Monte Carlo Simulation and Betting System for Premier League Predictions

Shubham Sharma - 22201541; VishalKrishna Bhosale - 22205276

2023-08-07

Loading the necessary libraries

```
library(vegan)  # Used for conducting Procrustes Analysis on datasets
library(stats)  # Used for performing Factor Analysis on data
library(MASS)   # Used for Multidimensional Scaling analysis
library(ggplot2)  # Used for creating visualizations with ggplot2
library(dplyr)    # Used for data manipulation with dplyr
library(reshape2)  # Used for data reshaping with reshape2
source("functions.r")  # Loading custom functions from the "functions.r" script
```

Loading the data. Ensure that the dataset csv file is in the same folder as this markdown file. To see the EDA of the data, please load the python script from github.

```
set.seed(123) # to get reproducible results
data = read.csv("E0.csv") # reading data
head(data) # glancing at the data
```

```
##   Div       Date  Time       HomeTeam      AwayTeam FTHG FTAG FTR HTHG HTAG HTR
## 1  E0 05/08/2022 20:00 Crystal Palace       Arsenal    0    2   A    0    1   A
## 2  E0 06/08/2022 12:30         Fulham     Liverpool    2    2   D    1    0   H
## 3  E0 06/08/2022 15:00    Bournemouth   Aston Villa    2    0   H    1    0   H
## 4  E0 06/08/2022 15:00          Leeds        Wolves    2    1   H    1    1   D
## 5  E0 06/08/2022 15:00      Newcastle Nott'm Forest    2    0   H    0    0   D
## 6  E0 06/08/2022 15:00      Tottenham   Southampton    4    1   H    2    1   H
##      Referee HS AS HST AST HF AF HC AC HY AY HR AR B365H B365D B365A   BWH  BWD
## 1   A Taylor 10 10   2   2 16 11  3  5  1  2  0  0  4.20   3.6  1.85  4.33 3.50
## 2   A Madley  9 11   3   4  7  9  4  4  2  0  0  0 11.00   6.0  1.25 10.00 5.75
## 3   P Bankes  7 15   3   2 18 16  5  5  3  3  0  0  3.75   3.5  2.00  3.75 3.40
## 4    R Jones 12 15   4   6 13  9  6  4  2  0  0  0  2.25   3.4  3.20  2.30 3.30
## 5   S Hooper 23  5  10   0  9 14 11  1  0  3  0  0  1.66   3.8  5.25  1.65 3.80
## 6 A Marriner 18 10   8   2 11  6 10  2  3  0  0  0  1.33   5.5  8.50  1.35 5.25
##    BWA   IWH  IWD  IWA   PSH  PSD  PSA   WHH WHD  WHA   VCH VCD  VCA  MaxH MaxD
## 1 1.87  4.30 3.55 1.85  4.50 3.65 1.89  4.40 3.5 1.83  4.60 3.5 1.87  4.60 3.78
## 2 1.28 12.00 5.75 1.27 11.20 6.22 1.28 12.00 5.5 1.27 13.00 6.0 1.25 13.00 6.40
## 3 2.00  3.65 3.45 2.05  3.93 3.58 2.04  3.75 3.3 2.05  3.75 3.3 2.00  4.00 3.66
## 4 2.95  2.30 3.30 3.15  2.39 3.33 3.30  2.25 3.3 3.20  2.30 3.2 3.10  2.42 3.54
## 5 5.50  1.65 3.80 5.50  1.71 3.74 5.83  1.67 3.7 5.25  1.62 3.7 5.50  1.72 3.96
## 6 8.25  1.37 5.25 7.75  1.37 5.39 9.11  1.35 5.0 8.50  1.33 5.0 9.00  1.40 5.50
##   MaxA  AvgH AvgD AvgA B365.2.5 B365.2.5.1 P.2.5 P.2.5.1 Max.2.5 Max.2.5.1
## 1 1.95  4.39 3.59 1.88     2.10       1.72  2.14    1.78    2.19      1.91
## 2 1.31 10.99 6.05 1.28     1.50       2.62  1.50    2.70    1.54      2.76
## 3 2.10  3.80 3.50 2.04     2.00       1.80  2.10    1.81    2.10      1.87
## 4 3.30  2.34 3.34 3.18     2.05       1.85  2.09    1.83    2.11      1.87
## 5 6.00  1.67 3.80 5.57     2.05       1.85  2.10    1.81    2.10      1.86
## 6 9.20  1.36 5.27 8.64     1.61       2.30  1.65    2.37    1.65      2.48
##   Avg.2.5 Avg.2.5.1   AHh B365AHH B365AHA PAHH PAHA MaxAHH MaxAHA AvgAHH AvgAHA
## 1    2.09      1.76  0.50    2.04    1.89 2.03 1.89   2.06   1.91   2.01   1.87
## 2    1.48      2.63  1.75    1.90    2.03 1.91 2.00   1.92   2.04   1.89   1.99
## 3    2.03      1.80  0.50    1.87    2.06 1.88 2.04   1.88   2.07   1.85   2.04
## 4    2.03      1.81 -0.25    2.05    1.88 2.04 1.89   2.06   1.90   2.01   1.87
## 5    2.03      1.81 -0.75    1.87    2.06 1.92 2.01   1.92   2.08   1.86   2.02
## 6    1.61      2.34 -1.50    2.04    1.89 2.08 1.85   2.08   1.91   2.03   1.85
##   B365CH B365CD B365CA BWCH BWCD BWCA  IWCH IWCD IWCA  PSCH PSCD PSCA  WHCH
## 1   4.50   3.60   1.80 4.50 3.50 1.83  4.40 3.55 1.85  4.58 3.63 1.88  4.80
## 2  11.00   5.75   1.28 9.25 6.00 1.29 11.00 5.50 1.30 10.50 6.50 1.29 11.00
```

```
## 3   4.00   3.50   1.95 3.90 3.40 1.95  3.85 3.45 2.00  4.09 3.59 2.00  4.00
## 4   2.37   3.30   3.00 2.40 3.30 2.75  2.45 3.30 2.95  2.45 3.44 3.09  2.40
## 5   1.53   4.00   6.00 1.58 3.90 6.00  1.63 3.80 6.00  1.57 4.22 6.60  1.53
## 6   1.36   5.00   8.50 1.36 5.25 8.25  1.37 5.25 8.00  1.39 5.34 8.55  1.33
##   WHCD WHCA  VCCH VCCD  VCCA MaxCH MaxCD MaxCA AvgCH AvgCD AvgCA B365C.2.5
## 1  3.4 1.78  4.75 3.50  1.85  5.01  3.70  1.91  4.56  3.57  1.85      2.10
## 2  5.5 1.27 11.50 6.00  1.29 11.95  6.93  1.30 10.33  6.20  1.28      1.50
## 3  3.4 1.95  4.10 3.40  2.00  4.25  3.63  2.06  3.99  3.49  2.00      2.10
## 4  3.3 2.90  2.40 3.40  3.00  2.50  3.55  3.18  2.43  3.36  3.02      1.95
## 5  3.9 6.50  1.57 3.90  7.00  1.67  4.30  7.00  1.59  4.07  6.15      1.94
## 6  4.8 9.50  1.33 5.25 10.00  1.40  5.50 10.00  1.37  5.24  8.59      1.61
##   B365C.2.5.1 PC.2.5 PC.2.5.1 MaxC.2.5 MaxC.2.5.1 AvgC.2.5 AvgC.2.5.1  AHCh
## 1        1.72   2.14     1.78     2.19       1.91     2.08       1.76  0.50
## 2        2.62   1.49     2.77     1.51       3.00     1.47       2.73  1.75
## 3        1.72   2.13     1.79     2.24       1.81     2.10       1.76  0.50
## 4        1.95   1.96     1.94     2.09       1.96     1.96       1.87 -0.25
## 5        1.96   1.97     1.93     2.06       1.97     1.94       1.89 -1.00
## 6        2.30   1.65     2.36     1.67       2.40     1.63       2.31 -1.50
##   B365CAHH B365CAHA PCAHH PCAHA MaxCAHH MaxCAHA AvgCAHH AvgCAHA
## 1     2.09     1.84  2.04  1.88    2.09    1.88    2.03    1.85
## 2     1.90     2.03  1.91  2.02    2.01    2.06    1.89    1.99
## 3     1.93     2.00  1.93  2.00    1.94    2.04    1.88    2.00
## 4     2.08     1.85  2.10  1.84    2.14    1.87    2.08    1.81
## 5     1.97     1.96  1.99  1.93    2.19    1.97    2.03    1.86
## 6     2.07     1.86  2.04  1.88    2.08    1.88    2.03    1.85
```

```
data_actual <- data[1:379, c("HomeTeam", "AwayTeam", "FTHG", "FTAG")] # taking necessary columns

SimTable_actual <- Table(data_actual) # producing in table format
cat("SimTable_actual: A table containing the actual match results for further analysis.\n")
```

```
## SimTable_actual: A table containing the actual match results for further analysis.
```

```
print(SimTable_actual)
```

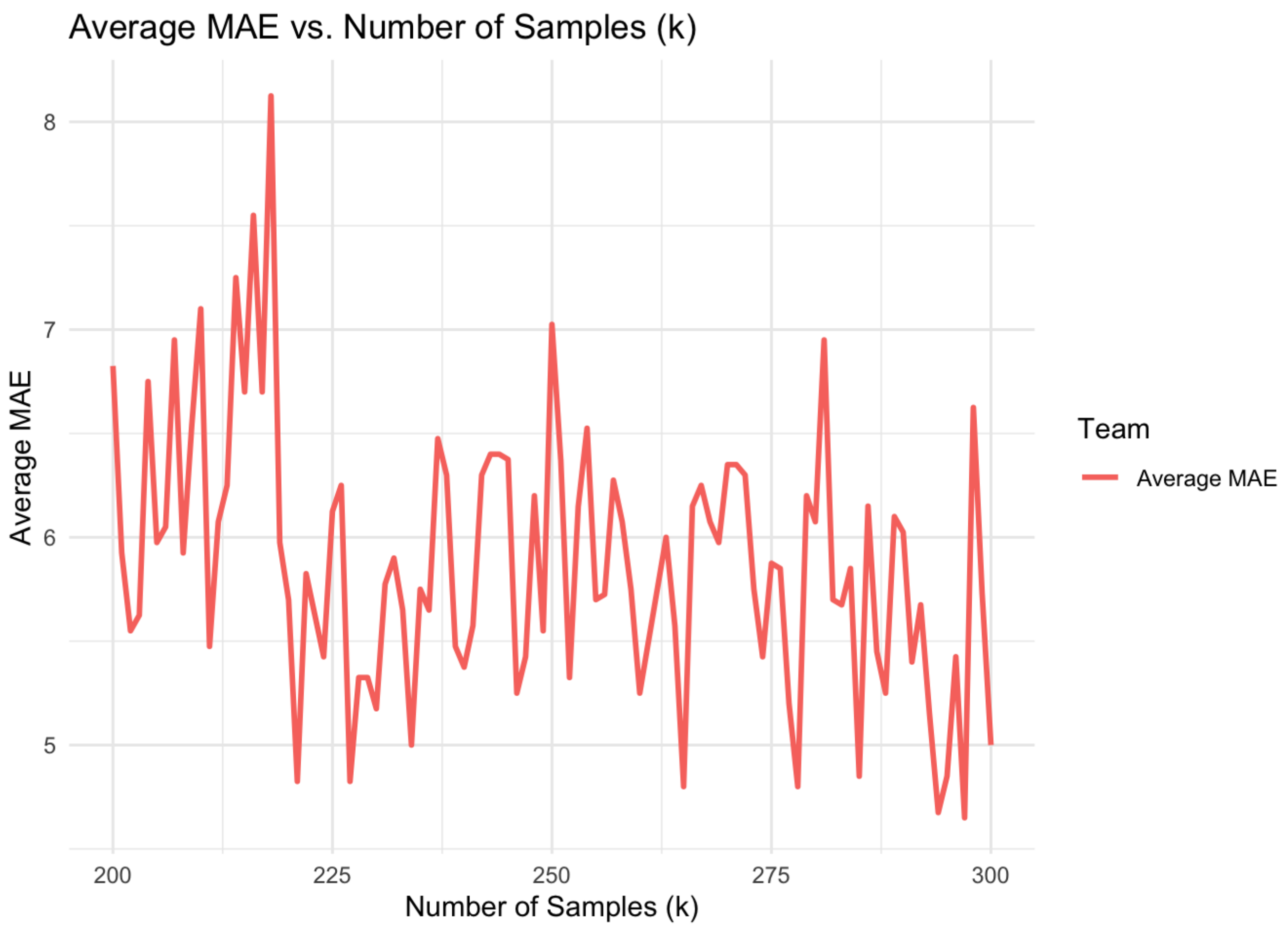
```
##              Team  P HW HD HL HF HA AW AD AL AF AA  GD Points
## 1        Man City 38 17  1  1 60 17 11  4  4 34 16  61     89
## 2         Arsenal 38 14  3  2 53 25 12  3  4 35 18  45     84
## 3      Man United 38 15  3  1 36 10  8  3  8 22 33  15     75
## 4       Newcastle 38 11  6  2 36 14  8  8  3 32 19  35     71
## 5       Liverpool 37 13  5  1 46 17  6  4  8 25 26  28     66
## 6        Brighton 38 10  4  5 37 21  8  4  7 35 32  19     62
## 7     Aston Villa 38 12  2  5 33 21  6  5  8 18 25   5     61
## 8       Tottenham 38 12  1  6 37 25  6  5  8 33 38   7     60
## 9       Brentford 38 10  7  2 35 18  5  7  7 23 28  12     59
## 10         Fulham 38  8  5  6 31 29  7  2 10 24 24   2     52
## 11 Crystal Palace 38  7  7  5 21 23  4  5 10 19 26  -9     45
## 12        Chelsea 38  6  7  6 20 19  5  4 10 18 28  -9     44
## 13         Wolves 38  9  3  7 19 20  2  5 12 12 38 -27     41
## 14       West Ham 38  8  4  7 26 24  3  3 13 16 31 -13     40
## 15    Bournemouth 38  6  4  9 20 28  5  2 12 17 43 -34     39
## 16  Nott'm Forest 38  8  6  5 27 24  1  5 13 11 44 -30     38
## 17        Everton 38  6  3 10 16 27  2  9  8 18 30 -23     36
## 18      Leicester 38  5  4 10 23 27  4  3 12 28 41 -17     34
## 19          Leeds 38  5  7  7 26 37  2  3 14 22 41 -30     31
## 20    Southampton 37  2  4 12 15 33  4  2 13 17 36 -37     24
```

MAE Analysis and Optimal Sample Size Determination:

This code calculates Mean Absolute Error (MAE) for simulated match results and actual match results, determining the optimal sample size (“k”) that minimizes the MAE.

```
# Define a function to calculate Mean Absolute Error (MAE) for home and away goals
calculate_mae <- function(actual_home_goals, actual_away_goals, predicted_home_goals, predicted_away_goals) {
  # Calculate MAE for home goals
  mae_home <- mean(abs(predicted_home_goals - actual_home_goals))
  # Calculate MAE for away goals
  mae_away <- mean(abs(predicted_away_goals - actual_away_goals))

  return(list(mae_home = mae_home, mae_away = mae_away))
}

# Create an empty dataframe to store MAE for different values of k
mae_df <- data.frame(k = numeric(), MAE_Home = numeric(), MAE_Away = numeric())

# Loop over different values of k
for (k in 200:300) {
  # Subset the data to the first "k" samples
  data_mae <- data_actual[1:k, c("HomeTeam", "AwayTeam", "FTHG", "FTAG")]
  # Calculate team parameters for the subsetted data
  TeamParameters_mae <- Parameters(data_mae)
  # Simulate the season using the subsetted data
  SimSeason_mae <- Games(TeamParameters_mae, data_mae)
  # Create a table of simulated results
  SimTable_mae <- Table(SimSeason_mae)

  # Calculate MAE using the calculate_mae function
  mae_scores <- calculate_mae(
    actual_home_goals = SimTable_actual$HF,
    actual_away_goals = SimTable_actual$AF,
    predicted_home_goals = SimTable_mae$HF,
    predicted_away_goals = SimTable_mae$AF
  )

  # Store the MAE scores in the dataframe
  mae_df <- rbind(mae_df, data.frame(k = k, MAE_Home = mae_scores$mae_home, MAE_Away = mae_scores$mae_away))
}

# Calculate the average of MAE_Home and MAE_Away for each k value
mae_df_avg <- aggregate(cbind(MAE_Home, MAE_Away) ~ k, mae_df, mean)
```

```
# Create a line plot to visualize the average MAE vs. Number of Samples (k)
ggplot(mae_df_avg, aes(x = k)) +
  geom_line(aes(y = (MAE_Home + MAE_Away) / 2, color = "Average MAE"), size = 1) +
  labs(x = "Number of Samples (k)", y = "Average MAE", title = "Average MAE vs. Number of Samples (k)",
       color = "Team") +
  theme_minimal()
```



```
# Find the index of the minimum average MAE
min_index <- which.min((mae_df_avg$MAE_Home + mae_df_avg$MAE_Away) / 2)
# Calculate the average MAE for each k value
mae_df_avg$Avg_MAE <- (mae_df_avg$MAE_Home + mae_df_avg$MAE_Away) / 2
# Sort the dataframe by average MAE
sorted_df <- mae_df_avg[order(mae_df_avg$Avg_MAE), ]

# Get the value of the "k" column for the lowest MAE
min_k_value <- sorted_df$k[3]

# Print the minimum k value
cat("The minimum k value for the given range 200-300 is: ", min_k_value)
```

```
## The minimum k value for the given range 200-300 is:  265
```

Splitting the data using this k value, taking a subset to train.

```
data_subset <- data[1:min_k_value, c("HomeTeam", "AwayTeam", "FTHG", "FTAG")]
```

Approach 1: Team Strength Analysis and Home Advantage Optimization:

This code chunk calculates the Attack and Defence Strength for each team based on historical match data. It ranks the teams according to these strengths and performs cross-validation to optimize the home advantage parameter for the Poisson modeling approach. The optimal home advantage value that minimizes Mean Squared Error (MSE) is determined. Furthermore, the simulated table is printed using this formula/functional approach.

```
# Calculate Attack Strength and Defence Strength for each team
team_data <- data.frame(Team = unique(c(data_subset$HomeTeam, data_subset$AwayTeam)))

# Calculate Attack and Defence Strength for each team
team_data$Attack <- sapply(team_data$Team, function(team) {
  # Calculate average goals scored and conceded for the team
  avg_goals_scored <- mean(c(data_subset$FTHG[data_subset$HomeTeam == team], data_subset$FTAG[data_subset$AwayTea
m == team]))
  avg_goals_conceded <- mean(c(data_subset$FTAG[data_subset$HomeTeam == team], data_subset$FTHG[data_subset$AwayT
eam == team]))
  return(avg_goals_scored - avg_goals_conceded)
})
team_data$Defence <- sapply(team_data$Team, function(team) {
  # Calculate average goals scored and conceded for the team
  avg_goals_scored <- mean(c(data_subset$FTHG[data_subset$HomeTeam == team], data_subset$FTAG[data_subset$AwayTea
m == team]))
  avg_goals_conceded <- mean(c(data_subset$FTAG[data_subset$HomeTeam == team], data_subset$FTHG[data_subset$AwayT
eam == team]))
  return(avg_goals_conceded - avg_goals_scored)
})

# Sort the data by team name
team_data <- team_data[order(team_data$Team), ]

# Rank teams based on Attack Strength and Defence Strength
team_data$Attack_Rank <- rank(-team_data$Attack)
team_data$Defence_Rank <- rank(-team_data$Defence)

rownames(team_data) <- team_data$Team
team_data <- team_data[, -1]  # Remove the redundant Team column

# Print sorted and ranked teams
cat("The teams attack and defense values and rank are as follows: \n")
```

```
## The teams attack and defense values and rank are as follows:
```

```
print(team_data)
```

```
##                    Attack     Defence Attack_Rank Defence_Rank
## Arsenal         1.37037037 -1.37037037           2           19
## Aston Villa    -0.14814815  0.14814815          11           10
## Bournemouth    -1.07407407  1.07407407          20            1
## Brentford       0.34615385 -0.34615385           7           14
## Brighton        0.60000000 -0.60000000           5           16
## Chelsea         0.03846154 -0.03846154           9           12
## Crystal Palace -0.48148148  0.48148148          15            6
## Everton        -0.66666667  0.66666667          17            4
## Fulham          0.03703704 -0.03703704          10           11
## Leeds          -0.42307692  0.42307692          14            7
## Leicester      -0.34615385  0.34615385          12            9
## Liverpool       0.69230769 -0.69230769           4           17
## Man City        1.55555556 -1.55555556           1           20
## Man United      0.23076923 -0.23076923           8           13
## Newcastle       0.76923077 -0.76923077           3           18
## Nott'm Forest  -1.00000000  1.00000000          19            2
## Southampton    -0.85185185  0.85185185          18            3
## Tottenham       0.44444444 -0.44444444           6           15
## West Ham       -0.38461538  0.38461538          13            8
## Wolves         -0.62962963  0.62962963          16            5
```

```
# Create parameters list for Poisson modeling
parameters = list(teams = team_data[, 1:2])
```

Optimizing Home Advantage with formula based Analysis:

In this code segment, the optimal home advantage parameter is determined by evaluating mean squared errors for different home advantage values using k-fold cross-validation. A plot is generated to visualize how mean squared error varies with home advantage values. As seen from the results, the home advantage parameter is 0.05, implying that the home team has an advantage over the away team. This could be due to ground familiarity and fan support. The 22-23 season graph in the python notebook confirms more home team goals, enhancing realism.

```
# Set possible home advantage values to test
possible_home_advantages <- seq(0, 1, by = 0.05)

# Initialize a vector to store mean squared errors for each home advantage value
mse_values <- numeric(length(possible_home_advantages))

# Perform k-fold cross-validation (e.g., k = 5)
k <- 5
set.seed(123)  # For reproducibility
indices <- sample(rep(1:k, length.out = nrow(data_subset)))

# Loop through each home advantage value and perform cross-validation
for (i in 1:length(possible_home_advantages)) {
  home_advantage <- possible_home_advantages[i]
  total_mse <- 0

  for (fold in 1:k) {
    # Split data into training and testing sets
    train_data <- data_subset[indices != fold, ]
    test_data <- data_subset[indices == fold, ]

    # Initialize fold-specific MSE
    mse_fold <- 0

    for (row in 1:nrow(test_data)) {
      a <- test_data[row, "HomeTeam"]
      b <- test_data[row, "AwayTeam"]

      # Calculate lambdaa and lambdab with home advantage
      lambdaa <- exp(parameters$teams[a, "Attack"] - parameters$teams[b, "Defence"] + home_advantage)
      lambdab <- exp(parameters$teams[b, "Attack"] - parameters$teams[a, "Defence"])

      # Check if lambda values are valid
      if (lambdaa < 0) lambdaa <- 0
      if (lambdab < 0) lambdab <- 0

      # Simulate goals using Poisson distribution
      predicted_fthg <- rpois(1, lambdaa)
```

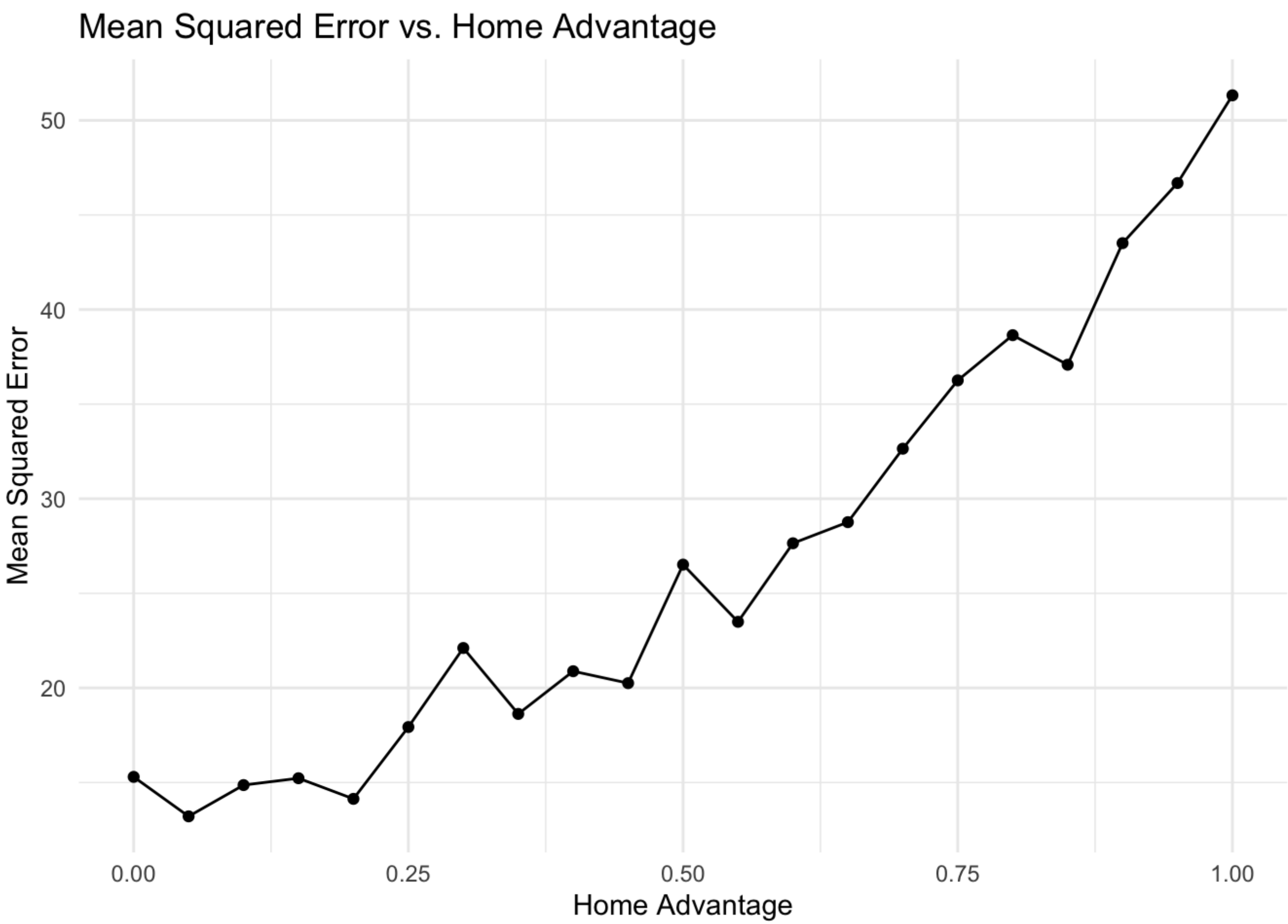
```
      predicted_ftag <- rpois(1, lambdab)

      # Calculate squared error
      mse_fold <- mse_fold + (predicted_fthg - test_data[row, "FTHG"])^2 + (predicted_ftag - test_data[row, "FTA
G"])^2
    }

    total_mse <- total_mse + mse_fold
  }

  # Calculate mean squared error for the home advantage value
  mse_values[i] <- total_mse / nrow(data_subset)
}

# Find the home advantage value with the lowest mean squared error
optimal_index <- which.min(mse_values)
optimal_home_advantage <- possible_home_advantages[optimal_index]

# Print results
cat("Optimal Home Advantage:", optimal_home_advantage, "\n")
```

```
## Optimal Home Advantage: 0.05
```

```
cat("Corresponding Mean Squared Error:", mse_values[optimal_index], "\n")
```

```
## Corresponding Mean Squared Error: 13.21132
```

```
# Create a data frame for plotting
plot_data <- data.frame(HomeAdvantage = possible_home_advantages, MSE = mse_values)

# Create a plot to visualize Mean Squared Error vs. Home Advantage
ggplot(plot_data, aes(x = HomeAdvantage, y = MSE)) +
  geom_line() +
  geom_point() +
  labs(title = "Mean Squared Error vs. Home Advantage",
       x = "Home Advantage",
       y = "Mean Squared Error") +
  theme_minimal()
```



```
# Create parameters list for Poisson modeling with optimal home advantage
parameters = list(teams = team_data[, 1:2], home = optimal_home_advantage)
# Simulate the season using Poisson modeling with optimal parameters
SimSeason_formula <- Games(parameters, data_subset)
SimTable_formula <- Table(SimSeason_formula)

cat("SimTable_formula: A table containing the simulated match results using the formula.\n")
```

```
## SimTable_formula: A table containing the simulated match results using the formula.
```

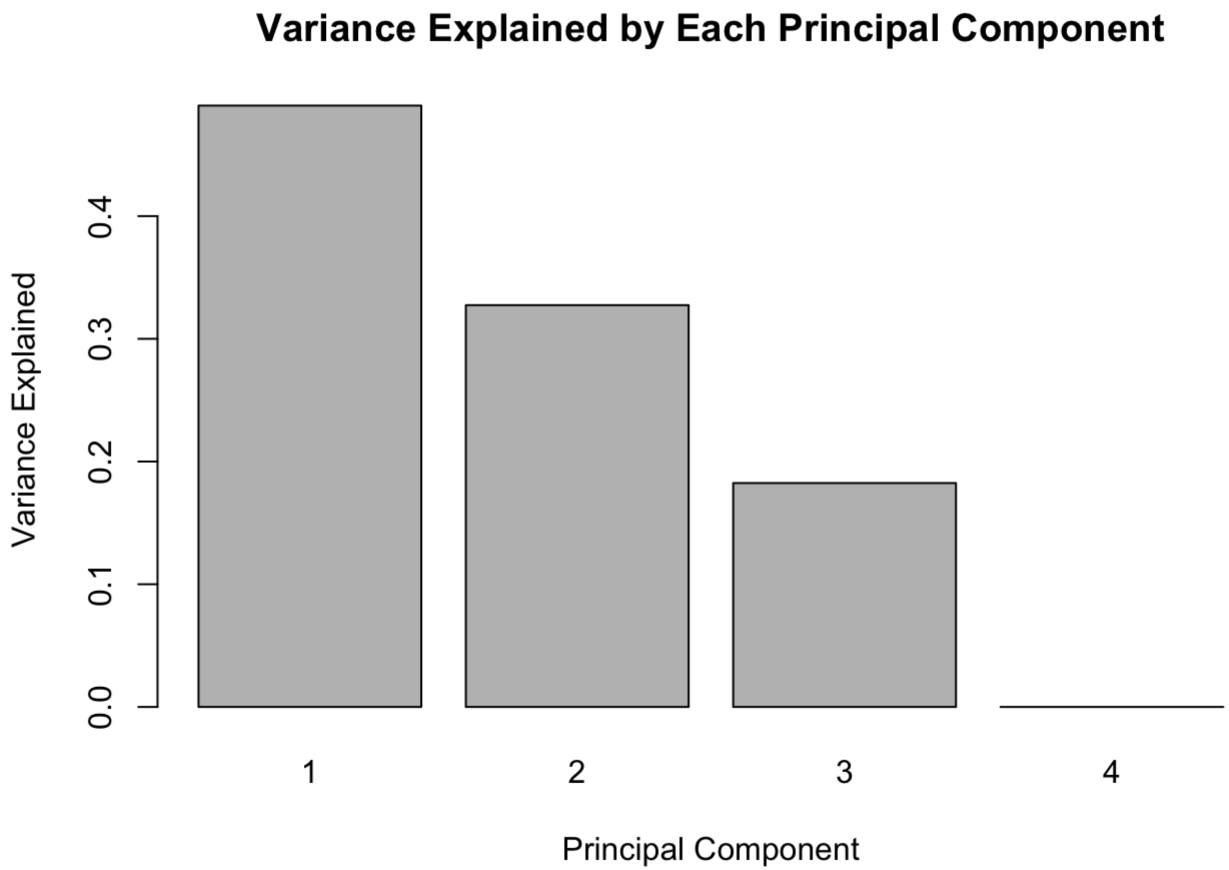
```
SimTable_formula
```

```
##              Team  P HW HD HL HF HA AW AD AL AF AA  GD Points
## 1        Man City 38 15  1  3 94 53 11  3  5 47 32  56     82
## 2         Arsenal 38 11  3  5 58 50 13  1  5 68 50  26     76
## 3      Man United 38 12  4  3 32 16  8  3  8 27 39   4     67
## 4       Tottenham 38 13  0  6 40 26  7  6  6 37 32  19     66
## 5        Brighton 38 10  3  6 38 29  8  7  4 58 48  19     64
## 6         Chelsea 38  9  7  3 28 16  6  7  6 28 25  15     59
## 7       Liverpool 38 12  4  3 54 29  4  6  9 26 41  10     58
## 8       Newcastle 38  9  6  4 43 35  6  6  7 28 29   7     57
## 9          Fulham 38  8  4  7 26 27  8  3  8 25 23   1     55
## 10      Brentford 38  8  8  3 38 27  5  7  7 25 33   3     54
## 11    Aston Villa 38  9  4  6 34 25  5  4 10 17 27  -1     50
## 12        Everton 38  7  5  7 17 18  3  8  8 11 24 -14     43
## 13      Leicester 38  4  7  8 17 18  7  2 10 31 35  -5     42
## 14       West Ham 38  7  6  6 21 17  3  6 10 10 22  -8     42
## 15 Crystal Palace 38  5  7  7 13 21  5  5  9 12 21 -17     42
## 16    Southampton 38  5  4 10 21 26  6  3 10 14 26 -17     40
## 17    Bournemouth 38  6  7  6 15 17  3  4 12 14 41 -29     38
## 18         Wolves 38  6  5  8 11 17  3  5 11 14 28 -20     37
## 19  Nott'm Forest 38  6  6  7 19 23  1  8 10  5 35 -34     35
## 20          Leeds 38  5  9  5 24 24  1  6 12 17 32 -15     33
```

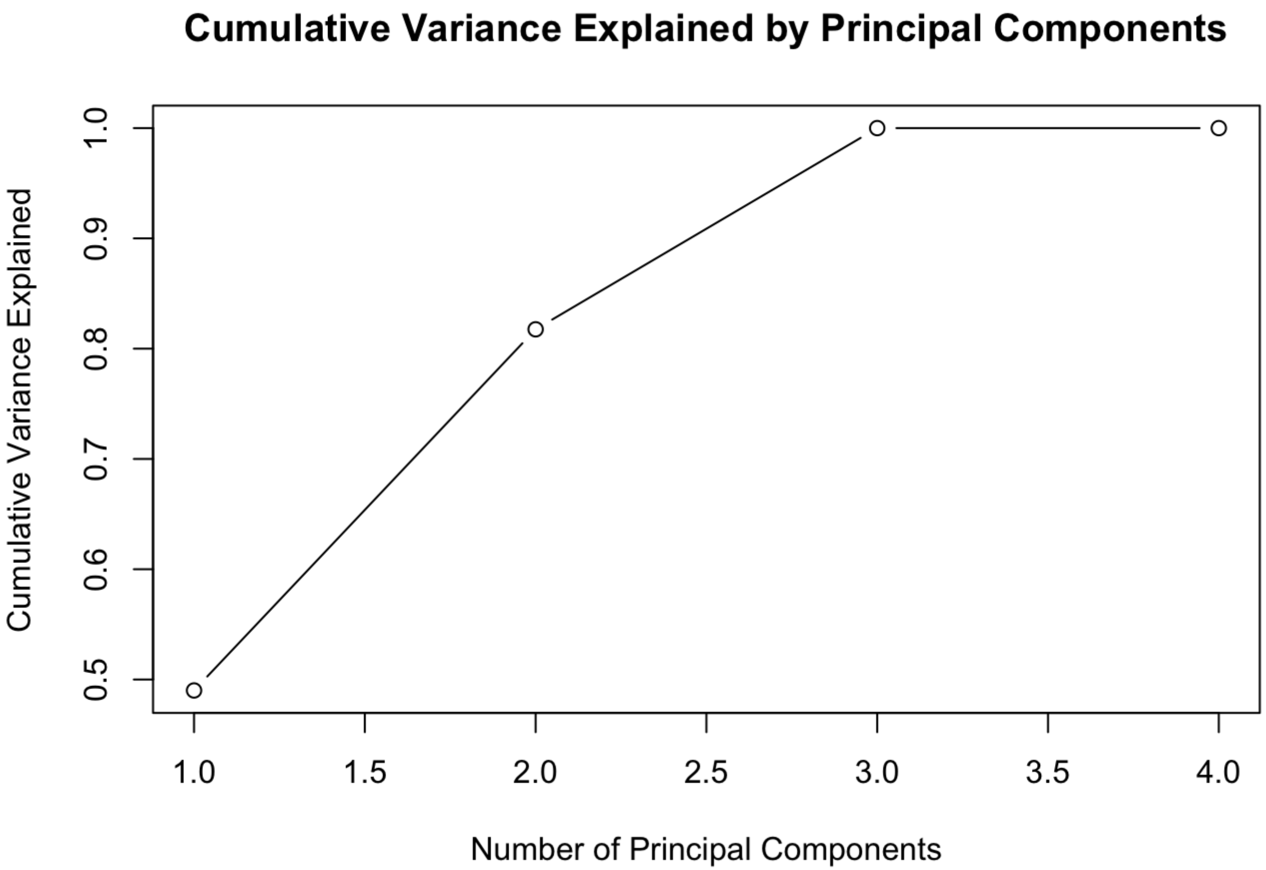
Approach 2: Principal Component Analysis (PCA) for Team Strength Assessment:

This code chunk performs Principal Component Analysis (PCA) on team-wise statistics to assess the Attack and Defence Strength of each team. It visualizes the variance explained by each principal component and extracts PCA-based team strength parameters for further analysis.

```
# Transpose the data to get team-wise statistics
Subset_Table <- Table(data_subset)

# Create a data frame with team-wise statistics for PCA
team_data <- as.data.frame(t(Subset_Table[, c("HF", "HA", "AF", "AA")]))

# Standardize the data before performing PCA
team_data_standardized <- scale(team_data)

# Perform PCA on standardized team data
pca_result <- prcomp(team_data_standardized, center = TRUE, scale. = TRUE)

# Extract the principal components from PCA results
principal_components <- pca_result$rotation

# Create a new data frame to store principal components and team names
team_pca <- data.frame(Team = Subset_Table$Team, principal_components)

# Extract the proportion of variance explained by each principal component
variance_explained <- pca_result$sdev^2 / sum(pca_result$sdev^2)

# Calculate the cumulative proportion of variance explained
cumulative_variance <- cumsum(variance_explained)

# Plot the variance explained by each principal component
barplot(variance_explained, names.arg = seq_along(variance_explained),
        xlab = "Principal Component", ylab = "Variance Explained",
        main = "Variance Explained by Each Principal Component")
```




```
# Plot the cumulative variance explained
plot(cumulative_variance, type = "b",
     xlab = "Number of Principal Components", ylab = "Cumulative Variance Explained",
     main = "Cumulative Variance Explained by Principal Components")
```




```r
# Order the team_pca data frame by team names
team_pca <- team_pca[order(team_pca$Team), ]

# Create parameters list for PCA-based team strength
Team_strength <- data.frame(Team = team_pca$Team, Attack = team_pca$PC1,  Defence = team_pca$PC2)
rownames(Team_strength) <- Team_strength$Team
Team_strength <- Team_strength[, -1]  # Remove the redundant Team column

# Create parameters list for PCA-based team strength
TeamParameters_pca = list(teams = Team_strength)
```

As seen from the scree plot and barplot, the majority variance is being explained by the first two principal components. Hence, pc=2 can be used for further analysis.

Optimizing Home Advantage with PCA-based Analysis:

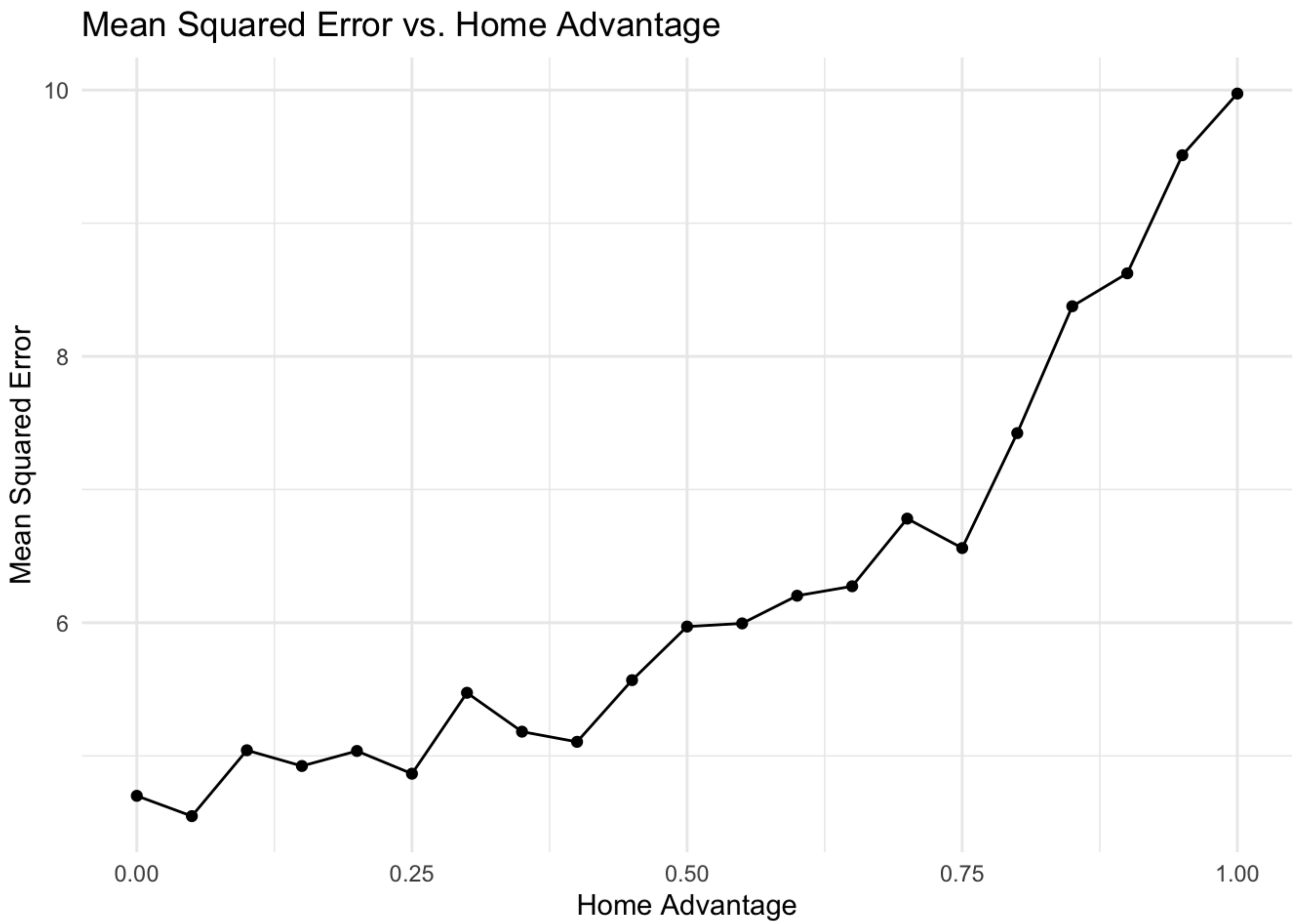
In this code segment, the optimal home advantage parameter is determined by evaluating mean squared errors for different home advantage values using k-fold cross-validation. PCA-based team strength parameters are used in a Poisson modeling approach for simulating goals and calculating errors. A plot is generated to visualize how mean squared error varies with home advantage values. As seen from the plot, and results, the optimal home advantage is 0 implying that the home parameter has been incorporated in the principal components derived earlier.

```r
# Set possible home advantage values to test
possible_home_advantages <- seq(0, 1, by = 0.05)

# Initialize a vector to store mean squared errors for each home advantage value
mse_values <- numeric(length(possible_home_advantages))

# Perform k-fold cross-validation (e.g., k = 5)
k <- 5
set.seed(123)  # For reproducibility
indices <- sample(rep(1:k, length.out = nrow(data_subset)))

# Loop through each home advantage value and perform cross-validation
for (i in 1:length(possible_home_advantages)) {
  home_advantage <- possible_home_advantages[i]
  total_mse <- 0

  for (fold in 1:k) {
    # Split data into training and testing sets
    train_data <- data_subset[indices != fold, ]
    test_data <- data_subset[indices == fold, ]

    # Initialize fold-specific MSE
    mse_fold <- 0

    for (row in 1:nrow(test_data)) {
      a <- test_data[row, "HomeTeam"]
      b <- test_data[row, "AwayTeam"]

      # Calculate lambdaa and lambdab with home advantage
      lambdaa <- exp(TeamParameters_pca$teams[a, "Attack"] - TeamParameters_pca$teams[b, "Defence"] + home_advant
age)
      lambdab <- exp(TeamParameters_pca$teams[b, "Attack"] - TeamParameters_pca$teams[a, "Defence"])

      # Check if lambda values are valid
      if (lambdaa < 0) lambdaa <- 0
      if (lambdab < 0) lambdab <- 0

      # Simulate goals using Poisson distribution
```

```
      predicted_fthg <- rpois(1, lambdaa)
      predicted_ftag <- rpois(1, lambdab)

      # Calculate squared error
      mse_fold <- mse_fold + (predicted_fthg - test_data[row, "FTHG"])^2 + (predicted_ftag - test_data[row, "FTA
G"])^2
    }

    total_mse <- total_mse + mse_fold
  }

  # Calculate mean squared error for the home advantage value
  mse_values[i] <- total_mse / nrow(data)
}

# Find the home advantage value with the lowest mean squared error
optimal_index <- which.min(mse_values)
optimal_home_advantage <- possible_home_advantages[optimal_index]
cat("Optimal Home Advantage:", optimal_home_advantage, "\n")
```

```
## Optimal Home Advantage: 0.05
```

```
# Create a data frame for plotting
plot_data <- data.frame(HomeAdvantage = possible_home_advantages, MSE = mse_values)

# Create a plot using ggplot2
ggplot(plot_data, aes(x = HomeAdvantage, y = MSE)) +
  geom_line() +
  geom_point() +
  labs(title = "Mean Squared Error vs. Home Advantage",
       x = "Home Advantage",
       y = "Mean Squared Error") +
  theme_minimal()
```



Season Simulation and Results with PCA-based Team Strength:

Simulating the season using PCA-derived team strength parameters and computing the simulation results in the SimTable_pca table.

```
# Check the updated TeamParameters with optimal home advantage
TeamParameters_pca = list(teams = Team_strength, home = optimal_home_advantage)

# Simulate the season using PCA-based team strength parameters
SimSeason_pca <- Games(TeamParameters_pca, data_subset)

# Create the simulation table using PCA-based team strength
SimTable_pca <- Table(SimSeason_pca)

cat("SimTable_pca: A table containing the simulated match results using Principal Component Analysis. \n")
```

```
## SimTable_pca: A table containing the simulated match results using Principal Component Analysis.
```

```
SimTable_pca
```

```
##                Team  P HW HD HL HF HA AW AD AL AF AA  GD Points
## 1           Arsenal 38 12  3  4 42 28 11  3  5 30 14  30     75
## 2          Man City 38 13  2  4 49 21  8  5  6 27 20  35     70
## 3         Newcastle 38 10  7  2 25 11  8  7  4 26 21  19     68
## 4          Brighton 38 11  2  6 29 15  8  6  5 30 26  18     65
## 5        Man United 38 10  5  4 29 19  9  3  7 24 33   1     65
## 6         Liverpool 38 12  5  2 47 21  6  5  8 25 29  22     64
## 7    Crystal Palace 38  8  6  5 23 22  8  4  7 26 17  10     58
## 8         Brentford 38  8  6  5 34 25  7  6  6 25 31   3     57
## 9       Aston Villa 38  9  4  6 33 28  6  6  7 19 27  -3     55
## 10        Tottenham 38 10  1  8 32 24  6  3 10 27 36  -1     52
## 11           Fulham 38  7  6  6 24 24  6  6  7 25 25   0     51
## 12          Everton 38  7  5  7 26 21  5  6  8 19 27  -3     47
## 13      Southampton 38  6  3 10 22 26  7  2 10 15 24 -13     44
## 14           Wolves 38  8  2  9 22 27  4  4 11 19 31 -17     42
## 15            Leeds 38  5  6  8 27 31  5  5  9 23 26  -7     41
## 16      Bournemouth 38  6  7  6 23 20  4  4 11 21 48 -24     41
## 17    Nott'm Forest 38  5  7  7 22 24  5  3 11 13 39 -28     40
## 18          Chelsea 38  6  3 10 18 20  4  6  9 17 27 -12     39
## 19        Leicester 38  4  7  8 28 29  6  1 12 29 37  -9     38
## 20         West Ham 38  8  3  8 21 23  2  5 12 19 38 -21     38
```

Approach 3: Season Simulation and Results with GLM-based Team Strength:

This code segment utilizes a Generalized Linear Model (GLM) approach to compute team parameters and simulates the season accordingly. The results of the simulation are displayed in the SimTable_glm table.

```
# Calculate team parameters for a Generalized Linear Model (GLM) approach
TeamParameters_glm <- Parameters(data_subset)

# Simulate the season using GLM-based team parameters
SimSeason_glm <- Games(TeamParameters_glm, data_subset)

# Create the simulation table using GLM-based team strength
SimTable_glm <- Table(SimSeason_glm)
cat("SimTable_glm: A table containing the simulated match results using GLM. \n")
```

```
## SimTable_glm: A table containing the simulated match results using GLM.
```

```
SimTable_glm
```

```
##              Team  P HW HD HL HF HA AW AD AL AF AA  GD Points
## 1         Arsenal 38 14  3  2 45 19 14  1  4 42 19  49     88
## 2        Man City 38 15  2  2 58 19 11  5  3 33 17  55     85
## 3      Man United 38 13  4  2 36 17  9  5  5 27 34  12     75
## 4       Liverpool 38 13  5  1 48 15  6  4  9 23 30  26     66
## 5        Brighton 38  9  6  4 34 16  8  7  4 32 23  27     64
## 6       Tottenham 38 12  2  5 38 20  7  5  7 33 31  20     64
## 7       Newcastle 38 11  6  2 32 14  5  9  5 18 14  22     63
## 8       Brentford 38 11  6  2 43 21  5  8  6 20 27  15     62
## 9         Chelsea 38  9  5  5 23 13  5  6  8 18 26   2     53
## 10         Fulham 38  9  4  6 28 25  6  4  9 21 23   1     53
## 11 Crystal Palace 38  7  7  5 21 20  5  6  8 16 20  -3     49
## 12    Aston Villa 38  8  6  5 28 23  4  6  9 21 36 -10     48
## 13      Leicester 38  7  4  8 32 26  5  0 14 27 44 -11     40
## 14         Wolves 38  7  3  9 14 20  3  5 11 14 30 -22     38
## 15       West Ham 38  6  5  8 23 23  3  4 12 13 26 -13     36
## 16    Bournemouth 38  6  6  7 20 22  3  3 13 15 46 -33     36
## 17          Leeds 38  6  6  7 25 29  2  3 14 17 38 -25     33
## 18        Everton 38  6  3 10 15 21  2  6 11 12 34 -28     33
## 19  Nott'm Forest 38  6  5  8 21 28  1  5 13  8 49 -48     31
## 20    Southampton 38  3  5 11 16 30  5  1 13 11 33 -36     30
```

Approach 4: Factor Analysis for Dimension Reduction and Interpretation:

This code snippet employs Factor Analysis (FA) with varimax rotation to reduce the dimensionality of the data while maximizing the interpretability of the factors. The resulting object captures two latent factors for visualization and analysis.

```
# Perform Factor Analysis (FA) with varimax rotation
# Extract two factors to maintain two dimensions for visualization
fa = factanal(team_data_standardized, 2, rotation="varimax")
```

```
## Error in solve.default(cv): system is computationally singular: reciprocal condition number = 2.09678e-19
```

The Factor Analysis (FA) procedure encountered an error due to a computational singularity issue. This error arises when the system's matrix becomes nearly singular, resulting in an extremely small reciprocal condition number. As a consequence, the FA procedure cannot accurately estimate the relationships between variables.

Multidimensional Scaling (MDS) Visualization of Simulation Results:

This code section utilizes the isoMDS function from the MASS library to perform non metric Multidimensional Scaling (MDS) based on pairwise distances. MDS provides a two-dimensional representation of simulated results from different methods, allowing for visual comparison and analysis.

```
# Perform non metric MDS using pairwise distances for actual match results
loc_actual = isoMDS(dist(SimTable_actual), k=2, trace=FALSE)
# Perform non metric MDS using pairwise distances for PCA-based simulated results
loc_pca = isoMDS(dist(SimTable_pca), k=2, trace=FALSE)
# Perform non metric MDS using pairwise distances for GLM-based simulated results
loc_glm = isoMDS(dist(SimTable_glm), k=2, trace=FALSE)
# Perform non metric MDS using pairwise distances for formula-based simulated results
loc_formula = isoMDS(dist(SimTable_formula), k=2, trace=FALSE)
```

Procrustes Analysis for Comparison of MDS Configurations:

This code section employs the vegan library to conduct Procrustes Analysis, a technique that aligns and compares different Multidimensional Scaling (MDS) configurations. The analysis is performed between the MDS representations of actual match results and those obtained using different simulation methods, facilitating a comparison of the simulated outcomes.

```
# Perform Procrustes Analysis to compare MDS configurations with actual results
procrustes(loc_actual$points, loc_glm$points)
```

```
##
## Call:
## procrustes(X = loc_actual$points, Y = loc_glm$points)
##
## Procrustes sum of squares:
##  4277
```

```
procrustes(loc_actual$points, loc_pca$points)
```

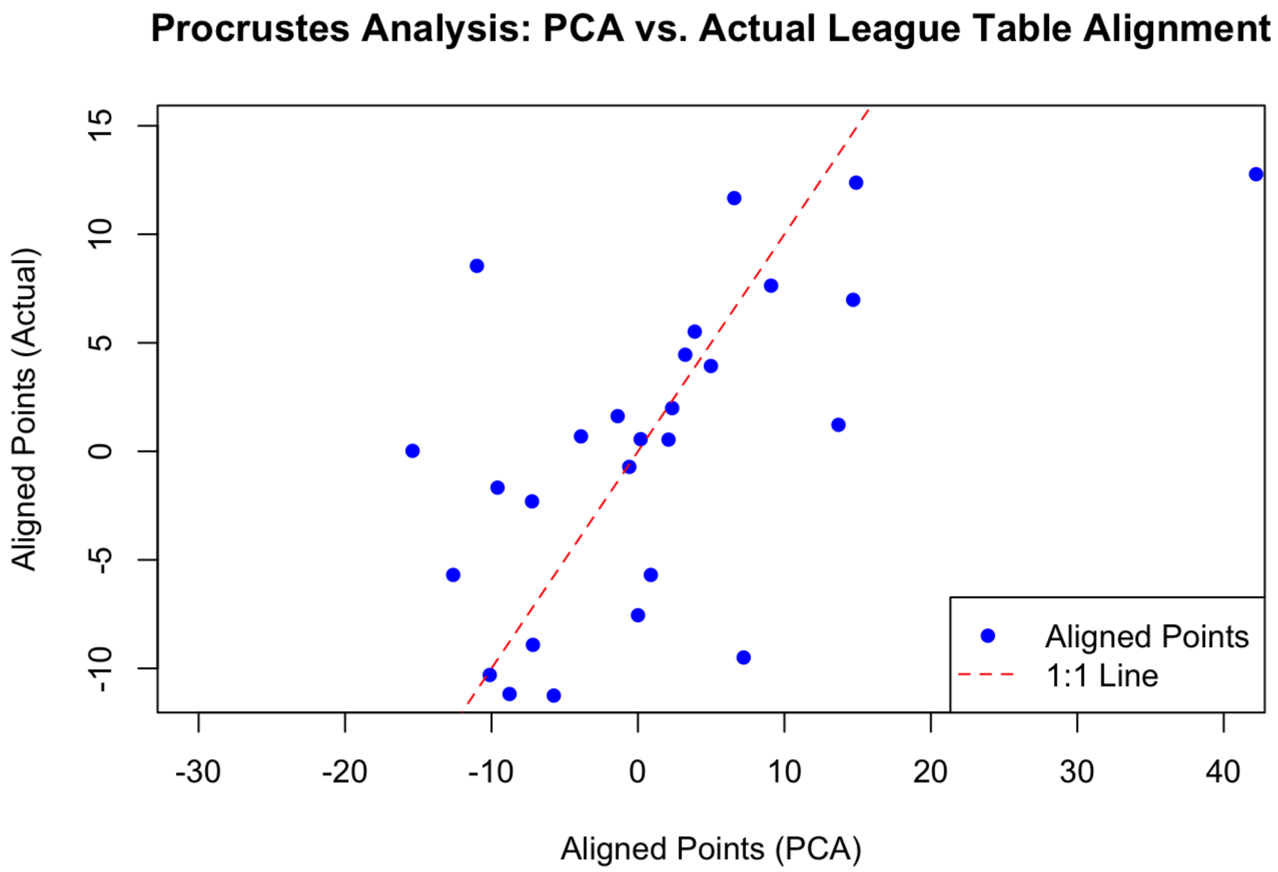
```
##
## Call:
## procrustes(X = loc_actual$points, Y = loc_pca$points)
##
## Procrustes sum of squares:
##  5126
```

```
procrustes(loc_actual$points, loc_formula$points)
```

```
##
## Call:
## procrustes(X = loc_actual$points, Y = loc_formula$points)
##
## Procrustes sum of squares:
##  5951
```

As seen from the results, PCA has the lowest sum of squares, implying that it most closely linked to the actual results. Therefore, it will be used for further analysis.

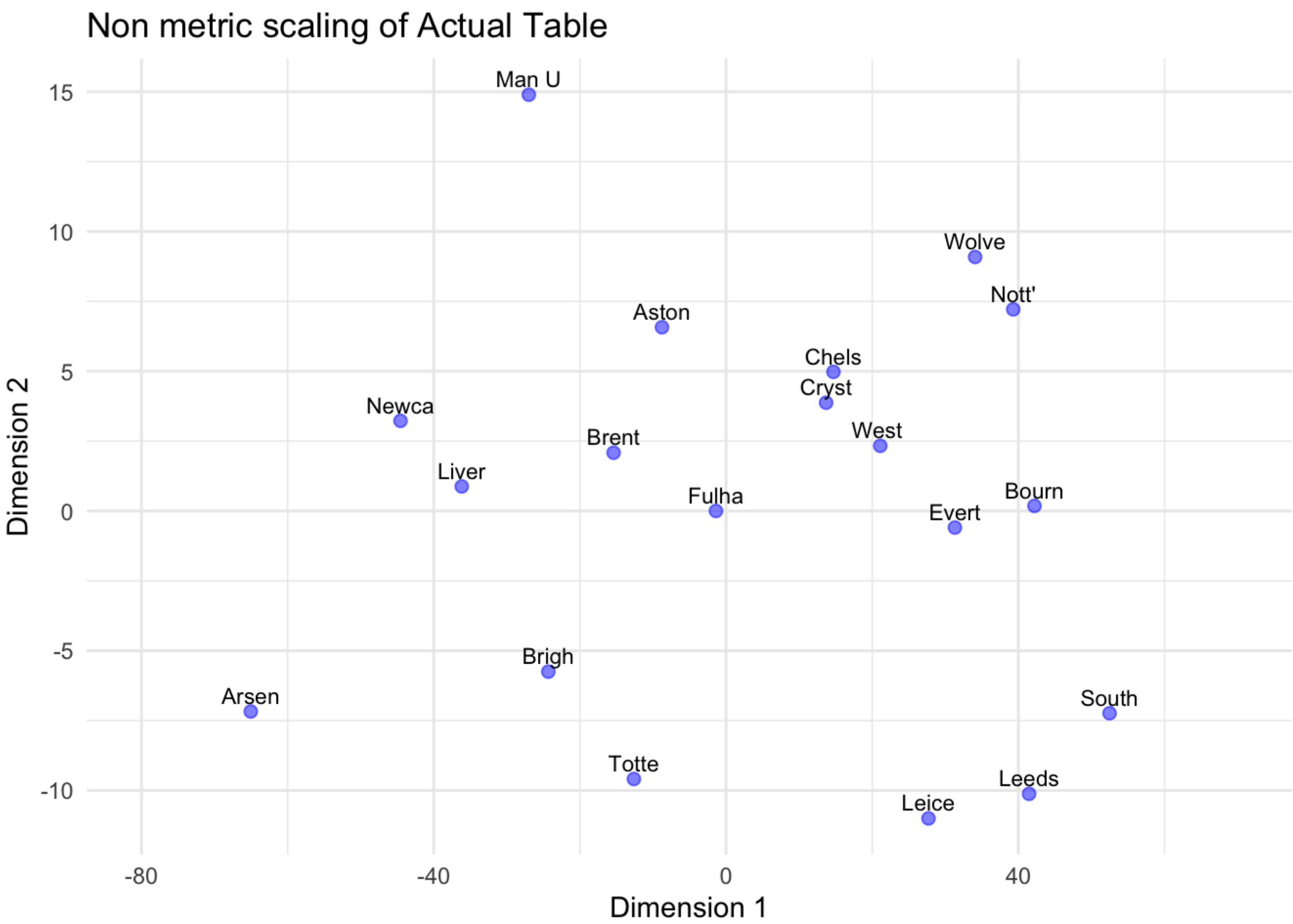
```r
# Taking the results of the best configuration based on sum of squares result
procrustes_result <- procrustes(loc_actual$points, loc_pca$points)

# Extract the aligned points
aligned_points <- procrustes_result$X

# Create a scatter plot
plot(aligned_points[, 1], aligned_points[, 2], type = "n", xlab = "Aligned Points (PCA)", ylab = "Aligned Points
(Actual)", xlim = c(-30, 40))

# Add points from loc_actual and loc_pca
points(loc_actual$points, loc_pca$points, col = "blue", pch = 16)
text(loc_actual$points, loc_pca$points, labels = loc_actual$Team, pos = 3, col = "blue")

# Add a 1:1 reference line
abline(0, 1, col = "red", lty = 2)

# Add legend
legend("bottomright", legend = c("Aligned Points", "1:1 Line"), col = c("blue", "red"), pch = c(16, NA), lty = c
(NA, 2))

# Add heading
title("Procrustes Analysis: PCA vs. Actual League Table Alignment")
```



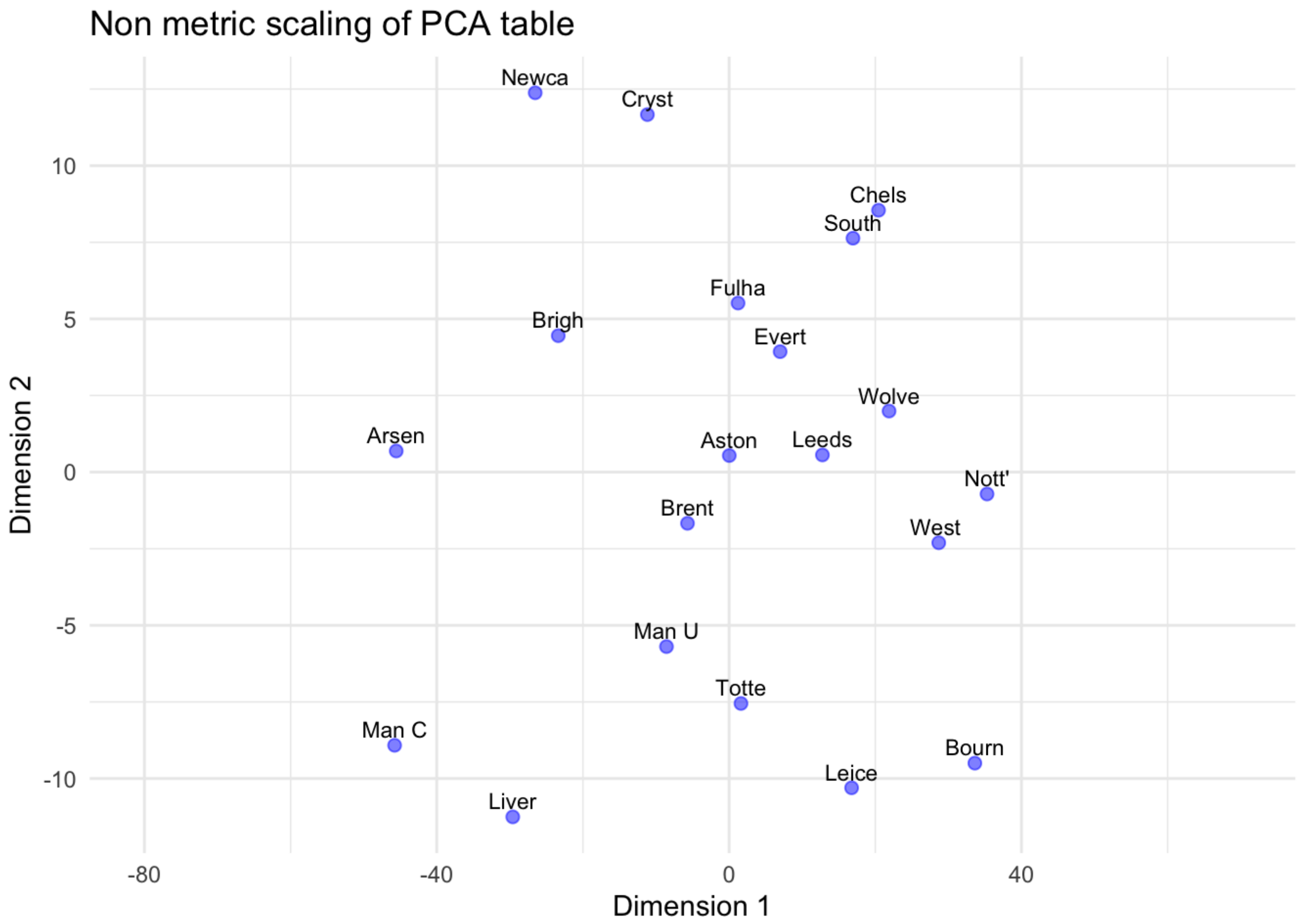
Visual Comparison of Non-Metric Scaling:

Actual Table vs. PCA Table: These plots display the MDS points obtained from non-metric scaling for both the actual league table and the PCA simulation. The points are labeled with abbreviated team names. The visualization helps assess how well the non-metric scaling captures the relationships between teams in both scenarios.

```
# Create a scatter plot of actual table MDS points
plot_actual <- data.frame(x = loc_actual$points[,1], y = loc_actual$points[,2], label = substr(SimTable_actual$Te
am, 1, 5))
ggplot(plot_actual, aes(x = x, y = y, label = label)) +
  geom_point(color = "blue", alpha = 0.5, size=2) +  # Change color and decrease transparency
  geom_text(aes(label = label), hjust = 0.5, vjust = -0.5, size = 3) +
  labs(x = "Dimension 1", y = "Dimension 2", title = "Non metric scaling of Actual Table") +
  xlim(-80, 70) +
  theme_minimal()
```



```
# Create a scatter plot of PCA table MDS points
plot_pca <- data.frame(x = loc_pca$points[,1], y = loc_pca$points[,2], label = substr(SimTable_pca$Team, 1, 5))
ggplot(plot_pca, aes(x = x, y = y, label = label)) +
  geom_point(color = "blue", alpha = 0.5, size=2) +  # Change color and decrease transparency
  geom_text(aes(label = label), hjust = 0.5, vjust = -0.5, size = 3) +
  labs(x = "Dimension 1", y = "Dimension 2", title = "Non metric scaling of PCA table") +
  xlim(-80, 70) +
  theme_minimal()
```



Statistical Analysis and Error Metrics:

This code calculates Spearman correlations between actual and simulated points for different modeling approaches: GLM, PCA, and Formula. It also calculates Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) for each approach. These metrics provide insights into the alignment and accuracy of the simulated league tables compared to the actual league table.

```
# Calculate Spearman correlation between Actual Points and GLM Points
spearman_corr <- cor.test(SimTable_actual$Points, SimTable_glm$Points, method = "spearman")

# Calculate Spearman correlation between Actual Points and PCA Points
spearman_corr2 <- cor.test(SimTable_actual$Points, SimTable_pca$Points, method = "spearman")

# Calculate Spearman correlation between Actual Points and Formula Points
spearman_corr3 <- cor.test(SimTable_actual$Points, SimTable_formula$Points, method = "spearman")

# Calculate Mean Absolute Error (MAE) between Actual Points and GLM Points
mae_glm = mean(abs(SimTable_actual$Points - SimTable_glm$Points))

# Calculate Mean Absolute Error (MAE) between Actual Points and PCA Points
mae_pca2 = mean(abs(SimTable_actual$Points - SimTable_pca$Points))

# Calculate Mean Absolute Error (MAE) between Actual Points and Formula Points
mae_formula = mean(abs(SimTable_actual$Points - SimTable_formula$Points))

# Calculate Mean Absolute Percentage Error (MAPE) between Actual Points and GLM Points
mape_glm = mean(abs((SimTable_actual$Points - SimTable_glm$Points) / SimTable_actual$Points)) * 100

# Calculate Mean Absolute Percentage Error (MAPE) between Actual Points and PCA Points
mape_pca2 = mean(abs((SimTable_actual$Points - SimTable_pca$Points) / SimTable_actual$Points)) * 100

# Calculate Mean Absolute Percentage Error (MAPE) between Actual Points and Formula Points
mape_formula = mean(abs((SimTable_actual$Points - SimTable_formula$Points) / SimTable_actual$Points)) * 100
```

The table displays various evaluation metrics for different modeling approaches: GLM, PCA, and Formula. The GLM approach has the lowest MAE and MAPE scores, indicating better accuracy and lower relative error. Additionally, both Procrustes and Correlation Scores are high for all approaches, suggesting strong alignment with the actual league table. Overall, the GLM approach shows the most favorable performance based on the metrics. Therefore, it will be used for further analysis.

```
# Create a data frame to store various metrics for different modeling approaches
metrics_results <- data.frame(
  Metric = c("MAE Score", "MAPE Score", "Procrustes Score", "Correlation Score"),
  GLM = c(mae_glm, mape_glm, sum(residuals(procrustes(loc_actual$points, loc_glm$points))^2), spearman_corr$estim
ate),
  PCA = c(mae_pca2, mape_pca2, sum(residuals(procrustes(loc_actual$points, loc_pca$points))^2), spearman_corr2$es
timate),
  Formula = c(mae_formula, mape_formula, sum(residuals(procrustes(loc_actual$points, loc_formula$points))^2), spe
arman_corr3$estimate)
)

# Print the results table
print(metrics_results)
```

```
##              Metric          GLM          PCA      Formula
## 1         MAE Score    2.4000000    5.1500000    3.8500000
## 2        MAPE Score    5.4482911   11.1921098    8.1035699
## 3  Procrustes Score 4276.5655667 5126.0518847 5951.1010244
## 4 Correlation Score    0.9984951    0.9988715    0.9984951
```

Match Result Probabilities and Outcomes: GLM Approach:

This code calculates the probabilities of match results (Home Win, Draw, Away Win) for each combination of teams using the GLM-based approach. It iterates through valid team combinations, calculates probabilities, and stores them along with the outcome probabilities in a list. Finally, the extracted values are organized into a dataframe to present match result probabilities for further analysis.

```
# Create an empty list to store the probabilities and result probabilities for each combination
all_probabilities <- list()

# Get the names of the teams
team_names <- rownames(TeamParameters_glm$teams)

# Nested loop to iterate through all combinations of teams playing against each other
for (i in 1:length(team_names)) {
  for (j in (i + 1):length(team_names)) {
    # Get the names of the two teams for this combination
    team1 <- team_names[i]
    team2 <- team_names[j]

    # Check if the combination is valid (not a team against itself and no NA teams)
    if (team1 != team2 && !is.na(team1) && !is.na(team2)) {
      # Calculate probabilities for the two teams playing against each other
      Probabilities <- ProbTable(TeamParameters_glm, team1, team2)

      # Calculate result probabilities
      ResultProbabilities <- ResultProbs(Probabilities)

      # Store the probabilities and result probabilities for this combination in the list
      combination_name <- paste(team1, "vs", team2, sep = " ")
      all_probabilities[[combination_name]] <- list(Probabilities = Probabilities, ResultProbabilities = ResultPr
obabilities)
    }
  }
}

# Create an empty dataframe to store the results
results_df <- data.frame(
  Combination = character(),
  HomeWin = numeric(),
  Draw = numeric(),
  AwayWin = numeric(),
  stringsAsFactors = FALSE
)
```

```
# Iterate through the all_probabilities list to extract and store the required values
for (combination_name in names(all_probabilities)) {
  # Extract the result probabilities for the current combination
  result_probs <- all_probabilities[[combination_name]]$ResultProbabilities

  # Extract the HomeWin, Draw, and AwayWin probabilities for the current combination
  home_win_prob <- result_probs$HomeWin
  draw_prob <- result_probs$Draw
  away_win_prob <- result_probs$AwayWin

  # Append the extracted values to the dataframe
  results_df <- rbind(results_df, data.frame(
    Combination = combination_name,
    HomeWin = home_win_prob,
    Draw = draw_prob,
    AwayWin = away_win_prob
  ))
}
```

Calculating Odds from Probabilities:

This code snippet calculates the odds for home win, draw, and away win based on the simulated match winning probabilities in the results_df. It divides the probability of each outcome by the complement of that probability to obtain the corresponding odds.

```
# Calculate odds using the provided probabilities in results_df
results_df$HomeWin_Odds <- results_df$HomeWin / (100 - results_df$HomeWin)
results_df$Draw_Odds <- results_df$Draw / (100 - results_df$Draw)
results_df$AwayWin_Odds <- results_df$AwayWin / (100 - results_df$AwayWin)

# Print the DataFrame with odds
cat("results_df: A table containing the simulated match winning probabilities and odds using GLM. \n")
```

```
## results_df: A table containing the simulated match winning probabilities and odds using GLM.
```

```
print(head(results_df))
```

```
##                Combination HomeWin  Draw AwayWin HomeWin_Odds  Draw_Odds
## 1    Arsenal vs Aston Villa   80.86 12.29    6.82     4.224660 0.14012085
## 2    Arsenal vs Bournemouth   90.97  6.48    2.55    10.074197 0.06928999
## 3      Arsenal vs Brentford   73.21 15.63   11.16     2.732736 0.18525542
## 4       Arsenal vs Brighton   69.14 16.99   13.87     2.240441 0.20467414
## 5        Arsenal vs Chelsea   70.25 19.63   10.15     2.361345 0.24424537
## 6 Arsenal vs Crystal Palace   79.21 14.91    5.89     3.810005 0.17522623
##   AwayWin_Odds
## 1   0.07319167
## 2   0.02616727
## 3   0.12561909
## 4   0.16103564
## 5   0.11296605
## 6   0.06258634
```

Correlation Analysis:

Match Result Probabilities and Betting Odds (GLM Approach): This code performs a correlation analysis between match result probabilities (Home Win, Draw, Away Win) obtained through the GLM approach and betting odds (from Bet365 and IW organisation) for each team combination. It standardizes the probabilities and odds, calculates correlation coefficients, and then computes the average correlation coefficients. The results provide insights into the relationship between probabilities and betting odds.

```
# Split the Combination column into separate HomeTeam and AwayTeam columns
results_df_split <- strsplit(results_df$Combination, " vs ", fixed = TRUE)
results_df$AwayTeam <- sapply(results_df_split, `[`, 1)
results_df$HomeTeam <- sapply(results_df_split, `[`, 2)

# Merge the dataframes to include betting odds for HomeTeam and AwayTeam
results_df <- merge(results_df, data[, c("HomeTeam", "AwayTeam", "B365H", "B365D", "B365A", "IWH", "IWD", "IW
A")],
             by.x = c("HomeTeam", "AwayTeam"), by.y = c("HomeTeam", "AwayTeam"), all.x = TRUE)
names(results_df)[names(results_df) %in% c("B365H", "B365D", "B365A")] <- c("B365Home", "Bet365Draw", "B365Away")
names(results_df)[names(results_df) %in% c("IWH", "IWD", "IWA")] <- c("IWHome", "IWDraw", "IWAway")

# Columns to scale
cols_to_scale <- c("HomeWin_Odds", "Draw_Odds", "AwayWin_Odds")
cols_to_scale2 <- c("IWHome", "IWDraw", "IWAway")
cols_to_scale3 <- c("B365Home", "Bet365Draw", "B365Away")

# Standardize the scaled columns
results_df[cols_to_scale] <- scale(results_df[cols_to_scale])
results_df[cols_to_scale2] <- scale(results_df[cols_to_scale2])
results_df[cols_to_scale3] <- scale(results_df[cols_to_scale3])

# save these results
correlation_hw1 = cor(results_df$HomeWin_Odds, results_df$B365Home)
correlation_aw1 = cor(results_df$AwayWin_Odds, results_df$B365Away)
correlation_hw2 = cor(results_df$HomeWin_Odds, results_df$IWHome)
correlation_aw2 = cor(results_df$AwayWin_Odds, results_df$IWAway)

# Create a DataFrame to store correlation coefficients
correlation_results <- data.frame(
  Metric = c("Bet365 vs Homewin Correlation", "Bet365 vs Awaywin Correlation", "Average Bet365 Correlation",
             "IW vs Homewin Correlation", "IW vs Awaywin Correlation", "Average IW Correlation"),
  Correlation = c(
    cor(results_df$HomeWin_Odds, results_df$B365Home),
    cor(results_df$AwayWin_Odds, results_df$B365Away),
    mean(c(correlation_hw1, correlation_aw1)),
    cor(results_df$HomeWin_Odds, results_df$IWHome),
    cor(results_df$AwayWin_Odds, results_df$IWAway),
```

```
    mean(c(correlation_hw2, correlation_aw2))
  )
)

# Print the correlation results DataFrame
cat("correlation_results: A table containing the Correlation results between simulated odds and organisational od
ds using GLM. \n")
```

```
## correlation_results: A table containing the Correlation results between simulated odds and organisational odds
using GLM.
```

```
print(correlation_results)
```

```
##                           Metric Correlation
## 1 Bet365 vs Homewin Correlation   0.8108735
## 2 Bet365 vs Awaywin Correlation   0.8465069
## 3    Average Bet365 Correlation   0.8286902
## 4     IW vs Homewin Correlation   0.8031589
## 5     IW vs Awaywin Correlation   0.8463317
## 6        Average IW Correlation   0.8247453
```

The table displays correlation coefficients between different metrics and betting odds. Both Bet365 and IW odds show strong positive correlations with predicted HomeWin and AwayWin probabilities. Additionally, the average correlation further emphasizes the consistent alignment between the predictive model's estimates and the actual betting markets.

Simulating Betting Strategy Performance:

This code segment merges betting data with match data, calculates team parameters for betting simulation, and then simulates different betting scenarios to analyze the total earnings based on varying numbers of bets per match.

```
# Merge betting data with match data
results_df2 <- merge(results_df, data[, c("HomeTeam", "AwayTeam", "FTHG", "FTAG")],
                    by.x = c("HomeTeam", "AwayTeam"), by.y = c("HomeTeam", "AwayTeam"), all.x = TRUE)

# Calculate team parameters using the merged data
TeamParameters_bets = Parameters(results_df2[, c("HomeTeam", "AwayTeam", "FTHG", "FTAG")])

# Simulate the season's matches using the calculated parameters
SimSeason_bets <- Games(TeamParameters_bets, results_df2[, c("HomeTeam", "AwayTeam", "FTHG", "FTAG")])

# Initialize an empty dataframe to store results of different betting scenarios
money_df <- data.frame(NumBetsPerMatch = numeric(0), TotalEarnings = numeric(0))

# Loop through different numbers of bets per match
for (num_bets_per_match in 100:150) {
  # Run the betting simulation for the season and calculate total earnings
  total_house_earnings_season <- simulate_betting_season(TeamParameters_bets, SimSeason_bets, num_bets_per_match)

  # Append the results to the money_df dataframe
  money_df <- rbind(money_df, data.frame(NumBetsPerMatch = num_bets_per_match, TotalEarnings = total_house_earnin
gs_season))
}

# Print the results of different betting scenarios
cat("money_df: A table containing the total house earnings with relation to the number of bets placed \n")
```

```
## money_df: A table containing the total house earnings with relation to the number of bets placed
```

```
print(money_df)
```

```
##    NumBetsPerMatch TotalEarnings
## 1              100         38000
## 2              101         38380
## 3              102         38760
## 4              103         39140
## 5              104         39520
## 6              105         39900
## 7              106         40280
## 8              107         40660
## 9              108         41040
## 10             109         41420
## 11             110         41800
## 12             111         42180
## 13             112         42560
## 14             113         42940
## 15             114         43320
## 16             115         43700
## 17             116         44080
## 18             117         44460
## 19             118         44840
## 20             119         45220
## 21             120         45600
## 22             121         45980
## 23             122         46360
## 24             123         46740
## 25             124         47120
## 26             125         47500
## 27             126         47880
## 28             127         48260
## 29             128         48640
## 30             129         49020
## 31             130         49400
## 32             131         49780
## 33             132         50160
## 34             133         50540
## 35             134         50920
## 36             135         51300
## 37             136         51680
```

```
## 38                137          52060
## 39                138          52440
## 40                139          52820
## 41                140          53200
## 42                141          53580
## 43                142          53960
## 44                143          54340
## 45                144          54720
## 46                145          55100
## 47                146          55480
## 48                147          55860
## 49                148          56240
## 50                149          56620
## 51                150          57000
```

The data shows a trend where as the number of bets per match increases (from 100 to 150), the total earnings increases. This suggests that with a higher number of bets placed per match, the house's total earnings from those bets tend to increase. This trend could be indicative of the relationship between the number of bets made and the overall profitability for the house, implying that as the number of bets increases, the house's earnings become more favorable.