A Bayesian Heirarchical Model Simulation of the English Premier League Season

Jack Banks, Yanqing Li, Yuan Su, Jingxuan Yang

December 12, 2022

Introduction

The 2022 FIFA World Cup is a unique tournament, taking place over 29 days from late November to mid-December in the Middle Eastern country of Qatar. It is the first World Cup to be hosted in an Arab country and the first to occur in the middle of Europe's domestic league schedule, which spans from August to May. Because of this, most European Leagues are on an extended break with around one third of the season already played.

Poisson distribution was developed by 19th century French mathematician Siméon Denis Poisson. It is a probability theory used to model the amount of times an event occurs in a specific length. One popular application of Poisson distribution is the number of goals scored by a team in a 90 minute football match. This can be further applied to model the results of matches over an entire season. By separating the home and away team effects, we can calculate the likelihood of each possible score.

Our goal in this project is to predict the final standings of the English Premier League through a Bayesian Hierarchical Model and Monte Carlo simulation. Using the completed portion of the Premier League season as a prior, we will simulate the rest of the season to predict final standings.

Data

The English Premier League follows a very simple schedule. The league consists of twenty teams, each of which plays every other team twice: once at home and once away. This leads to a total of thirty-eight games for each team over the entire season.

Here is the current real table:

Table 1: Premier League Table as of Dec 12, 2022

Team	Games	Wins	Draws	Losses	Scored	Allowed	Goal Diff	Points
Arsenal	14	12	1	1	33	11	22	37
Manchester City	14	10	2	2	40	14	26	32
Newcastle United	15	8	6	1	29	11	18	30
Tottenham Hotspur	15	9	2	4	31	21	10	29
Manchester United	14	8	2	4	20	20	0	26
Liverpool	14	6	4	4	28	17	11	22
Brighton & Hove Albion	14	6	3	5	23	19	4	21
Chelsea	14	6	3	5	17	17	0	21
Fulham	15	5	4	6	24	26	-2	19
Brentford	15	4	7	4	23	25	-2	19
Crystal Palace	14	5	4	5	15	18	-3	19
Aston Villa	15	5	3	7	16	22	-6	18
Leicester City	15	5	2	8	25	25	0	17
AFC Bournemouth	15	4	4	7	18	32	-14	16
Leeds United	14	4	3	7	22	26	-4	15
West Ham United	15	4	2	9	12	17	-5	14
Everton	15	3	5	7	11	17	-6	14
Nottingham Forest	15	3	4	8	11	30	-19	13
Southampton	15	3	3	9	13	27	-14	12

Team	Games	Wins	Draws	Losses	Scored	Allowed	Goal Diff	Points
Wolverhampton Wanderers	15	2	4	9	8	24	-16	10

Each team has played around fourteen to fifteen games, and it is at this point in the season where the table begins to resemble it's final form, after initial variation.

The simplicity of the schedule allows it to be represented by two 20-by-20 matrices displaying the goals scored by the home and away teams in each matchup, respectively. The row names display the name of the home team and the column names display the name of the away team.

Here is a sample of the *home* dataset:

Table 2: Sample of *home* dataset

	Arsenal	Aston Villa	Bournemouth	Brentford	Brighton
Arsenal	NA	2	NA	NA	NA
Aston Villa	NA	NA	NA	4	NA
Bournemouth	0	2	NA	0	NA
Brentford	0	NA	NA	NA	2
Brighton	NA	1	NA	NA	NA

The cells that are already filled represent games that already happened. The value **2** in the [Arsenal, Aston Villa] column represents the two goals scored by Arsenal in their 2-1 home victory over Aston Villa on August 31. In the *away* dataset, this cell is populated with the value **1**.

Cells that display an **NA** value represent games that have not happened yet. For example, the **NA** in [Aston Villa, Arsenal] shows that Aston Villa has not yet hosted Arsenal this season. The main objective in this project is to simulate the value for each unplayed game. While this game will not truly occur until February 18, we can use prior season information to predict the outcome.

This matrix also includes **NA** values through the diagonal. These are the games that will never happen, as a team does not play against itself. These values will later be removed from the model.

Model

In order to predict the outcome of a match, we need to model for the goals scored by each team. As mentioned above, this follows a Poisson distribution.

For game between teams i and j, the goals scored, $YHome_{i,j}$ and $YAway_{i,j}$ can be modeled as

$$YHome_{i,j} \sim Poisson(\lambda_{i,j}^{(A)})$$

$$YAway_{i,j} \sim Poisson(\lambda_{i,j}^{(A)})$$

The indices, i and j determine the home and away teams in a match, repectively. They are both discrete, beginning at 1 and ending at 20.

$$i, j = 1, 2, 3, ..., 19, 20$$

The λ value describes the mean expected goals for each side in the match. While it may be simpler to model a λ_i as the mean goals scored by team i, there are other factors at play in each game. Instead, it is better to model each $\lambda_{i,j}$ as a combination of one team's offensive ability and the other team's defensive ability.

Offensive ability for team i is modeled as α_i, which follows the assumes distribution:

$$\alpha_i \sim Normal(0, \sigma_\alpha^2)$$

The standard deviation value σ_α is set to follow a non-informative exponential prior.

$$\sigma_\alpha \sim Exponential(0.001)$$

Similarly, defensive ability for team i is modeled as β_i, which assumes the following distribution:

$$\beta_i \sim Normal(0, \sigma_\beta^2)$$

The standard deviation value σ_β^2 is set to follow a non-informative exponential prior.

$$\sigma_\beta \sim Exponential(0.001)$$

Teams with stronger offensive abilities will have larger α_i values. Teams with stronger defensive abilities will have larger β_i values.

Another important factor to consider is the game's location. Traditionally, the home team performs better than the away team, due to crowd, familiarity, and other factors.

To account for this, we implement adjusters for home and away into the model, μ_H and μ_A, respectively. The parameters also assume normal distribution, with large variation.

$$\mu_H \sim Normal(0, 1e + 6)$$

$$\mu_A \sim Normal(0, 1e + 6)$$

As the model iterates, it is expected for mu_H to approach a significantly larger value than mu_A. It is also expected for the best teams to exhibit higher values of α_i and β_i, with the lesser teams having negative values for these parameters.

It is also important to note that the JAGS language defaults to the use of *precision* (τ^2) instead of *variance* (σ^2). This is accounted for in the JAGS model code, and explains the perceived inconsistencies between the above distributions and written code.

Now that we have defined μ, α, and β, we return to the mean goals scored by each team in a single game, $\lambda_{i,j}$.

For home teams, $\lambda_{i,j}^{(H)}$ is now better represented as:

$$log(\lambda_{i,j}^{(H)}) = \mu_H + \alpha_i - \beta_j$$

For away teams, $\lambda_{i,j}^{(A)}$ is now better represented as:

$$log(\lambda_{i,j}^{(A)}) = \mu_A + \alpha_j - \beta_i$$

The purpose of the logarithm is to maintain the positive quality of goals scored. It is now clear to see how game location, team offensive strength, and opponent defensive strength play a role in determining the goals scored in a match.

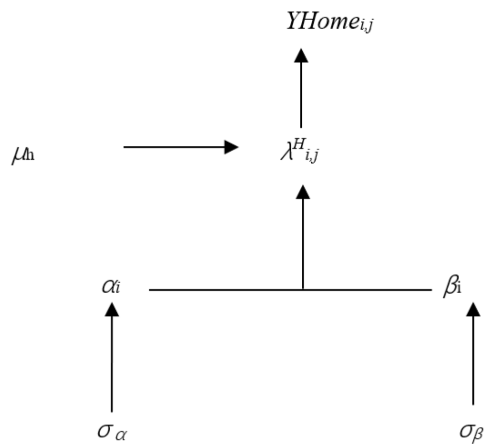


Figure 1: Bayesian Hierarchical Model for $YHome_{i,j}$

The following image depicts the relationship between the aforementioned parameters in the hierarchical model:

As an example, say we have three teams: A, B and C. From these teams, the games A vs. B and B vs. C have already occurred. The data from these two games is enough to roughly estimate μ_H, μ_A and an α and β for each team . In order to predict the result and goals in A vs. C (A is the home team and C is the away team), we can use the formulae:

$$log(\lambda_{A,C}^{(H)}) = \mu_H + \alpha_A - \beta_C$$

$$log(\lambda_{A,C}^{(A)}) = \mu_A + \alpha_C - \beta_A$$

Poisson distribution will sample from these lambdas in order to create simulated results for this game, just as every remaining game will be estimated and simulated for the remainder of the season.

The exact JAGS code used can be found in the appendix.

Results

JAGS was used to run through 10,000 iterations of a simulated season using Gibbs Sampling, substantially enough to estimate the true posterior. The values of μ_H, μ_A, σ_α, and σ_β were initialized into three chains at various values. Model parameters, most importantly μ_H and μ_A, demonstrated convergence near the 100 iteration mark.

Just as expected, mu_H converged to a higher value than mu_A.

- $mu_H = 0.48112$
- $mu_A = 0.05644$

This amounts to an expected difference of approximately 0.55 goals in favor of the home team in every match, a significant advantage that adequately models reality.

JAGS's `coda.samples()` function allowed for the extraction of statistics from each of the iterations, which were used to form a final table for each simulated season. As an example, here is the table from the very first simulation:

Table 3: Final Table from First SImulation

	Team	Points	GD
1.	Arsenal	85	47
2.	Newcastle	79	43
3.	Man City	78	44
4.	Man United	75	17
5.	Liverpool	66	20
6.	Chelsea	66	15
7.	Fulham	61	7
8.	Spurs	60	16
9.	Brighton	58	-1
10.	Brentford	53	1
11.	Leicester	46	0
12.	Aston Villa	46	-9
13.	Crystal Palace	45	-9
14.	Leeds	45	-14
15.	Everton	43	-12
16.	Southampton	37	-29
17.	West Ham	31	-24
18.	Nottingham Forest	31	-34
19.	Bournemouth	28	-37
20.	Wolves	27	-41

This simulation, like many others, resulted in a Premier League Championship for Arsenal, the league's current leader. To maintain a realistic prediction and avoid alphabetical bias, the real tiebreaker of goal difference was used to order teams who finished with equal points. This happens quite often, four times in this simulation alone. In this case, goal difference saved West Ham's season, as Nottingham Forest's inferior number sent them into a relegation position at 18.

Compliling data from every simulation produces the following table, describing each team's mean performance and posterior probabilities of certain accomplishments:

For background, UCL% shows the probability that each team will finish in a position to qualify for next season's UEFA Champions League. In the English Premier League, this is awarded to the top four teams. Relegated% shows the probability that each team will finish in a position that gets them relegated from the Premier League into England's second tier, The Championship, for the next season. Finishing positions 18, 19, 20 are relegated.

Table 4: Mean SUmmary Table of All Simulations

	Team	Points	SD	GoalDiff	First	UCL	Relegated
1.	Arsenal	82.2	7.4	42.8	57.47%	97.86%	0%
2.	Man City	78.4	7.5	49.9	33.86%	95.18%	0%
3.	Newcastle	68.5	7.6	28.7	4.74%	65.14%	0%
4.	Spurs	66.4	7.5	19.0	2.39%	52.58%	0%
5.	Man United	62.6	7.6	4.7	0.68%	30.3%	0.06%
6.	Liverpool	61.4	7.6	20.8	0.63%	27.15%	0.08%
7.	Brighton	57.4	7.7	8.6	0.16%	13.11%	0.35%
8.	Chelsea	53.7	7.5	-1.7	0.02%	5.44%	1.08%
9.	Brentford	52.2	7.5	-0.5	0.03%	4.02%	1.75%
10.	Fulham	51.7	7.4	-1.0	0%	2.99%	2.06%
11.	Leicester	50.6	7.3	1.9	0%	2.26%	2.31%

	Team	Points	SD	GoalDiff	First	UCL	Relegated
12.	Leeds	48.5	7.7	-4.8	0.02%	1.5%	5.34%
13.	Crystal Palace	48.5	7.5	-9.0	0%	1.68%	4.79%
14.	Aston Villa	46.4	7.2	-11.8	0%	0.54%	7.6%
15.	Bournemouth	42.7	7.3	-23.0	0%	0.21%	19.79%
16.	Everton	38.8	6.9	-16.8	0%	0.02%	35.68%
17.	West Ham	38.2	6.8	-17.2	0%	0.02%	37.77%
18.	Southampton	37.5	7.1	-24.2	0%	0%	43.88%
19.	Nottingham Forest	35.4	6.9	-33.8	0%	0%	58.33%
20.	Wolves	31.1	6.5	-32.7	0%	0%	79.13%

By our model, the Premier League is essentially a two-horse race, with Arsenal leading with a 57.47% chance of victory. Last season's champions, Manchester City, are currently in second place with a 33.86% chance. Other teams such as Newcastle United and Tottenham Hotspur are still in the race but have an uphill climb to the top.

The above table showed mean values, which are conservative by nature. In the immense volume of 10,000 iterations, wild responses can, and will, occur. The below table shows the most extreme results for each team.

Table 5: Summary of Extreme Simulation Values

	Team	BestRank	WorstRank	MaxPoints	MinPoints	MaxGD	MinGD
1.	Arsenal	1	10	106	55	98	-6
2.	Man City	1	13	102	45	109	-4
3.	Newcastle	1	16	94	43	82	-11
4.	Liverpool	1	19	91	36	79	-29
5.	Spurs	1	17	91	40	73	-29
6.	Man United	1	19	90	36	53	-37
7.	Chelsea	1	20	85	30	53	-50
8.	Brighton	1	20	83	29	65	-36
9.	Brentford	1	20	81	29	45	-41
10.	Leeds	1	20	78	22	46	-54
11.	Fulham	2	20	78	24	47	-43
12.	Leicester	2	20	77	23	47	-56
13.	Crystal Palace	2	20	77	24	30	-50
14.	Aston Villa	2	20	74	25	34	-52
15.	Bournemouth	2	20	71	20	28	-69
16.	West Ham	3	20	67	16	24	-56
17.	Everton	4	20	65	17	20	-57
18.	Southampton	5	20	67	16	14	-65
19.	Nottingham Forest	5	20	63	16	3	-75
20.	Wolves	7	20	58	12	3	-70

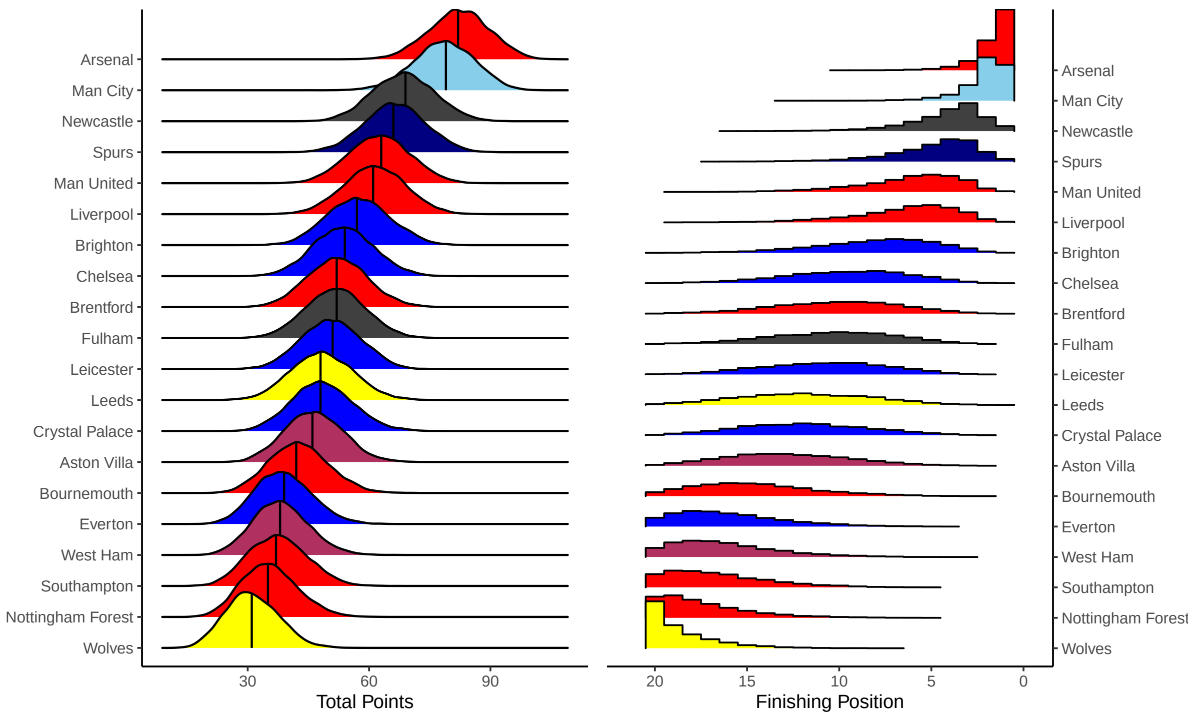
Despite the probabilistic two-horse race described above, the model demonstrates winning outcomes for ten different teams. On the flip side, only Arsenal, Man City, Newcastle, and Spurs are never relegated, with thirteen teams possibly finishing at the very bottom.

Only once has the 100 point mark ever been reached in a Premier League season, by Manchester City in 2017/18. There is still an avenue for both Arsenal and Manchester City to reach this mark, potentially eclipsing it and having the greatest season in history. Manchester City also had the best goal difference ever in that season with +79, which is in the realm of possibility for four clubs.

At the bottom, the fewest points ever recorded in a Premier League season was Derby County's 11 in 2007/08, a value below every team's minimum. Derby also had the worst goal difference ever that season at -69, a value that is unfortunately still possible for Bournemouth, Nottingham Forest, and Wolves.

The following graphs show the full distribution of points scored and finishing positions for each team:

Figure 3: Simulated Rest of Premier League Season 2022/23



Code for these tables and graphs can be found in the appendix.

Conclusion

In conclusion, our model takes into account each team's offensive and defensive abilities as well as the impact of home-and-away factors on team performance. It makes good use of goal data from the games already played this season to generate samples of the parameters and variables we need for each team. The final predictions are a good indicator of the teams' performances so far this season. For example, Arsenal has the best odds of winning the title this season (greater than 0.5) and Man City has the biggest sample mean of goal difference in our prediction. Both of teams are also the current leader in those respective categories.

However, there are some factors we didn't take into consideration in our model that might inhibit our predictive accuracy. For example, the future performance of some teams may be affected by injuries, manager changes, and winter acquisitions, especially after a month of the World Cup tournament. As a result, our predictions may be different from the actual league results in the future. Our prediction is also defined by the constraints of our prior data. We chose to only use the 14-15 games played for each team so far this season. Different popular season predictions may use other factors, such as previous season results, roster valuation, and schedule concentration from participation in outside tournaments.

In the future, we can use this exact model to predict the results in many of Europe's other leagues, such as Spain's La Liga and Germany's Bundesliga. Not only do these leagues follow identical schedule formats, but they begin and end at the same time in the calendar year. In general, this model can predict the results of any football league worldwide.

In its ultimate form, this can be translated into a publicly visible Shiny application that houses predictions for several football leagues around the world. Development of efficient data scraping techniques can permit the app to update daily, displaying the most recent and relevant information.

Not only did this project provide us with an interesting challenge, but it creates inspiration for future projects, deepening our knowledge and understanding of Bayesian Hierarchical Modeling while fueling one of our greatest interests.

References

[1] Lee, J., Kim, J., Kim, H., & Lee, J. S. (2022). A Bayesian Approach to Predict Football Matches with Changed Home Advantage in Spectator-Free Matches after the COVID-19 Break. Entropy (Basel, Switzerland), 24(3), 366. https://doi.org/10.3390/e24030366
[2] Azhari, Widyaningsih, Y., & Lestari, D. (2018). Predicting Final Result of Football Match Using Poisson Regression Model. Journal of Physics. Conference Series, 1108(1), 12066–. https://doi.org/10.1088/1742-6596/1108/1/012066
[3] Dingwei Wang. (2010). Soccer tournament simulation and analysis for South Africa World Cup with Poisson model of goal probability. 2010 Chinese Control and Decision Conference, Control and Decision Conference (CCDC), 2010 Chinese, 3654–3659. https://doi-org.proxy2.library.illinois.edu/10.1109/CCDC.2010.5498512
[4] English Premier League Stats, STATS, CRICKET, https://sports.ndtv.com/english-premier-league/stats /most-yellow-cards-team-statsdetail
[5] Shahtahmassebi, & Moyeed, R. (2016). An application of the generalized Poisson difference distribution to the Bayesian modelling of football scores. Statistica Neerlandica, 70(3), 260–273. https://doi.org/10.1111/stan.12087
[6] Fedrizzi, G., Canal, L., & Micciolo, R. (2022). UEFA EURO 2020: An exciting match between football and probability. Teaching Statistics, 44(3), 119–125. https://doi- org.proxy2.library.illinois.edu/10.1111/test.12315
[7] Footbal-Data.co.uk.,Accessed:2019-06-01. <URL: http://www.football-data.co.uk/englandm.php>
[8] L. A. Asimow and M. M. Maxwell. Probability and Statistics with Applications: A Problem Solving Text. 2nd. ACTEX Publications, 2015. ISBN: 9781625424723.
[9] K. Singh, S. Shastri, A. Bhadwal, et al. "Implementation of Exponential Smoothing for Forecasting Time Series Data". In: International Journal of Scientific Research in Computer Science Applications and Management Studies (Jan. 2019).
[10] S. Yang and G. Berdine. "Poisson Regression". In: The Southwest Respiratory and Critical Care Chronicles 3 (Jan. 2015), p. 61. DOI: 10.12746/swrccc.v3i9.191

Appendix

Code for Current Table The current table was copied and pasted from https://www.premierleague.com/tables and manually edited in excel to create the table shown in the report.

Code for Creating Data The data is derived from the real schedule and results from the Premier League so far in the 2022/23 season.

The results are found here: https://www.premierleague.com/results.

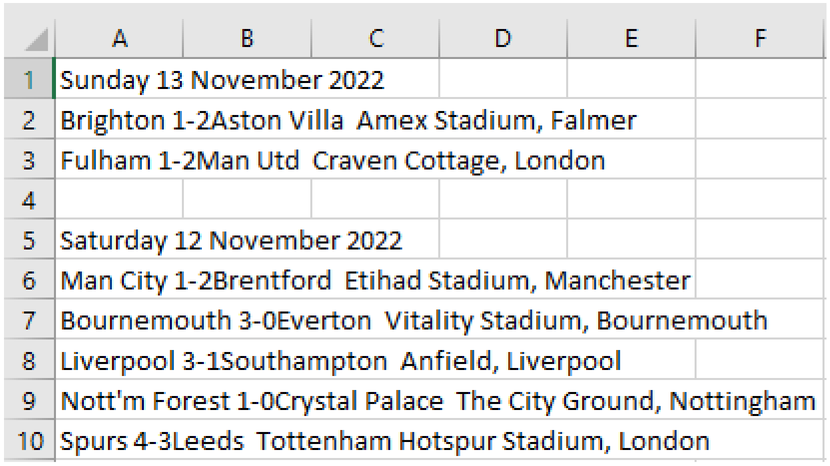
The tables in the website are copied and pasted into an Excel CSV, which looks as such:

Initial Clean

```
# load in packages and data
library(tidyverse)
games = read_csv("data/past-games-raw.csv")

# rename first column
names(games)[1] = "info"

# rename teams with two-word names
for(i in 1:nrow(games)) {
  games$info[i] = str_replace_all(games$info[i], "Aston Villa",
                                  "AstonVilla")
```



	A	B	C	D	E	F
1	Sunday 13 November 2022					
2	Brighton 1-2Aston Villa Amex Stadium, Falmer					
3	Fulham 1-2Man Utd Craven Cottage, London					
4						
5	Saturday 12 November 2022					
6	Man City 1-2Brentford Etihad Stadium, Manchester					
7	Bournemouth 3-0Everton Vitality Stadium, Bournemouth					
8	Liverpool 3-1Southampton Anfield, Liverpool					
9	Nott'm Forest 1-0Crystal Palace The City Ground, Nottingham					
10	Spurs 4-3Leeds Tottenham Hotspur Stadium, London					

Figure 2: Screenshot of Copied Code in Excel

```
  games$info[i] = str_replace_all(games$info[i], "Crystal Palace",
                               "CrystalPalace")
  games$info[i] = str_replace_all(games$info[i], "Man ", "Man")
  games$info[i] = str_replace_all(games$info[i], "Nott'm Forest",
                               "NottinghamForest")
  games$info[i] = str_replace_all(games$info[i], "West Ham", "WestHam")
}

# add new columns to fill later
games = {
  games %>%
    mutate(matchup = NA_character_,
           home = NA_character_,
           away = NA_character_,
           homeTeam = NA_character_,
           awayTeam = NA_character_,
           homeGoals = NA_integer_,
           awayGoals = NA_integer_)
}

# split info column by space
info_split = strsplit(games$info, " ")

# insert first split into matchup
for(i in 1:nrow(games)) {
  games$matchup[i] = info_split[[i]][1]
}

# remove date headers
games = {
  games %>%
    filter(matchup != "Sunday" &
           matchup != "Monday" &
           matchup != "Tuesday" &
           matchup != "Wednesday" &
```

```r
           matchup != "Thursday" &
           matchup != "Friday" &
           matchup != "Saturday")
}

# split matchup by hyphen
matchup_split = strsplit(games$matchup, "-")

# fill columns
for(i in 1:nrow(games)) {
  games$home[i] = matchup_split[[i]][1]
  games$away[i] = matchup_split[[i]][2]

  games$homeTeam[i] = substr(games$home[i], 1, nchar(games$home[i])-2)
  games$homeGoals[i] = as.numeric(substr(games$home[i],
                                         nchar(games$home[i]),
                                         nchar(games$home[i])))

  games$awayTeam[i] = substr(games$away[i], 2, nchar(games$away[i])-1)
  games$awayGoals[i] = as.numeric(substr(games$away[i], 1, 1))
}

# select necessary columns
games = {
  games %>%
    select(homeTeam, homeGoals, awayTeam, awayGoals)
}

knitr::kable(head(games,5),
             caption = "Sample of Initial Data Clean")
```

Table 6: Sample of Initial Data Clean

homeTeam	homeGoals	awayTeam	awayGoals
Brighton	1	AstonVilla	2
Fulham	1	ManUtd	2
ManCity	1	Brentford	2
Bournemouth	3	Everton	0
Liverpool	3	Southampton	1

Into Matrix

```r
# vector of teams
teams = sort(unique(games$homeTeam))
teams[c(2,7,13,14,16,19)] =
  c("Aston Villa", "Crystal Palace", "Man City",
    "Man United", "Nottingham Forest", "West Ham"
    )
# empty matrix
data = matrix(nrow = 20,
              ncol = 20)

# add team names to matrix
```

```r
rownames(data) = teams
colnames(data) = teams

# duplicate into home and away
home = data
away = data

# reformat data list
for(i in 1:20) {
  for(j in 1:20) {
    game = {
      games %>%
        filter(homeTeam == teams[i] &
                 awayTeam == teams[j])
    }
    if(nrow(game) == 0) {
      home[i, j] = NA
      away[i, j] = NA
    } else {
      home[i,j] = game$homeGoals[1]
      away[i,j] = game$awayGoals[1]
    }
  }
}

knitr::kable(home[c(1:5), c(1:5)],
             caption = "Sample of Data Cleaned into Matrix: home")
```

Table 7: Sample of Data Cleaned into Matrix: home

	Arsenal	Aston Villa	Bournemouth	Brentford	Brighton
Arsenal	NA	NA	NA	NA	NA
Aston Villa	NA	NA	NA	NA	NA
Bournemouth	0	NA	NA	0	NA
Brentford	0	NA	NA	NA	2
Brighton	NA	NA	NA	NA	NA

```r
knitr::kable(away[c(1:5), c(1:5)],
             caption = "Sample of Data Cleaned into Matrix: away")
```

Table 8: Sample of Data Cleaned into Matrix: away

	Arsenal	Aston Villa	Bournemouth	Brentford	Brighton
Arsenal	NA	NA	NA	NA	NA
Aston Villa	NA	NA	NA	NA	NA
Bournemouth	3	NA	NA	0	NA
Brentford	3	NA	NA	NA	0
Brighton	NA	NA	NA	NA	NA

```
model {
  for(i in 1:20) {
    for(j in 1:20) {
      YHome[i,j] ~ dpois(lambdaHome[i,j])
      YAway[i,j] ~ dpois(lambdaAway[i,j])
      log(lambdaHome[i,j]) <- muHome+alpha[i]-beta[j]
      log(lambdaAway[i,j]) <- muAway+alpha[j]-beta[i]
      resultsHome[i,j] <- ifelse(YHome[i,j] > YAway[i,j], 3,
                     ifelse(YHome[i,j] == YAway[i,j], 1,
                     ifelse(YHome[i,j] < YAway[i,j], 0, -1)))
      resultsAway[i,j] <- ifelse(YHome[i,j] > YAway[i,j], 0,
                     ifelse(YHome[i,j] == YAway[i,j], 1,
                     ifelse(YHome[i,j] < YAway[i,j], 3, -1)))
    }

    alpha[i] ~ dnorm(0, 1 / sigma.alpha^2)
    beta[i] ~ dnorm(0, 1 / sigma.beta^2)

    scoreHome[i] <- sum(YHome[i,]) - YHome[i,i]
    scoreAway[i] <- sum(YAway[,i]) - YAway[i,i]
    goalsScored[i] <- scoreHome[i] + scoreAway[i]

    allowHome[i] <- sum(YAway[i,]) - YAway[i,i]
    allowAway[i] <- sum(YHome[,i]) - YHome[i,i]
    goalsAllowed[i] <- allowHome[i] + allowAway[i]

    goalDif[i] <- goalsScored[i] - goalsAllowed[i]
    points[i] <- sum(resultsHome[i,]) + sum(resultsAway[,i]) - resultsHome[i,i] - resultsAway[i,i]

  }

  muHome ~ dnorm(0.0,1.0E-6)
  muAway ~ dnorm(0.0,1.0E-6)

  sigma.alpha ~ dexp(0.001)
  sigma.beta ~ dexp(0.001)
}
```

JAGS Model Code

```
# load libraries and data
library(rjags)
home = read.table("data/home.txt")
away = read.table("data/away.txt")

# set seed for reproducibility
set.seed(2023)

# declare data and initials
d =  list(YHome = home,
          YAway = away)
```

```
inits = list(list(muHome = 0, muAway = 0, sigma.alpha = 1000, sigma.beta = 1000),
             list(muHome = 1, muAway = -1, sigma.alpha = 0.1, sigma.beta = 0.1),
             list(muHome = -1, muAway = 1, sigma.alpha = 10, sigma.beta = 10))

# fit model
m = jags.model("R/model.bug", d, inits, n.chains = 3)
```

Code for Fitting JAGS Model

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 292
##    Unobserved stochastic nodes: 552
##    Total graph size: 5317
##
## Initializing model
```

```
# initial run and convergence check of muHome and muAway
x = coda.samples(m, c("muHome", "muAway"), n.iter=1000)

plot(x, smooth=FALSE, ask=TRUE)
```

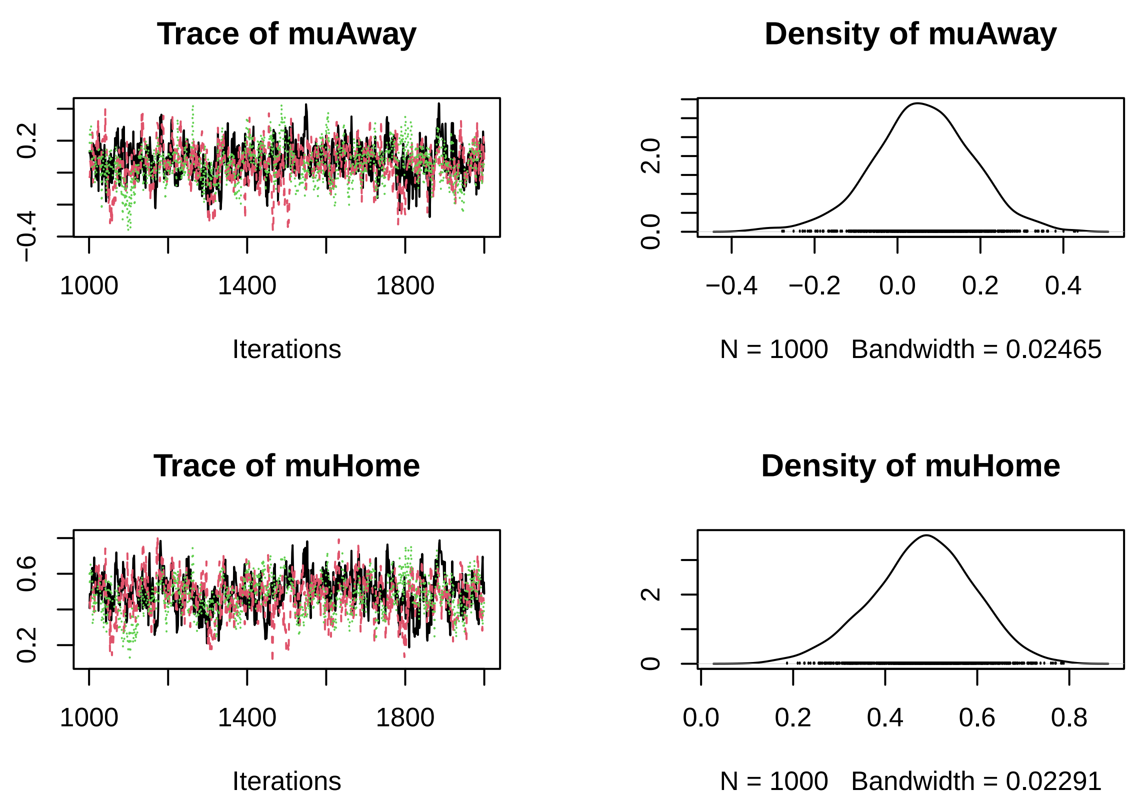


Figure 4: Convergence Plots of μ_H and μ_A

```
autocorr.plot(x[1], ask=TRUE)
```

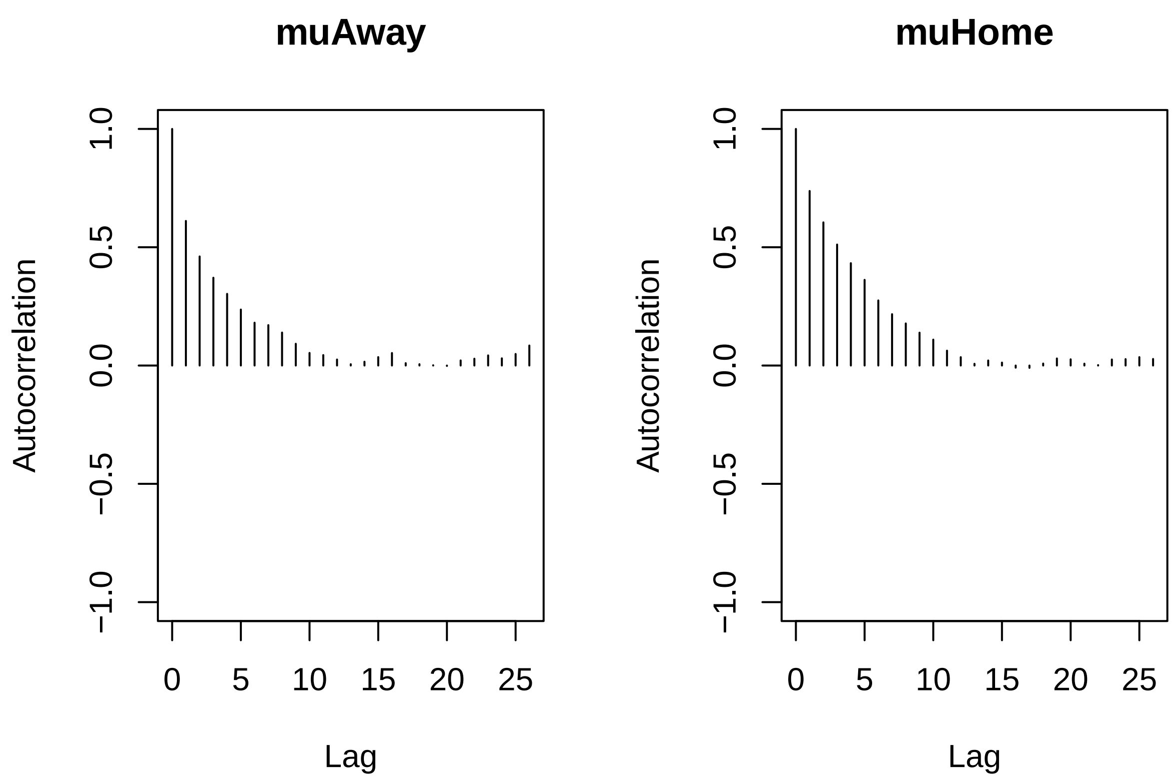


Figure 5: Autocorrelation Plots of μ_H and μ_A

```
# full run of 10,000 iterations
x = coda.samples(m, c("muHome", "muAway", "points", "goalDif"), n.iter = 10000)

# Final Value of muHome and muAway after burn in
as.matrix(summary(window(x, 2000))$statistics)[21:22,]
```

```
##              Mean        SD     Naive SE Time-series SE
## muAway 0.05918293 0.1246513 0.0007196748    0.002383765
## muHome 0.48405469 0.1163174 0.0006715588    0.002352721
```

Table 9: Final Summary Value of μ_H and μ_A

```
# Recall x from previous section
data = x[[1]]

# initialize ranking matrix
ranks_all = matrix(nrow = 10000, ncol = 20)
colnames(ranks_all) = teams
tables = list()

# extract rankings
for(i in 1:10000) {
  table = data.frame(Team = teams,
                     Points = data[i,23:42],
```

```r
                    GD = data[i,1:20]) %>%
    arrange(desc(Points), desc(GD))
  for(j in 1:20) {
    ranks_all[i, j] = which(table$Team == teams[j])
  }
  tables[[i]] = table
}

# initialize clean ranks matrix
ranks = matrix(nrow = 20,
               ncol = 20)
colnames(ranks) = teams
rownames(ranks) = as.character(seq(1, 20))

# extract clean ranks
for(i in 1:20) {
  for(j in 1:20) {
    ranks[i, j] = sum(ranks_all[,j] == i)
  }
}

# initialize mean and extreme table columns
gd = colMeans(data)[1:20]
pts = colMeans(data)[23:42]
sd = rep(NA_real_, 20)
win_pct = rep(NA_real_, 20)
ucl_pct = rep(NA_real_, 20)
rel_pct = rep(NA_real_, 20)
maxP = rep(NA_integer_, 20)
minP = rep(NA_integer_, 20)
maxG = rep(NA_integer_, 20)
minG = rep(NA_integer_, 20)
maxR = rep(NA_integer_, 20)
minR = rep(NA_integer_, 20)

# extract columns
for(i in 1:20) {
  sd[i] = sd(data[,22+i])
  win_pct[i] = sum(ranks[1,i]) / 10000
  ucl_pct[i] = sum(ranks[1:4,i]) / 10000
  rel_pct[i] = sum(ranks[18:20,i]) / 10000
  maxP[i] = max(data[,22+i])
  minP[i] = min(data[,22+i])
  maxG[i] = max(data[,i])
  minG[i] = min(data[,i])
  maxR[i] = max(ranks_all[,i])
  minR[i] = min(ranks_all[,i])
}

# create first, mean, extreme tables
s1 = data.frame(Team = teams,
                Points = data[1,23:42],
                GD = data[1, 1:20]) %>% arrange(desc(Points), desc(GD))
```

```r
rownames(s1) = paste0(as.character(seq(1,20,1)), ".")

mean_table = data.frame(Team = teams,
                   Points = round(pts,1),
                   SD = round(sd,1),
                   GoalDiff = round(gd,1),
                   First = paste0(as.character(win_pct*100),"%"),
                   UCL = paste0(as.character(ucl_pct*100),"%"),
                   Relegated = paste0(as.character(rel_pct*100),"%")
                   ) %>% arrange(desc(Points), desc(GoalDiff))
rownames(mean_table) = paste0(as.character(seq(1,20,1)), ".")

extremes = data.frame(Team = teams,
                      BestRank = minR,
                      WorstRank = maxR,
                      MaxPoints = maxP,
                      MinPoints = minP,
                      MaxGD = maxG,
                      MinGD = minG) %>% arrange(BestRank, desc(MaxPoints), desc(MaxGD))
rownames(extremes) = paste0(as.character(seq(1,20,1)), ".")
```

Code for Extracting Results into Tables

Code for Creating Graphs **Preparation**

```r
# load packages
library(tidyverse)
library(ggplot2)
library(ggridges)
library(gridExtra)

# create graphing data from previous data
points_all = as.data.frame(data[,23:42])
names(points_all) = teams
points_gg = gather(points_all) %>% arrange(key)

ranks_all = as.data.frame(ranks_all)
ranks_gg = gather(ranks_all) %>% arrange(key)

# team colors vector
team_colors = c("red", "skyblue", "gray25", "navy", "red",
                "red", "blue", "blue", "red", "gray25",
                "blue", "yellow", "blue", "maroon", "red",
                "blue", "maroon", "red", "red", "yellow")
names(team_colors) = mean_table$Team
```

Graphing

```r
# graph of team points
points_graph = {
  ggplot(points_gg, aes(x = value, y = key, fill = key)) +
    geom_density_ridges(color = "black", lwd = 0.6,
                        quantile_lines = TRUE, quantiles = 2) +
    scale_y_discrete(limits = rev(mean_table$Team)) +
    scale_fill_manual(values = team_colors) +
```

```r
    guides(fill = FALSE) +
    labs(x = "Total Points", y = "") +
    theme_classic()
}

# graph of team rankings
ranks_graph = {
  ggplot(ranks_gg, aes(x = value, y = key, fill = key)) +
    geom_density_ridges(stat = "binline", bins = 20,
                        scale = 2, draw_baseline = FALSE) +
    scale_x_reverse() +
    scale_y_discrete(limits = rev(mean_table$Team),
                     position = "right") +
    scale_fill_manual(values = team_colors) +
    guides(fill = FALSE) +
    labs(x = "Finishing Position", y = "") +
    theme(plot.title = element_text(hjust=0.5)) +
    theme_classic()
}

grid.arrange(points_graph, ranks_graph, ncol = 2,
             top = "Simulated Rest of Premier League Season 2022/23")
```