Comparing Rating Methods in NCAA Division III Women's Soccer

Hope Donoghue

Honors in Data Science 2023-2024

Advisor: Dr. Robin Lock

Department of Mathematics, Statistics, and Computer Science

St. Lawrence University

Abstract

Currently, the NCAA uses a metric called Rating Percentage Index (RPI) to rate teams in women's soccer. RPI considers a team's strength of schedule in addition to their number of wins and losses. Is this metric the most effective way to rate a team in soccer? This project aims to compare various rating methods for ranking Division III women's soccer teams. The rating methods used in this project are: RPI, Elo (originally developed for chess) and a model based on Poisson scoring rates. An interactive Shiny App, either sorted by region or league, displays the ratings generated by each method for the respective teams. We then simulate results for many seasons with known team abilities to compare how each rating method performs.

Table of Contents

Introduction

How do we effectively and fairly rank teams in Division III women's soccer when each team has differing schedules and opponents? At the moment, there are 435 teams and 10 regions in Division III women's soccer. Should we rank teams based off of highest number of points, highest points per game or even maybe highest win percentage? Currently, the NCAA in women's soccer uses a metric called RPI to decide which teams make it to the NCAA tournament at the end of the year. RPI considers a team's strength of schedule in addition to their number of wins and losses. The goal of this project is to compare various rating methods to one another and see which one performs the best. The rating methods we will look at are RPI, Elo and a Poisson scoring rate model. Below is a short description of each of the chapters.

In Chapter 1, we explain how we web scrape the data so that we can create a game results file for each season. Next in Chapter 2, we describe two functions that we built to generate a schedule file and a standings file from the game results file. The schedule file and the standings file can be subsetted either by region or by conference. In Chapter 3, we explain what RPI ratings are and demonstrate how to generate them. Likewise, in Chapter 4, we describe what Elo ratings are and how to calculate them. In Chapter 5, we explain the Poisson scoring rate model that we built and how to calculate an overall rating for each team based on their offensive and defensive ratings. In Chapter 6, we describe the process of making our Shiny App to display the various rating methods by any league or region. Lastly, in Chapter 7, we simulate results over many seasons to compare the rating methods to one another.

Chapter 1: Web Scraping Data

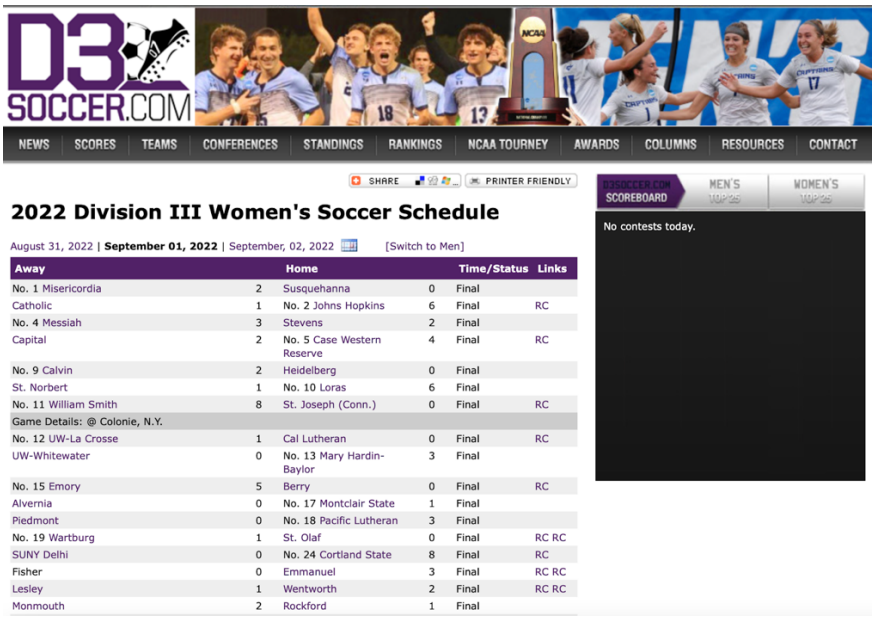
Our goal for this section is to get the data that we web scrape into a clean table that looks like the image below. We need five columns: an away team column, an away score column, a home team column, a home score column and a date column. Each row in the table represents a game played in the season.

	Away	Away Score	Home	Home Score	Date
1	Misericordia	2	Susquehanna	0	2022-09-01
2	Catholic	1	Johns Hopkins	6	2022-09-01
3	Messiah	3	Stevens	2	2022-09-01
4	Capital	2	Case Western Reserve	4	2022-09-01
5	Calvin	2	Heidelberg	0	2022-09-01
6	St. Norbert	1	Loras	6	2022-09-01
7	William Smith	8	St. Joseph (Conn.)	0	2022-09-01
8	UW-La Crosse	1	Cal Lutheran	0	2022-09-01
9	UW-Whitewater	0	Mary Hardin-Baylor	3	2022-09-01
10	Emory	5	Berry	0	2022-09-01
11	Alvernia	0	Montclair State	1	2022-09-01
12	Piedmont	0	Pacific Lutheran	3	2022-09-01
13	Wartburg	1	St. Olaf	0	2022-09-01
14	SUNY Delhi	0	Cortland State	8	2022-09-01
15	Fisher	0	Emmanuel	3	2022-09-01
16	Lesley	1	Wentworth	2	2022-09-01
17	Monmouth	2	Rockford	1	2022-09-01
18	Fitchburg State	3	Dean	0	2022-09-01

For this project, our data comes from two sources. We initially web scraped data from the website, *d3soccer.prestosports.com*. However, we soon found out that the data for this website ends at the 2022 season. Halfway through the project, we switched to the website, *masseyratings.com* to get data for the 2023 season. This section displays both sources and how we web scraped the data for the project.

Source 1: D3Soccer

The website, *d3soccer.prestosports.com*, provides game results from each day for a given season. The last season recorded on the website is the 2022 Division III soccer season. Figure 1.1 shows an example of what the website displays for games played on September 1, 2022.



Away		Home		Time/Status	Links
No. 1 Misericordia	2	Susquehanna	0	Final	
Catholic	1	No. 2 Johns Hopkins	6	Final	RC
No. 4 Messiah	3	Stevens	2	Final	
Capital	2	No. 5 Case Western Reserve	4	Final	RC
No. 9 Calvin	2	Heidelberg	0	Final	
St. Norbert	1	No. 10 Loras	6	Final	
No. 11 William Smith	8	St. Joseph (Conn.)	0	Final	RC
Game Details: @ Colonie, N.Y.					
No. 12 UW-La Crosse	1	Cal Lutheran	0	Final	RC
UW-Whitewater	0	No. 13 Mary Hardin-Baylor	3	Final	
No. 15 Emory	5	Berry	0	Final	RC
Alvernia	0	No. 17 Montclair State	1	Final	
Piedmont	0	No. 18 Pacific Lutheran	3	Final	
No. 19 Wartburg	1	St. Olaf	0	Final	RC RC
SUNY Delhi	0	No. 24 Cortland State	8	Final	RC
Fisher	0	Emmanuel	3	Final	RC RC
Lesley	1	Wentworth	2	Final	RC RC
Monmouth	2	Rockford	1	Final	

Figure 1.1: Overview of the *d3soccer.prestosports.com* results page for 9/1/2022

Importing the Games into R:

The game results are imported into R using the rvest package (1). The rvest package helps users web scrape data from web pages more easily by taking in an url link and storing the information into tables in R. A given url link for each day in the season is almost identical on the website. The only changes that occur in the url link is the date format at the end. For example, the url for the game results for September 1, 2022 is: https://d3soccer.prestosports.com/seasons/women/2022/schedule?date=2022-09-01.

The code below shows the web scrape function that takes in a given date for a season and compiles all the game results from that day into a single table.

```
function_webscrap <- function(date) {
  year <- substr(date, 1, 4)
  url <- paste("https://d3soccer.prestosports.com/seasons/women/", year,
"/schedule?date=", date, sep = "")
  g <- read_html(url)
  tab <- g %>% html_nodes("table")
  objs <- tab %>% html_table()
  table_clean <- modify_stats_function(objs[[1]], date)

  return(table_clean)
}
```

An example of how to call the above function to get the game results for September 1, 2022 is below.

```
game_results_sept_1_2022 <- function_webscrap("2022-09-01")
```

The tables are initially very messy when first imported into R using rvest. Figure 1.2 shows the messy table that results from web scraping the game results on September 1, 2022. The table is missing column names and there are rankings in the team columns that need to be eliminated to ensure consistency.

	X1	X2	X3	X4	X5
1	Away		Home		Time/Status
2	No. 1 Misericordia	2	Susquehanna	0	Final
3	Catholic	1	No. 2 Johns Hopkins	6	Final
4	No. 4 Messiah	3	Stevens	2	Final
5	Capital	2	No. 5 Case Western Reserve	4	Final
6	No. 9 Calvin	2	Heidelberg	0	Final
7	St. Norbert	1	No. 10 Loras	6	Final
8	No. 11 William Smith	8	St. Joseph (Conn.)	0	Final
9	@ Colonie, N.Y.	@ Colonie, N.Y.	@ Colonie, N.Y.	@ Colonie, N.Y.	@ Colonie, N.Y.
10	No. 12 UW-La Crosse	1	Cal Lutheran	0	Final
11	UW-Whitewater	0	No. 13 Mary Hardin-Baylor	3	Final
12	No. 15 Emory	5	Berry	0	Final
13	Alvernia	0	No. 17 Montclair State	1	Final
14	Piedmont	0	No. 18 Pacific Lutheran	3	Final
15	No. 19 Wartburg	1	St. Olaf	0	Final
16	SUNY Delhi	0	No. 24 Cortland State	8	Final
17	Fisher	0	Emmanuel	3	Final
18	Lesley	1	Wentworth	2	Final
19	Monmouth	2	Rockford	1	Final

Showing 1 to 20 of 186 entries, 6 total columns

Figure 1.2: Messy game results for 9/1/22

The *modify_stats_function_d3* can be used to clean up the game result tables. This function takes in the messy table and also the date of that table. The code below displays the *modify_stats_function_d3* function.

```
modify_stats_function_d3 <- function(date_results, date) {
date_results %>%
slice(-1) %>%
rename("Away" = X1) %>%
rename("Away Score" = X2) %>%
rename("Home" = X3) %>%
rename("Home Score" = X4)%>%
select(1:5) %>%
filter(X5 == "Final")%>%
select(1:4) %>%
# code below removes the numerical ranks in front of some of
the teams
mutate(Date = date) %>%
mutate(Home = str_remove(Home, "No\\. \\d+")) %>%
mutate(Away = str_remove(Away, "No\\. \\d+")) %>%
mutate(Home = str_remove(Home, "\r\n\r\n")) %>%
mutate(Away = str_remove(Away, "\r\n\r\n"))
}
```

When the *modify_stats_function_d3* is applied to the web scrape function, the game results are now cleaned and readable. Figure 1.3 shows the cleaned game results for September 1, 2022. Note that only 18 games out of 173 games played that day are shown.

	Away	Away Score	Home	Home Score	Date
1	Misericordia	2	Susquehanna	0	2022-09-01
2	Catholic	1	Johns Hopkins	6	2022-09-01
3	Messiah	3	Stevens	2	2022-09-01
4	Capital	2	Case Western Reserve	4	2022-09-01
5	Calvin	2	Heidelberg	0	2022-09-01
6	St. Norbert	1	Loras	6	2022-09-01
7	William Smith	8	St. Joseph (Conn.)	0	2022-09-01
8	UW-La Crosse	1	Cal Lutheran	0	2022-09-01
9	UW-Whitewater	0	Mary Hardin-Baylor	3	2022-09-01
10	Emory	5	Berry	0	2022-09-01
11	Alvernia	0	Montclair State	1	2022-09-01
12	Piedmont	0	Pacific Lutheran	3	2022-09-01
13	Wartburg	1	St. Olaf	0	2022-09-01
14	SUNY Delhi	0	Cortland State	8	2022-09-01
15	Fisher	0	Emmanuel	3	2022-09-01
16	Lesley	1	Wentworth	2	2022-09-01
17	Monmouth	2	Rockford	1	2022-09-01
18	Fitchburg State	3	Dean	0	2022-09-01

Figure 1.3: Cleaned game results for 9/1/22

To generate the game results for an entire season, a while loop can be used to read, clean and bind the game results from each date in the season together. The code below displays the loop

and how the 2022 game results file is constructed. The while loop starts after getting the cleaned game results table from September 1, 2022 and ends when the game results reach December 4, 2022, which was the last game of the 2022 season.

```
start_date <- as.Date("2022-09-02")
end_date <- as.Date("2022-12-04")
dates <- seq(start_date,end_date, by = "day")

D3_2022 <- function_webscrap("2022-09-01")

new_date <- start_date


while(new_date <= end_date){

  temp <- function_webscrap(as.character(new_date))
  D3_2022 <- rbind(D3_2022, temp)
  new_date = new_date + 1

}
```

Figure 1.4 displays some of the games in the 2022 game results file, D3_2022_FINAL.csv. The 2022 season had 3753 games played.

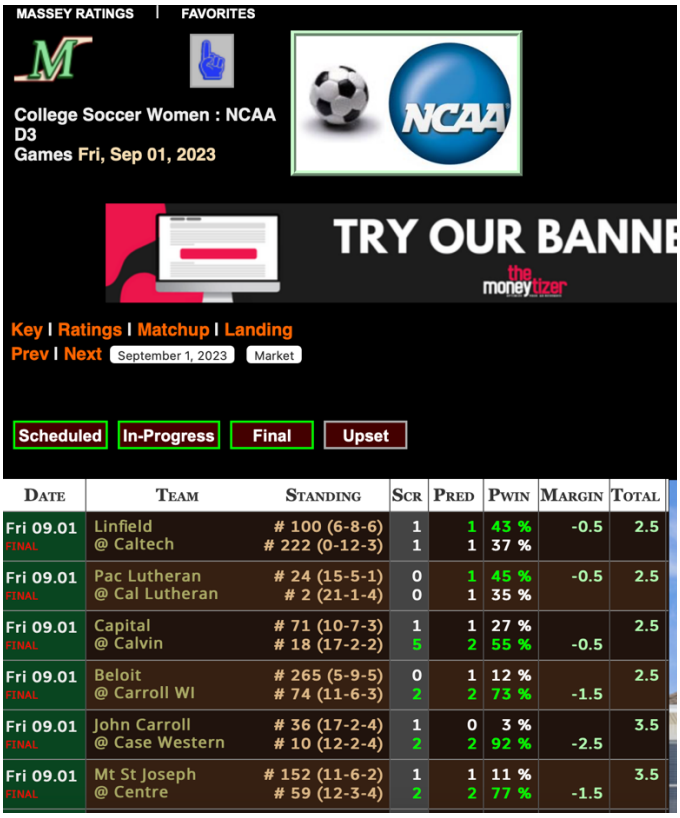
	Away	Away Score	Home	Home Score	Date
139	Chicago	0	Pomona-Pitzer	3	2022-09-01
140	SUNY-Cobleskill	0	Morrisville State	1	2022-09-01
141	Salem State	0	WPI	4	2022-09-01
142	Benedictine	0	North Park	2	2022-09-01
143	Millikin	5	Illinois College	0	2022-09-01
144	Aurora	0	North Central (Ill.)	1	2022-09-01
145	Greenville	0	Washington U.	2	2022-09-01
146	Carthage	1	Illinois Tech	2	2022-09-01
147	McMurry	1	Texas Lutheran	1	2022-09-01
148	St. Scholastica	0	Hardin-Simmons	2	2022-09-01
149	Centenary (La.)	0	East Texas Baptist	2	2022-09-01
150	Lewis and Clark	0	Whittier	1	2022-09-01
151	Whitworth	0	La Verne	3	2022-09-01
152	Saint Benedict	0	Christopher Newport	1	2022-09-02
153	Lynchburg	1	Rowan	1	2022-09-02
154	McDaniel	0	Virginia Wesleyan	1	2022-09-02
155	UW-Eau Claire	0	Wartburg	2	2022-09-02
156	Yeshiva	0	New Jersey City	10	2022-09-02
157	Alverno	0	SUNY-Maritime	7	2022-09-02

Showing 139 to 157 of 3,753 entries, 5 total columns

Figure 1.4: Sample of 2022 game results file

Source 2: Massey Ratings

Similarly the website, *masseyratings.com*, provides game results from each day for a given season. Figure 1.5 shows an example of what the website displays for games played on September 1, 2023 https://masseyratings.com/csocw/ncaa-d3/games?dt=20230901.



Date	Team	Standing	Scr	Pred	Pwin	Margin	Total
Fri 09.01 FINAL	Linfield	# 100 (6-8-6)	1	1	43 %	-0.5	2.5
	@ Caltech	# 222 (0-12-3)	1	1	37 %		
Fri 09.01 FINAL	Pac Lutheran	# 24 (15-5-1)	0	1	45 %	-0.5	2.5
	@ Cal Lutheran	# 2 (21-1-4)	0	1	35 %		
Fri 09.01 FINAL	Capital	# 71 (10-7-3)	1	1	27 %		2.5
	@ Calvin	# 18 (17-2-2)	5	2	55 %	-0.5	
Fri 09.01 FINAL	Beloit	# 265 (5-9-5)	0	1	12 %		2.5
	@ Carroll WI	# 74 (11-6-3)	2	2	73 %	-1.5	
Fri 09.01 FINAL	John Carroll	# 36 (17-2-4)	1	0	3 %		3.5
	@ Case Western	# 10 (12-2-4)	2	2	92 %	-2.5	
Fri 09.01 FINAL	Mt St Joseph	# 152 (11-6-2)	1	1	11 %		3.5
	@ Centre	# 59 (12-3-4)	2	2	77 %	-1.5	

Figure 1.5: Overview of the *masseyratings.com* results page for 9/1/2023

Importing the Games to R

The game results are imported into R using a ChromoteSession, which is a headless browser that allows the users to interact with the HTML code and web scrape the data off of it. The Massey Ratings website uses JavaScript to generate the HTML code. Using a ChromoteSession provides an alternative interface that runs a live web browser through Google Chrome to access the data tables (2). Within the live web browser, we use CSS selectors to help select the HTML elements from the tables that we want to web scrape (3). For example, ".vatop"

is a CSS selector that outputs the standings column and all the columns to the right in Figure 1.5. Likewise, “#tbl a” gives the date column and the team name column.

Below is the code that shows the web scrape function.

```
function_webscrap <- function(date) {
  url <- paste("https://masseyratings.com/csocw/ncaa-d3/games?dt=", date,
sep = "")
  c <- ChromoteSession$new()
  c$Page$navigate(url, wait = FALSE)
  c$Page$loadEventFired()

html <-
c$Runtime$evaluate("document.querySelector('html').outerHTML")$result$value
%>%
  read_html()

# use CSS selectors to get the elements we want from the tables
# .vatop pulls standings column plus all the columns to the right of it
  half_table <- ".vatop"

  html2 <- html %>%
  html_elements(half_table) %>%
  html_text2() %>%
  matrix(ncol = 6, byrow = TRUE) %>%
  data.frame()

# eliminate the repeated headers in the code on these lines
html2 <- html2[-c(1, 22, 43, 64, 85, 106, 127, 148, 169, 190, 211, 232,
253, 274, 295),]

# tbl a pulls the date and team name columns
half_table2 <- "#tbl a"

html3 <- html %>%
  html_elements(half_table2) %>%
  html_text2() %>%
  matrix(ncol = 4, byrow = TRUE) %>%
  data.frame()

schedule <- cbind(html2, html3)

# use a different modify_stats_function for Massey data
# similar structure to D3Soccer website modify_stats_function
table_clean <- modify_stats_function(schedule, date)

c$close()

return(table_clean)

}
```

An example of how to call the above function to get the game results for September 1, 2023 is below.

```
game_results_sept_1_2023 <- function_webscrap("20230901")
```

The tables are initially very cluttered when imported into R using ChromoteSession. Figure 1.6 displays the messy table from the game results on 9/1/23 for the first 20 games played on that day.

	X1	X2	X3	X4	X5	X6	X1.1	X2.1	X3.1	X4.1
2	# 100 (6-8-6) # 222 (0-12-3)	1 1	1 1	43 % 37 %	-0.5 ---	2.5 ---	Fri 09.01	FINAL	Linfield	@ Caltech
3	# 24 (15-5-1) # 2 (21-1-4)	0 0	1 1	45 % 35 %	-0.5 ---	2.5 ---	Fri 09.01	FINAL	Pac Lutheran	@ Cal Lutheran
4	# 71 (10-7-3) # 18 (17-2-2)	1 5	1 2	27 % 55 %	--- -0.5	2.5 ---	Fri 09.01	FINAL	Capital	@ Calvin
5	# 265 (5-9-5) # 74 (11-6-3)	0 2	1 2	12 % 73 %	--- -1.5	2.5 ---	Fri 09.01	FINAL	Beloit	@ Carroll WI
6	# 36 (17-2-4) # 10 (12-2-4)	1 2	0 2	3 % 92 %	--- -2.5	3.5 ---	Fri 09.01	FINAL	John Carroll	@ Case Western
7	# 152 (11-6-2) # 59 (12-3-4)	1 2	1 2	11 % 77 %	--- -1.5	3.5 ---	Fri 09.01	FINAL	Mt St Joseph	@ Centre
8	# 178 (2-13-5) # 56 (7-6-5)	0 2	1 2	27 % 55 %	--- -0.5	2.5 ---	Fri 09.01	FINAL	Willamette	@ Chapman
9	# 43 (20-2-3) # 75 (6-2-10)	2 0	1 1	38 % 43 %	--- -0.5	2.5 ---	Fri 09.01	FINAL	Milwaukee Eng	@ Albion
10	# 221 (8-8-1) # 21 (10-3-7)	0 3	0 2	5 % 86 %	--- -1.5	3.5 ---	Fri 09.01	FINAL	Fontbonne	@ Chicago
11	# 299 (4-12-2) # 313 (13-3-1)	0 2	2 1	70 % 17 %	-1.5 ---	3.5 ---	Fri 09.01	FINAL	Albright	@ Bryn Athyn
12	# 5 (20-1-1) # 157 (5-10-4)	3 1	2 1	78 % 10 %	-1.5 ---	2.5 ---	Fri 09.01	FINAL	Chris Newport	WI Oshkosh
13	# 97 (6-9-5) # 52 (7-7-3)	0 2	1 1	33 % 48 %	--- -0.5	2.5 ---	Fri 09.01	FINAL	George Fox	@ Claremont M.S.
14	# 182 (5-13-1) # 61 (11-5-2)	0 1	1 1	39 % 43 %	--- -0.5	3.5 ---	Fri 09.01	FINAL	DeSales	@ Clarkson
15	# 226 (10-4-7) # 283 (12-7-3)	1 1	2 1	55 % 27 %	-0.5 ---	3.5 ---	Fri 09.01	FINAL	Keene St	@ Colby-Sawyer
16	# 193 (5-6-6) # 84 (7-4-7)	0 2	0 1	5 % 84 %	--- -1.5	2.5 ---	Fri 09.01	FINAL	Alfred	@ Geneseo St
17	# 144 (8-11-1) # 40 (16-4-1)	0 2	1 2	14 % 72 %	--- -1.5	2.5 ---	Fri 09.01	FINAL	WI Stevens Pt	@ St Catherine
18	# 122 (10-3-5) # 300 (3-12-3)	3 0	2 1	70 % 15 %	-1.5 ---	2.5 ---	Fri 09.01	FINAL	WI Platteville	@ Concordia IL
19	# 260 (6-10-1) # 150 (7-8-2)	0 2	1 2	19 % 66 %	--- -0.5	2.5 ---	Fri 09.01	FINAL	Allegheny	@ Oberlin
20	# 53 (11-3-5) # 79 (9-5-4)	1 1	1 0	54 % 27 %	-0.5 ---	2.5 ---	Fri 09.01	FINAL	Cortland St	@ Rensselaer

Figure 1.6: Messy game results table for 9/1/23

To clean the game result tables, a modify_stats_function specific to Massey Ratings data can be employed. The *modify_stats_function* creates five columns: home_team, home_score, away_team, away_score and date. The code below shows the *modify_stats_function*. Figure 1.7 displays the cleaned game results for the first 20 games played on 9/1/23.

```
modify_stats_function <- function(date_results, date) {
   date_results <- date_results
names(date_results)[2] <- "score"
names(date_results)[9] <- "home_team"
names(date_results)[10] <- "away_team"
names(date_results)[7] <- "date"

d3_schedule <- date_results %>%
  select(score, home_team, away_team, date) %>%
  separate(score, into = c("home_score", "away_score"), sep = "\n") %>%
  mutate(away_team = str_remove(away_team, "@")) %>%
  mutate(away_team = str_trim(away_team, "left")) %>%
  select(home_team, home_score, away_team, away_score, date)

}
```

away_team	away_score	home_team	home_score	date
Linfield	1	Caltech	1	Fri 09.01
Pac Lutheran	0	Cal Lutheran	0	Fri 09.01
Capital	1	Calvin	5	Fri 09.01
Beloit	0	Carroll WI	2	Fri 09.01
John Carroll	1	Case Western	2	Fri 09.01
Mt St Joseph	1	Centre	2	Fri 09.01
Willamette	0	Chapman	2	Fri 09.01
Milwaukee Eng	2	Albion	0	Fri 09.01
Fontbonne	0	Chicago	3	Fri 09.01
Albright	0	Bryn Athyn	2	Fri 09.01
Chris Newport	3	WI Oshkosh	1	Fri 09.01
George Fox	0	Claremont M.S.	2	Fri 09.01
DeSales	0	Clarkson	1	Fri 09.01
Keene St	1	Colby-Sawyer	1	Fri 09.01
Alfred	0	Geneseo St	2	Fri 09.01
WI Stevens Pt	0	St Catherine	2	Fri 09.01
WI Platteville	3	Concordia IL	0	Fri 09.01
Allegheny	0	Oberlin	2	Fri 09.01
Cortland St	1	Rensselaer	1	Fri 09.01
Transylvania	3	Covenant	1	Fri 09.01

Figure 1.7: Cleaned game results for 9/1/23

The final part is to generate the game results for an entire season, in this case, the 2023 season. As with the data from D3Soccer in Source 1, we use a while loop to read, clean and bind game results from each date in the season together. The while loop starts after getting the cleaned game results table from September 1, 2023 and ends when the game results reach November 30, 2023.

Figure 1.8 displays some of the games in the 2023 game results file, D3_2023.csv. The 2023 season had 3964 games played.

away_team	away_score	home_team	home_score	date
Lynchburg	2	Guilford	0	Sat 10.14
Immaculata	0	Gwynedd-Mercy	1	Sat 10.14
Carleton MN	5	Hamline	0	Sat 10.14
Hanover	6	Bluffton	1	Sat 10.14
Otterbein	4	Heidelberg	0	Sat 10.14
Hollins	0	Shenandoah	9	Sat 10.14
Houghton	2	Hartwick	0	Sat 10.14
Husson	2	Lesley	4	Sat 10.14
Lawrence	4	Illinois Col	2	Sat 10.14
North Park	0	IL Wesleyan	1	Sat 10.14
Skidmore	0	Ithaca	1	Sat 10.14
Johns Hopkins	1	Frank & Marsh	0	Sat 10.14
N England Col	1	J&W RI	8	Sat 10.14
Asbury	0	Piedmont	2	Sat 10.14
Scranton	1	Juniata	0	Sat 10.14

Figure 1.8: Sample of 2023 game results file

Note: For the remainder of this project, we use data from the website, *masseyratings.com* (Source 2).

Chapter 2: Schedule/Standings

Schedule Function

The schedule function enables the user to subset the game results file for any season by various conferences and regions. Each of the game results files consist of 45 leagues and 10 regions that the user can filter by. The schedule function takes in a game results file, a team list that states all the Division III women's soccer teams, a conference, a region and whether the games in the outputted schedule are league only games or also out of league games. The default schedule that the function outputs is all the games results from the 2022 season. To generate a different schedule, the user can alter the *schedule, conf* or *reg* parameters. The code for the schedule function is below.

```
schedule_function <- function(schedule = D3_2022, team_list =
D3_teams_massey, conf = "all", reg = "all", league_only = TRUE){

  if(conf != "all"){
        team_list <- team_list %>%
          filter(Conference == conf)
      }

      if(reg != "all"){
        team_list <- team_list %>%
          filter(Region == reg)
      }

      is_team_in_league_AWAY <- ifelse(schedule[['away_team']] %in%
team_list$School, TRUE, FALSE)
      is_team_in_league_HOME <- ifelse(schedule[['home_team']] %in%
team_list$School, TRUE, FALSE)
      df_Away <- as.data.frame(is_team_in_league_AWAY)
      df_Home <- as.data.frame(is_team_in_league_HOME)
      df_HOME_AWAY <- bind_cols(df_Away, df_Home, schedule)

# schedule of teams in league since league_only is true
      if(league_only){

        games <- df_HOME_AWAY %>%
          filter(is_team_in_league_AWAY == TRUE & is_team_in_league_HOME ==
TRUE) %>%
          select(away_team, away_score, home_team, home_score, date)
```

```
} else { # want out of conference team schedule

        games <- df_HOME_AWAY %>%
          filter(is_team_in_league_AWAY == TRUE | is_team_in_league_AWAY ==
TRUE) %>%
          select(away_team, away_score, home_team, home_score, date)
      }

      return (games)
}
```

For example, to generate a schedule of game results from the 2022 Liberty League (LL) season, the user can change *conf* = "all" to *conf* = "LL". Below is a code chunk that shows this along with the resulting schedule.

```
LL_schedule_2022 <- schedule_function(conf = "LL")
```

	away_team	away_score	home_team	home_score	date
1	Rensselaer	0	St Lawrence	0	Sat 09.17
2	Union NY	1	Bard	0	Tue 09.20
3	Clarkson	0	Skidmore	1	Wed 09.21
4	Rochester Tech	1	Hobart & Smith	4	Wed 09.21
5	Ithaca	2	Clarkson	0	Sat 09.24
6	Hobart & Smith	1	Skidmore	1	Sat 09.24
7	Rensselaer	3	Bard	0	Sat 09.24
8	Union NY	0	St Lawrence	1	Sat 09.24
9	Vassar	1	Rochester Tech	0	Sat 09.24
10	Hobart & Smith	3	Ithaca	0	Wed 09.28
11	Bard	0	Vassar	7	Wed 09.28
12	Clarkson	0	Rochester Tech	0	Sat 10.01
13	Bard	0	Hobart & Smith	5	Sat 10.01
14	St Lawrence	0	Ithaca	1	Sat 10.01
15	Rensselaer	0	Skidmore	1	Sat 10.01
16	Vassar	0	Union NY	1	Sat 10.01
17	Union NY	1	Rensselaer	1	Fri 10.07
18	Hobart & Smith	2	Clarkson	0	Sat 10.08
19	Bard	0	Ithaca	6	Sat 10.08
20	Rochester Tech	3	St Lawrence	1	Sat 10.08

Figure 2.1: First 20 games out of 45 games from 2022 LL schedule

Another example of how to generate a schedule for all games played by teams in Region III in the 2023 season is below.

```
region_3_2023 <- schedule_function(schedule = D3_2023, reg = "Region III",
league_only = FALSE)
```

	away_team	away_score	home_team	home_score	date
1	Alfred	0	Geneseo St	2	Fri 09.01
2	Cortland St	1	Rensselaer	1	Fri 09.01
3	Elmira	1	Ithaca	10	Fri 09.01
4	Keuka	0	Buffalo St	3	Fri 09.01
5	Nazareth	1	St Lawrence	1	Fri 09.01
6	Rochester NY	3	Rochester Tech	0	Fri 09.01
7	Brockport St	0	Skidmore	1	Fri 09.01
8	Hartwick	1	SUNY Oneonta	1	Fri 09.01
9	Mt St Vincent	0	Vassar	4	Fri 09.01
10	Russell Sage	0	SUNY New Paltz	2	Fri 09.01
11	Yeshiva	0	St Joseph's NY	4	Fri 09.01
12	Nazareth	0	Clarkson	3	Sat 09.02
13	Buffalo St	2	Houghton	1	Sat 09.02
14	St John Fisher	1	Skidmore	1	Sat 09.02
15	Plattsburgh St	3	Rochester Tech	1	Sun 09.03
16	SUNY New Paltz	1	Rensselaer	2	Sun 09.03
17	Old Westbury	6	Yeshiva	1	Sun 09.03
18	Cortland St	0	Rochester NY	2	Mon 09.04
19	Alfred	1	SUNY Oneonta	4	Tue 09.05
20	Farmingdale	12	Yeshiva	0	Tue 09.05

Figure 2.2: First 20 games out of 275 from 2023 Region III schedule

Standings Function

In soccer, three points are awarded for a win, one point for a tie and zero points for a loss. A standings table is used to keep track of the number of points a team gets in a given season. The standings function works in a similar way to the schedule function above. The function takes in the same parameters: a game results file, a team list, a conference, a region and whether the

games from the schedule should include only in league games or also include out of league games. Below is an outline of the function.

```
# standings_function outline

standings_function <- function(schedule = D3_2022, team_list =
D3_teams_massey, conf = "all", reg = "all", league_only = TRUE){

# get each team's home goals for and against, away goals for and against
and total goals/goal differential

# get each team's home points, away points and total points

# compile statistics into standings table and return the table

return (standings)

}
```

The first part of the function calculates the goals scored and conceded by each team in the conference or region specified. These calculations are used to find goal differential which is a metric used if two teams are tied on the same number of points. The team with the higher goal differential gets ranked above the other team.

```
# to get the home goals a team scores and concedes
home_goals <- games %>%
group_by(home_team) %>%
summarize(home_goals_for = sum(home_score, na.rm = TRUE),
home_goals_against = sum(away_score, na.rm = TRUE)) %>%
rename("School" = home_team)

standings <- left_join(team_list, home_goals, by = "School")


# to get the away goals a team scores and concedes
away_goals <- games %>%
group_by(away_team) %>%
summarize(away_goals_for = sum(away_score, na.rm = TRUE),
away_goals_against = sum(home_score, na.rm = TRUE)) %>%
rename("School" = away_team)

standings <- left_join(standings, away_goals, by = "School")


# to get total goals and goal differential for each team
standings <- standings %>%
mutate(goals_scored = (home_goals_for + away_goals_for), goals_conceded =
(home_goals_against + away_goals_against), goal_differential = goals_scored
- goals_conceded)
```

The next part of the function calculates points: a team's home points, a team's away points and then a team's total number of points. These calculations along with the goal calculations from above help find the average number of points a team receives each game (PPG) and their respective winning percentages. We denote winning percentage as:

$$(\text{wins} + 0.5 * \text{ties}) / \text{total\_games}$$

```
# to get a team's home points
home_points <- games %>%
  group_by(home_team) %>%
  mutate(points = ifelse(home_score > away_score, 3,
                         ifelse(home_score == away_score, 1, 0))) %>%
  summarize(home_points = sum(points, na.rm = TRUE),
            home_wins = sum(points == 3, na.rm = TRUE),
            home_ties = sum(points == 1, na.rm = TRUE),
            home_losses = sum(points == 0, na.rm = TRUE)) %>%
  rename("School" = home_team)

standings <- left_join(standings, home_points, by="School")

# to get a team's away points
away_points <- games %>%
  group_by(away_team) %>%
  mutate(points = ifelse(away_score > home_score, 3,
                         ifelse(home_score == away_score, 1, 0))) %>%
  summarize(away_points = sum(points, na.rm = TRUE),
            away_wins = sum(points == 3, na.rm = TRUE),
            away_ties = sum(points == 1, na.rm = TRUE),
            away_losses = sum(points == 0, na.rm = TRUE)) %>%
  rename("School" = away_team)

standings <- left_join(standings, away_points, by = "School")

# to get total points
standings <- standings %>%
  mutate(points = home_points + away_points,
         wins = home_wins + away_wins,
         ties = home_ties + away_ties,
         losses = home_losses + away_losses,
         record = paste(wins, losses, ties, sep = "-"),
         games_played = wins + ties + losses,
         ppg = round((points / games_played),3),
         goal_differential_pg = round((goal_differential /
games_played),3),
         win_percentage = round(((wins + (0.5*ties))/games_played),3)
         )
```

The final part of the standings function organizes the output into a table. The table is sorted by total number of points.

```
# to get standings
standings <- standings %>%
  arrange(desc(points), desc(goal_differential)) %>%
  select(School, points, goal_differential, goals_scored, goals_conceded,
wins, losses, ties, record, games_played, ppg, goal_differential_pg,
win_percentage) %>%
  rename(Points = points,
'Goal Differential' = goal_differential, 'Goals Scored' = goals_scored,
'Goals Conceded' = goals_conceded,
Wins = wins,
Losses = losses,
Ties = ties,
'Record (W-L-T)' = record,
Games = games_played,
PPG = ppg,
'Goal Differential Per Game' = goal_differential_pg,
'Win Percentage' = win_percentage)

return(standings)

}
```

An example of the code for generating a standings table for the 2023 Liberty League season is below.

```
ll_standings_2023 <- standings_function(schedule = D3_2023, conf = "LL")
```

	School	Points	Goal Differential	Goals Scored	Goals Conceded	Wins	Losses	Ties	Record (W-L-T)	Games	PPG	Goal Differential Per Game	Win Percentage
1	Hobart & Smith	27	26	28	2	8	0	3	8-0-3	11	2.455	2.364	0.864
2	Rochester Tech	22	15	21	6	6	1	4	6-1-4	11	2.000	1.364	0.727
3	Ithaca	21	8	15	7	6	2	3	6-2-3	11	1.909	0.727	0.682
4	Clarkson	15	2	12	10	5	5	0	5-5-0	10	1.500	0.200	0.500
5	Rensselaer	14	-8	11	19	4	5	2	4-5-2	11	1.273	-0.727	0.455
6	Vassar	11	0	3	3	2	2	5	2-2-5	9	1.222	0.000	0.500
7	St Lawrence	11	-9	5	14	3	5	2	3-5-2	10	1.100	-0.900	0.400
8	Union NY	8	-3	8	11	2	5	2	2-5-2	9	0.889	-0.333	0.333
9	Skidmore	7	-1	9	10	1	4	4	1-4-4	9	0.778	-0.111	0.333
10	Bard	1	-30	1	31	0	8	1	0-8-1	9	0.111	-3.333	0.056

Figure 2.3: Standings Table for 2023 LL season

We create a standings2 function that computes the same output, but instead of taking in multiple parameters (a game results file, a team list, a conference, a region and whether the games from the schedule are in league or also out of league games), the function only takes in a game results file. Below is an outline of the code for the function.

```
# standings2 function outline

standings2 <- function(games){

# get every teams' home goals for and against, away goals for and against
and total goals/goal differential

# get every teams' home points, away points and total points

# compile statistics into standings table and return the table

return(standings)
}
```

An example of how to use the standings2 function to get a standings table for the 2023 Liberty League season is below. Note that the only difference between the standings2 function and the standings_function above is, the standings2 function requires a game results file that has already been filtered by league or region to be passed in.

```
ll_standings_2023_2 <- standings2(games = LL_2023_schedule)
```

Chapter 3: RPI Ratings

Rating Percentage Index (RPI) is a method used to rank teams based off a team's winning percentage and their strength of schedule. Currently, the NCAA uses RPI based on winning percentage to rank teams in Division III women's soccer. The formula below displays how RPI is calculated.

$$RPI = 0.25 * WP + 0.50 * OWP + 0.25 * OOWP$$

In this formula, WP refers to the team's winning percentage. Winning percentage is calculated by taking a team's wins plus 0.5 times a team's ties divided by the number of games that they have played (4). OWP refers to the team's opponents' winning percentage, which is the average WP of all opponents played. OOWP refers to the team's opponents' opponents' winning percentage. OOWP is calculated by taking the average OWP for each team. The code for RPI WP function can be found below. The function takes in a game results file and each of the weights from the formula (w1, w2 and w3). Note the default settings correspond to the weights from the NCAA's formula, so no need to pass in those parameters to the function.

```
rpi_function_wp <- function(games, w1=(0.25), w2=(0.5), w3=(0.25)){

# get the standings table from the game results file that was passed in
  stand <- standings2(games)

# rename the columns in the game results file
  games <- games %>%
    rename(team = home_team, opponent = away_team, team_score = home_score,
opp_score = away_score)

# duplicate the game results file and switch the labels, so team and
opponent are now switched along with team_score and opp_score
  games_2 <- games %>%
    rename(team = opponent, opponent = team, team_score = opp_score,
opp_score = team_score)

# combine the two data frames together to better calculate win percentage
for opponents and opponents of the opponents
  double_table <- bind_rows(games, games_2)
```

```
# calculate opponents' wp
wp1 <- left_join(double_table, stand, by = c("opponent" = "School")) %>%
    select(team, PPG, 'Win Percentage')%>%
    rename(opp_wp = 'Win Percentage')

  opp_wp <- wp1 %>%
    group_by(team) %>%
    mutate(avg_opp_wp = mean(opp_wp)) %>%
    distinct(team, .keep_all = T) %>%
    select(team, avg_opp_wp)

new_teams_wp <- left_join(stand, opp_wp, by = c("School" = "team")) %>%
  rename("team" = "School")

# calculate opponents' opponents' wp
  wp2 <- left_join(double_table, new_teams_wp, by = c("opponent" =
"team")) %>%
    select(-c(date, opp_score, team_score)) %>%
    rename(opp_wp = 'Win Percentage')

  new_opp_wp <- wp2 %>%
    group_by(team) %>%
    mutate(avg_opp_opp_wp = mean(avg_opp_wp)) %>%
    distinct(team, .keep_all = T) %>%
    select(team, avg_opp_opp_wp)

# combine together to calculate RPI metric and outputted table
  full_teams_wp <- left_join(new_teams_wp, new_opp_wp, by = "team") %>% #
to add opponents opponents WP
    rename(WP = 'Win Percentage') %>%
    mutate(RPI = (WP*w1+avg_opp_wp*w2+avg_opp_opp_wp*w3)) %>%
    arrange(desc(RPI)) %>%
    select(team, RPI, WP, avg_opp_wp, avg_opp_opp_wp) %>%
    rename('Team' = team, 'RPI' = RPI, 'Winning Percentage (WP)' = WP,
'Opponent\'s WP' = avg_opp_wp, 'Opponent\'s Opponent\'s WP' =
avg_opp_opp_wp) %>%
    mutate_if(is.numeric, round, digits = 2)

  return(full_teams_wp)
}
```

An example of the code and output table to find the RPI WP ratings for the 2023 Liberty League season is below.

```
ll_rpi_wp_2023 <- rpi_function_wp(games = LL_2023_schedule)
```

	Team	RPI	Winning Percentage (WP)	Opponent's WP	Opponent's Opponent's WP
1	Hobart & Smith	0.58	0.86	0.47	0.51
2	Rochester Tech	0.56	0.73	0.52	0.49
3	Ithaca	0.54	0.68	0.48	0.50
4	Rensselaer	0.50	0.45	0.52	0.49
5	Vassar	0.49	0.50	0.48	0.50
6	Clarkson	0.49	0.50	0.48	0.50
7	St Lawrence	0.48	0.40	0.51	0.50
8	Union NY	0.46	0.33	0.50	0.50
9	Skidmore	0.46	0.33	0.50	0.50
10	Bard	0.40	0.06	0.53	0.50

Figure 3.1: RPI WP Ratings for 2023 LL season

From Figure 3.1, we see that Rensselaer has a RPI WP rating of 0.50, while Clarkson has a RPI WP rating of 0.49. Clarkson has the better record with regards to winning percentage (5 wins, 5 losses and 0 ties) compared to Rensselaer's record (4 wins, 5 losses, 2 ties). However, we see that Rensselaer has a higher RPI WP rating. This is because Rensselaer has a higher OWP and in the RPI WP formula, OWP gets 50% of the weight. Playoffs games are included in the dataset and Rensselaer played an additional game compared to Clarkson. Rensselaer played Hobart & Smith during the second round of the playoffs on Friday, November 3, 2023. Hobart & Smith has the highest WP in the Liberty League, so naturally Rensselaer's OWP will increase when they play them.

Another way to account for ties differently is to use points per game (PPG) instead of win percentage in the RPI formula. The new formula appears below.

$$RPI = 0.25 * PPG + 0.50 * OPPG + 0.25 * OOPPG$$

This time, ties are only given 1/3 the weight of a win in the RPI PPG formula. The code for the RPI PPG function is almost identical to the RPI WP function above, except instead of using WP as the metric, we are now using PPG. The weights on the components stay the same. Below is an

example of what the code looks like to generate a RPI PPG ratings table for the 2023 Liberty League season.

```
ll_rpi_ppg_2023 <- rpi_function_ppg(games = LL_2023_schedule)
```

	Team	RPI	Points per game (PPG)	Opponent's PPG	Opponent's Opponent's PPG
1	Hobart & Smith	1.60	2.45	1.28	1.39
2	Rochester Tech	1.55	2.00	1.42	1.35
3	Ithaca	1.48	1.91	1.31	1.38
4	Rensselaer	1.38	1.27	1.45	1.35
5	Clarkson	1.37	1.50	1.30	1.39
6	St Lawrence	1.32	1.10	1.40	1.36
7	Vassar	1.32	1.22	1.33	1.37
8	Union NY	1.25	0.89	1.37	1.37
9	Skidmore	1.23	0.78	1.38	1.37
10	Bard	1.10	0.11	1.46	1.36

Figure 3.2: RPI PPG Ratings for 2023 LL season

We see from Figure 3.2 that Clarkson has a higher RPI PPG compared to Vassar. In Figure 3.1, Vassar and Clarkson have the same RPI WP of 0.49 as their WP's are identical. It is interesting how using PPG in the RPI formula instead of WP, significantly changes the rankings. Clarkson has a higher PPG than Vassar because they have no ties compared to Vassar's five. Vassar's record was 2 wins, 2 losses and 5 ties.

Chapter 4: Elo Ratings

Elo is another popular method used for ranking teams. Elo was originally created in the 1960s to rank players in chess, but more recently has been applied to other contexts like sports (5). The basic idea of how Elo ratings work is quite simple. Each team starts with an initial rating of 1500 points. Before a match is played between two teams, an expected win probability (E) is calculated based on the ratings for each of the teams. *Ra* refers to the rating for Team A and *Rb* refers to the rating for Team B.

$$E = \frac{1}{1 + 10^{\frac{Ra - Rb}{400}}}$$

After every match, both teams either gain points or lose points depending on the outcome (O). If a team beats a stronger opponent, then the team receives more points than what they would receive if they beat a weaker opponent. The update ratings formula is below, where HA is a home field adjustment and K is a scaling factor. O is the outcome of the match (1 for win, 0.5 for tie and 0 for loss).

$$R_{new} = R_{old} + HA + K*(O-E)$$

To implement Elo in R, we use the elo package and more specifically the elo.run function (6). The elo.run function updates each team's Elo rating after every match is finished (6). The score function within the elo.run function calculates the outcome of the match and returns 1 for a win, 0.5 for a tie and 0 for a loss. The code for the elo function can be found below. The function takes in a game results file, a home advantage value and a K scaling factor (k). The default home advantage is set at 30. The default k value is set at 20 plus the absolute value of the difference between the home score and the away score times 10. In other words, k adjusts based off the

score of the game. Since the scores in soccer are relatively small, 20 is added to make the value of k larger and the adjustment more noticeable.

```
# Elo ratings function
elo_rating_auto <- function(schedule, KValue = 20, HomeAdv = 30) {

elo_ratings <- elo.run(score(home_score, away_score) ~ adjust(home_team,
HomeAdv) + away_team + k(KValue + abs(home_score - away_score)*10), data =
schedule)

# make elo_ratings a matrix
elo_object <- as.matrix(elo_ratings)

# select only the last row
elo_object <- elo_object %>%
  tail(n=1)

# use the transpose function in R to switch columns and rows
elo_object <- elo_object %>%
  t()

elo_df <- as.data.frame(elo_object) %>%
  cbind(rownames(elo_object), elo_object) %>%
  select(2:3) %>%
  rename(Team = "rownames(elo_object)", "Elo Rating" = "elo_object") %>%
  arrange(desc("Elo Rating"))

# to set the row headers to null so it is not just the college names
rownames(elo_df) <- NULL

return (elo_df)
}
```

An example of how to call the function to get the Elo ratings for the 2023 Liberty League season is below.

```
ll_elo_ratings_2023 <- elo_rating_auto(schedule = LL_2023_schedule)
```

It also works to call the schedule function from within the elo_rating_auto function.

```
ll_elo_ratings_2023_2 <- elo_rating_auto(schedule_function(schedule =
D3_2023, conf = "LL"))
```

Team	Elo Rating
Hobart & Smith	1656.837
Rochester Tech	1602.977
Ithaca	1562.980
Clarkson	1508.296
Vassar	1501.597
Rensselaer	1472.467
Skidmore	1469.421
Union NY	1457.336
St Lawrence	1442.966
Bard	1325.123

Figure 4.1: Elo ratings for 2023 LL season

From Figure 4.1, we see that Rensselaer drops to sixth place and Clarkson jumps up to fourth place with Elo ratings. This is the opposite of what we see in Chapter 3 when we use RPI WP and RPI PPG ratings on the same season. In Figures 3.1 and 3.2, Rensselaer is above Clarkson in both RPI WP and RPI PPG. Additionally, we see in Figure 4.1 that St. Lawrence drops to ninth place with Elo ratings, when it is in sixth place and seventh place respectively with RPI PPG and RPI WP. The low rank for St. Lawrence in Figure 4.1, comes from their hefty defeat to Hobart & Smith 6-0 on the last game of the season. Having that result along with the timing of it makes the Elo rating for SLU drop.

Optimizing Parameters

We are interested in finding the optimal values for both the home adjustment parameter and the K scaling factor parameter that help make the Elo ratings as accurate as possible. We

start by optimizing the K scaling factor. Using opisthokonta.net's blog as inspiration, we create a graph that plots the MSE value for each k value in the specified range (7). The schedule that we use consists of all the games from the 2022 season. We are interested in finding the k value that has the smallest MSE. After lots of adjustment with the interval, we find that the range 120 to 160 illustrates that the optimal k value is at 140. Below is the code for creating the graph.

```
kk <- 0
elo_MSE <- 0
a <- 120
b <- 1
ndots <- 40
for(dot in (1:ndots)){
  kk[dot] = a + b * dot
  elo_d3 <- elo.run(score(home_score, away_score) ~ home_team + away_team,
data = schedule_function(), k = kk[dot])
  elo_MSE[dot] <- mse(elo_d3)
}

# plot the graph
plot(elo_MSE~kk, xlab = "k", main = "Optimal k Value")
```

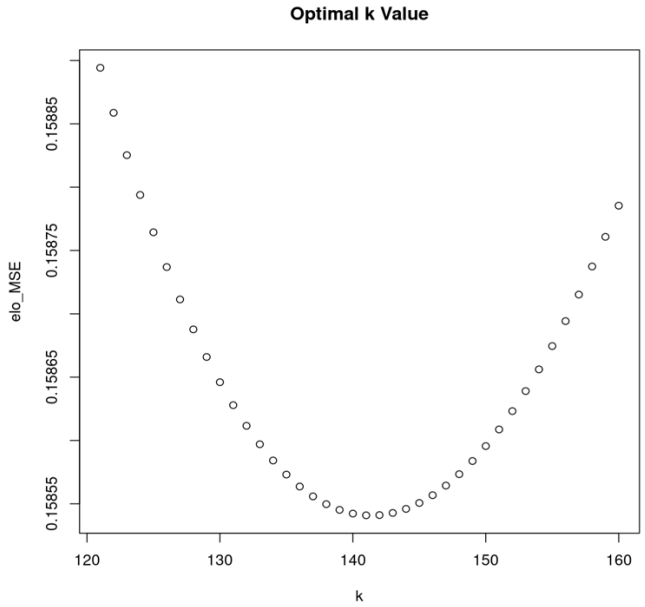


Figure 4.2: Optimal k value for Elo using all games from 2022 season

The k value of 140 is relatively high because we start each team at a 1500 Elo rating. In order to differentiate between the various strengths of all 435 Division III women's soccer teams, the k

value has to be high as each team only plays on average 11 games. This is as far as we got, but future work includes seeing how the k value is impacted when each team either starts at the Elo rating that they ended the previous season at or at the mean value ($\frac{OldEloRating+1500}{2}$). Also, we plan on applying this same approach to finding the optimal home adjustment value.

Chapter 5: Poisson Ratings

In soccer, we might assume that goals are independent events that happen at random throughout a game. A Poisson distribution is used for modeling the probability of a certain number of random events occurring within a fixed interval. We can use a Poisson distribution to predict the number of goals scored in a game. But before we get to the prediction, we need to estimate what the scoring rates (λ) would be.

We illustrate this idea with an example. Suppose that SLU plays at Clarkson in a Liberty League game. We assume that the scoring rates are 0.47 for SLU and 1.63 for Clarkson. In other words, SLU is expected to score about 0.47 goals against Clarkson and Clarkson is expected to score about 1.63 goals against SLU. We are interested in finding the probability that SLU wins 2-1.

We first find the probability using the Poisson probability function that SLU scores two goals given that their scoring rate is 0.47.

$$\text{P(SLU = 2)} = \frac{0.47^2 * e^{-0.47}}{2!} = 0.069$$

Next, we find the probability that Clarkson scores one goal given that their scoring rate is 1.63.

$$\text{P(Clarkson = 1)} = \frac{1.63^1 * e^{-1.63}}{1!} = 0.32$$

After finding these individual probabilities, we then multiply them together because we assume that goals in soccer are independent events.

$$\text{P(2, 1)} = 0.069 * 0.32 = 0.022$$

We find the probability that SLU wins 2-1 against Clarkson is 0.022.

How are the scoring rates of 1.63 for Clarkson and 0.47 for SLU calculated?

We assume that a team's ability to score goals against an average defense playing on a neutral field is reflected by an offensive rating (Off_{team}). Similarly, a defensive rating (Def_{opp}) represents the expected number of goals a team concedes playing against an average offense on a neutral field. Additionally, to account for home field advantage, we multiply by a parameter we call *Location*. When a team plays at home, we multiply by *Location* and when a team plays away, we divide by *Location*. Thus, to predict the goal scoring rate for each team, the following formula is used:

$$\lambda_{team} = \frac{Off_{team} * Def_{opp}}{AvgRating} * Location^{\pm 1}$$

Let's assume that SLU's mean scoring rate is 0.54 goals per game against an average team, while on average they give up 1.32 goals per game. Clarkson scores an average of 1.16 goals per game and on average they give up 1.01 goals per game. Also, let's assume that the average scoring rate is 1.044 goals per game and the location adjustment is 1.11. We will soon demonstrate where these values come from, but once we have the values, we can compute the scoring rates for both teams.

$$\lambda_{SLU} = \frac{0.54 * 1.01}{1.044} * 1.11^{-1} = 0.47$$

$$\lambda_{Clarkson} = \frac{1.16 * 1.32}{1.044} * 1.11^{1} = 1.63$$

How do we get the offensive and defensive ratings?

To get these ratings, we build a Poisson regression model that uses maximum likelihood estimation to predict a team's offensive and defensive rating based off the results from the season. Before we get to fitting the Poisson regression model, we need to manipulate the data to build the location variable. The data is currently in the format of home team, home score, away

team, and away score. To add location, we will first rename all the variables from "home" and "away" to "team" and "opponent." We will then make a new table where we flip the team and opponent, and then paste it on to the bottom of our original table. We consider the first half of the dataset to be "home" (*Location* = 1) and the second half of the dataset to be "away" (*Location* = -1). We call this function *doubleTableFunc* and the code for it is below.

```
doubleTableFunc <- function(games){
      games <- games %>%
        rename(team = home_team, opponent = away_team,
               team_score = home_score, opp_score = away_score) %>%
        filter(!is.na(team_score & opp_score))
      games2 <- games %>% # new data table flipping the columns
        rename(team = opponent, opponent = team,
               team_score = opp_score, opp_score = team_score)
      double_table <- bind_rows(games, games2) # paste them together
      double_table$Location = NA # add location variable
      double_table$Location[0:length(double_table$date)/2] = 1
      double_table[is.na(double_table)] = -1

      return(double_table)
  }
```

An example of how to call the above function for the 2023 Liberty League schedule is below along with the sample output.

```
ll_doubletable <- doubleTableFunc(games = LL_2023)
```

	opponent	opp_score	team	team_score	Date	Location
1	Union NY	0	St Lawrence	2	Tue 09.19	1
2	Clarkson	2	Skidmore	0	Wed 09.20	1
3	Rochester Tech	1	Hobart & Smith	1	Wed 09.20	1
4	Clarkson	0	Rochester Tech	1	Sat 09.23	1
5	Hobart & Smith	1	Skidmore	1	Sat 09.23	1
6	Bard	0	Ithaca	4	Sat 09.23	1
7	Rensselaer	0	St Lawrence	0	Sat 09.23	1
8	Vassar	0	Union NY	0	Sat 09.23	1
9	Hobart & Smith	1	Ithaca	0	Wed 09.27	1
10	Bard	0	Vassar	0	Wed 09.27	1
11	Hobart & Smith	3	Clarkson	0	Sat 09.30	1
12	St Lawrence	0	Ithaca	2	Sat 09.30	1
13	Skidmore	1	Vassar	2	Sat 09.30	1
14	Bard	0	Rochester Tech	7	Sat 09.30	1
15	Union NY	2	Rensselaer	0	Sat 09.30	1

Figure 5.1: Sample output from doubleTableFunc for 2023 LL schedule

Now since the table is doubled and we have the location variable, we can fit the Poisson regression model. In the model, "team" refers to the offensive ratings for a team and "opponent" refers to the defensive ratings for a team. Below is the code used to fit the model for the 2023 Liberty League schedule, followed by sample output for the initial values before adjustment. In Figure 5.2, "x" is the coefficient ($\widehat{B}_i$) from the Poisson regression model and "rating" is the coefficient exponentiated.

```
model <- glm(team_score ~ team + opponent + Location, data =
ll_doubletable),
family = poisson) # to build the poisson model
original <- data.frame(x = model$coefficients, rating =
exp(model$coefficients)) # to extract the coefficients
original$team <- rownames(original)
row.names(original) <- NULL
original <- original %>%
select(team, x, rating)
```

	team	x	rating
1	(Intercept)	-0.87218716	0.41803624
2	teamClarkson	2.06292724	7.86897052
3	teamHobart & Smith	2.79983449	16.44192528
4	teamIthaca	2.24205802	9.41268285
5	teamRensselaer	2.04802779	7.75259623
6	teamRochester Tech	2.65713036	14.25532274
7	teamSkidmore	1.92430793	6.85040605
8	teamSt Lawrence	1.30404354	3.68416364
9	teamUnion NY	1.80523569	6.08140463
10	teamVassar	0.77558540	2.17186317
11	opponentClarkson	-1.15544070	0.31491872
12	opponentHobart & Smith	-2.81028360	0.06018792
13	opponentIthaca	-1.61532776	0.19882549
14	opponentRensselaer	-0.72287533	0.48535469
15	opponentRochester Tech	-1.80367323	0.16469282
16	opponentSkidmore	-1.08187275	0.33896014
17	opponentSt Lawrence	-0.89111957	0.41019625
18	opponentUnion NY	-0.97147015	0.37852614
19	opponentVassar	-2.31608410	0.09865917
20	Location	-0.04299299	0.95791810

Figure 5.2: Sample output from Poisson model for LL 2023 before adjustments

From Figure 5.2, we see that Bard does not have any offensive and defensive coefficients. In the Poisson regression model, the first team alphabetically is treated as the intercept. This is a problem because with Bard not being included, it means that all the other offensive and defensive ratings for the other teams are computed relative to Bard and not the average team. To fix this issue we followed Brenden Bready's thesis paper (8). See his thesis for more details on the adjustments made to rescale the offensive and defensive ratings, while keeping the predicted scoring rates the same.

After the adjustments are made, we have an offensive and defensive rating for each team. We interpret the offensive ratings as the goal scoring rate against an average team. Likewise, we interpret defensive ratings as the goal concession rate against an average team. Figure 5.3 shows the offensive and defensive ratings for the 2023 Liberty League season.

	Team	OffRating	DefRating
1	Hobart & Smith	2.415	0.194
2	Rochester Tech	2.094	0.530
3	Ithaca	1.383	0.639
4	Clarkson	1.156	1.013
5	Vassar	0.319	0.317
6	Skidmore	1.006	1.090
7	Union NY	0.893	1.217
8	Rensselaer	1.139	1.561
9	St Lawrence	0.541	1.319
10	Bard	0.147	3.215

Figure 5.3: Offensive and defensive ratings for LL 2023

The final step is to generate an overall rating for each team. We assume every team plays each other once including themselves on a neutral field. We calculate the overall rating metric by

finding the probability of a win, loss and tie in each game (9). We find the probability of a tie using the formula below:

$$P(Tie) = \sum_{n=0}^{\infty} \frac{(\lambda_1\lambda_2)^n e^{-(\lambda_1\lambda_2)}}{(n!)^2}$$

where λ_1 and $\lambda_2$are the scoring rates for the two teams

This formula is conveniently equivalent to:

$$J_0(2\sqrt{\lambda_1\lambda_2})e^{-(\lambda_1\lambda_2)}$$

where $J_0(x)$ is a zero order Bessel function.

Likewise to find the probability of a win, we start by finding the probability of winning by k goals:

$$P(Team\ 1\ wins\ by\ k\ goals) = \sum_{n=0}^{\infty} \frac{{\lambda_1}^{n+k}{\lambda_2}^{n} e^{-(\lambda_1\lambda_2)}}{(n+k)!\,n!}$$

Similar to ties, this formula is equivalent to a kth order Bessel function:

$$P(Team\ 1\ wins\ by\ k\ goals) = J_k(2\sqrt{\lambda_1\lambda_2})(\frac{\lambda_1}{\lambda_2})^{k/2} e^{-(\lambda_1\lambda_2)}$$

We find the probability of a win by summing over all the possible winning margins using the formula below:

$$P(Win) = \sum_{k=1}^{\infty} J_k(2\sqrt{\lambda_1\lambda_2})(\frac{\lambda_1}{\lambda_2})^{k/2} e^{-(\lambda_1\lambda_2)}$$

For soccer, we capture most of the probability with:

$$P(Win) = \sum_{k=1}^{20} J_k(2\sqrt{\lambda_1\lambda_2})(\frac{\lambda_1}{\lambda_2})^{k/2} e^{-(\lambda_1\lambda_2)}$$

Finally, to get the probability of a loss, we use the following formula:

$$P(Loss) = 1 - P(Tie) - P(Win)$$

We use the Bessel package in R and specifically the besselI function (10). The code for the function that finds all three probabilities is below. The function takes in the two scoring rates of the teams (lambda1 and lambda2) and returns a list containing the probabilities.

```
# bessel function calculations used to find probabilities of
# win, tie, and loss
probs <- function(lambda1, lambda2){

prob_tie <- besselI(2*sqrt(lambda1*lambda2), 0)*exp(-(lambda1+lambda2))

total <- 0
  for(i in 1:20){
   prob_win_iteration <- besselI(2*sqrt(lambda1*lambda2), i)*
(lambda1/lambda2)^(i/2) *exp(-(lambda1+lambda2))

    total = total + prob_win_iteration
    i = i + 1
  }
  prob_win <- total

prob_loss <- 1 - (prob_tie) - (prob_win)

return(list(prob_win, prob_tie, prob_loss))
}
```

After we calculate the probabilities of a win, loss, and tie for each team in every game, we convert to expected points (3 points for a win, 1 point for a tie and 0 points for a loss). We compute the average expected points over all the teams and this is the overall ratings metric. Figure 5.4 displays the new overall rating metric for the 2023 Liberty League season. We see for example that Hobart & Smith is expected to get 2.40 points per game, while St. Lawrence is expected to get 0.87 points per game.

	Team	OverallRating	OffRating	DefRating
1	Hobart & Smith	2.40	2.415	0.194
2	Rochester Tech	2.10	2.094	0.530
3	Ithaca	1.76	1.383	0.639
4	Clarkson	1.41	1.156	1.013
5	Vassar	1.21	0.319	0.317
6	Skidmore	1.28	1.006	1.090
7	Union NY	1.15	0.893	1.217
8	Rensselaer	1.15	1.139	1.561
9	St Lawrence	0.87	0.541	1.319
10	Bard	0.24	0.147	3.215

Figure 5.4: Overall ratings metric for LL 2023

We see from Figure 5.4 that SLU's offensive rating is 0.54 and its defensive rating is 1.32. Clarkson's offensive rating is 1.16 and its defensive rating is 1.01. The average rating is 1.044 and it is computed by averaging the offensive ratings from all the teams in the league. The location adjustment is 1.11 and it is calculated within the Poisson regression model. To put all these steps together, we create a function called *poisfunction* that takes in a schedule and outputs a list that has a table of offensive, defensive and overall ratings, the location adjustment, and the mean of the scoring rates. See Appendix for the code for the function.

An example of how to call the *poisfunction* to get the Poisson ratings for the 2023 Liberty League season is below.

```
x <- poisfunction(schedule = LL_2023)
```

The results are:

- x[[1]] gives the table shown in Figure 5.4
- x[[2]] = 1.11 (location adjustment)
- x[[3]] = 1.044 (mean of the scoring rates)

Chapter 6: Shiny App

The app displays all the rating methods described in the previous chapters. The user can select a year and then filter by conference or region. Although the app will find new ranks for all 435 teams in Division III women's soccer, the Poisson ratings fail when trying to fit this many teams. The link to the app is: https://stlawu.shinyapps.io/D3WSOCRatings/. Below is a picture of what the app looks like when you first click on the link.

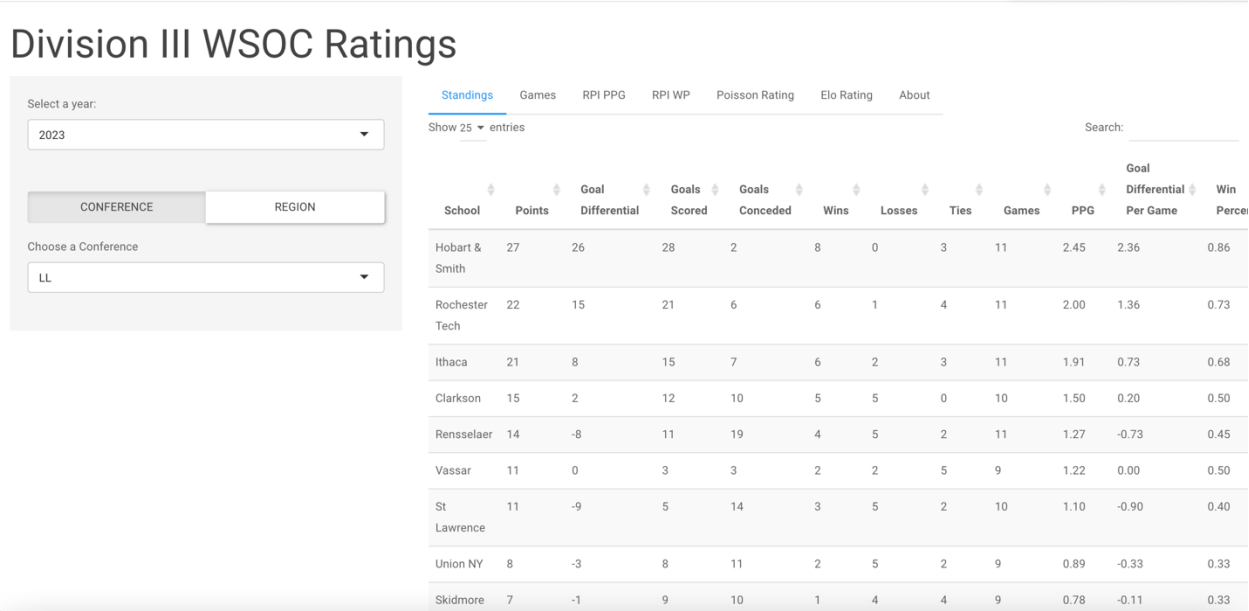


Figure 6.1: App when first opened

The app has six major components all indicated in the top tabs. The tabs are standings, games, RPI PPG, RPI WP, Poisson rating and Elo rating. The user can click on one of these tabs and find the rankings of teams based on that method. For example, if the user selects the RPI WP tab for 2022 and the NESCAC conference, the following is what the app will display.

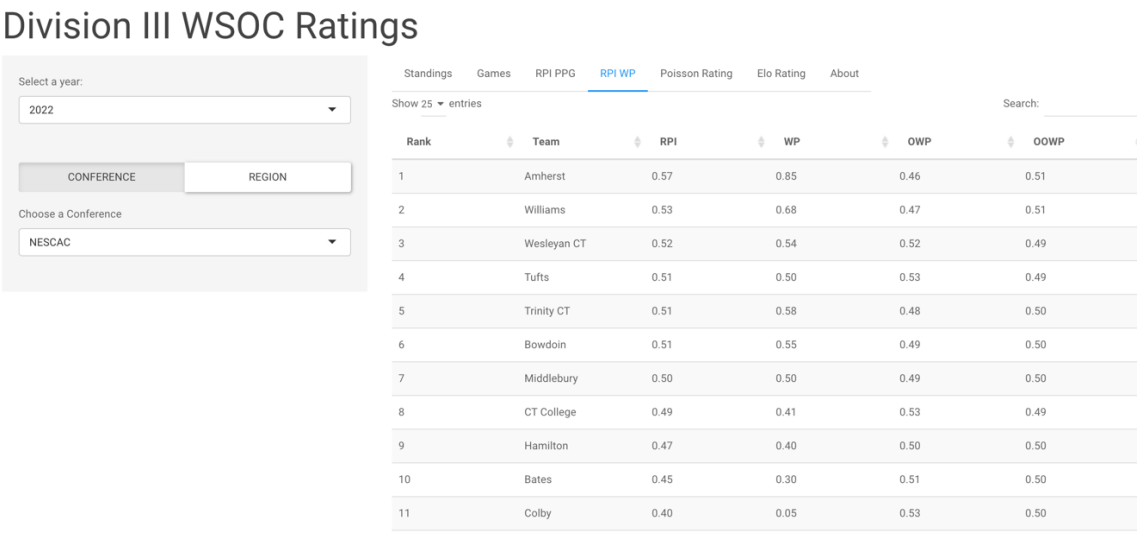


Figure 6.2: App when RPI WP is selected for 2022 NESCAC season

Additionally, if a user wants to perform their own analysis, the app allows them to download the data from any of the tabs.

ui.R

The app has two files that interact to build the shiny app. The two files are a server.R file and a ui.R file. The ui.R file has two main components to it, a sidebar and a main panel. The first part of the ui creates the sidebar with user options. The user can select a year from a dropdown menu and choose whether they want to see rankings based on region or conference.

```
ui <- (FluidPage(

# application title
titlePanel("Division III WSOC Ratings"),

# sidebar
sidebarLayout(
    sidebarPanel(
      # to select the year
      selectInput(
        "Year", "Select a year: ", choices = c("2022", "2023"), selected =
"2023"),
      # to choose Conference or Region
      radioGroupButtons(
        inputId = "ConfReg", label = "", choices = c("Conference",
"Region"), justified = TRUE, selected = "Conference"),
uiOutput("ConfRegBox")),
```

The second part of the ui builds each tab from the elements on the server side, adding the correct table and download button to each tab. There are seven total tabs, six tabs for the various rating methods and one tab for the about section. Six out of the seven tabs have a similar format. For example in addition to the data table and the download button, the Elo tab has two sliders to adjust the home field and the scaling factor. The code below shows an example of how the standings tab along with the Elo tabs are created.

```
mainPanel(
      tabsetPanel(
        tabPanel(
          "Standings", dataTableOutput("standings"),
          downloadButton("downloadstandings", "Download Standings File")),
      tabPanel(
          "Elo Rating",
          flowLayout(
            sliderInput("home_advantage", "Value for home field:",
                        min = 0, max = 100, value = 30, width = "100%"),
            sliderInput("k_value", "Choose value for k:", min = 0, max =
50, value = 20, width = "100%")),
          dataTableOutput("elo_rating"),

            …
        tabPanel(
            # last tab panel
            ),

      )

))))
```

server.R

The server.R file of the app is where all the functions for running every tab are found. The functions are displayed throughout the paper, so details about them can be found in specific chapters.

```
server <- function(input, output){

  D3_year <- reactive(
    if(input$Year == "2022"){
      D3_2022
    } else {
      D3_2023
    }
  )
```

```
# Schedule function
schedule_function <- function(schedule = D3_year(), team_list =
team_details, conf = "all", reg = "all", league_only = TRUE){
# see chapter 2 for details
}

# Standings function
standings_function <- function(schedule = D3_year(), team_list =
team_details, conf = "all", reg = "all", league_only = TRUE){
# see chapter 2 for details
}

# Standings2 function
standings2 <- function(games){
# see chapter 2 for details
}

# RPI_PPG function
rpi_function_ppg <- function(games, w1=(0.25), w2=(0.5), w3=(0.25)){
# see chapter 3 for details
}

# RPI_WP function
rpi_function_wp <- function(games, w1=(0.25), w2=(0.5), w3=(0.25)){
# see chapter 3 for details
}

# Double table function
doubleTableFunc <- function(games){
# see chapter 5 for details
}

# Bessel probabilities function
probs <- function(lambda1, lambda2) {
# see chapter 5 for details
}

# Poisson ratings function
poisfunction <- function(schedule) {
# see chapter 5 for details
}

# Elo ratings function
elo_rating_function <- function(schedule) {
# see chapter 4 for details

}
```

After the functions are defined, output code is written to call the functions and display the data tables and download buttons. Before getting to that, we first need to connect the

radioGroupButton from the ui side to the server side. This enables the user to click on either conference or region and a separate drop-down menu will appear displaying the region choices or conference choices depending on their selection. The code below displays this result.

```
output$ConfRegBox <- renderUI({
    if(input$ConfReg == "Conference"){

      selectInput("Conference", "Choose a Conference", choices =
conferences, selected = "LL")
    } else {
      selectInput("Region", "Choose a Region", choices = regions, selected
= "Region III")
    }
  })
```

The last step is to create reactive elements for each of the functions, so a data table and a download button can be displayed. Below is an example of how to create the reactive element for standings, put the element into a table and build the download button.

```
standings <- reactive(
    if(input$ConfReg == "Conference"){
      standings_function(conf = input$Conference)
    } else {
      standings_function(reg = input$Region)
    })

  output$standings <- renderDataTable(
    standings())

  output$downloadstandings <- downloadHandler(
    filename = "Standings.csv",
    content = function(file) {
      write.csv(standings(), file, row.names = FALSE)
    }
  )

}
```

Overall, when using the app users can:

- Sort by any column. For example, the users can click on the goal differential column in the standings tab and the standings are now sorted by that.

- Subset the results. For example, the users can type in St. Lawrence in the search bar in any of the tabs and the results will only show information specific to St. Lawrence.
- Download the results from any tab. For example, to get the game results for the 2023 Liberty League season as a csv file, the user selects the “Games” tab and clicks on the “Download Games File” button.

Chapter 7: Simulation

We use simulation to compare the rating methods to one another. We are first interested in seeing how the rating methods compare to one another within one league where strength of schedule is similar as every team only plays each other. To test this out, we simulate 1000 seasons using the 2023 Liberty League schedule. We assume known abilities of each of the teams. The known abilities (ratings) of each of the teams are summarized in the table below.

Rank	Team	Overall	Off	Def
1	Hobart & Smith	2.22	2.3	0.7
2	Rochester Tech	1.97	2	0.9
3	Ithaca	1.79	1.5	0.8
4	Vassar	1.53	1.5	1.2
5	Rensselaer	1.41	1.3	1.2
6	Clarkson	1.35	1.3	1.3
7	St Lawrence	1.24	1.3	1.5
8	Skidmore	1.13	1.2	1.6
9	Union NY	0.96	1	1.7
10	Bard	0.33	0.6	3.1

Next, we join the schedule file that contains the actual game schedule from the 2023 season to the ratings file, so we know both the offensive and defensive ratings of each team for every game in the season. The following code displays how to get the joined dataset, which we call *big_sched*.

```
big_sched  <- left_join(schedule, ratings, by = c("away_team" =
                                                  "team"))

big_sched <- left_join(big_sched, ratings, by = c("home_team" =
                                                  "team"))

big_sched <- big_sched %>%
  rename("away_off" = OffRating.x, "away_def" = DefRating.x, "home_off" =
OffRating.y, "home_def" = DefRating.y)
```

	home_team	home_score	away_team	away_score	Date	away_off	away_def	home_off	home_def
1	Union NY	0	St Lawrence	2	Tue 09.19	1.3	1.5	1.0	1.7
2	Clarkson	2	Skidmore	0	Wed 09.20	1.2	1.6	1.3	1.3
3	Rochester Tech	1	Hobart & Smith	1	Wed 09.20	2.3	0.7	2.0	0.9
4	Clarkson	0	Rochester Tech	1	Sat 09.23	2.0	0.9	1.3	1.3
5	Hobart & Smith	1	Skidmore	1	Sat 09.23	1.2	1.6	2.3	0.7
6	Bard	0	Ithaca	4	Sat 09.23	1.5	0.8	0.6	3.1
7	Rensselaer	0	St Lawrence	0	Sat 09.23	1.3	1.5	1.3	1.2
8	Vassar	0	Union NY	0	Sat 09.23	1.0	1.7	1.5	1.2
9	Hobart & Smith	1	Ithaca	0	Wed 09.27	1.5	0.8	2.3	0.7
10	Bard	0	Vassar	0	Wed 09.27	1.5	1.2	0.6	3.1
11	Hobart & Smith	3	Clarkson	0	Sat 09.30	1.3	1.3	2.3	0.7
12	St Lawrence	0	Ithaca	2	Sat 09.30	1.5	0.8	1.3	1.5
13	Skidmore	1	Vassar	2	Sat 09.30	1.5	1.2	1.2	1.6
14	Bard	0	Rochester Tech	7	Sat 09.30	2.0	0.9	0.6	3.1
15	Union NY	2	Rensselaer	0	Sat 09.30	1.3	1.2	1.0	1.7

Figure 7.1: Sample of output from joining two datasets

After joining the two datasets together, we need to find the scoring rates of the teams for each game. To do this, we create a score_rate function that outputs the scoring rate using the formula displayed in Chapter 5. The score_rate function takes in the team's offensive rating, the opponent's defensive rating, the average rating, and the location adjustment. The default value for the location adjustment is 1.11 and the default average rating is 1.044. The code for the function is displayed below.

```
score_rate <- function(OffRating, DefRating, Home, HomeAdj = 1.11,
AvgRating = 1.044){
  return(OffRating*DefRating*HomeAdj^Home / AvgRating)
}
```

Before we get to the simulation, we need to initialize the data frames that will keep the ranks for each team from the simulated seasons. We do this with the following code.

```
nteams=length(ratings$team)

ranks_elo_2023 <- data.frame(matrix(ncol = nteams, nrow = 0))
names(ranks_elo_2023) = ratings$team

ranks_pois_2023 <- data.frame(matrix(ncol = nteams, nrow = 0))
names(ranks_pois_2023) = ratings$team

ranks_ppg_2023 <- data.frame(matrix(ncol = nteams, nrow = 0))
names(ranks_ppg_2023) = ratings$team

ranks_points_2023 <- data.frame(matrix(ncol = nteams, nrow = 0))
names(ranks_points_2023) = ratings$team

ranks_rpi_ppg_2023 <- data.frame(matrix(ncol = nteams, nrow = 0))
names(ranks_rpi_ppg_2023) = ratings$team

ranks_rpi_wp_2023 <- data.frame(matrix(ncol = nteams, nrow = 0))
names(ranks_rpi_wp_2023) = ratings$team
```

Now, we have all the pieces to simulate 1000 seasons for each rating method and save those ranks in the data frames described above. We use the Poisson probability model to generate actual scores for each team in each game based on the given scoring rate. After simulating scores for each game, we apply each of the rating methods to rank the teams for the simulated season and store those ranks in the data frames mentioned above. Below is the code for running the simulation for the 2023 Liberty League schedule.

```
n_games <- length(big_sched$away_team)
home <- 1.11
avg_rating <- mean(ratings$OffRating) # get from ratings file passed in
numseasons=1000

for(i in 1:numseasons){

  v <- rep(1, n_games)

  one_season <- big_sched %>%

    mutate(away_rate = score_rate(away_off, home_def, -1, home,
avg_rating)) %>%
    mutate(home_rate = score_rate(home_off, away_def, 1, home, avg_rating))
%>%
    mutate(home_score = rpois(v, home_rate)) %>%
    mutate(away_score = rpois(v, away_rate)) %>%
    select(c(1:5,19,20))

  elo_season <- elo_rating_auto(one_season,KValue=30)

  ranks_elo_2023[i,] <- (nteams+1) - rank(elo_season$`Elo Rating`)
```

```
  poisson_season <- poisfunction(one_season)[[1]] %>%
  arrange((Team))

  ranks_pois_2023[i,] <- poisson_season$Rank

  ppg_season <- standings2(one_season)
  ppg_season <- ppg_season[order(ppg_season$School),]

  ranks_ppg_2023[i,] <- (nteams+1) - rank(ppg_season$PPG)


  points_season <- standings2(one_season)
  points_season <- points_season[order(points_season$School),]

  ranks_points_2023[i,] <- (nteams+1) - rank(points_season$Points)

  rpi_wp_season <- rpi_function_wp(one_season)
  rpi_wp_season <- rpi_wp_season[order(rpi_wp_season$Team),]

  ranks_rpi_wp_2023[i,] <- (nteams+1) - rank(rpi_wp_season$RPI)

  rpi_ppg_season <- rpi_function_ppg(one_season)
  rpi_ppg_season <- rpi_ppg_season[order(rpi_ppg_season$Team),]

  ranks_rpi_ppg_2023[i,] <- (nteams+1) - rank(rpi_ppg_season$RPI)

}
```

Results

The table below shows the average rank for each team over the 1000 simulated seasons based on four rating methods: points, RPI WP, Elo and Poisson.

Rank	Team	Points	RPI WP	Elo	Poisson
1	Hobart & Smith	1.60	1.62	1.49	1.41
2	Rochester Tech	2.51	2.51	2.31	2.30
3	Ithaca	3.21	3.17	3.10	3.14
4	Vassar	5.01	4.97	4.82	4.79
5	Rensselaer	5.49	5.48	5.59	5.52
6	Clarkson	5.92	5.92	5.92	5.88
7	St Lawrence	6.47	6.47	6.58	6.55
8	Skidmore	6.99	7.04	7.06	7.22
9	Union NY	7.88	7.89	8.14	8.23
10	Bard	9.43	9.95	9.99	9.98

From the table, we see that the Poisson rating method does the best because the simulated averages for each of the teams are the closest to the actual ranks. For example, Hobart & Smith's

actual rank is 1 and the Poisson rating method has Hobart & Smith's average rank at 1.41. This average is the closest out of the other rating methods to 1. To illustrate this point graphically, we create boxplots that show the average simulated ranks for each of the teams based on RPI WP and Poisson rating methods.

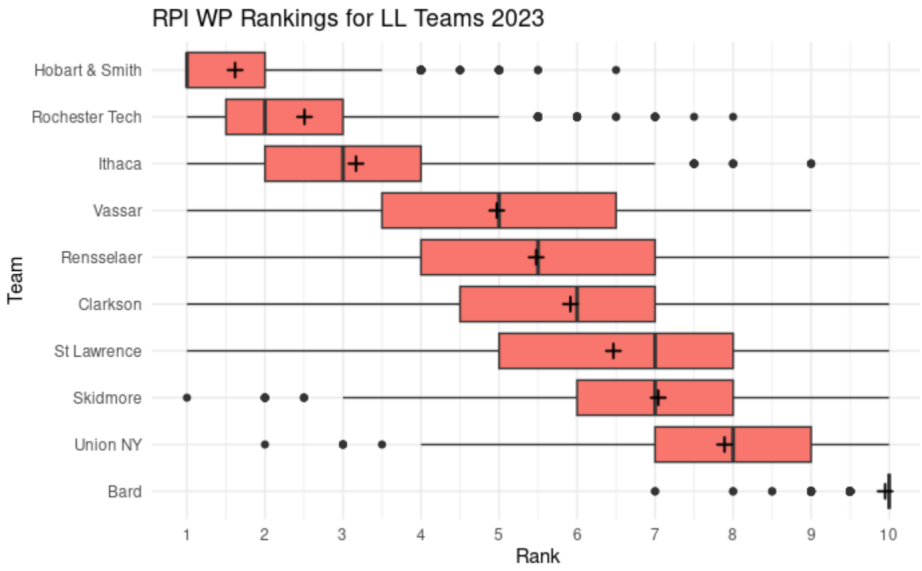


Figure 7.2: Boxplot of simulated ranks for RPI WP

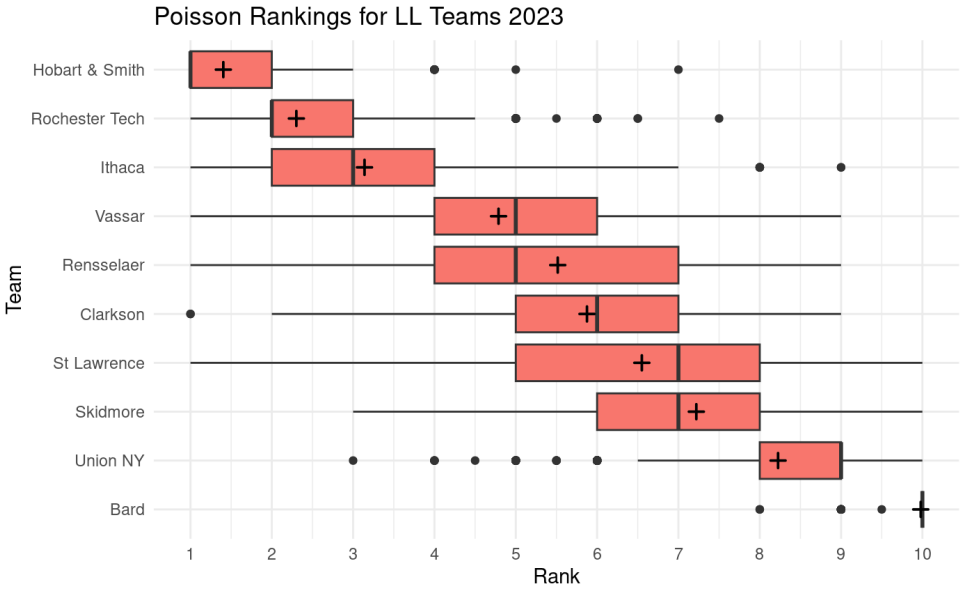


Figure 7.3: Boxplot of simulated ranks for Poisson

From Figure 7.2, we see that more variability exists within the RPI WP boxplot compared to the Poisson boxplot in Figure 7.2. We see in Figure 7.2 that the interquartile ranges in the boxes are wider and more spread out particularly for ranking Vassar, Rensselaer, Clarkson, and St. Lawrence. In addition to graphically illustrating the differences, we also calculate the mean squared error (MSE) and mean absolute deviation (MAD) for the rating methods. We want the MSE and MAD to be as small as possible. The table below summarizes the results.

	Points	RPI WP	Elo	Poisson
MSE	2.76	2.66	2.22	2.02
MAD	1.17	1.15	1.01	0.95

We see from the table that the Poisson rating method has the smallest MSE and MAD compared to the other rating methods. The Elo rating method is not far behind the Poisson rating method, while RPI WP and points are worse with much higher MSE and MAD. Overall, we conclude that the Poisson rating method works the best for accurately ranking teams within the same league.

Simulating Multiple Leagues

We are now interested in seeing how the rating methods compare to one another when the schedule consists of games from three different leagues instead of just one league. The three leagues we use are the Empire 8, SUNYAC, and the Liberty League. There are 29 teams from those three leagues and the schedule file consists of 171 games. Like mentioned above, we assume known abilities for each of the teams. The assumed abilities (ratings) of each of the teams are summarized in the table below.

Rank	Team	Overall	Off	Def
1	Hobart & Smith	2.41	3.57	0.22
2	Ithaca	2.28	2.56	0.31
3	Clarkson	2.19	2.67	0.47
4	Cortland St	2.14	1.89	0.31
5	Suny New Paltz	2.10	2.65	0.61
6	Rochester Tech	2.05	2.63	0.70
7	Rensselaer	1.86	2.02	0.79
8	Brockport St	1.78	1.08	0.36
9	Skidmore	1.75	1.32	0.56
10	Geneseo St	1.73	0.96	0.33
11	Vassar	1.66	0.81	0.30
12	Plattsburgh St	1.62	1.35	0.79
13	St Lawrence	1.57	1.27	0.81
14	Union NY	1.49	1.14	0.84
15	Nazareth	1.45	0.84	0.61
16	Oswego St	1.37	0.70	0.58
17	St John Fisher	1.32	1.11	1.17
18	Suny Fredonia	1.29	1.42	1.59
19	Suny Oneonta	1.10	1.19	1.89
20	Utica	0.98	0.83	1.64
21	Russell Sage	0.91	0.32	0.79
22	Suny Potsdam	0.86	0.63	1.63
23	Buffalo St	0.81	1.08	2.88
24	Alfred	0.76	0.59	1.87
25	Hartwick	0.51	0.32	2.09
26	Bard	0.48	0.25	1.99
27	Houghton	0.42	0.47	3.40
28	Elmira	0.35	0.20	2.52
29	Keuka	0.30	0.38	4.20

We go through the same process mentioned previously in the chapter to simulate 1000 seasons for each of the rating methods using the three league schedule file. The location adjustment is 1.17 and the average rating is 1.25. The results from the simulations are below.

Results

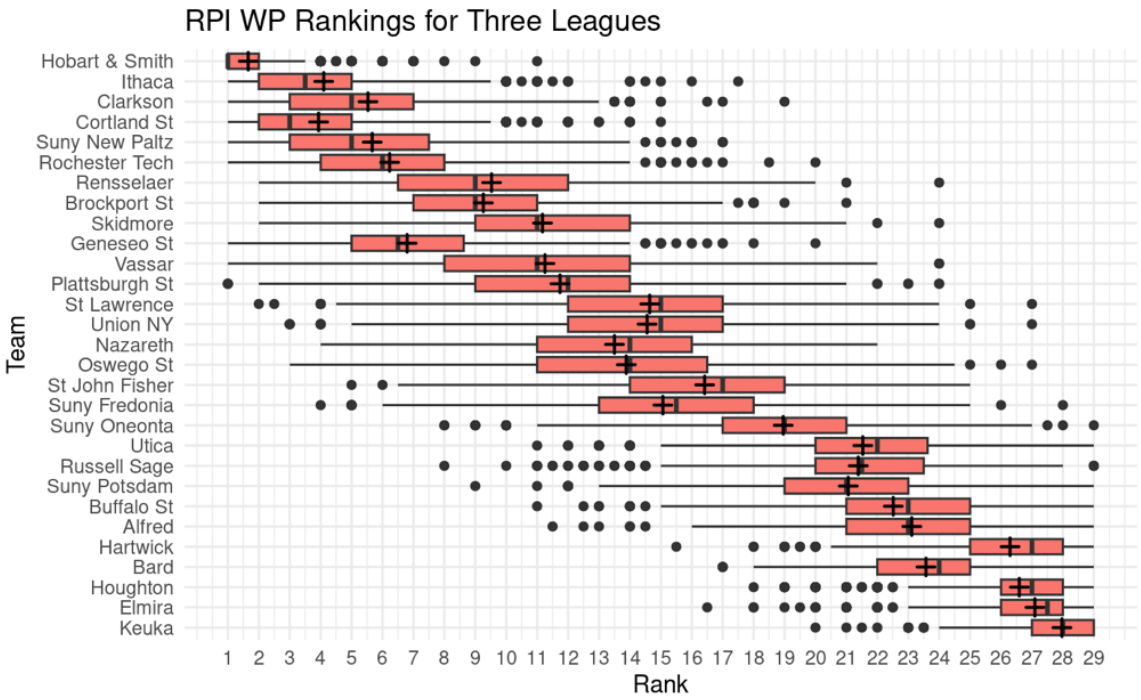


Figure 7.4: Boxplot of simulated ranks for RPI WP

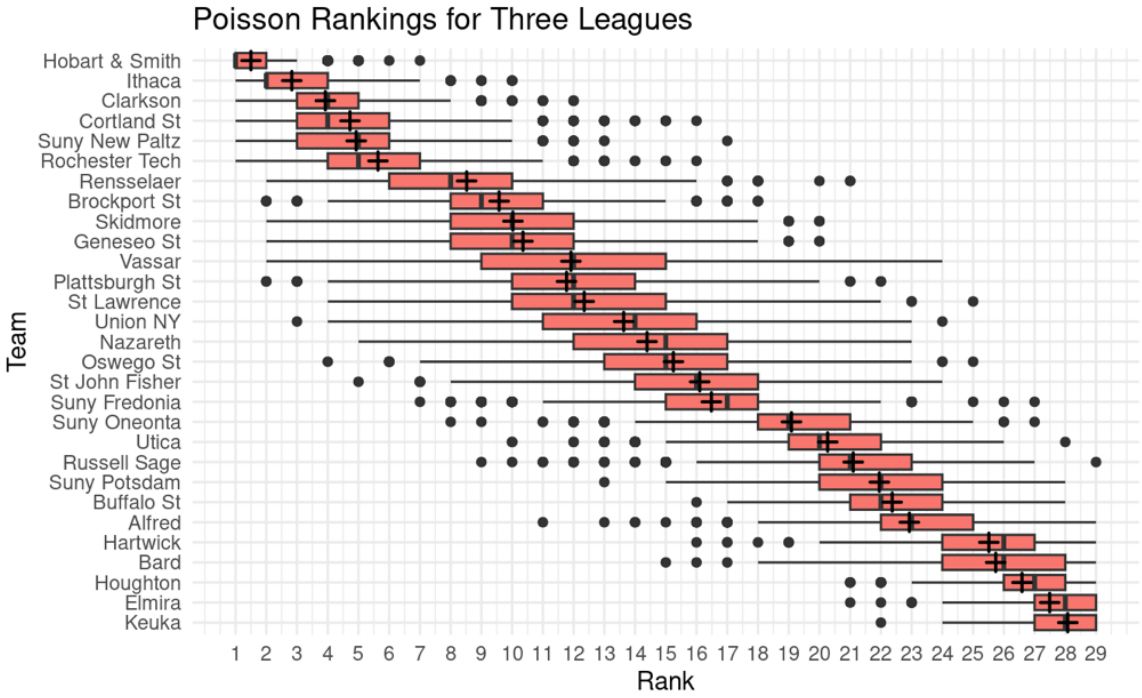


Figure 7.5: Boxplot of simulated ranks for Poisson

We see from Figure 7.4 for RPI WP, that the boxplot has a lot of variability between the simulated average rank and the actual rank of the teams. We expect the ranking method to simulate ranks that match the actual ranks of the teams. This does not seem to be the case when using RPI WP for this three league simulation. For example, we see that Geneseo St. has an average simulated rank of 7, when their actual rank is 10. On the other hand, Figure 7.5 shows that the average simulated ranks from the Poisson rating method more closely match the actual ranks of the teams. The table below displays the MSE and MAD for the four rating methods.

	Points	RPI WP	Elo	Poisson
MSE	29.51	11.74	24.53	7.4
MAD	4.23	2.58	3.84	1.99

It is evident that the Poisson rating method has the smallest MSE and MAD compared to the other rating methods. Interestingly, we see that the Elo rating method performs worse on a three league simulation compared to a one league simulation. The Elo rating method has a very high MSE of 24.53 for the three league simulation, whereas for the one league simulation, its MSE is 2.22. We believe that the Elo rating method performs worse in this situation because strength of schedule is not adequately taken into account. For our setup of the Elo rating method, we start each team at a rating of 1500. This leaves relatively little room for the ratings to adjust over a fairly short season. Consequently, when a schedule includes teams from different leagues of various strengths, the Elo rating method struggles to account for the differences that exist within the teams and inaccurately ranks them.

Conclusion

All in all, in this project we investigated different rating methods that can be used to rank teams in Division III women's soccer. The rating methods that we looked at were: RPI, Elo and a Poisson scoring rate model. We employed simulation to compare the methods to one another and see which one performs the best. From the simulations, we found that the Poisson rating method performs the best on both single league schedules and on multiple league schedules. This is due to the fact that the Poisson rating method takes into account the actual scores and margins of victories, instead of just solely relying on win/tie/loss results. Therefore, we recommend using a Poisson rating method to rank teams in Division III women's soccer.

Some future work includes optimizing the home adjustment and K scaling factor parameters in the Elo rating method mentioned in Chapter 4. We also want to update the Shiny App so that rating method predictions are made throughout the season instead of just at the end of the season. Lastly, we could use these methods to explore the question – is it more beneficial for rankings to be a good team in a weak league or a bad team in a strong league?

\

References

1. **Rvest package for web scraping:** *Rvest Overview*. (n.d.). Rvest Tidyverse. Retrieved May 11, 2024, from https://rvest.tidyverse.org/

2. **Web scraping with ChromoteSession:** *Live web scraping (with chromote) — read_html_live*. (n.d.). Rvest Tidyverse. Retrieved May 11, 2024, from https://rvest.tidyverse.org/reference/read_html_live.html

3. **CSS selectors for web scraping HTML:** Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). 24: Web scraping. In *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data* (2nd ed.). O'Reilly Media. https://r4ds.hadley.nz/webscraping#sec-css-selectors

4. **RPI formula information:** *RPI for Division I Women's Soccer - RPI: Formula*. (n.d.). https://sites.google.com/site/rpifordivisioniwomenssoccer/rpi-formula?authuser=0

5. **History of Elo:** Arrabal, A. T. (2023, August 25). Elo Ratings: Explaining Their Application and Power through League Simulation. *Medium*. Retrieved May 11, 2024, from https://medium.com/@atilarrabal/elo-ratings-explaining-their-application-and-power-through-league-simulation-1bf640475d2

6. **Elo package documentation:** Heinzen, E. (n.d.). *Calculating running ELO updates*. Retrieved May 11, 2024, from https://cran.r-project.org/web/packages/elo/vignettes/running_elos.html

7. **Optimizing parameters for Elo:** *Tuning the Elo ratings: The K-factor and home field advantage | opisthokonta.net*. (2016, August 12). Retrieved May 11, 2024, from https://opisthokonta.net/?p=1387

8. **Brenden Bready's thesis:** Bready, B. (2022). *BIFA: Bready's International Football App* [Undergrad Thesis]. St. Lawrence University.

9. **Poisson Scoring Rate Model**: Lock, R. H., & Danehy, T. J. (1997). Using a Poisson model to rate teams and predict scores in ice hockey. In *ASA Proceedings of the Section on Statistics in Sports* (pp. 25-30).

10. **BesselI function documentation:** *Bessel function - RDocumentation*. (n.d.). Retrieved May 11, 2024, from https://www.rdocumentation.org/packages/Bessel/versions/0.6-0/topics/Bessel

Appendix

Web scraping D3soccer

```
function_webscrap <- function(date) {
  year <- substr(date, 1, 4)
  url <- paste("https://d3soccer.prestosports.com/seasons/women/", year,
"/schedule?date=", date, sep = "")
  g <- read_html(url)
  tab <- g %>% html_nodes("table")
  objs <- tab %>% html_table()
  table_clean <- modify_stats_function(objs[[1]], date)
  return(table_clean)
}

modify_stats_function_d3 <- function(date_results, date) {
  date_results %>%
  slice(-1) %>%
  rename("Away" = X1) %>%
  rename("Away Score" = X2) %>%
  rename("Home" = X3) %>%
  rename("Home Score" = X4)%>%
  select(1:5) %>%
  filter(X5 == "Final")%>%
  select(1:4) %>%
  mutate(Date = date) %>%
  mutate(Home = str_remove(Home, "No\\. \\d+")) %>%
  mutate(Away = str_remove(Away, "No\\. \\d+")) %>%
  mutate(Home = str_remove(Home, "\r\n\r\n")) %>%
  mutate(Away = str_remove(Away, "\r\n\r\n"))
}
```

Web scraping Massey

```
function_webscrap <- function(date) {
  url <- paste("https://masseyratings.com/csocw/ncaa-d3/games?dt=", date, sep
= "")
  c <- ChromoteSession$new()
  c$Page$navigate(url, wait = FALSE)
  c$Page$loadEventFired()

html <-
c$Runtime$evaluate("document.querySelector('html').outerHTML")$result$value
%>%
  read_html()

# use CSS selectors to get the elements we want from the tables
# .vatop pulls standings column plus all the columns to the right of it
  half_table <- ".vatop"

  html2 <- html %>%
  html_elements(half_table) %>%
  html_text2() %>%
  matrix(ncol = 6, byrow = TRUE) %>%
```

```
  data.frame()

# eliminate the repeated headers in the code on these lines
html2 <- html2[-c(1, 22, 43, 64, 85, 106, 127, 148, 169, 190, 211, 232, 253,
274, 295),]

# tbl a pulls the date and team name columns
half_table2 <- "#tbl a"

html3 <- html %>%
  html_elements(half_table2) %>%
  html_text2() %>%
  matrix(ncol = 4, byrow = TRUE) %>%
  data.frame()

schedule <- cbind(html2, html3)

# use a different modify_stats_function for Massey data
# similar structure to D3Soccer website modify_stats_function
table_clean <- modify_stats_function(schedule, date)

c$close()

return(table_clean)

}

modify_stats_function <- function(date_results, date) {
   date_results <- date_results
names(date_results)[2] <- "score"
names(date_results)[9] <- "home_team"
names(date_results)[10] <- "away_team"
names(date_results)[7] <- "date"

d3_schedule <- date_results %>%
  select(score, home_team, away_team, date) %>%
  separate(score, into = c("home_score", "away_score"), sep = "\n") %>%
  mutate(away_team = str_remove(away_team, "@")) %>%
  mutate(away_team = str_trim(away_team, "left")) %>%
  select(home_team, home_score, away_team, away_score, date)
}
```

ui.R Code for Shiny App

```
library(stringr)
library(shiny)
library(dplyr)
library(rvest)
library(readr)
library(shinyWidgets)
library(elo)

#Define the UI for application
ui <- (fluidPage(
  # pick a theme
```

```
  theme = shinythemes::shinytheme("paper"),
  # application title
  titlePanel("Division III WSOC Ratings"),
  # sidebar
  sidebarLayout(
    sidebarPanel(
      selectInput(
        "Year", "Select a year: ", choices = c("2022", "2023"), selected =
"2023"),
      radioGroupButtons(
        inputId = "ConfReg", label = "", choices = c("Conference", "Region"),
justified = TRUE, selected = "Conference"),
      uiOutput("ConfRegBox")),
    mainPanel(
      tabsetPanel(
        tabPanel(
          "Standings", dataTableOutput("standings"),
          downloadButton("downloadstandings", "Download Standings File")),
        tabPanel(
          "Games", dataTableOutput("games"),
          downloadButton("downloadgames", "Download Games File")),
        tabPanel(
          "RPI PPG", dataTableOutput("rpi_ppg"),
          downloadButton("downloadrpi_ppg", "Download RPI_PPG File")),
        tabPanel(
          "RPI WP", dataTableOutput("rpi_wp"),
          downloadButton("downloadrpi_wp", "Download RPI_WP File")),
        tabPanel(
          "Poisson Rating", textOutput("poisson_stats"),
dataTableOutput("poisson_table"),
          downloadButton("downloadpoisson", "Download Poisson Ratings")),
        tabPanel(
          "Elo Rating",
          flowLayout(

            sliderInput("home_advantage", "Value for home field:",
                        min = 0, max = 100, value = 30, width = "100%"),
            sliderInput("k_value", "Choose value for k:", min = 0, max = 50,
value = 20, width = "100%")),
          dataTableOutput("elo_rating"),
          downloadButton("downloadelo", "Download Elo Ratings")),
        tabPanel("About",
                 h6("This R shiny app displays the standings and game
schedule based on conference or region for the 2022 and 2023 Division III
women's soccer teams. In addition, the app displays RPI rankings, Elo
rankings and offensive and defensive rankings based on a Poisson probability
model for each of the teams."),


                 h6("This R shiny app was developed by Hope Donoghue for a
honors Senior Year Experience in Data Science at St. Lawrence University
during the academic school year of 2023-2024. This project was supervised by
Dr. Robin Lock."),

                 h6("Data for this app is scraped from masseyratings.com"))
      )))))
```

Server.R Code for Shiny App

```
D3_2022 <- read_csv("D3_2022_massey.csv")
D3_2023 <- read_csv("D3_2023.csv")
team_details <- read_csv("D3_teams_Massey.csv")
team_details[nrow(team_details) + 1, ] = list("", "", "", "all", "all")
regions = c(
  "Region I",
  "Region II",
  "Region III",
  "Region IV",
  "Region V",
  "Region VI",
  "Region VII",
  "Region VIII",
  "Region IX",
  "Region X",
  "all"
)
conferences = sort(unique(team_details$Conference))
server <- function(input, output) {
  D3_year <- reactive(if (input$Year == "2022") {
    D3_2022
  } else {
    D3_2023
  })

  # SCHEDULE FUNCTION
  schedule_function <-
    function(schedule = D3_year(),
             team_list = team_details,
             conf = "all",
             reg = "all",
             league_only = TRUE) {
      if (conf != "all") {
        team_list <- team_list %>%
          filter(Conference == conf)
      }
      if (reg != "all") {
        team_list <- team_list %>%
          filter(Region == reg)
      }
      is_team_in_league_AWAY <-
        ifelse(schedule[['away_team']] %in% team_list$School, TRUE, FALSE)
      is_team_in_league_HOME <-
        ifelse(schedule[['home_team']] %in% team_list$School, TRUE, FALSE)
      df_Away <- as.data.frame(is_team_in_league_AWAY)
      df_Home <- as.data.frame(is_team_in_league_HOME)
      df_HOME_AWAY <- bind_cols(df_Away, df_Home, schedule)
      # schedule of teams in league since league_only is true
      if (league_only) {
        games <- df_HOME_AWAY %>%
          filter(is_team_in_league_AWAY == TRUE &
                   is_team_in_league_HOME == TRUE) %>%
          select(away_team, away_score, home_team, home_score, date)
      } else {
```

```
        # want out of conference team schedule
        games <- df_HOME_AWAY %>%
          filter(is_team_in_league_AWAY == TRUE |
                   is_team_in_league_AWAY == TRUE) %>%
          select(away_team, away_score, home_team, home_score, date)
      }
      return (games)
    }
  # STANDINGS FUNCTION
  standings_function <-
    function(schedule = D3_year(),
             team_list = team_details,
             conf = "all",
             reg = "all",
             league_only = TRUE) {
      # if the user picks a specific conference rather than default of all
conferences
      if (conf != "all") {
        team_list <- team_list %>%
          filter(Conference == conf)
      }
      # if the user picks a specific region rather than default of all
regions
      if (reg != "all") {
        team_list <- team_list %>%
          filter(Region == reg)
      }
      is_team_in_league_AWAY <-
        ifelse(schedule[['away_team']] %in% team_list$School, TRUE, FALSE)
      is_team_in_league_HOME <-
        ifelse(schedule[['home_team']] %in% team_list$School, TRUE, FALSE)
      df_Away <- as.data.frame(is_team_in_league_AWAY)
      df_Home <- as.data.frame(is_team_in_league_HOME)
      df_HOME_AWAY <- bind_cols(df_Away, df_Home, schedule)

      # schedule of teams in league since league_only is true
      if (league_only) {
        games <- df_HOME_AWAY %>%
          filter(is_team_in_league_AWAY == TRUE &
                   is_team_in_league_HOME == TRUE) %>%
          select(home_team, home_score, away_team, away_score, date)

      } else {
        # want out of conference team schedule
        games <- df_HOME_AWAY %>%
          filter(is_team_in_league_AWAY == TRUE |
                   is_team_in_league_AWAY == TRUE) %>%
          select(home_team, home_score, away_team, away_score, date)
      }
      # to get the home goals a team's scores and concedes
      home_goals <- games %>%
        group_by(home_team) %>%
        summarize(
          home_goals_for = sum(home_score, na.rm = TRUE),
          home_goals_against = sum(away_score, na.rm = TRUE)
        ) %>%
        rename("School" = home_team)
```

```
standings <- left_join(team_list, home_goals, by = "School")
# to get the away goals a team scores and concedes
away_goals <- games %>%
  group_by(away_team) %>%
  summarize(
    away_goals_for = sum(away_score, na.rm = TRUE),
    away_goals_against = sum(home_score, na.rm = TRUE)
  ) %>%
  rename("School" = away_team)
standings <- left_join(standings, away_goals, by = "School")
# to get total goals and goal differential for each team
standings <- standings %>%
  mutate(
    goals_scored = (home_goals_for + away_goals_for),
    goals_conceded = (home_goals_against + away_goals_against),
    goal_differential = goals_scored - goals_conceded
  )
# to get a team's home points
home_points <- games %>%
  group_by(home_team) %>%
  mutate(points = ifelse(
    home_score > away_score,
    3,
    ifelse(home_score == away_score, 1, 0)
  )) %>%
  summarize(
    home_points = sum(points, na.rm = TRUE),
    home_wins = sum(points == 3, na.rm = TRUE),
    home_ties = sum(points == 1, na.rm = TRUE),
    home_losses = sum(points == 0, na.rm = TRUE)
  ) %>%
  rename("School" = home_team)
standings <- left_join(standings, home_points, by = "School")
# to get a team's away points
away_points <- games %>%
  group_by(away_team) %>%
  mutate(points = ifelse(
    away_score > home_score,
    3,
    ifelse(home_score == away_score, 1, 0)
  )) %>%
  summarize(
    away_points = sum(points, na.rm = TRUE),
    away_wins = sum(points == 3, na.rm = TRUE),
    away_ties = sum(points == 1, na.rm = TRUE),
    away_losses = sum(points == 0, na.rm = TRUE)
  ) %>%
  rename("School" = away_team)

standings <- left_join(standings, away_points, by = "School")
# to get total points
standings <- standings %>%
  mutate(
    points = home_points + away_points,
    wins = home_wins + away_wins,
    ties = home_ties + away_ties,
    losses = home_losses + away_losses,
```

```
          games_played = wins + ties + losses,
          ppg = round((points / games_played), 2),
          goal_differential_pg = round((goal_differential / games_played),
2),
          win_percentage = round((wins + 0.5 * ties) / games_played, 2)
        )
      # to format the standings
      standings <- standings %>%
        arrange(desc(points), desc(goal_differential)) %>%
        select(
          School,
          points,
          goal_differential,
          goals_scored,
          goals_conceded,
          wins,
          losses,
          ties,
          games_played,
          ppg,
          goal_differential_pg,
          win_percentage
        ) %>%
        rename(
          Points = points,
          'Goal Differential' = goal_differential,
          'Goals Scored' = goals_scored,
          'Goals Conceded' = goals_conceded,
          Wins = wins,
          Losses = losses,
          Ties = ties,
          Games = games_played,
          PPG = ppg,
          'Goal Differential Per Game' = goal_differential_pg,
          'Win Percentage' = win_percentage
        )
      return(standings)
    }

    # SECOND FUNCTION FOR STANDINGS
    # function that takes in schedule of games with no extraneous teams: use
    # this function for rpi_wp and rpi_ppg
    standings2 <- function(games) {
      # to get the home goals a teams scores and concedes
      home_goals <- games %>%
        group_by(home_team) %>%
        summarize(
          home_goals_for = sum(home_score, na.rm = TRUE),
          home_goals_against = sum(away_score, na.rm = TRUE)
        ) %>%
        rename("School" = home_team)
      # to get the away goals a team scores and concedes
      away_goals <- games %>%
        group_by(away_team) %>%
        summarize(
          away_goals_for = sum(away_score, na.rm = TRUE),
          away_goals_against = sum(home_score, na.rm = TRUE)
```

```
  ) %>%
  rename("School" = away_team)
standings <- left_join(home_goals, away_goals, by = "School")
# to get total goals and goal differential for each team
standings <- standings %>%
  mutate(
    goals_scored = (home_goals_for + away_goals_for),
    goals_conceded = (home_goals_against + away_goals_against),
    goal_differential = goals_scored - goals_conceded
  )
# to get a team's home points
home_points <- games %>%
  group_by(home_team) %>%
  mutate(points = ifelse(
    home_score > away_score,
    3,
    ifelse(home_score == away_score, 1, 0)
  )) %>%
  summarize(
    home_points = sum(points, na.rm = TRUE),
    home_wins = sum(points == 3, na.rm = TRUE),
    home_ties = sum(points == 1, na.rm = TRUE),
    home_losses = sum(points == 0, na.rm = TRUE)
  ) %>%
  rename("School" = home_team)
standings <- left_join(standings, home_points, by = "School")
# to get a team's away points
away_points <- games %>%
  group_by(away_team) %>%
  mutate(points = ifelse(
    away_score > home_score,
    3,
    ifelse(home_score == away_score, 1, 0)
  )) %>%
  summarize(
    away_points = sum(points, na.rm = TRUE),
    away_wins = sum(points == 3, na.rm = TRUE),
    away_ties = sum(points == 1, na.rm = TRUE),
    away_losses = sum(points == 0, na.rm = TRUE)
  ) %>%
  rename("School" = away_team)
standings <- left_join(standings, away_points, by = "School")
# to get total points
standings <- standings %>%
  mutate(
    points = home_points + away_points,
    wins = home_wins + away_wins,
    ties = home_ties + away_ties,
    losses = home_losses + away_losses,
    record = paste(wins, losses, ties, sep = "-"),
    games_played = wins + ties + losses,
    ppg = (points / games_played),
    goal_differential_pg = (goal_differential / games_played),
    win_percentage = ((wins + 0.5 * ties) / games_played)
  )
# to format the standings
standings <- standings %>%
```

```
    arrange(desc(points), desc(goal_differential)) %>%
    select(
      School,
      points,
      goal_differential,
      goals_scored,
      goals_conceded,
      wins,
      losses,
      ties,
      record,
      games_played,
      ppg,
      goal_differential_pg,
      win_percentage
    ) %>%
    rename(
      Points = points,
      'Goal Differential' = goal_differential,
      'Goals Scored' = goals_scored,
      'Goals Conceded' = goals_conceded,
      Wins = wins,
      Losses = losses,
      Ties = ties,
      'Record (W-L-T)' = record,
      Games = games_played,
      PPG = ppg,
      'Goal Differential Per Game' = goal_differential_pg,
      'Win Percentage' = win_percentage
    )
  return(standings)
}
# RPI_PPG FUNCTION
rpi_function_ppg <-
  function(games,
           w1 = (0.25),
           w2 = (0.5),
           w3 = (0.25)) {
    # w1, w2, w3 are weights
    stand <- standings2(games)
    games <- games %>%
      rename(
        team = home_team,
        opponent = away_team,
        team_score = home_score,
        opp_score = away_score
      )

    games_2 <- games %>%
      rename(
        team = opponent,
        opponent = team,
        team_score = opp_score,
        opp_score = team_score
      )
    double_table <- bind_rows(games, games_2)
    ppg1 <-
```

```
      left_join(double_table, stand, by = c("opponent" = "School")) %>%
      select(team, PPG, 'Win Percentage') %>%
      rename(opp_ppg = PPG)
    opp_ppg <- ppg1 %>%
      group_by(team) %>%
      mutate(avg_opp_ppg = mean(opp_ppg)) %>%
      distinct(team, .keep_all = T) %>%
      select(team, avg_opp_ppg)
    new_teams_ppg <-
      left_join(stand, opp_ppg, by = c("School" = "team")) %>%
      rename("team" = "School")
    ppg2 <-
      left_join(double_table, new_teams_ppg, by = c("opponent" =
                                                      "team")) %>%
      select(-c(date, opp_score, team_score)) %>%
      rename(opp_ppg = PPG)
    new_opp_ppg <- ppg2 %>%
      group_by(team) %>%
      mutate(avg_opp_opp_ppg = mean(avg_opp_ppg)) %>%
      distinct(team, .keep_all = T) %>%
      select(team, avg_opp_opp_ppg)
    full_teams_ppg <-
      left_join(new_teams_ppg, new_opp_ppg, by = "team") %>% # to add
opponents opponents points per game
      mutate(RPI = (PPG * w1 + avg_opp_ppg * w2 + avg_opp_opp_ppg * w3))
%>%
      arrange(desc(RPI)) %>%
      select(team, RPI, PPG, avg_opp_ppg, avg_opp_opp_ppg) %>%
      rename(
        'Team' = team,
        'RPI' = RPI,
        'PPG' = PPG,
        'OPPG' = avg_opp_ppg,
        'OOPPG' = avg_opp_opp_ppg
      ) %>%
      mutate_if(is.numeric, round, digits = 2)
    full_teams_ppg$Rank = 1:nrow(full_teams_ppg)
    full_teams_ppg <- full_teams_ppg %>%
      relocate(Rank)
    return(full_teams_ppg)
  }
# RPI_WP FUNCTION
rpi_function_wp <-
  function(games,
           w1 = (0.25),
           w2 = (0.5),
           w3 = (0.25)) {
    stand <- standings2(games)

    games <- games %>%
      rename(
        team = home_team,
        opponent = away_team,
        team_score = home_score,
        opp_score = away_score
      )
    games_2 <- games %>%
```

```
        rename(
          team = opponent,
          opponent = team,
          team_score = opp_score,
          opp_score = team_score
        )
      wp1 <-
        left_join(double_table, stand, by = c("opponent" = "School")) %>%
        select(team, PPG, 'Win Percentage') %>%
        rename(opp_wp = 'Win Percentage')
      opp_wp <- wp1 %>%
        group_by(team) %>%
        mutate(avg_opp_wp = mean(opp_wp)) %>%
        distinct(team, .keep_all = T) %>%
        select(team, avg_opp_wp)
      new_teams_wp <-
        left_join(stand, opp_wp, by = c("School" = "team")) %>%
        rename("team" = "School")
      wp2 <- left_join(double_table, new_teams_wp, by = c("opponent" =
                                                            "team")) %>%
        select(-c(date, opp_score, team_score)) %>%
        rename(opp_wp = 'Win Percentage')
      new_opp_wp <- wp2 %>%
        group_by(team) %>%
        mutate(avg_opp_opp_wp = mean(avg_opp_wp)) %>%
        distinct(team, .keep_all = T) %>%
        select(team, avg_opp_opp_wp)
      full_teams_wp <-
        left_join(new_teams_wp, new_opp_wp, by = "team") %>% # to add
opponents opponents WP
        rename(WP = 'Win Percentage') %>%
        mutate(RPI = (WP * w1 + avg_opp_wp * w2 + avg_opp_opp_wp * w3)) %>%
        arrange(desc(RPI)) %>%
        select(team, RPI, WP, avg_opp_wp, avg_opp_opp_wp) %>%
        rename(
          'Team' = team,
          'RPI' = RPI,
          'WP' = WP,
          'OWP' = avg_opp_wp,
          'OOWP' = avg_opp_opp_wp
        ) %>%
        mutate_if(is.numeric, round, digits = 2)
      full_teams_wp$Rank = 1:nrow(full_teams_wp)
      full_teams_wp <- full_teams_wp %>%
        relocate(Rank)
      return(full_teams_wp)
    }

  # DOUBLE TABLE FUNCTION
  doubleTableFunc <- function(games) {
    games <- games %>%
      rename(
        team = home_team,
        opponent = away_team,
        team_score = home_score,
        opp_score = away_score
      ) %>%
```

```
    filter(!is.na(team_score & opp_score))
  games2 <- games %>% # new data table flipping the columns
    rename(
      team = opponent,
      opponent = team,
      team_score = opp_score,
      opp_score = team_score
    )
  double_table <- bind_rows(games, games2) # paste them together
  double_table$Location = NA # add location variable
  double_table$Location[0:length(double_table$date) / 2] = 1
  double_table[is.na(double_table)] = -1
  return(double_table)
}

# FUNCTION to Calculate Expected Poisson Points (with Bessel functions)
probs <- function(lambda1, lambda2) {
  prob_tie <-
    besselI(2 * sqrt(lambda1 * lambda2), 0) * exp(-(lambda1 + lambda2))

  total <- 0
  for (i in 1:20) {
    prob_win_iteration <-
      besselI(2 * sqrt(lambda1 * lambda2), i) * (lambda1 / lambda2) ^
          (i / 2) * exp(-(lambda1 + lambda2))
    total = total + prob_win_iteration
    i = i + 1
  }
  prob_win <- total
  prob_loss <- 1 - (prob_tie) - (prob_win)
  return(list(prob_win, prob_tie, prob_loss))
}

# POISSON FUNCTION
poisfunction <- function(schedule) {
  # based on a team's offensive and defensive ratings
  # team_score is the home team score
  model <-
    glm(
      team_score ~ team + opponent + Location,
      data = doubleTableFunc(schedule),
      family = poisson
    ) # to build the poisson model
  original <- data.frame(x = model$coefficients,
                rating = exp(model$coefficients))
  original$team <- rownames(original)
  row.names(original) <- NULL
  original <- original %>%
    select(team, x, rating)
  defAVG <- (original %>%
                filter(str_detect(team, "opponent") == TRUE) %>%
                summarise(sum(x) / (n() + 1)))[1, 1]
  offAVG <- (original %>%
                filter(str_detect(team, "team") == TRUE) %>%
                summarise(sum(x) / (n() + 1)))[1, 1]
  intercept <- (original$x)[1]
  homeAdj <- original$rating[length(original$rating)]
```

```
    adjusted <-
      data.frame(team = unique(doubleTableFunc(schedule)$team))
    adjusted <- data.frame(team = adjusted[order(adjusted$team),])
    adjusted$Off = append(0, original$x[2:(length(original$x) / 2)])
    adjusted$Def = append(0, original$x[(length(original$x) / 2 +
1):(length(original$x) - 1)])
    adjusted$OffAdj = adjusted$Off + defAVG + intercept
    adjusted$DefAdj = adjusted$Def + offAVG + intercept
    adjusted$expOff = exp(adjusted$Off + defAVG + intercept)
    adjusted$expDef = exp(adjusted$Def + offAVG + intercept)

    # bump scoring rate to 0.05
    adjusted$expOff[adjusted$expOff < 0.05] <- 0.05

    # bump defensive rate to 0.05
    adjusted$expDef[adjusted$expDef < 0.05] <- 0.05

    #average offensive
    avgPoisOffCoef <- mean(adjusted$OffAdj)
    # average defensive
    avgPoisDefCoef <- mean(adjusted$DefAdj)
    avgPoisCoef <-
      exp((avgPoisDefCoef + avgPoisOffCoef) / 2) # avg coefficient
    scoreRate <-
      mean(doubleTableFunc(schedule)$team_score) # avg scoring rate
    C = scoreRate / avgPoisCoef
    poissonOFFavg = mean(adjusted$expOff)
    poissonDEFavg = mean(adjusted$expDef)
    D = sqrt(C * (poissonDEFavg / poissonOFFavg))
    adjusted$newOffExp = adjusted$expOff * D
    adjusted$newDefExp = adjusted$expDef * (C / D)

    returnedTable <- adjusted %>%
      mutate(
        OffRating = newOffExp,
        DefRating = newDefExp,
        OverallRating = newOffExp / newDefExp
      ) %>%
      rename(Team = "team") %>%
      select(Team,
             OffRating,
             DefRating,
             OverallRating) %>%
      arrange(desc(OffRating))
    returnedTable$OffRank = 1:nrow(returnedTable)

    returnedTable <- returnedTable %>%
      arrange(DefRating)
    returnedTable$DefRank = 1:nrow(returnedTable)

    returnedTable <- returnedTable %>%
      arrange(desc(OverallRating)) %>%
      mutate_if(is.numeric, round, digits = 2) %>%
      select(Team,
             OverallRating,
             OffRating,
             OffRank,
```

```
            DefRating,
            DefRank)

    avgOffRating <- mean(returnedTable$OffRating)
    sum = rep(0, length(returnedTable$Team))

    for (i in 1:length(returnedTable$Team)) {
      for (j in 1:length(returnedTable$Team)) {
        lambda1 = (returnedTable$OffRating[i] *
                   returnedTable$DefRating[j]) / (avgOffRating)

        lambda2 = (returnedTable$OffRating[j] *
                   returnedTable$DefRating[i]) / (avgOffRating)

        x <- probs(lambda1, lambda2)
        sum[i] = sum[i] + (3 * x[[1]]) + x[[2]]
      }
    }
    expected_points <- sum / (length(returnedTable$Team))
    returnedTable$OverallRating <- round(expected_points, 2)

    returnedTable <- returnedTable %>%
      arrange(desc(OverallRating))

    returnedTable$Rank = 1:nrow(returnedTable)

    returnedTable <- returnedTable %>%
      select(Rank,
             Team,
             OverallRating,
             OffRating,
             OffRank,
             DefRating,
             DefRank)

    return(list(returnedTable, homeAdj, avgOffRating))
  }

  # ELO FUNCTION
  elo_rating_function <- function(schedule) {
    elo_ratings <-
      elo.run(
        score(home_score, away_score) ~ adjust(home_team,
input$home_advantage) + away_team + k(input$k_value + abs(home_score -
away_score) * 10),
        data = schedule
      )
    elo_object <- as.matrix(elo_ratings)
    elo_object <- elo_object %>%
      tail(n = 1) # select only the last row
    rownames(elo_object) <- NULL
    elo_object <- elo_object %>%
      t() # use the transpose function in R to switch columns and rows
    elo_df <- as.data.frame(elo_object) %>%
      cbind(rownames(elo_object), elo_object) %>%
      select(2:3) %>%
      rename(Team = "rownames(elo_object)", "Elo Rating" = "elo_object") %>%
```

```
    mutate_if(is.numeric, round, digits = 2) %>%
    arrange(desc(`Elo Rating`))
  rownames(elo_df) <- NULL
  elo_df$Rank = 1:nrow(elo_df)
  elo_df <- elo_df %>%
    relocate(Rank)
  return (elo_df)
}

# Output code to call functions
output$ConfRegBox <- renderUI({
  if (input$ConfReg == "Conference") {
    selectInput("Conference",
                "Choose a Conference",
                choices = conferences,
                selected = "LL")
  } else {
    selectInput("Region",
                "Choose a Region",
                choices = regions,
                selected = "Region III")
  }
})

games <- reactive(if (input$ConfReg == "Conference") {
  schedule_function(conf = input$Conference)
} else {
  schedule_function(reg = input$Region)
})

output$games <- renderDataTable(games())

standings <- reactive(if (input$ConfReg == "Conference") {
  standings_function(conf = input$Conference)
} else {
  standings_function(reg = input$Region)
})
output$standings <- renderDataTable(standings())

rpi_ppg <- reactive(rpi_function_ppg(games()))
output$rpi_ppg <- renderDataTable(rpi_ppg())

rpi_wp <- reactive(rpi_function_wp(games()))
output$rpi_wp <- renderDataTable(rpi_wp())

poisson_x <- reactive(poisfunction(games()))
output$poisson_table <- renderDataTable(poisson_x()[[1]])

output$poisson_stats <- renderText(paste(
  "Mean Rating = ",
  round(poisson_x()[[2]], 3),
  "   Home Adjustment = ",
  round(poisson_x()[[3]], 3)
))

elo_rating_1 <- reactive(elo_rating_function(games()))
output$elo_rating <- renderDataTable(elo_rating_1())
```

```
  output$downloadgames <- downloadHandler(
    filename = "Games.csv",
    content = function(file) {
      write.csv(games(), file, row.names = FALSE)
    }
  )

  output$downloadstandings <- downloadHandler(
    filename = "Standings.csv",
    content = function(file) {
      write.csv(standings(), file, row.names = FALSE)
    }
  )

  output$downloadpoisson <- downloadHandler(
    filename = "Poisson.csv",
    content = function(file) {
      write.csv(poisson_x()[[1]], file, row.names = FALSE)
    }
  )
  output$downloadelo <- downloadHandler(
    filename = "Elo.csv",
    content = function(file) {
      write.csv(elo_rating_1(), file, row.names = FALSE)
    }
  )
  output$downloadrpi_ppg <- downloadHandler(
    filename = "RPI_PPG.csv",
    content = function(file) {
      write.csv(rpi_ppg(), file, row.names = FALSE)
    }
  )
  output$downloadrpi_wp <- downloadHandler(
    filename = "RPI_WP.csv",
    content = function(file) {
      write.csv(rpi_wp(), file, row.names = FALSE)
    }
  )
}
```

Simulation Code

```
schedule <- read_csv("schedule_LL_reg_season_massey.csv")
ratings <- read_csv("LL_pois_ratings.csv")

# Joining schedule file to ratings file

big_sched  <- left_join(schedule, ratings, by = c("away_team" =
                                                  "team"))
big_sched <- left_join(big_sched, ratings, by = c("home_team" =
                                                  "team"))
big_sched <- big_sched %>%
  rename("away_off" = OffRating.x, "away_def" = DefRating.x, "home_off" =
OffRating.y, "home_def" = DefRating.y)
```

```
# Score rate function
score_rate <- function(OffRating, DefRating, Home, HomeAdj = 1.11, AvgRating
= 1.044){
  return(OffRating*DefRating*HomeAdj^Home / AvgRating)
}

# Initialize data frames to hold ranks
ranks_elo_2023 <- data.frame(matrix(ncol = nteams, nrow = 0))
names(ranks_elo_2023) = ratings$team

ranks_pois_2023 <- data.frame(matrix(ncol = nteams, nrow = 0))
names(ranks_pois_2023) = ratings$team

ranks_ppg_2023 <- data.frame(matrix(ncol = nteams, nrow = 0))
names(ranks_ppg_2023) = ratings$team

ranks_points_2023 <- data.frame(matrix(ncol = nteams, nrow = 0))
names(ranks_points_2023) = ratings$team

ranks_rpi_ppg_2023 <- data.frame(matrix(ncol = nteams, nrow = 0))
names(ranks_rpi_ppg_2023) = ratings$team

ranks_rpi_wp_2023 <- data.frame(matrix(ncol = nteams, nrow = 0))
names(ranks_rpi_wp_2023) = ratings$team

# Loop for simulating many seasons
n_games <- length(big_sched$away_team)
home <- 1.11
avg_rating <- mean(ratings$OffRating) # get from ratings file passed in
numseasons=1000
for(i in 1:numseasons){

  if((i/100) == round(i/100,0)){
    print(i)
  }

  v <- rep(1, n_games)

  one_season <- big_sched %>%
    mutate(away_rate = score_rate(away_off, home_def, -1, home, avg_rating))
%>%
    mutate(home_rate = score_rate(home_off, away_def, 1, home, avg_rating))
%>%
    mutate(home_score = rpois(v, home_rate)) %>%
    mutate(away_score = rpois(v, away_rate)) %>%
    select(c(1:5,19,20))

  elo_season <- elo_rating_auto(one_season,KValue=30)

  ranks_elo_2023[i,] <- (nteams+1) - rank(elo_season$`Elo Rating`)

  poisson_season <- poisfunction(one_season)[[1]] %>%
  arrange((Team))

  ranks_pois_2023[i,] <- poisson_season$Rank

  ppg_season <- standings2(one_season)
```

```
  ppg_season <- ppg_season[order(ppg_season$School),]

  ranks_ppg_2023[i,] <- (nteams+1) - rank(ppg_season$PPG)

  points_season <- standings2(one_season)
  points_season <- points_season[order(points_season$School),]

  ranks_points_2023[i,] <- (nteams+1) - rank(points_season$Points)

  rpi_wp_season <- rpi_function_wp(one_season)
  rpi_wp_season <- rpi_wp_season[order(rpi_wp_season$Team),]

  ranks_rpi_wp_2023[i,] <- (nteams+1) - rank(rpi_wp_season$RPI)

  rpi_ppg_season <- rpi_function_ppg(one_season)
  rpi_ppg_season <- rpi_ppg_season[order(rpi_ppg_season$Team),]

  ranks_rpi_ppg_2023[i,] <- (nteams+1) - rank(rpi_ppg_season$RPI)
}
```

MSE Function

```
# function for calculating mean squared error
rank_mse <- function(ranks_file){
  stat_longer <- stack(ranks_file) %>%
  rename("Rank" = "values") %>%
  rename("Team" = "ind") %>%
  mutate(order = rep(ratings$Rank, each = numseasons))
mse <- round(sum((stat_longer$order -
stat_longer$Rank)^2)/(numseasons*nteams), 2)
return(mse)
}
```

MAD Function

```
# function for calculating mean absolute deviation
rank_mad <- function(ranks_file){
  stat_longer <- stack(ranks_file) %>%
  rename("Rank" = "values") %>%
  rename("Team" = "ind") %>%
 mutate(order = rep(ratings$Rank, each = numseasons))
mad <- round(sum(abs(stat_longer$order -
stat_longer$Rank))/(numseasons*nteams), 3)
return(mad)
}
```

Simulation Boxplots

```
pois_longer <- stack(ranks_pois_2023) %>%
  rename("Rank" = "values") %>%
  rename("Team" = "ind") %>%
  mutate(order = rep(ratings$Rank, each = numseasons))
```

```
rpi_wp_longer <- stack(ranks_rpi_wp_2023) %>%
  rename("Rank" = "values") %>%
  rename("Team" = "ind") %>%
 mutate(order = rep(ratings$Rank, each = numseasons))

# RPI WP ranking boxplot
ggplot(data = rpi_wp_longer, aes(x = fct_reorder(Team, .x = order, .desc =
TRUE), y = Rank, fill = "orange")) + geom_boxplot() + coord_flip() +
stat_summary(fun.y="mean", shape=3) + theme_minimal() + labs(x = "Team", y =
"Rank", title = "RPI WP Rankings for Three Leagues") + theme(legend.position
= "none") + scale_y_continuous(breaks=c(1:nteams))

# Poisson ranking boxplot
ggplot(data = pois_longer, aes(x = fct_reorder(Team, .x = order, .desc =
TRUE), y = Rank, fill = "orange")) + geom_boxplot() +
stat_summary(fun.y="mean", shape=3) + coord_flip() + theme_minimal() + labs(x
= "Team", y = "Rank", title = "Poisson Rankings for Three Leagues") +
theme(legend.position = "none") + scale_y_continuous(breaks=c(1:nteams))
```