifelse(hart_data$RESULT == "Win", 1, 0)
hart_data$RESULT <- factor(hart_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
hart_data <- hart_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
hart_stats_mean <- hart_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

hart_stats_sd <- hart_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one and two standard deviations above and below the mean
hart_stats_high <- hart_stats_mean + hart_stats_sd
hart_stats_very_high <- hart_stats_mean + 2 * hart_stats_sd
hart_stats_low <- hart_stats_mean - hart_stats_sd
hart_stats_very_low <- hart_stats_mean - 2 * hart_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(hart_data$RESULT, p = 0.8, list = FALSE)
train_data <- hart_data[train_index, ]
test_data <- hart_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)
```

```r
# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 16 13
##          1  8  7
##
##                Accuracy : 0.5227
##                  95% CI : (0.3669, 0.6754)
##     No Information Rate : 0.5455
##     P-Value [Acc > NIR] : 0.6762
##
##                   Kappa : 0.017
##
##  Mcnemar's Test P-Value : 0.3827
##
##             Sensitivity : 0.6667
##             Specificity : 0.3500
##          Pos Pred Value : 0.5517
##          Neg Pred Value : 0.4667
##              Prevalence : 0.5455
##          Detection Rate : 0.3636
##    Detection Prevalence : 0.6591
##       Balanced Accuracy : 0.5083
##
##        'Positive' Class : 0
##
```

```r
# Get model accuracy
accuracy <- conf_matrix$overall['Accuracy']

# Step 6: Predict Team's Expected Win Percentage for the Next Game

# 6.1 Using Very High Stats (two standard deviations above average)
next_game_data_very_high <- as.data.frame(t(hart_stats_very_high))
next_game_data_very_high <- as.data.frame(t(next_game_data_very_high))
colnames(next_game_data_very_high) <- colnames(train_data)[-16]
win_prob_very_high <- predict(rf_model, next_game_data_very_high, type = "prob")[,2]

# 6.2 Using High Stats (one standard deviation above average)
next_game_data_high <- as.data.frame(t(hart_stats_high))
next_game_data_high <- as.data.frame(t(next_game_data_high))
colnames(next_game_data_high) <- colnames(train_data)[-16]
win_prob_high <- predict(rf_model, next_game_data_high, type = "prob")[,2]

# 6.3 Using Average Stats
next_game_data_avg <- as.data.frame(t(hart_stats_mean))
next_game_data_avg <- as.data.frame(t(next_game_data_avg))
colnames(next_game_data_avg) <- colnames(train_data)[-16]
```

```r
win_prob_avg <- predict(rf_model, next_game_data_avg, type = "prob")[,2]

# 6.4 Using Low Stats (one standard deviation below average)
next_game_data_low <- as.data.frame(t(hart_stats_low))
next_game_data_low <- as.data.frame(t(next_game_data_low))
colnames(next_game_data_low) <- colnames(train_data)[-16]
win_prob_low <- predict(rf_model, next_game_data_low, type = "prob")[,2]

# 6.5 Using Very Low Stats (two standard deviations below average)
next_game_data_very_low <- as.data.frame(t(hart_stats_very_low))
next_game_data_very_low <- as.data.frame(t(next_game_data_very_low))
colnames(next_game_data_very_low) <- colnames(train_data)[-16]
win_prob_very_low <- predict(rf_model, next_game_data_very_low, type = "prob")[,2]

# Print combined output
print(paste("We can predict with", round(accuracy * 100, 2),
            "% accuracy that the win probability for the team when Josh Hart has an excellent game is:"
            "%, a good game is:", round(win_prob_high * 100, 1),
            "%, an average game is:", round(win_prob_avg * 100, 2),
            "%, a bad game is:", round(win_prob_low * 100, 2),
            "%, and a terrible game is:", round(win_prob_very_low * 100, 2), "%"))
```

```
## [1] "We can predict with 52.27 % accuracy that the win probability for the team when Josh Hart has ar
```

Tj Mcconnel

```r
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games T.J. McConnell Played
player_name <- "T.J. McConnell"
mcconnell_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(mcconnell_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT",

# Ensure RESULT is a factor with two levels
mcconnell_data$RESULT <- ifelse(mcconnell_data$RESULT == "Win", 1, 0)
mcconnell_data$RESULT <- factor(mcconnell_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
mcconnell_data <- mcconnell_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
mcconnell_stats_mean <- mcconnell_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

mcconnell_stats_sd <- mcconnell_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))
```

```r
# Calculate one and two standard deviations above and below the mean
mcconnell_stats_high <- mcconnell_stats_mean + mcconnell_stats_sd
mcconnell_stats_very_high <- mcconnell_stats_mean + 2 * mcconnell_stats_sd
mcconnell_stats_low <- mcconnell_stats_mean - mcconnell_stats_sd
mcconnell_stats_very_low <- mcconnell_stats_mean - 2 * mcconnell_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(mcconnell_data$RESULT, p = 0.8, list = FALSE)
train_data <- mcconnell_data[train_index, ]
test_data <- mcconnell_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 0 1
##          0 5 5
##          1 8 1
##
##                Accuracy : 0.3158
##                  95% CI : (0.1258, 0.5655)
##     No Information Rate : 0.6842
##     P-Value [Acc > NIR] : 0.9998
##
##                   Kappa : -0.3955
##
##  Mcnemar's Test P-Value : 0.5791
##
##             Sensitivity : 0.3846
##             Specificity : 0.1667
##          Pos Pred Value : 0.5000
##          Neg Pred Value : 0.1111
##              Prevalence : 0.6842
##          Detection Rate : 0.2632
##    Detection Prevalence : 0.5263
##       Balanced Accuracy : 0.2756
##
##        'Positive' Class : 0
##
```

```r
# Get model accuracy
accuracy <- conf_matrix$overall['Accuracy']

# Step 6: Predict Team's Expected Win Percentage for the Next Game

# 6.1 Using Very High Stats (two standard deviations above average)
next_game_data_very_high <- as.data.frame(t(mcconnell_stats_very_high))
next_game_data_very_high <- as.data.frame(t(next_game_data_very_high))
colnames(next_game_data_very_high) <- colnames(train_data)[-16]
win_prob_very_high <- predict(rf_model, next_game_data_very_high, type = "prob")[,2]

# 6.2 Using High Stats (one standard deviation above average)
next_game_data_high <- as.data.frame(t(mcconnell_stats_high))
next_game_data_high <- as.data.frame(t(next_game_data_high))
colnames(next_game_data_high) <- colnames(train_data)[-16]
win_prob_high <- predict(rf_model, next_game_data_high, type = "prob")[,2]

# 6.3 Using Average Stats
next_game_data_avg <- as.data.frame(t(mcconnell_stats_mean))
next_game_data_avg <- as.data.frame(t(next_game_data_avg))
colnames(next_game_data_avg) <- colnames(train_data)[-16]
win_prob_avg <- predict(rf_model, next_game_data_avg, type = "prob")[,2]

# 6.4 Using Low Stats (one standard deviation below average)
next_game_data_low <- as.data.frame(t(mcconnell_stats_low))
next_game_data_low <- as.data.frame(t(next_game_data_low))
colnames(next_game_data_low) <- colnames(train_data)[-16]
win_prob_low <- predict(rf_model, next_game_data_low, type = "prob")[,2]

# 6.5 Using Very Low Stats (two standard deviations below average)
next_game_data_very_low <- as.data.frame(t(mcconnell_stats_very_low))
next_game_data_very_low <- as.data.frame(t(next_game_data_very_low))
colnames(next_game_data_very_low) <- colnames(train_data)[-16]
win_prob_very_low <- predict(rf_model, next_game_data_very_low, type = "prob")[,2]

# Print combined output
print(paste("We can predict with", round(accuracy * 100, 2),
            "% accuracy that the win probability for the team when T.J. McConnell has an excellent game
            "%, a good game is:", round(win_prob_high * 100, 1),
            "%, an average game is:", round(win_prob_avg * 100, 2),
            "%, a bad game is:", round(win_prob_low * 100, 2),
            "%, and a terrible game is:", round(win_prob_very_low * 100, 2), "%"))
```

```
## [1] "We can predict with 31.58 % accuracy that the win probability for the team when T.J. McConnell l
```

Grayson Allen

```r
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games Grayson Allen Played
```

```r
player_name <- "Grayson Allen"
allen_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(allen_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A",

# Ensure RESULT is a factor with two levels
allen_data$RESULT <- ifelse(allen_data$RESULT == "Win", 1, 0)
allen_data$RESULT <- factor(allen_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
allen_data <- allen_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
allen_stats_mean <- allen_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

allen_stats_sd <- allen_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one and two standard deviations above and below the mean
allen_stats_high <- allen_stats_mean + allen_stats_sd
allen_stats_very_high <- allen_stats_mean + 2 * allen_stats_sd
allen_stats_low <- allen_stats_mean - allen_stats_sd
allen_stats_very_low <- allen_stats_mean - 2 * allen_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(allen_data$RESULT, p = 0.8, list = FALSE)
train_data <- allen_data[train_index, ]
test_data <- allen_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  3  5
##          1 16 26
##
##                Accuracy : 0.58
```

```
##                  95% CI : (0.4321, 0.7181)
##     No Information Rate : 0.62
##     P-Value [Acc > NIR] : 0.7683
##
##                   Kappa : -0.0038
##
##  Mcnemar's Test P-Value : 0.0291
##
##             Sensitivity : 0.1579
##             Specificity : 0.8387
##          Pos Pred Value : 0.3750
##          Neg Pred Value : 0.6190
##              Prevalence : 0.3800
##          Detection Rate : 0.0600
##    Detection Prevalence : 0.1600
##       Balanced Accuracy : 0.4983
##
##        'Positive' Class : 0
##
```

```r
# Get model accuracy
accuracy <- conf_matrix$overall['Accuracy']

# Step 6: Predict Team's Expected Win Percentage for the Next Game

# 6.1 Using Very High Stats (two standard deviations above average)
next_game_data_very_high <- as.data.frame(t(allen_stats_very_high))
next_game_data_very_high <- as.data.frame(t(next_game_data_very_high))
colnames(next_game_data_very_high) <- colnames(train_data)[-16]
win_prob_very_high <- predict(rf_model, next_game_data_very_high, type = "prob")[,2]

# 6.2 Using High Stats (one standard deviation above average)
next_game_data_high <- as.data.frame(t(allen_stats_high))
next_game_data_high <- as.data.frame(t(next_game_data_high))
colnames(next_game_data_high) <- colnames(train_data)[-16]
win_prob_high <- predict(rf_model, next_game_data_high, type = "prob")[,2]

# 6.3 Using Average Stats
next_game_data_avg <- as.data.frame(t(allen_stats_mean))
next_game_data_avg <- as.data.frame(t(next_game_data_avg))
colnames(next_game_data_avg) <- colnames(train_data)[-16]
win_prob_avg <- predict(rf_model, next_game_data_avg, type = "prob")[,2]

# 6.4 Using Low Stats (one standard deviation below average)
next_game_data_low <- as.data.frame(t(allen_stats_low))
next_game_data_low <- as.data.frame(t(next_game_data_low))
colnames(next_game_data_low) <- colnames(train_data)[-16]
win_prob_low <- predict(rf_model, next_game_data_low, type = "prob")[,2]

# 6.5 Using Very Low Stats (two standard deviations below average)
next_game_data_very_low <- as.data.frame(t(allen_stats_very_low))
next_game_data_very_low <- as.data.frame(t(next_game_data_very_low))
colnames(next_game_data_very_low) <- colnames(train_data)[-16]
win_prob_very_low <- predict(rf_model, next_game_data_very_low, type = "prob")[,2]
```

```r
# Print combined output
print(paste("We can predict with", round(accuracy * 100, 2),
            "% accuracy that the win probability for the team when Grayson Allen has an excellent game i
            "%, a good game is:", round(win_prob_high * 100, 1),
            "%, an average game is:", round(win_prob_avg * 100, 2),
            "%, a bad game is:", round(win_prob_low * 100, 2),
            "%, and a terrible game is:", round(win_prob_very_low * 100, 2), "%"))
```

```
## [1] "We can predict with 58 % accuracy that the win probability for the team when Grayson Allen has a
```

Wemby

```r
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games Victor Wembanyama Played
player_name <- "Victor Wembanyama"
wemby_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(wemby_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A",

# Ensure RESULT is a factor with two levels
wemby_data$RESULT <- ifelse(wemby_data$RESULT == "Win", 1, 0)
wemby_data$RESULT <- factor(wemby_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
wemby_data <- wemby_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
wemby_stats_mean <- wemby_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

wemby_stats_sd <- wemby_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one and two standard deviations above and below the mean
wemby_stats_high <- wemby_stats_mean + wemby_stats_sd
wemby_stats_very_high <- wemby_stats_mean + 2 * wemby_stats_sd
wemby_stats_low <- wemby_stats_mean - wemby_stats_sd
wemby_stats_very_low <- wemby_stats_mean - 2 * wemby_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(wemby_data$RESULT, p = 0.8, list = FALSE)
train_data <- wemby_data[train_index, ]
test_data <- wemby_data[-train_index, ]
```

```r
# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 10  2
##          1  0  1
##
##                Accuracy : 0.8462
##                  95% CI : (0.5455, 0.9808)
##     No Information Rate : 0.7692
##     P-Value [Acc > NIR] : 0.3936
##
##                   Kappa : 0.4348
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 1.0000
##             Specificity : 0.3333
##          Pos Pred Value : 0.8333
##          Neg Pred Value : 1.0000
##              Prevalence : 0.7692
##          Detection Rate : 0.7692
##    Detection Prevalence : 0.9231
##       Balanced Accuracy : 0.6667
##
##        'Positive' Class : 0
##
```

```r
# Get model accuracy
accuracy <- conf_matrix$overall['Accuracy']

# Step 6: Predict Team's Expected Win Percentage for the Next Game

# 6.1 Using Very High Stats (two standard deviations above average)
next_game_data_very_high <- as.data.frame(t(wemby_stats_very_high))
next_game_data_very_high <- as.data.frame(t(next_game_data_very_high))
colnames(next_game_data_very_high) <- colnames(train_data)[-16]
win_prob_very_high <- predict(rf_model, next_game_data_very_high, type = "prob")[,2]

# 6.2 Using High Stats (one standard deviation above average)
next_game_data_high <- as.data.frame(t(wemby_stats_high))
next_game_data_high <- as.data.frame(t(next_game_data_high))
colnames(next_game_data_high) <- colnames(train_data)[-16]
```

```
win_prob_high <- predict(rf_model, next_game_data_high, type = "prob")[,2]

# 6.3 Using Average Stats
next_game_data_avg <- as.data.frame(t(wemby_stats_mean))
next_game_data_avg <- as.data.frame(t(next_game_data_avg))
colnames(next_game_data_avg) <- colnames(train_data)[-16]
win_prob_avg <- predict(rf_model, next_game_data_avg, type = "prob")[,2]

# 6.4 Using Low Stats (one standard deviation below average)
next_game_data_low <- as.data.frame(t(wemby_stats_low))
next_game_data_low <- as.data.frame(t(next_game_data_low))
colnames(next_game_data_low) <- colnames(train_data)[-16]
win_prob_low <- predict(rf_model, next_game_data_low, type = "prob")[,2]

# 6.5 Using Very Low Stats (two standard deviations below average)
next_game_data_very_low <- as.data.frame(t(wemby_stats_very_low))
next_game_data_very_low <- as.data.frame(t(next_game_data_very_low))
colnames(next_game_data_very_low) <- colnames(train_data)[-16]
win_prob_very_low <- predict(rf_model, next_game_data_very_low, type = "prob")[,2]

# Print combined output
print(paste("We can predict with", round(accuracy * 100, 2),
            "% accuracy that the win probability for the team when Victor Wembanyama has an excellent ga
            "%, a good game is:", round(win_prob_high * 100, 1),
            "%, an average game is:", round(win_prob_avg * 100, 2),
            "%, a bad game is:", round(win_prob_low * 100, 2),
            "%, and a terrible game is:", round(win_prob_very_low * 100, 2), "%"))
```

```
## [1] "We can predict with 84.62 % accuracy that the win probability for the team when Victor Wembanyar
```

Luka

```
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games Luka Dončić Played
player_name <- "Luka Doncic"
doncic_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(doncic_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A"

# Ensure RESULT is a factor with two levels
doncic_data$RESULT <- ifelse(doncic_data$RESULT == "Win", 1, 0)
doncic_data$RESULT <- factor(doncic_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
doncic_data <- doncic_data %>% na.omit()
```

```r
# Step 4: Calculate Mean and Standard Deviation
doncic_stats_mean <- doncic_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

doncic_stats_sd <- doncic_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one and two standard deviations above and below the mean
doncic_stats_high <- doncic_stats_mean + doncic_stats_sd
doncic_stats_very_high <- doncic_stats_mean + 2 * doncic_stats_sd
doncic_stats_low <- doncic_stats_mean - doncic_stats_sd
doncic_stats_very_low <- doncic_stats_mean - 2 * doncic_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(doncic_data$RESULT, p = 0.8, list = FALSE)
train_data <- doncic_data[train_index, ]
test_data <- doncic_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 14 16
##          1 25 34
##
##                Accuracy : 0.5393
##                  95% CI : (0.4304, 0.6456)
##     No Information Rate : 0.5618
##     P-Value [Acc > NIR] : 0.7043
##
##                   Kappa : 0.04
##
##  Mcnemar's Test P-Value : 0.2115
##
##             Sensitivity : 0.3590
##             Specificity : 0.6800
##          Pos Pred Value : 0.4667
##          Neg Pred Value : 0.5763
##              Prevalence : 0.4382
##          Detection Rate : 0.1573
##    Detection Prevalence : 0.3371
```

```
##       Balanced Accuracy : 0.5195
##
##        'Positive' Class : 0
##
```

```r
# Get model accuracy
accuracy <- conf_matrix$overall['Accuracy']

# Step 6: Predict Team's Expected Win Percentage for the Next Game

# 6.1 Using Very High Stats (two standard deviations above average)
next_game_data_very_high <- as.data.frame(t(doncic_stats_very_high))
next_game_data_very_high <- as.data.frame(t(next_game_data_very_high))
colnames(next_game_data_very_high) <- colnames(train_data)[-16]
win_prob_very_high <- predict(rf_model, next_game_data_very_high, type = "prob")[,2]

# 6.2 Using High Stats (one standard deviation above average)
next_game_data_high <- as.data.frame(t(doncic_stats_high))
next_game_data_high <- as.data.frame(t(next_game_data_high))
colnames(next_game_data_high) <- colnames(train_data)[-16]
win_prob_high <- predict(rf_model, next_game_data_high, type = "prob")[,2]

# 6.3 Using Average Stats
next_game_data_avg <- as.data.frame(t(doncic_stats_mean))
next_game_data_avg <- as.data.frame(t(next_game_data_avg))
colnames(next_game_data_avg) <- colnames(train_data)[-16]
win_prob_avg <- predict(rf_model, next_game_data_avg, type = "prob")[,2]

# 6.4 Using Low Stats (one standard deviation below average)
next_game_data_low <- as.data.frame(t(doncic_stats_low))
next_game_data_low <- as.data.frame(t(next_game_data_low))
colnames(next_game_data_low) <- colnames(train_data)[-16]
win_prob_low <- predict(rf_model, next_game_data_low, type = "prob")[,2]

# 6.5 Using Very Low Stats (two standard deviations below average)
next_game_data_very_low <- as.data.frame(t(doncic_stats_very_low))
next_game_data_very_low <- as.data.frame(t(next_game_data_very_low))
colnames(next_game_data_very_low) <- colnames(train_data)[-16]
win_prob_very_low <- predict(rf_model, next_game_data_very_low, type = "prob")[,2]

# Print combined output
print(paste("We can predict with", round(accuracy * 100, 2),
            "% accuracy that the win probability for the team when Luka Dončić has an excellent game is
            "%, a good game is:", round(win_prob_high * 100, 1),
            "%, an average game is:", round(win_prob_avg * 100, 2),
            "%, a bad game is:", round(win_prob_low * 100, 2),
            "%, and a terrible game is:", round(win_prob_very_low * 100, 2), "%"))
```

```
## [1] "We can predict with 53.93 % accuracy that the win probability for the team when Luka Dončić has
```

John Wall

```r
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games John Wall Played
player_name <- "John Wall"
wall_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(wall_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A",

# Ensure RESULT is a factor with two levels
wall_data$RESULT <- ifelse(wall_data$RESULT == "Win", 1, 0)
wall_data$RESULT <- factor(wall_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
wall_data <- wall_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
wall_stats_mean <- wall_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

wall_stats_sd <- wall_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one and two standard deviations above and below the mean
wall_stats_high <- wall_stats_mean + wall_stats_sd
wall_stats_very_high <- wall_stats_mean + 2 * wall_stats_sd
wall_stats_low <- wall_stats_mean - wall_stats_sd
wall_stats_very_low <- wall_stats_mean - 2 * wall_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(wall_data$RESULT, p = 0.8, list = FALSE)
train_data <- wall_data[train_index, ]
test_data <- wall_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction  0  1
##          0 16 14
##          1 13 14
##
##                Accuracy : 0.5263
##                  95% CI : (0.3897, 0.6602)
##     No Information Rate : 0.5088
##     P-Value [Acc > NIR] : 0.4477
##
##                   Kappa : 0.0518
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.5517
##             Specificity : 0.5000
##          Pos Pred Value : 0.5333
##          Neg Pred Value : 0.5185
##              Prevalence : 0.5088
##          Detection Rate : 0.2807
##    Detection Prevalence : 0.5263
##       Balanced Accuracy : 0.5259
##
##        'Positive' Class : 0
##
```

```r
# Get model accuracy
accuracy <- conf_matrix$overall['Accuracy']

# Step 6: Predict Team's Expected Win Percentage for the Next Game

# 6.1 Using Very High Stats (two standard deviations above average)
next_game_data_very_high <- as.data.frame(t(wall_stats_very_high))
next_game_data_very_high <- as.data.frame(t(next_game_data_very_high))
colnames(next_game_data_very_high) <- colnames(train_data)[-16]
win_prob_very_high <- predict(rf_model, next_game_data_very_high, type = "prob")[,2]

# 6.2 Using High Stats (one standard deviation above average)
next_game_data_high <- as.data.frame(t(wall_stats_high))
next_game_data_high <- as.data.frame(t(next_game_data_high))
colnames(next_game_data_high) <- colnames(train_data)[-16]
win_prob_high <- predict(rf_model, next_game_data_high, type = "prob")[,2]

# 6.3 Using Average Stats
next_game_data_avg <- as.data.frame(t(wall_stats_mean))
next_game_data_avg <- as.data.frame(t(next_game_data_avg))
colnames(next_game_data_avg) <- colnames(train_data)[-16]
win_prob_avg <- predict(rf_model, next_game_data_avg, type = "prob")[,2]

# 6.4 Using Low Stats (one standard deviation below average)
next_game_data_low <- as.data.frame(t(wall_stats_low))
next_game_data_low <- as.data.frame(t(next_game_data_low))
colnames(next_game_data_low) <- colnames(train_data)[-16]
win_prob_low <- predict(rf_model, next_game_data_low, type = "prob")[,2]
```

```r
# 6.5 Using Very Low Stats (two standard deviations below average)
next_game_data_very_low <- as.data.frame(t(wall_stats_very_low))
next_game_data_very_low <- as.data.frame(t(next_game_data_very_low))
colnames(next_game_data_very_low) <- colnames(train_data)[-16]
win_prob_very_low <- predict(rf_model, next_game_data_very_low, type = "prob")[,2]

# Print combined output
print(paste("We can predict with", round(accuracy * 100, 2),
            "% accuracy that the win probability for the team when John Wall has an excellent game is:"
            "%, a good game is:", round(win_prob_high * 100, 1),
            "%, an average game is:", round(win_prob_avg * 100, 2),
            "%, a bad game is:", round(win_prob_low * 100, 2),
            "%, and a terrible game is:", round(win_prob_very_low * 100, 2), "%"))
```

```
## [1] "We can predict with 52.63 % accuracy that the win probability for the team when John Wall has ar
```

Mo Bamba

```r
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games Mo Bamba Played
player_name <- "Mo Bamba"
bamba_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(bamba_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A",

# Ensure RESULT is a factor with two levels
bamba_data$RESULT <- ifelse(bamba_data$RESULT == "Win", 1, 0)
bamba_data$RESULT <- factor(bamba_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
bamba_data <- bamba_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
bamba_stats_mean <- bamba_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

bamba_stats_sd <- bamba_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one and two standard deviations above and below the mean
bamba_stats_high <- bamba_stats_mean + bamba_stats_sd
bamba_stats_very_high <- bamba_stats_mean + 2 * bamba_stats_sd
bamba_stats_low <- bamba_stats_mean - bamba_stats_sd
bamba_stats_very_low <- bamba_stats_mean - 2 * bamba_stats_sd
```

```r
# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(bamba_data$RESULT, p = 0.8, list = FALSE)
train_data <- bamba_data[train_index, ]
test_data <- bamba_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 12  5
##          1  1  0
##
##                Accuracy : 0.6667
##                  95% CI : (0.4099, 0.8666)
##     No Information Rate : 0.7222
##     P-Value [Acc > NIR] : 0.7893
##
##                   Kappa : -0.102
##
##  Mcnemar's Test P-Value : 0.2207
##
##             Sensitivity : 0.9231
##             Specificity : 0.0000
##          Pos Pred Value : 0.7059
##          Neg Pred Value : 0.0000
##              Prevalence : 0.7222
##          Detection Rate : 0.6667
##    Detection Prevalence : 0.9444
##       Balanced Accuracy : 0.4615
##
##        'Positive' Class : 0
##
```

```r
# Get model accuracy
accuracy <- conf_matrix$overall['Accuracy']

# Step 6: Predict Team's Expected Win Percentage for the Next Game

# 6.1 Using Very High Stats (two standard deviations above average)
next_game_data_very_high <- as.data.frame(t(bamba_stats_very_high))
next_game_data_very_high <- as.data.frame(t(next_game_data_very_high))
```

```r
colnames(next_game_data_very_high) <- colnames(train_data)[-16]
win_prob_very_high <- predict(rf_model, next_game_data_very_high, type = "prob")[,2]

# 6.2 Using High Stats (one standard deviation above average)
next_game_data_high <- as.data.frame(t(bamba_stats_high))
next_game_data_high <- as.data.frame(t(next_game_data_high))
colnames(next_game_data_high) <- colnames(train_data)[-16]
win_prob_high <- predict(rf_model, next_game_data_high, type = "prob")[,2]

# 6.3 Using Average Stats
next_game_data_avg <- as.data.frame(t(bamba_stats_mean))
next_game_data_avg <- as.data.frame(t(next_game_data_avg))
colnames(next_game_data_avg) <- colnames(train_data)[-16]
win_prob_avg <- predict(rf_model, next_game_data_avg, type = "prob")[,2]

# 6.4 Using Low Stats (one standard deviation below average)
next_game_data_low <- as.data.frame(t(bamba_stats_low))
next_game_data_low <- as.data.frame(t(next_game_data_low))
colnames(next_game_data_low) <- colnames(train_data)[-16]
win_prob_low <- predict(rf_model, next_game_data_low, type = "prob")[,2]

# 6.5 Using Very Low Stats (two standard deviations below average)
next_game_data_very_low <- as.data.frame(t(bamba_stats_very_low))
next_game_data_very_low <- as.data.frame(t(next_game_data_very_low))
colnames(next_game_data_very_low) <- colnames(train_data)[-16]
win_prob_very_low <- predict(rf_model, next_game_data_very_low, type = "prob")[,2]

# Print combined output
print(paste("We can predict with", round(accuracy * 100, 2),
            "% accuracy that the win probability for the team when Mo Bamba has an excellent game is:",
            "%, a good game is:", round(win_prob_high * 100, 1),
            "%, an average game is:", round(win_prob_avg * 100, 2),
            "%, a bad game is:", round(win_prob_low * 100, 2),
            "%, and a terrible game is:", round(win_prob_very_low * 100, 2), "%"))
```

```
## [1] "We can predict with 66.67 % accuracy that the win probability for the team when Mo Bamba has an
```

Tyler Herro

```r
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games Tyler Herro Played
player_name <- "Tyler Herro"
herro_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(herro_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A",
```

```r
# Ensure RESULT is a factor with two levels
herro_data$RESULT <- ifelse(herro_data$RESULT == "Win", 1, 0)
herro_data$RESULT <- factor(herro_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
herro_data <- herro_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
herro_stats_mean <- herro_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

herro_stats_sd <- herro_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one and two standard deviations above and below the mean
herro_stats_high <- herro_stats_mean + herro_stats_sd
herro_stats_very_high <- herro_stats_mean + 2 * herro_stats_sd
herro_stats_low <- herro_stats_mean - herro_stats_sd
herro_stats_very_low <- herro_stats_mean - 2 * herro_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(herro_data$RESULT, p = 0.8, list = FALSE)
train_data <- herro_data[train_index, ]
test_data <- herro_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 10  7
##          1  5  8
##
##                Accuracy : 0.6
##                  95% CI : (0.406, 0.7734)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.1808
##
##                   Kappa : 0.2
##
##  Mcnemar's Test P-Value : 0.7728
##
```

```
##             Sensitivity : 0.6667
##             Specificity : 0.5333
##          Pos Pred Value : 0.5882
##          Neg Pred Value : 0.6154
##              Prevalence : 0.5000
##          Detection Rate : 0.3333
##    Detection Prevalence : 0.5667
##       Balanced Accuracy : 0.6000
##
##        'Positive' Class : 0
##
```

```
# Get model accuracy
accuracy <- conf_matrix$overall['Accuracy']

# Step 6: Predict Team's Expected Win Percentage for the Next Game

# 6.1 Using Very High Stats (two standard deviations above average)
next_game_data_very_high <- as.data.frame(t(herro_stats_very_high))
next_game_data_very_high <- as.data.frame(t(next_game_data_very_high))
colnames(next_game_data_very_high) <- colnames(train_data)[-16]
win_prob_very_high <- predict(rf_model, next_game_data_very_high, type = "prob")[,2]

# 6.2 Using High Stats (one standard deviation above average)
next_game_data_high <- as.data.frame(t(herro_stats_high))
next_game_data_high <- as.data.frame(t(next_game_data_high))
colnames(next_game_data_high) <- colnames(train_data)[-16]
win_prob_high <- predict(rf_model, next_game_data_high, type = "prob")[,2]

# 6.3 Using Average Stats
next_game_data_avg <- as.data.frame(t(herro_stats_mean))
next_game_data_avg <- as.data.frame(t(next_game_data_avg))
colnames(next_game_data_avg) <- colnames(train_data)[-16]
win_prob_avg <- predict(rf_model, next_game_data_avg, type = "prob")[,2]

# 6.4 Using Low Stats (one standard deviation below average)
next_game_data_low <- as.data.frame(t(herro_stats_low))
next_game_data_low <- as.data.frame(t(next_game_data_low))
colnames(next_game_data_low) <- colnames(train_data)[-16]
win_prob_low <- predict(rf_model, next_game_data_low, type = "prob")[,2]

# 6.5 Using Very Low Stats (two standard deviations below average)
next_game_data_very_low <- as.data.frame(t(herro_stats_very_low))
next_game_data_very_low <- as.data.frame(t(next_game_data_very_low))
colnames(next_game_data_very_low) <- colnames(train_data)[-16]
win_prob_very_low <- predict(rf_model, next_game_data_very_low, type = "prob")[,2]

# Print combined output
print(paste("We can predict with", round(accuracy * 100, 2),
            "% accuracy that the win probability for the team when Tyler Herro has an excellent game is
            "%, a good game is:", round(win_prob_high * 100, 1),
            "%, an average game is:", round(win_prob_avg * 100, 2),
            "%, a bad game is:", round(win_prob_low * 100, 2),
            "%, and a terrible game is:", round(win_prob_very_low * 100, 2), "%"))
```

```
## [1] "We can predict with 60 % accuracy that the win probability for the team when Tyler Herro has an
```

Joe Ingles

```
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games Joe Ingles Played
player_name <- "Joe Ingles"
ingles_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(ingles_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT", "A"

# Ensure RESULT is a factor with two levels
ingles_data$RESULT <- ifelse(ingles_data$RESULT == "Win", 1, 0)
ingles_data$RESULT <- factor(ingles_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
ingles_data <- ingles_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
ingles_stats_mean <- ingles_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

ingles_stats_sd <- ingles_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one and two standard deviations above and below the mean
ingles_stats_high <- ingles_stats_mean + ingles_stats_sd
ingles_stats_very_high <- ingles_stats_mean + 2 * ingles_stats_sd
ingles_stats_low <- ingles_stats_mean - ingles_stats_sd
ingles_stats_very_low <- ingles_stats_mean - 2 * ingles_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(ingles_data$RESULT, p = 0.8, list = FALSE)
train_data <- ingles_data[train_index, ]
test_data <- ingles_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 12 11
##          1 13 22
##
##                Accuracy : 0.5862
##                  95% CI : (0.4493, 0.714)
##     No Information Rate : 0.569
##     P-Value [Acc > NIR] : 0.4497
##
##                   Kappa : 0.1481
##
##  Mcnemar's Test P-Value : 0.8383
##
##             Sensitivity : 0.4800
##             Specificity : 0.6667
##          Pos Pred Value : 0.5217
##          Neg Pred Value : 0.6286
##              Prevalence : 0.4310
##          Detection Rate : 0.2069
##    Detection Prevalence : 0.3966
##       Balanced Accuracy : 0.5733
##
##        'Positive' Class : 0
##
```

```r
# Get model accuracy
accuracy <- conf_matrix$overall['Accuracy']

# Step 6: Predict Team's Expected Win Percentage for the Next Game

# 6.1 Using Very High Stats (two standard deviations above average)
next_game_data_very_high <- as.data.frame(t(ingles_stats_very_high))
next_game_data_very_high <- as.data.frame(t(next_game_data_very_high))
colnames(next_game_data_very_high) <- colnames(train_data)[-16]
win_prob_very_high <- predict(rf_model, next_game_data_very_high, type = "prob")[,2]

# 6.2 Using High Stats (one standard deviation above average)
next_game_data_high <- as.data.frame(t(ingles_stats_high))
next_game_data_high <- as.data.frame(t(next_game_data_high))
colnames(next_game_data_high) <- colnames(train_data)[-16]
win_prob_high <- predict(rf_model, next_game_data_high, type = "prob")[,2]

# 6.3 Using Average Stats
next_game_data_avg <- as.data.frame(t(ingles_stats_mean))
next_game_data_avg <- as.data.frame(t(next_game_data_avg))
colnames(next_game_data_avg) <- colnames(train_data)[-16]
win_prob_avg <- predict(rf_model, next_game_data_avg, type = "prob")[,2]

# 6.4 Using Low Stats (one standard deviation below average)
next_game_data_low <- as.data.frame(t(ingles_stats_low))
next_game_data_low <- as.data.frame(t(next_game_data_low))
```

```r
colnames(next_game_data_low) <- colnames(train_data)[-16]
win_prob_low <- predict(rf_model, next_game_data_low, type = "prob")[,2]

# 6.5 Using Very Low Stats (two standard deviations below average)
next_game_data_very_low <- as.data.frame(t(ingles_stats_very_low))
next_game_data_very_low <- as.data.frame(t(next_game_data_very_low))
colnames(next_game_data_very_low) <- colnames(train_data)[-16]
win_prob_very_low <- predict(rf_model, next_game_data_very_low, type = "prob")[,2]

# Print combined output
print(paste("We can predict with", round(accuracy * 100, 2),
            "% accuracy that the win probability for the team when Joe Ingles has an excellent game is:
            "%, a good game is:", round(win_prob_high * 100, 1),
            "%, an average game is:", round(win_prob_avg * 100, 2),
            "%, a bad game is:", round(win_prob_low * 100, 2),
            "%, and a terrible game is:", round(win_prob_very_low * 100, 2), "%"))
```

```
## [1] "We can predict with 58.62 % accuracy that the win probability for the team when Joe Ingles has a
```

Pat Connaughton

```r
# Load necessary packages
library(dplyr)
library(caret)
library(randomForest)

# Step 1: Filter Data for Games Pat Connaughton Played
player_name <- "Pat Connaughton"
connaughton_data <- Data %>%
  filter(Player == player_name) %>%
  select(PTS, FG, FGA, `3P`, `3PA`, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL, RESULT)

# Step 2: Rename columns to avoid special characters
colnames(connaughton_data) <- c("PTS", "FG", "FGA", "ThreeP", "ThreePA", "FT", "FTA", "OR", "DR", "TOT"

# Ensure RESULT is a factor with two levels
connaughton_data$RESULT <- ifelse(connaughton_data$RESULT == "Win", 1, 0)
connaughton_data$RESULT <- factor(connaughton_data$RESULT, levels = c(0, 1))

# Step 3: Remove rows with missing values
connaughton_data <- connaughton_data %>% na.omit()

# Step 4: Calculate Mean and Standard Deviation
connaughton_stats_mean <- connaughton_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), mean))

connaughton_stats_sd <- connaughton_data %>%
  summarise(across(c(PTS, FG, FGA, ThreeP, ThreePA, FT, FTA, OR, DR, TOT, A, PF, ST, TO, BL), sd))

# Calculate one and two standard deviations above and below the mean
connaughton_stats_high <- connaughton_stats_mean + connaughton_stats_sd
connaughton_stats_very_high <- connaughton_stats_mean + 2 * connaughton_stats_sd
connaughton_stats_low <- connaughton_stats_mean - connaughton_stats_sd
```

```r
connaughton_stats_very_low <- connaughton_stats_mean - 2 * connaughton_stats_sd

# Step 5: Train the Classification Model
# Split data into training and test sets
set.seed(123)
train_index <- createDataPartition(connaughton_data$RESULT, p = 0.8, list = FALSE)
train_data <- connaughton_data[train_index, ]
test_data <- connaughton_data[-train_index, ]

# Train Random Forest model for classification
rf_model <- randomForest(RESULT ~ ., data = train_data, ntree = 500, mtry = 3, importance = TRUE)

# Make predictions on test data
rf_pred <- predict(rf_model, test_data)

# Evaluate model performance
conf_matrix <- confusionMatrix(rf_pred, test_data$RESULT)
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 0 1
##          0 1 2
##          1 4 6
##
##                Accuracy : 0.5385
##                  95% CI : (0.2513, 0.8078)
##     No Information Rate : 0.6154
##     P-Value [Acc > NIR] : 0.8051
##
##                   Kappa : -0.0541
##
##  Mcnemar's Test P-Value : 0.6831
##
##             Sensitivity : 0.20000
##             Specificity : 0.75000
##          Pos Pred Value : 0.33333
##          Neg Pred Value : 0.60000
##              Prevalence : 0.38462
##          Detection Rate : 0.07692
##    Detection Prevalence : 0.23077
##       Balanced Accuracy : 0.47500
##
##        'Positive' Class : 0
##
```

```r
# Get model accuracy
accuracy <- conf_matrix$overall['Accuracy']

# Step 6: Predict Team's Expected Win Percentage for the Next Game

# 6.1 Using Very High Stats (two standard deviations above average)
```

```r
next_game_data_very_high <- as.data.frame(t(connaughton_stats_very_high))
next_game_data_very_high <- as.data.frame(t(next_game_data_very_high))
colnames(next_game_data_very_high) <- colnames(train_data)[-16]
win_prob_very_high <- predict(rf_model, next_game_data_very_high, type = "prob")[,2]

# 6.2 Using High Stats (one standard deviation above average)
next_game_data_high <- as.data.frame(t(connaughton_stats_high))
next_game_data_high <- as.data.frame(t(next_game_data_high))
colnames(next_game_data_high) <- colnames(train_data)[-16]
win_prob_high <- predict(rf_model, next_game_data_high, type = "prob")[,2]

# 6.3 Using Average Stats
next_game_data_avg <- as.data.frame(t(connaughton_stats_mean))
next_game_data_avg <- as.data.frame(t(next_game_data_avg))
colnames(next_game_data_avg) <- colnames(train_data)[-16]
win_prob_avg <- predict(rf_model, next_game_data_avg, type = "prob")[,2]

# 6.4 Using Low Stats (one standard deviation below average)
next_game_data_low <- as.data.frame(t(connaughton_stats_low))
next_game_data_low <- as.data.frame(t(next_game_data_low))
colnames(next_game_data_low) <- colnames(train_data)[-16]
win_prob_low <- predict(rf_model, next_game_data_low, type = "prob")[,2]

# 6.5 Using Very Low Stats (two standard deviations below average)
next_game_data_very_low <- as.data.frame(t(connaughton_stats_very_low))
next_game_data_very_low <- as.data.frame(t(next_game_data_very_low))
colnames(next_game_data_very_low) <- colnames(train_data)[-16]
win_prob_very_low <- predict(rf_model, next_game_data_very_low, type = "prob")[,2]

# Print combined output
print(paste("We can predict with", round(accuracy * 100, 2),
            "% accuracy that the win probability for the team when Pat Connaughton has an excellent game
            "%, a good game is:", round(win_prob_high * 100, 1),
            "%, an average game is:", round(win_prob_avg * 100, 2),
            "%, a bad game is:", round(win_prob_low * 100, 2),
            "%, and a terrible game is:", round(win_prob_very_low * 100, 2), "%"))
```

```
## [1] "We can predict with 53.85 % accuracy that the win probability for the team when Pat Connaughton
```

‘