Scraping and Analyzing NCAA Data with Applications to Liberty League Volleyball

Cassie Richter

Advisor: Dr. Ivan Ramler

Data Science Honors Thesis

Department of Mathematics, Computer Science, and Statistics

St. Lawrence University

May 11th, 2023

ABSTRACT

In this project, we collect and store NCAA Sports Data in a way that makes it more accessible for researchers and instructors. Using ECAC Division 3 Volleyball data, we demonstrate effective scraping in R and how to efficiently store multiple tables into a relational database. Finally, we show sample statistical analyses that make use of different features of the match and team data to visualize and investigate win percentages.

1 INTRODUCTION

Data science is a growing field, and a popular sector continues to be sports analytics. Professional sports all have teams of analysts working to organize and analyze statistics of the game, and much of these data is widely available through R, github, python etc.; however, NCAA/collegiate data is much less accessible (Morgulev, 2018). The NCAA has done a great job of standardizing their sports pages to make scraping data easier. The new format began in 2013, across all divisions and sports, and has continued through this year's seasons. This project focuses on the collection process for the NCAA Division 3 Volleyball data, specifically for the Liberty League, which contains St. Lawrence University.

In order to understand the organization and analysis of these data, it's important to first know some basics of volleyball. Each overall match that is played contains sets within it. Each set is played to 25 points, where the winner must win by 2, and to win a match, a team must win the best of 5 sets. Positions in volleyball which are also associated with a court number, include setters (1) , liberos (defensive specialists) (6), and outside (4/5), middle (3) and right-side hitters(2). A diagram of these positions and where they play on the court can be seen in Figure 1.

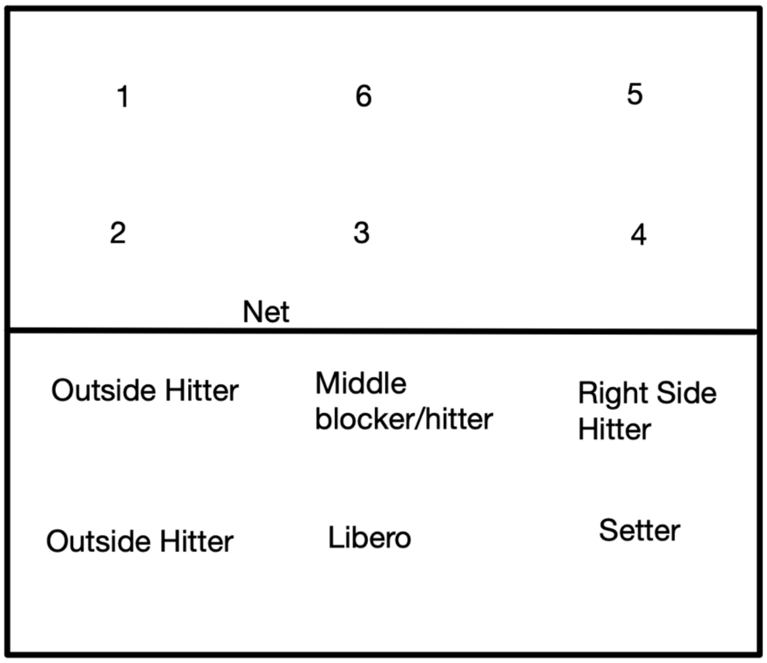


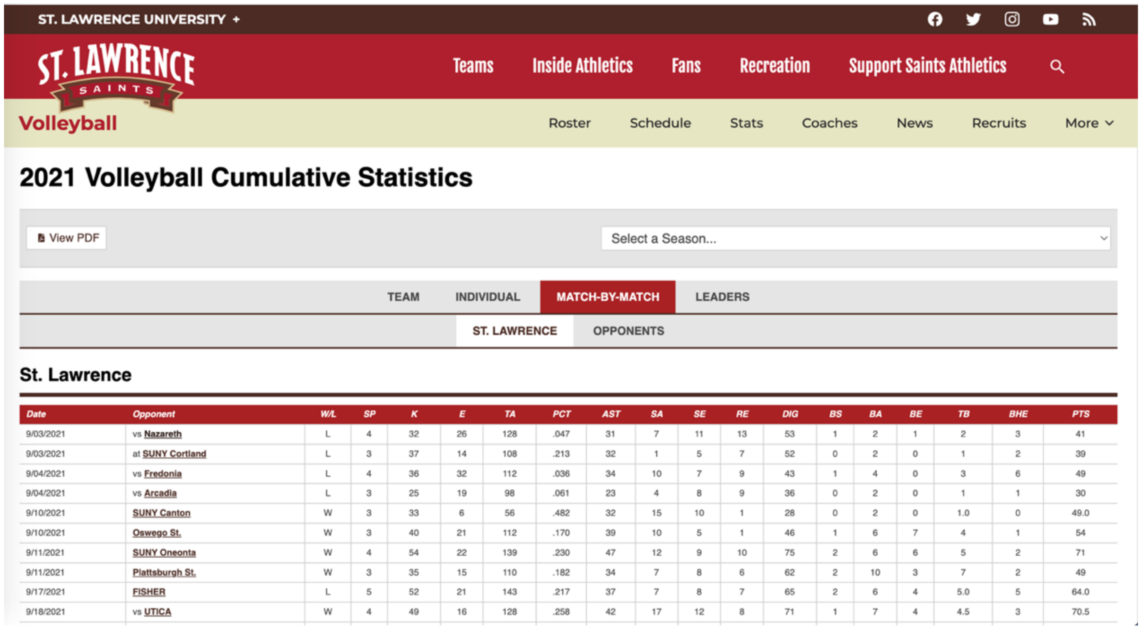
Figure 1. Volleyball Court Positions

The goal with these data is to be able to collect and process them in a systematic and repeatable way, and then organize them in a fashion for analysis. This will make Liberty League Volleyball more available for instructors and anyone who wishes to analyze it, as well as provide a blueprint for collecting data on other leagues, divisions, or sports in the NCAA. The Data Collection and Processing section of this paper discusses how we scrape the data for the Liberty League 2013-2022 seasons and store it into tables for Boxscores, Per Set Statistics, Individual Statistics, and Play-By-Play data. The Databases and Analysis sections of the paper then will describe how we can store and visualize this in a relational database and access it through SQL and a python interface, and how analysis can be done in R. The code for functions that we discuss throughout these sections will all be available on github and linked in the appendix of the paper.

2 DATA COLLECTION AND PROCESSING

2.1 Gathering Proper URLs

This section will discuss how we scrape and clean the data from the team webpages into csv files to be analyzed. Figure 2 exhibits the current form of the volleyball statistics homepages. The match-by-match tab of each statistics home page is what we are interested in. As shown in Figure 2, each school in the opponent column is a hyperlink that takes you to the Match-by-match data we want. First, we create a function, `get_first_google_link`(), that takes in a vector of teams, and uses the `URLencode()` function from utils package to Google search these teams with the search bar "women's volleyball stats", and paste the year and team name, to then grab their home team URLs as a base (ex https://saintsathletics.com). Then we paste on "/sports/womens-volleyball/stats/" + year, to get the main stats pages for each team in the format https://saintsathletics.com/sports/womens-volleyball/stats/2021. We did this for the 9 teams in the Liberty League: St. Lawrence, Clarkson, Ithaca, Union, Vassar, RIT, William Smith, Bard and Skidmore.



2021 Volleyball Cumulative Statistics

St. Lawrence

Date	Opponent	W/L	SP	K	E	TA	PCT	AST	SA	SE	RE	DIG	BS	BA	BE	TB	BHE	PTS
9/03/2021	vs Nazareth	L	4	32	26	128	.047	31	7	11	13	53	1	2	1	2	3	41
9/03/2021	at SUNY Cortland	L	3	37	14	108	.213	32	1	5	7	52	0	2	0	1	2	39
9/04/2021	vs Fredonia	L	4	36	32	112	.036	34	10	7	9	43	1	4	0	3	6	49
9/04/2021	vs Arcadia	L	3	25	19	98	.061	23	4	8	9	36	0	2	0	1	1	30
9/10/2021	SUNY Canton	W	3	33	6	56	.482	32	15	10	1	28	0	2	0	1.0	0	49.0
9/10/2021	Oswego St.	W	3	40	21	112	.170	39	10	5	1	46	1	6	7	4	1	54
9/11/2021	SUNY Oneonta	W	4	54	22	139	.230	47	12	9	10	75	2	6	6	5	2	71
9/11/2021	Plattsburgh St.	W	3	35	15	110	.182	34	7	8	6	62	2	10	3	7	2	49
9/17/2021	FISHER	L	5	52	21	143	.217	37	7	8	7	65	2	6	4	5.0	5	64.0
9/18/2021	vs UTICA	W	4	49	16	128	.258	42	17	12	8	71	1	7	4	4.5	3	70.5

Figure 2. Volleyball Statistics Homepage.

Next, we get the match url that is associated with each opponent hyperlink for all of the teams that we have just acquired the URLs for. We pass the teams Volleyball Statistics homepage URLs into a function `get_complete_match_urls()`, which uses `read_html()`, `html_nodes()` and `html_attr()` from the rvest package to scrape the desired match links for all matches for each team and then gets rid of any non-unique URLs. The format of a match url has a base of the school:

https://saintsathletics.com,

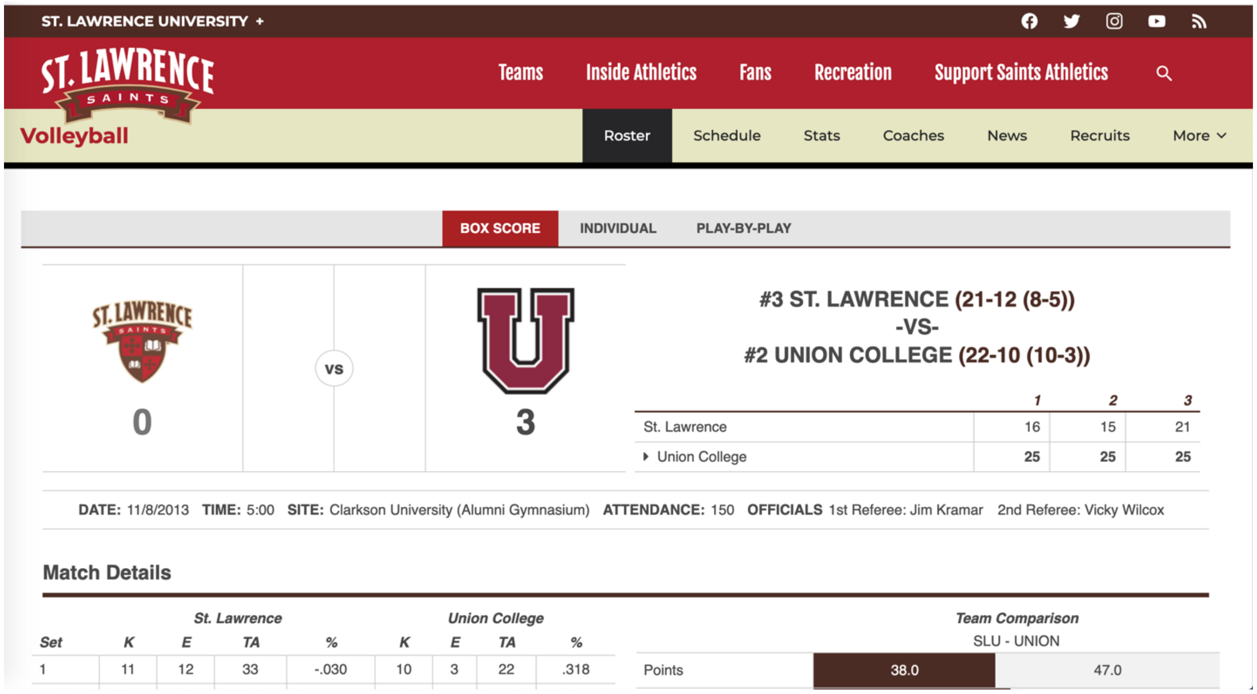
and has the following attachment,

/boxscore.aspx?id=3485&path=wvball

with a unique id= for each match, to give a match url formatted as:

https://saintsathletics.com/boxscore.aspx?id=3485&path=wvball.

Figure 3 shows the match statistics page from where we will go on to scrape the data. The main section of the page in Figure 3 is where the boxscore data comes from, and below in "Match Details" is what will become the per set statistics data. To get all the data desired, we put the match urls into a file complete_url_list.txt, so we are able to loop through and pull data from all URLs.



0 | 3

#3 ST. LAWRENCE (21-12 (8-5))
-VS-
#2 UNION COLLEGE (22-10 (10-3))

	1	2	3
St. Lawrence	16	15	21
Union College	25	25	25

DATE: 11/8/2013 TIME: 5:00 SITE: Clarkson University (Alumni Gymnasium) ATTENDANCE: 150 OFFICIALS 1st Referee: Jim Kramar 2nd Referee: Vicky Wilcox

Match Details

	St. Lawrence				Union College			
Set	K	E	TA	%	K	E	TA	%
1	11	12	33	-.030	10	3	22	.318

Team Comparison		
	SLU - UNION	
Points	38.0	47.0

Figure 3. Match homepage

2.2 Looping through each URL

When getting the data from each of these match URLs, we first scrape the data using our `scrape_url`() function, then clean the data with an individualized function for each table. We then pass the cleaned data into a `save_table`() function, which saves our cleaned table into a csv and appends for each succeeding URL. This loop is provided below, showing how we read in the complete_url_list file, and pass each URL through our functions. This loop is specifically made for boxscores data, but the loops is of identical structure for each of the other tables, with different function names.

```
complete_url_list <- scan("complete_url_list_ALL.txt", what = "character")

for (u in complete_url_list) {
   print(u)
   if (url.exists(u) == TRUE){
     ## scraped takes url, outputs obj tables
     scraped <- scrape_url(u)
     ## clean takes scraped obj table and outputs list of cleaned
     cleaned <- clean_boxscore(scraped)
     ## takes list of cleaned tables
```

```
        keep_names = FALSE
        if (u == complete_url_list[1]){
          keep_names = TRUE
        }
        save_boxscores(cleaned, keep_names)
      }
    }
```

2.3 The Scraping Function

The `scrape_url`() function is consistent over each of the tables and it takes in a url, and outputs tables that can be pulled. This function uses `html_node`() and `html_text()` to pull all of the data frames from the page so we then can store each section of the webpage in objects that can be identified for boxscores, per set stats, individuals and play by play tabs. The scrape function also allows us to pull the match information from the html text, which can be attached to each of our tables as identifying variables. The function cleans and formats the match information within it to select Date, Time, Attendance, and Location, and then attach the match's unique URL. The `scrape_url`() function outputs a list with all of the separate objects that our cleaning functions then pull to process and format the tables into our desired form.

3 PROCESS AND STORE

After acquiring the tables through our scraping function, we pass the data into a cleaning function and a saving function to put the formatted data into a csv file. This section will discuss how we clean and store each of our four csv files.

3.1 Boxscores Data

The first cleaning function we use is `clean_boxscore()` which takes an outputted object from the `scrape_url`() function, and cleans it into a boxscores table.This function is the most simple of the tables, as it just has to pull the table out and do some light cleaning using functions in the dplyr package. A home and away variable is created for identification, and we attach the match information (identifying columns) to the right side.

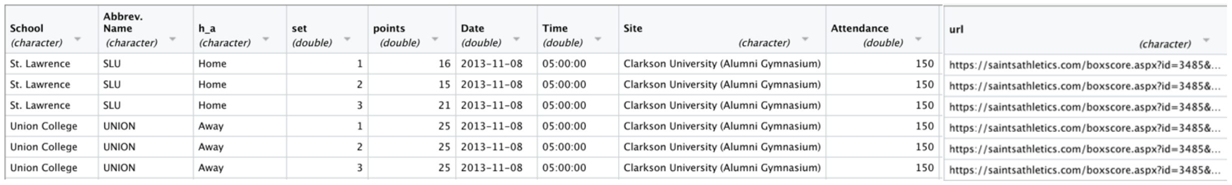
Then we pass the data into the save function, `save_boxscores`(). This function takes the cleaned data and sets the column names of the csv using the first table passed in. The rows of the boxscore data contains one teams points for one set, so each match has 2*number of sets rows. So, each match will have 6-10 rows in the boxscores data. When each match is appended to this csv it will use those initial column headers and append the data below the previous match. The `save_boxscores`() function is below, and this structure is the same for latter tables, with just the name of the csv file changing.

```
save_boxscores <- function(cleaned, set_the_names){
    write_csv(file='boxscoresNEW.csv',x=cleaned,col_names= set_the_names,
  append=TRUE)
}
```

Figure 4 shows the boxscores data final format, with each set per match and the associated number of points scored per team, with the match information column for that match attached to each row. The h_a variable is to identify if that team was the home or away team. In volleyball however, this is sometimes arbitrarily assigned variable, mostly in the case of tournaments when there is no true home team, but one is still always assigned. This table has 10,146 rows for 1,207 matches, on 10 variables, which are described in Table 1.

Name of Variable	**Variable Measure**
School	School/team name
Abbrev. Name	School abbreviation (ex. SLU for St. Lawrence University)
h_a	Whether the match is home or away for that team
set	Set number of the match
points	Number of points scored in that set
Date	Date of the match
Time	Time of the match
Site	Location of the match
Attendance	Recorded attendance of the match
URL	Unique match url

Table 1. A Description of Boxscores Variables



School (character)	Abbrev. Name (character)	h_a (character)	set (double)	points (double)	Date (double)	Time (double)	Site (character)	Attendance (double)	url (character)
St. Lawrence	SLU	Home	1	16	2013-11-08	05:00:00	Clarkson University (Alumni Gymnasium)	150	https://saintsathletics.com/boxscore.aspx?id=3485&...
St. Lawrence	SLU	Home	2	15	2013-11-08	05:00:00	Clarkson University (Alumni Gymnasium)	150	https://saintsathletics.com/boxscore.aspx?id=3485&...
St. Lawrence	SLU	Home	3	21	2013-11-08	05:00:00	Clarkson University (Alumni Gymnasium)	150	https://saintsathletics.com/boxscore.aspx?id=3485&...
Union College	UNION	Away	1	25	2013-11-08	05:00:00	Clarkson University (Alumni Gymnasium)	150	https://saintsathletics.com/boxscore.aspx?id=3485&...
Union College	UNION	Away	2	25	2013-11-08	05:00:00	Clarkson University (Alumni Gymnasium)	150	https://saintsathletics.com/boxscore.aspx?id=3485&...
Union College	UNION	Away	3	25	2013-11-08	05:00:00	Clarkson University (Alumni Gymnasium)	150	https://saintsathletics.com/boxscore.aspx?id=3485&...

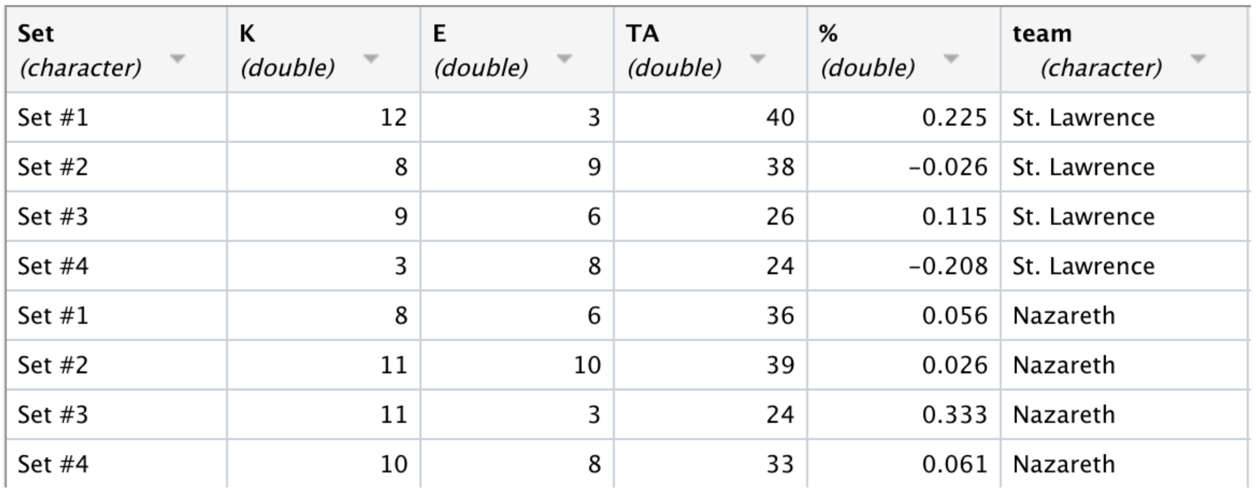
Figure 4. Boxscores Data Final Format

3.2 Per Set Statistics

For the next table, per set stats, we pass the scraped tables into `clean_perset`(), a function for formatting. This sets the column headers and uses `pivot_longer()` from the tidyr package to have each team's set data stacked on top of each other per match. These metrics are the total team's statistics for each set within the match. The function then attaches the identifying variable columns on the right, and outputs the table. We use the `save_persetstats`() function to append each of the match's set statistics into one csv file "per_set_stats.csv'. Figure 5 shows the final format of these data in the csv file, with the match information column unpictured, but attached on the right side of this table. This table has 10,146 rows, for 1,207 matches, on 11 variables, which are identified in Table 2.

Name of Variable	Variable Measure
Set	Set number of the match
K	Kills
E	Errors
TA	Total attempts made
%	((K-E)/TA *100)
Team	The school/team name

Table 2. A Description of the first 6 Variables in the Per Set Stats Data (not including the identifying variables)

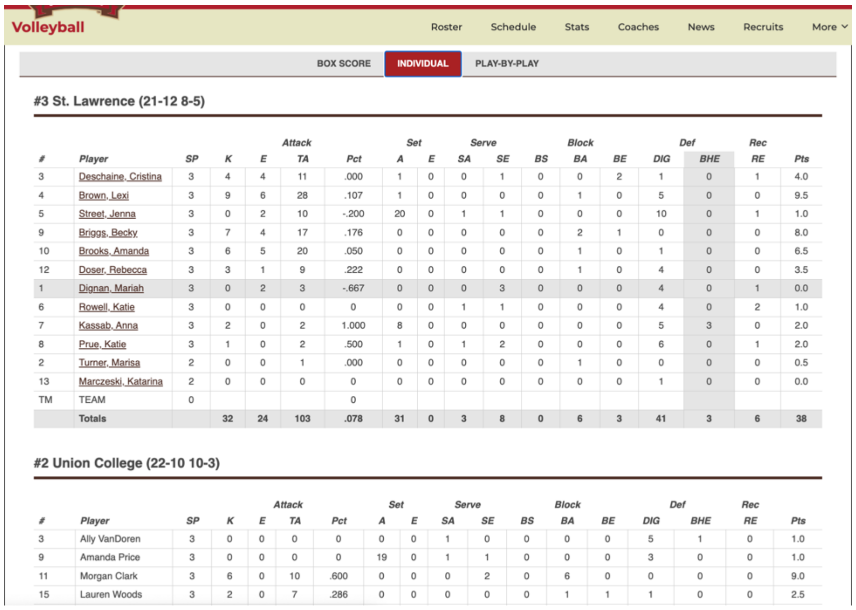


Set (character)	K (double)	E (double)	TA (double)	% (double)	team (character)
Set #1	12	3	40	0.225	St. Lawrence
Set #2	8	9	38	−0.026	St. Lawrence
Set #3	9	6	26	0.115	St. Lawrence
Set #4	3	8	24	−0.208	St. Lawrence
Set #1	8	6	36	0.056	Nazareth
Set #2	11	10	39	0.026	Nazareth
Set #3	11	3	24	0.333	Nazareth
Set #4	10	8	33	0.061	Nazareth

Figure 5. Per Set Stats Table Structure

3.3 Individual Player Data

Individual player data were gathered for every player that played in the match, and the function `clean_individuals`(), formats the data. The previously extracted data from the individuals tab of the match webpage outputs separate data on home and away players. The form of the individuals webpage tab can be seen in Figure 6. This function takes those, and cleans the column names, and recombines them so all players (home and away) are put into one table. We now attach match information variables, and save the data to 'individuals.csv' with the `save_individuals`() function. The final form can be seen in Figure 7 (with identifying variables attached but unpictured). Each player's data is per match, so their statistics from each set are totaled and in the table for that player and match. This table has 30,716 rows on 23 variables, which are described in Table 3.



#3 St. Lawrence (21-12 8-5)

					Attack		Set		Serve			Block		Def		Rec	
#	Player	SP	K	E	TA	Pct	A	E	SA	SE	BS	BA	BE	DIG	BHE	RE	Pts
3	Deschaine, Cristina	3	4	4	11	.000	1	0	0	1	0	0	2	1	0	1	4.0
4	Brown, Lexi	3	9	6	28	.107	1	0	0	0	0	1	0	5	0	0	9.5
5	Street, Jenna	3	0	2	10	-.200	20	0	1	1	0	0	0	10	0	1	1.0
9	Briggs, Becky	3	7	4	17	.176	0	0	0	0	0	2	1	0	0	0	8.0
10	Brooks, Amanda	3	6	5	20	.050	0	0	0	0	0	1	0	1	0	0	6.5
12	Doser, Rebecca	3	3	1	9	.222	0	0	0	0	0	1	0	4	0	0	3.5
1	Dignan, Mariah	3	0	2	3	-.667	0	0	0	3	0	0	0	4	0	1	0.0
6	Rowell, Katie	3	0	0	0	0	0	0	1	1	0	0	0	4	0	2	1.0
7	Kassab, Anna	3	2	0	2	1.000	8	0	0	0	0	0	0	5	3	0	2.0
8	Prue, Katie	3	1	0	2	.500	1	0	1	2	0	0	0	6	0	1	2.0
2	Turner, Marisa	2	0	0	1	.000	0	0	0	0	0	1	0	0	0	0	0.5
13	Marczeski, Katarina	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0.0
TM	TEAM	0				0											
	Totals		32	24	103	.078	31	0	3	8	0	6	3	41	3	6	38

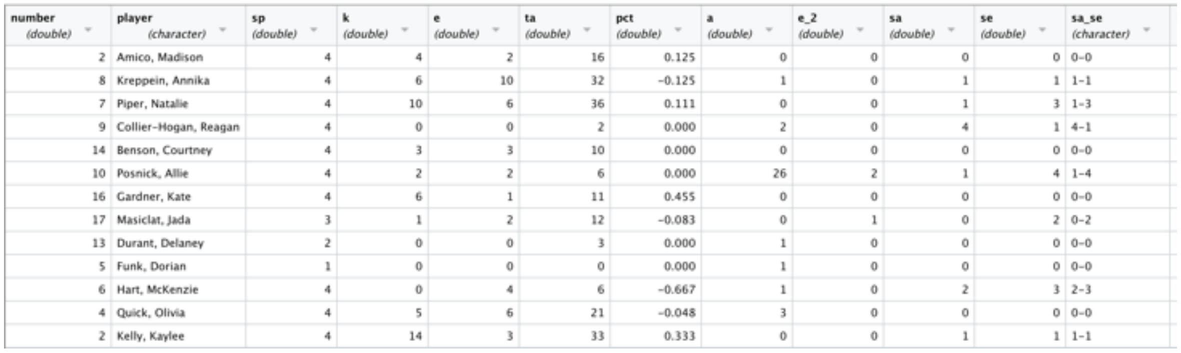
#2 Union College (22-10 10-3)

					Attack		Set		Serve			Block		Def		Rec	
#	Player	SP	K	E	TA	Pct	A	E	SA	SE	BS	BA	BE	DIG	BHE	RE	Pts
3	Ally VanDoren	3	0	0	0	0	0	0	1	0	0	0	0	5	1	0	1.0
9	Amanda Price	3	0	0	0	0	19	0	1	1	0	0	0	3	0	0	1.0
11	Morgan Clark	3	6	0	10	.600	0	0	0	2	0	6	0	0	0	0	9.0
15	Lauren Woods	3	2	0	7	.286	0	0	0	0	0	1	1	1	0	0	2.5

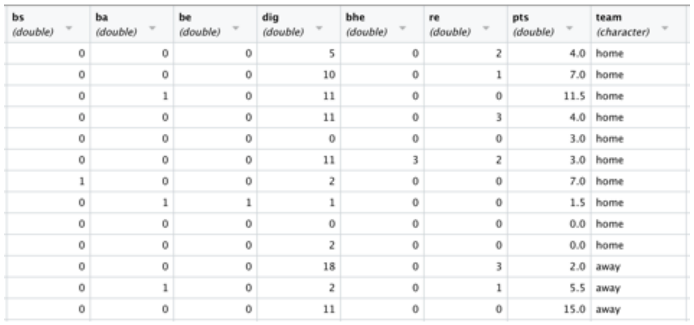
Figure 6. Individuals Tab on Match Homepage

Name of Variable	Variable Measure
Player	First and last name
Sp	Sets played in match
K	Kills
E	Errors
TA	Total Attempts
Pct	Percent ((K-E)/TA *100)
A	Assists
Sa	Service aces
Se	Service errors
Sa_se	Ratio aces-errors (Ex: 4-1)
Bs	Solo blocks
Ba	Block assists
Be	Block errors
Dig	Digs (defensive pass)
Bhe	Ball Handling Errors
Re	Reception Error
Pts	Points
Team	Team/School

Table 3. Individual Player Stats Variables and Meaning



number (double)	player (character)	sp (double)	k (double)	e (double)	ta (double)	pct (double)	a (double)	e_2 (double)	sa (double)	se (double)	sa_se (character)
2	Amico, Madison	4	4	2	16	0.125	0	0	0	0	0-0
8	Kreppein, Annika	4	6	10	32	-0.125	1	0	1	1	1-1
7	Piper, Natalie	4	10	6	36	0.111	0	0	1	3	1-3
9	Collier-Hogan, Reagan	4	0	0	2	0.000	2	0	4	1	4-1
14	Benson, Courtney	4	3	3	10	0.000	0	0	0	0	0-0
10	Posnick, Allie	4	2	2	6	0.000	26	2	1	4	1-4
16	Gardner, Kate	4	6	1	11	0.455	0	0	0	0	0-0
17	Masiclat, Jada	3	1	2	12	-0.083	0	1	0	2	0-2
13	Durant, Delaney	2	0	0	3	0.000	1	0	0	0	0-0
5	Funk, Dorian	1	0	0	0	0.000	1	0	0	0	0-0
6	Hart, McKenzie	4	0	4	6	-0.667	1	0	2	3	2-3
4	Quick, Olivia	4	5	6	21	-0.048	3	0	0	0	0-0
2	Kelly, Kaylee	4	14	3	33	0.333	0	0	1	1	1-1

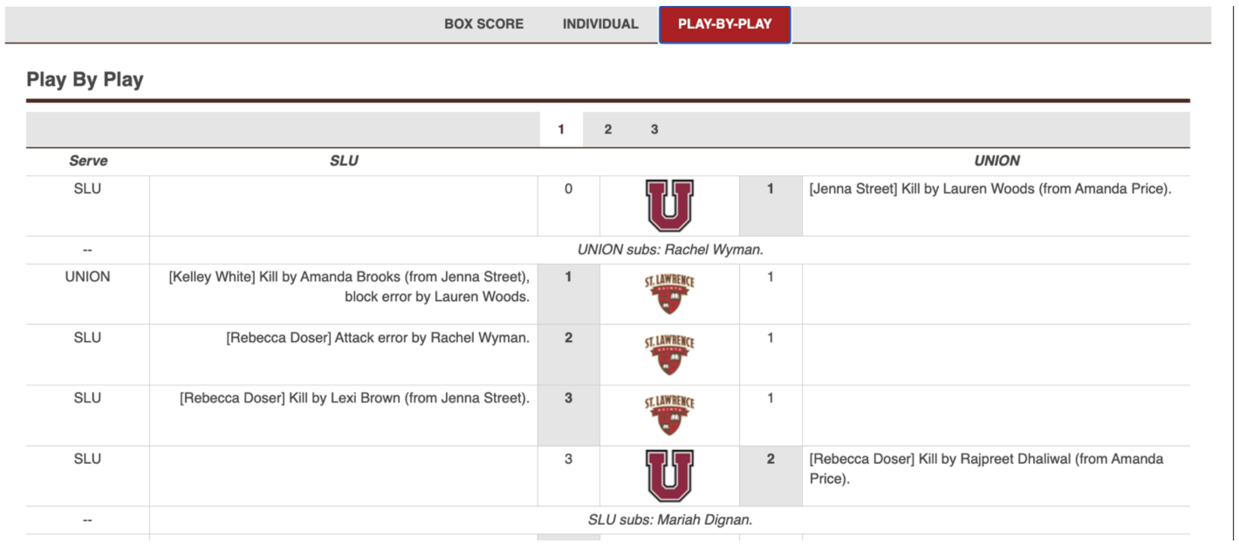


bs (double)	ba (double)	be (double)	dig (double)	bhe (double)	re (double)	pts (double)	team (character)
0	0	0	5	0	2	4.0	home
0	0	0	10	0	1	7.0	home
0	1	0	11	0	0	11.5	home
0	0	0	11	0	3	4.0	home
0	0	0	0	0	0	3.0	home
0	0	0	11	3	2	3.0	home
1	0	0	2	0	0	7.0	home
0	1	1	1	0	0	1.5	home
0	0	0	0	0	0	0.0	home
0	0	0	2	0	0	0.0	home
0	0	0	18	0	3	2.0	away
0	1	0	2	0	1	5.5	away
0	0	0	11	0	0	15.0	away

Figure 7. Individual Statistics Final Form with the First 12 Variables and then next 7 (match information column unpictured)

3.4 Play-by-Play Data

Next, `clean_playbyplay()` cleans the play by play data. Because volleyball has a point scored every play, this is particularly interesting data because we can see how each point was won/lost throughout a set rather than looking at the set overall or the match overall. The cleaning function for this is more complicated, since some matches have play by play data on 3, 4, or 5 sets, and each is in its own tab online. The online format of these data can be viewed in Figure 8. This function creates headers for the data, and a base format for the first three sets put together, since there must be at least three sets. Then, based on the length of the tables, we pull data on the fourth and fifth sets if necessary, and append those to the initial 3. The function makes sure all sets are bound together and attaches the match information onto the final table. The saved version of the play by play, from save_playpbyplay(), looks as pictured in Figure 9 (with identifying variables attached but unpictured), and has 232,260 rows on 12 variables, which can be seen in Table 4.



BOX SCORE INDIVIDUAL PLAY-BY-PLAY

Play By Play

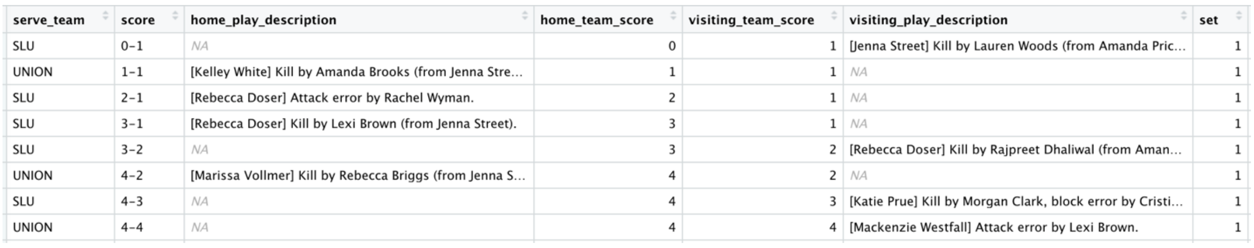
1 2 3

Serve	SLU				UNION
SLU		0		1	[Jenna Street] Kill by Lauren Woods (from Amanda Price).
--	UNION subs: Rachel Wyman.				
UNION	[Kelley White] Kill by Amanda Brooks (from Jenna Street), block error by Lauren Woods.	1		1	
SLU	[Rebecca Doser] Attack error by Rachel Wyman.	2		1	
SLU	[Rebecca Doser] Kill by Lexi Brown (from Jenna Street).	3		1	
SLU		3		2	[Rebecca Doser] Kill by Rajpreet Dhaliwal (from Amanda Price).
--	SLU subs: Mariah Dignan.				

Figure 8. Play-By-Play Tab of Match Webpage

Name of Variable	Variable Measure
serve_team	Which team (school abbrev. name) was serving
Score	Current score home-away (Ex. 4-5)
home_play_description	Description of home team play if they caused the point
home_team_score	Home team current points
visiting_team_score	Visiting team current points
visiting_play_description	Description of visiting team play, if they caused the point
set	Set number (int 1-5)

Table 4. Play By Play Data Variables



serve_team	score	home_play_description	home_team_score	visiting_team_score	visiting_play_description	set
SLU	0-1	NA	0	1	[Jenna Street] Kill by Lauren Woods (from Amanda Pric…	1
UNION	1-1	[Kelley White] Kill by Amanda Brooks (from Jenna Stre…	1	1	NA	1
SLU	2-1	[Rebecca Doser] Attack error by Rachel Wyman.	2	1	NA	1
SLU	3-1	[Rebecca Doser] Kill by Lexi Brown (from Jenna Street).	3	1	NA	1
SLU	3-2	NA	3	2	[Rebecca Doser] Kill by Rajpreet Dhaliwal (from Aman…	1
UNION	4-2	[Marissa Vollmer] Kill by Rebecca Briggs (from Jenna S…	4	2	NA	1
SLU	4-3	NA	4	3	[Katie Prue] Kill by Morgan Clark, block error by Cristi…	1
UNION	4-4	NA	4	4	[Mackenzie Westfall] Attack error by Lexi Brown.	1

Figure 9. Play by Play Data

4 DATABASE DESIGN

The R functions store the data into csv files but these data is very versatile and can be organized and seen in different ways, including as a relational database. To show this connection, we create a database from the 2021 Liberty League matches and store it using PostgresSQL. The storage design is depicted in the Entity-Relationship Diagram in Figure 10. An entity in this case is each of the tables, such as “matches,” and all of its attributes are the variables within that entity’s box. The relationships between entities are represented by the arrows to show which attributes are shared. On the left-hand column of each attribute row, a pk means that the variable forms the primary key (which are attributes that can be used to uniquely identify any row), and a fk symbol means that variable is a primary key in another entity (Chen, 1976).

The current state of the database only has one season, because we created it before scraping more seasons, but it could be easily expanded upon. This relational database was designed properly so that more seasons could be added seamlessly by having the “year” variable in the team’s entity, as well as “date” in matches. The match url, which is a super key in this design, is a feature that naturally lends us to having as many seasons as desired, since that url will never be repeated for more than one match.

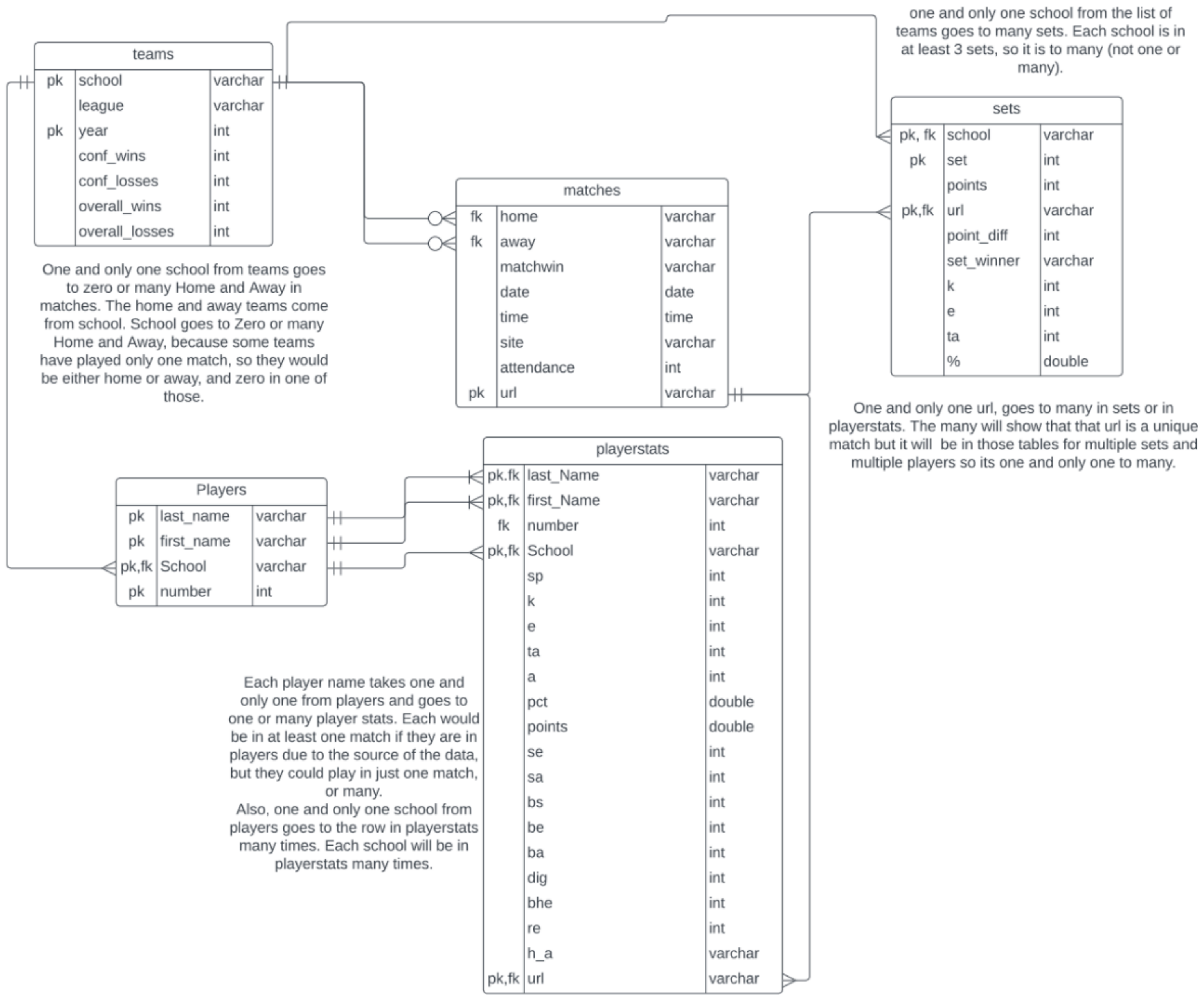

teams
pk | school | varchar
league | varchar
pk | year | int
conf_wins | int
conf_losses | int
overall_wins | int
overall_losses | int
matches
fk | home | varchar
fk | away | varchar
matchwin | varchar
date | date
time | time
site | varchar
attendance | int
pk | url | varchar
sets
pk, fk | school | varchar
pk | set | int
points | int
pk,fk | url | varchar
point_diff | int
set_winner | varchar
k | int
e | int
ta | int
% | double
Players
pk | last_name | varchar
pk | first_name | varchar
pk,fk | School | varchar
pk | number | int
playerstats
pk.fk | last_Name | varchar
pk,fk | first_Name | varchar
fk | number | int
pk,fk | School | varchar
sp | int
k | int
e | int
ta | int
a | int
pct | double
points | double
se | int
sa | int
bs | int
be | int
ba | int
dig | int
bhe | int
re | int
h_a | varchar
pk,fk | url | varchar
one and only one school from the list of teams goes to many sets. Each school is in at least 3 sets, so it is to many (not one or many).
One and only one school from teams goes to zero or many Home and Away in matches. The home and away teams come from school. School goes to Zero or many Home and Away, because some teams have played only one match, so they would be either home or away, and zero in one of those.
One and only one url, goes to many in sets or in playerstats. The many will show that that url is a unique match but it will be in those tables for multiple sets and multiple players so its one and only one to many.
Each player name takes one and only one from players and goes to one or many player stats. Each would be in at least one match if they are in players due to the source of the data, but they could play in just one match, or many.
Also, one and only one school from players goes to the row in playerstats many times. Each school will be in playerstats many times.

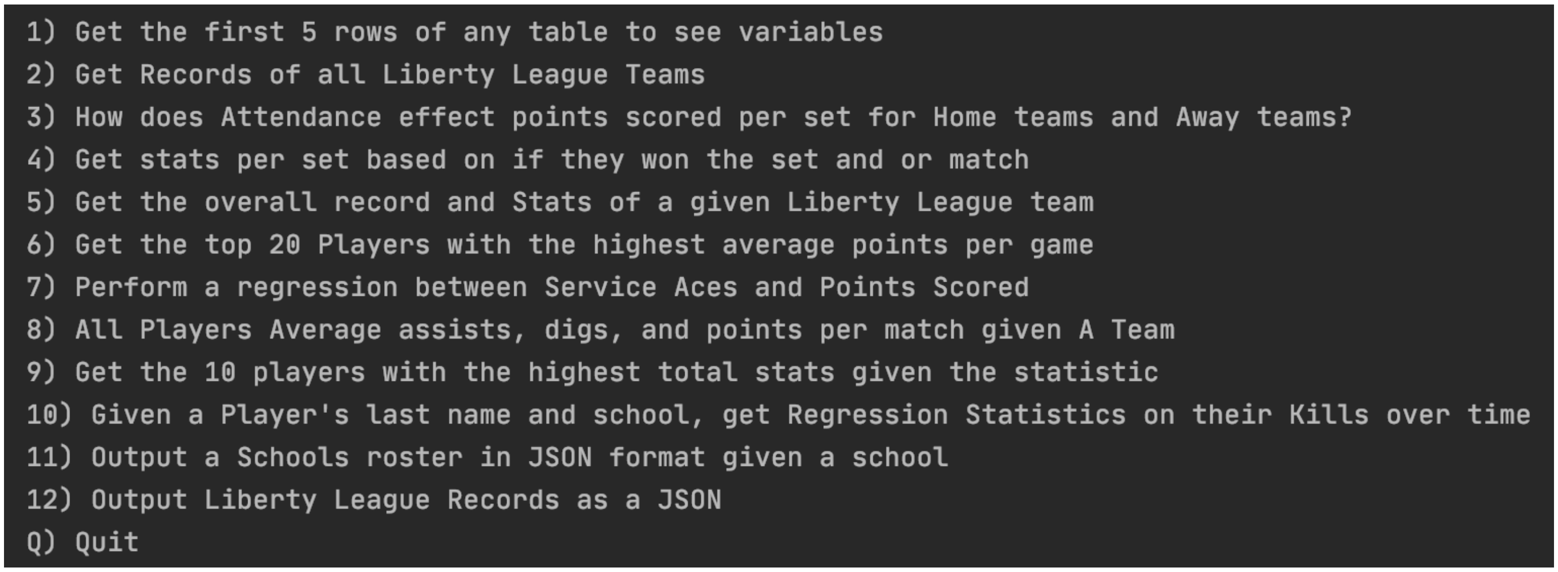
Figure 10. Entity Relationship Diagram Representing the Data

5 ANALYSIS

5.1 SQL Analysis on 2021 season

When creating SQL queries for analysis the main goal was to decipher what a user may want from this database. We create a python interface for these data that allows a user to select from a menu of SQL queries and input some wanted variables to pull information. We want users to be able to pull basic statistics, run regressions, and be able to pull information into JSON format. JSON structure, which is outlined below for roster data, is another way these data can be represented visually and used for easily pulling data. The python prompts we create to run the queries for these goals are in Figure 11. The interface allows users to select queries and input what information they want, and then runs the queries we created to output the requested information back to the user. In the subsequent sections, we illustrate how to use the database to answer several of our posed queries.

```
roster[ {
      "last_name": "Amico",
      "first_name": "Madison",
      "number": 2,
      "school": "St. Lawrence"
     },
     {
      "last_name": "Benson",
      "first_name": "Courtney",
      "number": 14,
      "school": "St. Lawrence"
     },
     {
      "last_name": "Boudreau",
      "first_name": "Rachel",
      "number": 12,
      "school": "St. Lawrence"
     },  …]
```



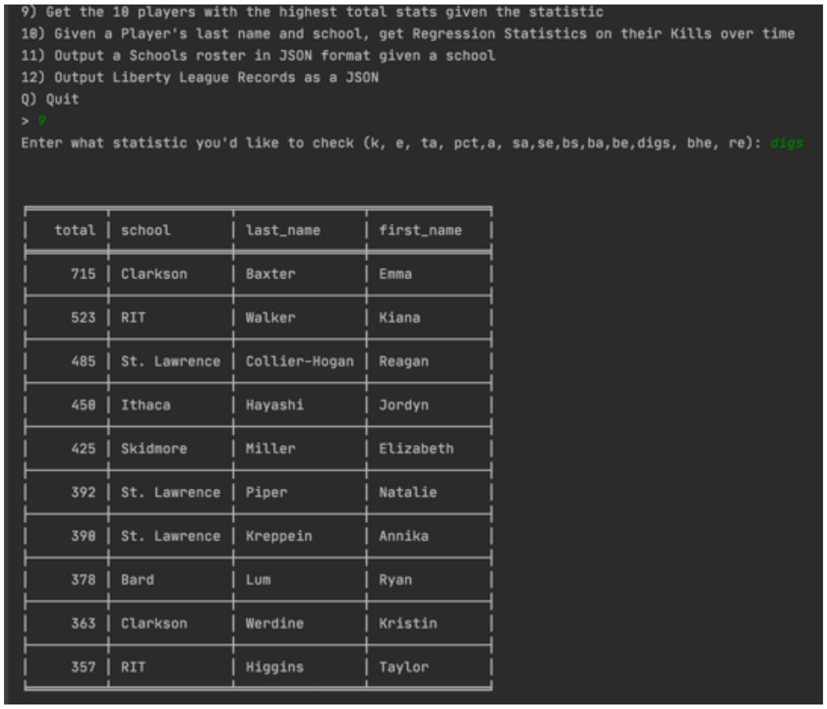
```
1) Get the first 5 rows of any table to see variables
2) Get Records of all Liberty League Teams
3) How does Attendance effect points scored per set for Home teams and Away teams?
4) Get stats per set based on if they won the set and or match
5) Get the overall record and Stats of a given Liberty League team
6) Get the top 20 Players with the highest average points per game
7) Perform a regression between Service Aces and Points Scored
8) All Players Average assists, digs, and points per match given A Team
9) Get the 10 players with the highest total stats given the statistic
10) Given a Player's last name and school, get Regression Statistics on their Kills over time
11) Output a Schools roster in JSON format given a school
12) Output Liberty League Records as a JSON
Q) Quit
```

Figure 11. Python Interface User Prompt

5.1.1 Top 10 Statistics

One of the first queries we create is to pull basic statistics/data on the season. The interface allows the user to enter menu prompt 9 from Figure 11 and enter what metric they wish to view. Then python runs the SQL associated with that prompt to get the top 10 players of the season with that statistic. Python then outputs this information in a table, which is shown in Figure 12 when the user inputs "digs". The query used to pull this information from the database is:

```
SELECT
      sum(dig) as total, school, last_name, first_name
FROM
      playerstats
GROUP BY
      school, last_name, first_name
ORDER BY
      total desc
LIMIT 10;
```



```
9) Get the 10 players with the highest total stats given the statistic
10) Given a Player's last name and school, get Regression Statistics on their Kills over time
11) Output a Schools roster in JSON format given a school
12) Output Liberty League Records as a JSON
Q) Quit
> 9
Enter what statistic you'd like to check (k, e, ta, pct,a, sa,se,bs,ba,be,digs, bhe, re): digs


┌───────┬─────────────┬───────────────┬────────────┐
│ total │ school      │ last_name     │ first_name │
╞═══════╪═════════════╪═══════════════╪════════════╡
│   715 │ Clarkson    │ Baxter        │ Emma       │
├───────┼─────────────┼───────────────┼────────────┤
│   523 │ RIT         │ Walker        │ Kiana      │
├───────┼─────────────┼───────────────┼────────────┤
│   485 │ St. Lawrence│ Collier-Hogan │ Reagan     │
├───────┼─────────────┼───────────────┼────────────┤
│   450 │ Ithaca      │ Hayashi       │ Jordyn     │
├───────┼─────────────┼───────────────┼────────────┤
│   425 │ Skidmore    │ Miller        │ Elizabeth  │
├───────┼─────────────┼───────────────┼────────────┤
│   392 │ St. Lawrence│ Piper         │ Natalie    │
├───────┼─────────────┼───────────────┼────────────┤
│   390 │ St. Lawrence│ Kreppein      │ Annika     │
├───────┼─────────────┼───────────────┼────────────┤
│   378 │ Bard        │ Lum           │ Ryan       │
├───────┼─────────────┼───────────────┼────────────┤
│   363 │ Clarkson    │ Werdine       │ Kristin    │
├───────┼─────────────┼───────────────┼────────────┤
│   357 │ RIT         │ Higgins       │ Taylor     │
└───────┴─────────────┴───────────────┴────────────┘
```

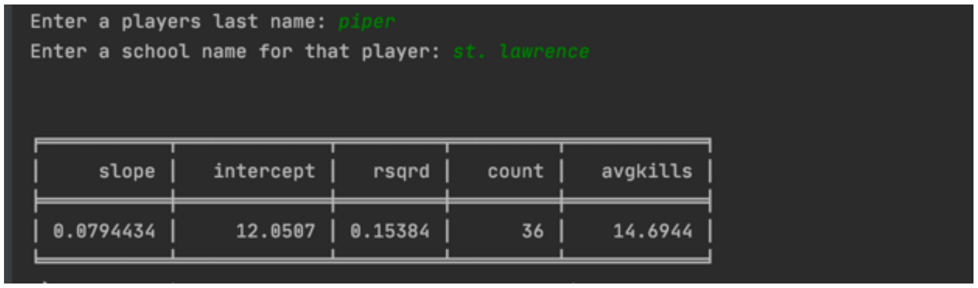
Figure 12. Example Output for top 10 digs

5.1.2 Kills over Time

The next analysis goal is to look at a player's kills over time, and how that can be done as a regression (number 10 in Figure 11). In SQL, you can use REGR_ to run a regression between two variables, and then there are different aspects of a regression you can request in the query. As seen in the query below, we can call for slope, intercept, R^2, count, and average X and Y values. The interface allows for the user to select this regression menu prompt and then input a given Player last name and Team. Since the user can also view the head of any table or JSON format of a roster, which can be used if they did not know a name and team to input. An example output in python is seen in Figure 13, which can then be plotted/visualized in R (Figure 14). The example we use for these outputs is with Natalie Piper, a former St. Lawrence outside hitter. In this case, we see there is a steady increase in Piper's kills as the season goes on, showing a lot of improvement. The query used for this is:

```
WITH
    days as (
        SELECT
              date - '2021-09-01' AS days, m.k as kills, m.last_name
              as last_name, school
        FROM
              matches natural join playerstats as m )
```

```
SELECT
    REGR_SLOPE(kills, days) SLOPE,
    REGR_INTERCEPT(kills, days ) INTCPT,
    REGR_R2(kills, days ) RSQR,
    REGR_COUNT(kills, days) COUNT,
    REGR_AVGY(kills, days ) AVGKills
FROM
    days
WHERE
    lower(last_name) = 'Piper' and lower(school) = 'St. Lawrence';
```



```
Enter a players last name: piper
Enter a school name for that player: st. lawrence


|     slope |   intercept |   rsqrd |   count |   avgkills |
|-----------+-------------+---------+---------+------------|
| 0.0794434 |     12.0507 | 0.15384 |      36 |    14.6944 |
```

Figure 13. Regression output from SQL/Python

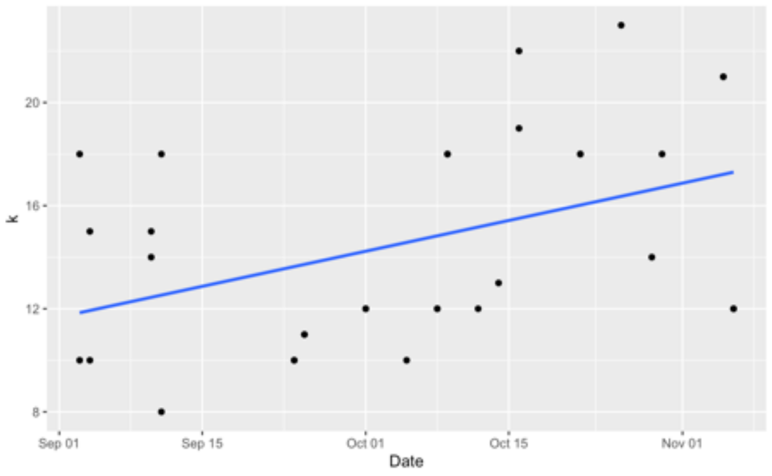


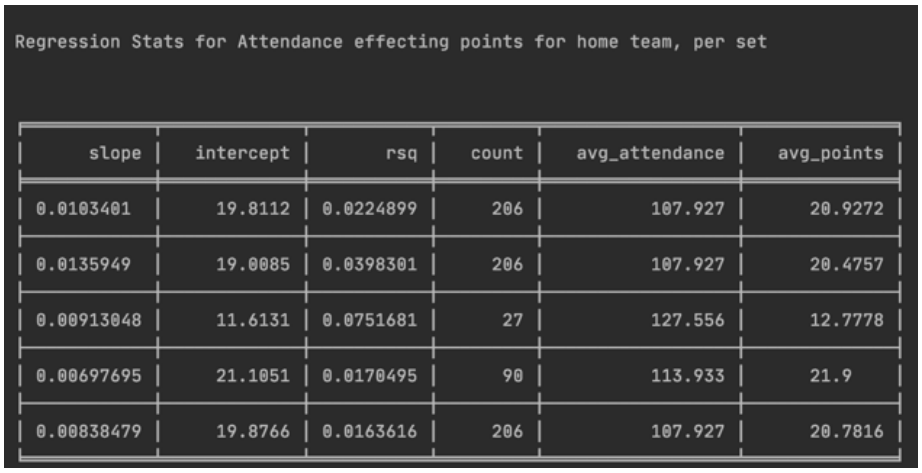
Figure 14. Plot of regression in R

5.1.3 Attendance

The next query from the python interface we use is to assess home team advantage by looking at how attendance of a match effects the points scored for both home and away teams, and if that relationship changes depending on the set of the match (query 3 in Figure 11). This interface prompt allow a user to input either "home" or "away" and runs a regression between attendance and points, for either the home or away team. The output for these is seen below for both home and away teams. In each, the row of the output is each set of the match. The regression runs attendance vs points scored per set within the match, and outputs the regressions into one match table. Figure 15 shows the output for home teams, and Figure 16 shows the output in python for away teams, gotten from the following query:

```
SELECT
    REGR_SLOPE(t.points,attendance) SLOPE,
    REGR_INTERCEPT(t.points,attendance) INTCPT,
    REGR_R2(t.points,attendance) RSQR,
    REGR_COUNT(t.points,attendance) COUNT,
    REGR_AVGX(t.points,attendance) avgattendance,
    REGR_AVGY(t.points,attendance) avgpoints
FROM
      matches natural join sets as t
WHERE
      home = t.school
GROUP BY
      t.set;
```

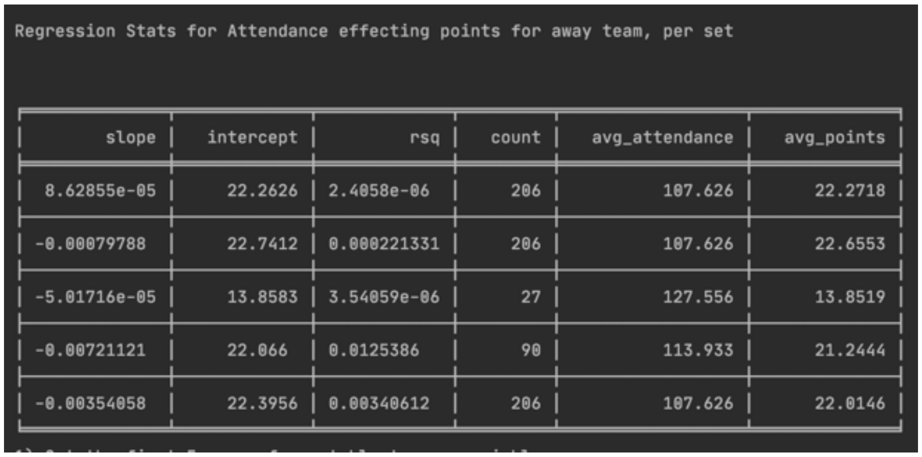
With 'home = t.school' being exchanged for 'away = t.school' for away team analysis.



```
Regression Stats for Attendance effecting points for home team, per set


┌───────────┬───────────┬───────────┬───────┬────────────────┬────────────┐
│     slope │ intercept │       rsq │ count │ avg_attendance │ avg_points │
╞═══════════╪═══════════╪═══════════╪═══════╪════════════════╪════════════╡
│ 0.0103401 │   19.8112 │ 0.0224899 │   206 │        107.927 │    20.9272 │
├───────────┼───────────┼───────────┼───────┼────────────────┼────────────┤
│ 0.0135949 │   19.0085 │ 0.0398301 │   206 │        107.927 │    20.4757 │
├───────────┼───────────┼───────────┼───────┼────────────────┼────────────┤
│ 0.00913048│   11.6131 │ 0.0751681 │    27 │        127.556 │    12.7778 │
├───────────┼───────────┼───────────┼───────┼────────────────┼────────────┤
│ 0.00697695│   21.1051 │ 0.0170495 │    90 │        113.933 │    21.9    │
├───────────┼───────────┼───────────┼───────┼────────────────┼────────────┤
│ 0.00838479│   19.8766 │ 0.0163616 │   206 │        107.927 │    20.7816 │
└───────────┴───────────┴───────────┴───────┴────────────────┴────────────┘
```

Figure 15. Output for home python



```
Regression Stats for Attendance effecting points for away team, per set


┌──────────────┬───────────┬─────────────┬───────┬────────────────┬────────────┐
│        slope │ intercept │         rsq │ count │ avg_attendance │ avg_points │
╞══════════════╪═══════════╪═════════════╪═══════╪════════════════╪════════════╡
│  8.62855e-05 │   22.2626 │ 2.4058e-06  │   206 │        107.626 │    22.2718 │
├──────────────┼───────────┼─────────────┼───────┼────────────────┼────────────┤
│ -0.00079788  │   22.7412 │ 0.000221331 │   206 │        107.626 │    22.6553 │
├──────────────┼───────────┼─────────────┼───────┼────────────────┼────────────┤
│ -5.01716e-05 │   13.8583 │ 3.54059e-06 │    27 │        127.556 │    13.8519 │
├──────────────┼───────────┼─────────────┼───────┼────────────────┼────────────┤
│ -0.00721121  │   22.066  │ 0.0125386   │    90 │        113.933 │    21.2444 │
├──────────────┼───────────┼─────────────┼───────┼────────────────┼────────────┤
│ -0.00354058  │   22.3956 │ 0.00340612  │   206 │        107.626 │    22.0146 │
└──────────────┴───────────┴─────────────┴───────┴────────────────┴────────────┘
```

Figure 16. Output for away python

We do see that increased attendance positively affects points for home teams and negatively affects points for away teams, and that the effect is slightly different for each set. But these are very slight changes, and the slopes are not large. There was also little testing done due to the nature of SQL regressions. The R^2 for these data is the only thing used for evaluation of the regression, and they are all very small, which could be fixed by adding in more predictors or executing the model more accurately outside of SQL. Another problem with this analysis is, as stated earlier, home and away teams could be arbitrarily assigned in terms of a tournament, so the home and away team variables may not always be representative of a home gym. Since attendance is our measure of "home" team here, this arbitrary assignment as a home team in a tournament would not give an advantage. It is also somewhat unclear how attendance was recorded at these games, so this analysis may not be based on an accurate record of how many people were at the game.

SQL limits the way we can analyze data, so it is not the best place for in depth analysis. A better way to do this would be in R with more sophisticated regressions including interactions between attendance and location; however, a different way of analyzing home team advantage would be best. Looking into Location of Games to determine if the home team is really at "home" and not a tournament, would be more accurate. Also, checking win rates may be better than points, or looking in depth at play-by-play data to determine if being at home makes it more likely to score tough points in close sets at the end. Looking specifically at close games and getting a more accurate variable for Home and away teams, as well as doing this analysis outside of SQL would be a better way to assess home team advantage.

5.2 R Analysis on the 2013-2022 Seasons

5.2.1 Cluster Analysis

Moving into R analysis, our first goal is to use hierarchical clustering with individual player's stats, to group players from each team into 'positions' based on what area they had higher stats in. Hierarchical clustering is a type used to group based on similarities and is a good clustering to use since it does not force you to choose a set number of clusters prior to running it. It also allows visual representation in a dendrogram to better pick clusters, which we used in our analysis (Galili, 2021). The complete method of hierarchical clustering which computes the distance of the elements of different clusters that are furthest, rather than an average method which would compute and use average distance (Nielson, 2016). Complete hierarchical clustering is what we use here, which is important to volleyball because we want to create more compact clusters, since players have statistics in so many categories regardless of position, so we need to use complete to be able to separate into positions.

When beginning to cluster, we filter out players that did not play at least 3 sets per match, to ensure we were recording starters and players that would have enough data so we can decipher between positions. We sum each player's statistics over the season, with the variables found in Table 3. We take just the numerical variables, scale them, and use hierarchical clusters in R with `hclust()` and the "complete" method, to get an initial cluster dendogram. We decide to cut this at 7 using `member = cutree(mydata.hclust,7)` , and then re-cluster on subgroups within this.

```
scaled <- scale(just_num)
distance = dist(scaled)
mydata.hclust = hclust(distance)
plot(mydata.hclust)
```

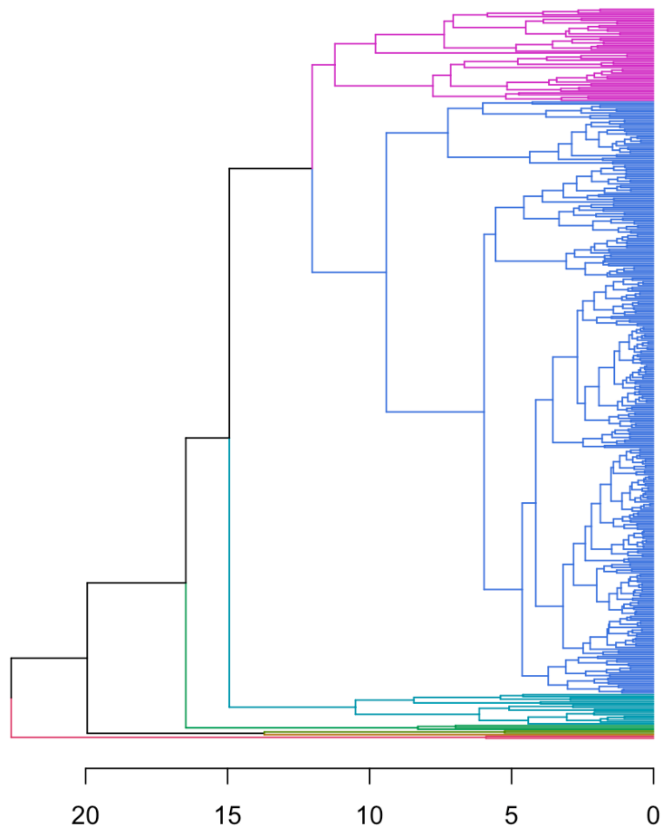


Figure 17. First Cluster Dendogram

From Figure 17, we see there is one large blue cluster in the middle, and a large pink cluster towards the top, which should be broken up further. To do this, we attach member ID numbers to the above cluster data and created subsets with just the two we want to recluster. With the blue group, we rerun the cluster code to scale and then reclustered, outputting Figure 18 as our new dendogram, which we chose to cut into these 7 groups. The pink cluster, which we also chose to split up further into two clusters seen in Figure 19. We then put these 7 and 2 clusters into our original data into the place of the blue and pink clusters respectively.

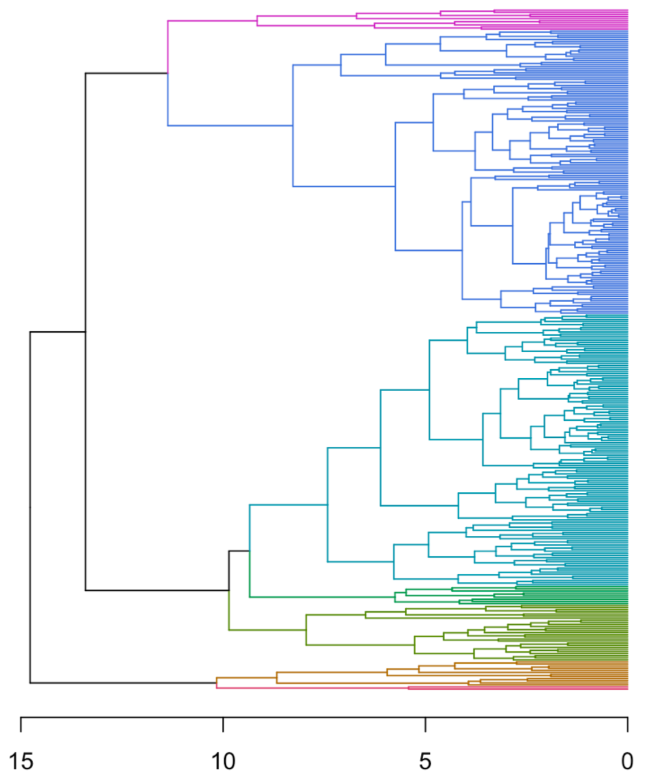


Figure 18. Dendogram from Blue cluster subclustering

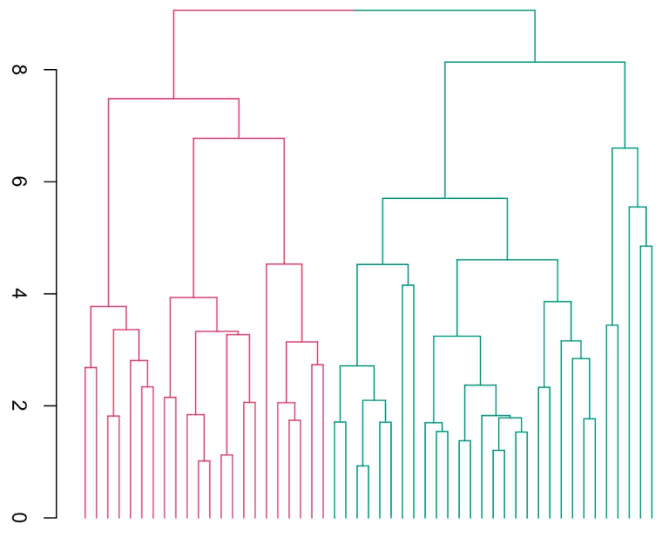


Figure 19. Dendogram from Pink Cluster subclustering

Row/Group	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# of Players	17	3	1	2	2	136	2	130	10	12	27	9	22	29

Table 5. Final Player Allocation to each Cluster

The final group memberships are seen in Table 5, showing our final 14 groups (rows of Figure 19), and how many players are in each. We paste the clusters back together, and then analyze. We take the mean scaled value (z score) from each statistical category and then create a heat map seen in Figure 19. Something to note in the heat map is that we take any statistic with a mean of higher than 3 (which was not many), and set those all equal to 3, to be able to compare colors better and distinguish between low and high statistics in a category. This will help visualize which clusters are higher in which statistics to attempt to identify types of players or positions of players in these clusters. The code use to create the heat map from these data is:

```
heat_9 <- scaledall2 %>%
        group_by(member)%>%
        summarise(across(everything(),mean))%>%
        ungroup()%>%
        select(-member)

heat_9[which(heat_9 > 3, arr.ind = TRUE)] = 3

pheatmap(heat_9,  cluster_rows = FALSE, fontsize = 7)
```

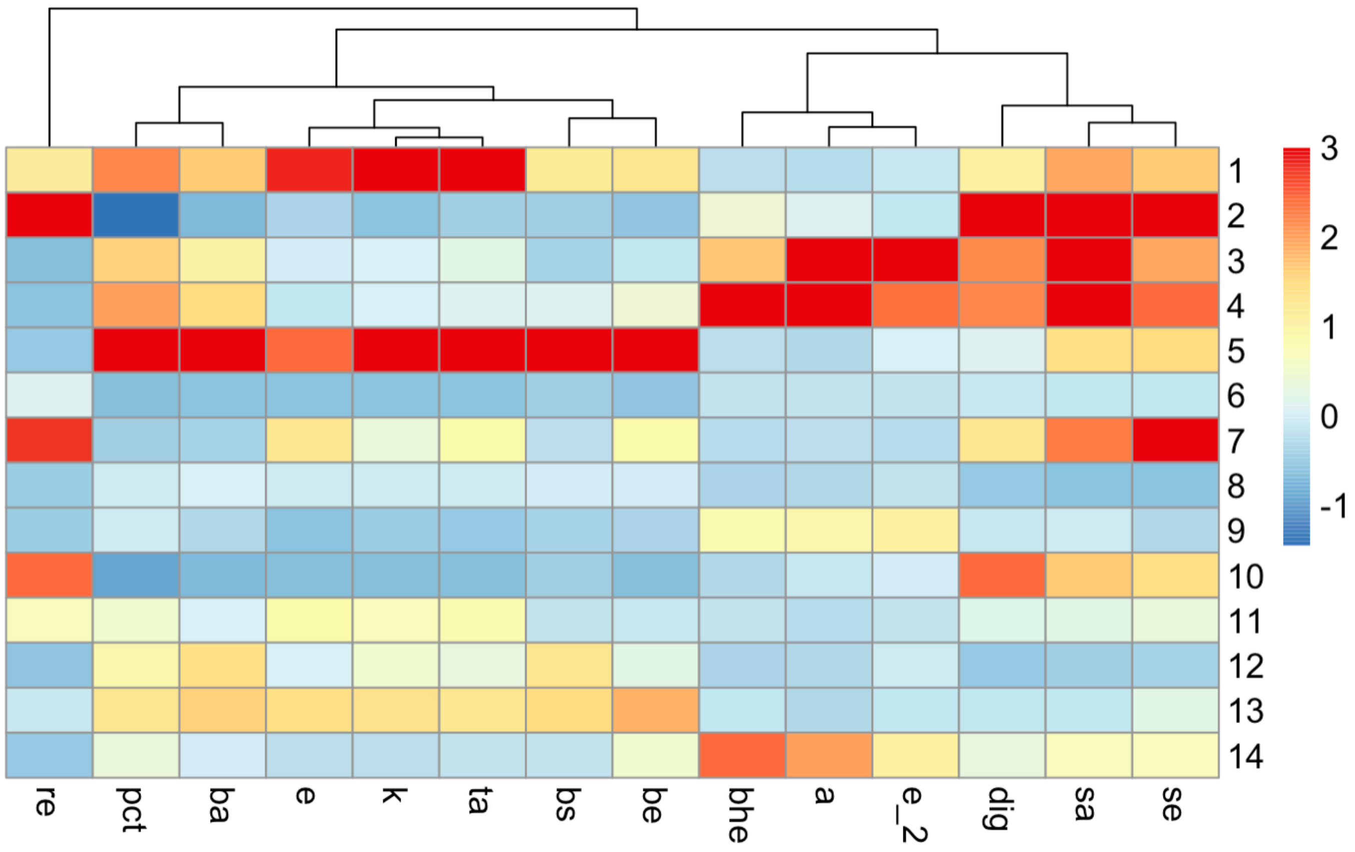


Figure 20. Heat Map showing Cluster Results

From this heat map (Figure 20), we see that some rows have a fairly clear identification, and some are more questionable. Row 1, which we see has 17 individuals, has very high values for Kills, errors and attempts, and high values for multiple blocking and serving statistics and in digs/receptions errors, but has low stats in assists. This appears to represent outside hitters, who usually hit a good amount and play back row, having a chance to defend, but do not usually have to back-up set. Similar statistics can be seen in row 11, but to a lesser extent. These 27 individuals seem to be outside hitters as well, but row 1 may be better/more skilled outside hitters compared to this row.

Rows 12, 13, and 5 seem to have a similar relationship. Row 5 shows very high statistics in kills and blocks, but low in assists and digs, with some average to high serving metrics. Row 13, with 22 players, have middle of the line stats for hitting and blocking with almost no digs or assists. Row 12, which has 9 players, has similarities but to a lesser extent (lower stats in all categories). These players fit the category of middle hitters, because they hit and block a lot and serve some, but usually do not play back row so it makes sense that would have low defensive statistics.

Next, we can identify liberos (defensive specialists). These players have many digs, and usually serve and can occasionally backup assist and back row hit. The rows matching these statistics are 7, which has 2 players 10 with 12 players, and 2 with 3 players. These have high concentrations of digs and reception errors, which is positive and negative defense statistics.

Looking at rows 3, 4 and 14, they only have 1, 2, and 29 players, and seem to fit the description of setters. They have high stats in assists and errors, and have good serving statistics, with some blocks. Setters generally pass some but not a lot, so the moderate dig level makes sense, but the highest number of sets makes sense for setters who have the majority of assists.

Rows 6 and 8, which contain 136 and 130 individuals respectively, appear to have no high valued statistics. These rows are most likely players who do not play often in matches, so they do not have high statistics, or it could be players who are so well rounded that each category has lower numbers, as they are spread out. These players most likely do not have distinctive enough stats to be clustered effectively. Alternatively, these players could be right side hitters. We don't have any of these types of players in our cluster, which could be due to these players being harder to identify as they hit, block, pass, and are backup setters (if the setter passes the first ball). This could be causing them to get clustered into other groups, or have the well-rounded stats discussed above, causing them to blend in.

5.2.2 Win Rates throughout a Set based on Point Gap

The next thing we investigate in these data is how we can use the play-by-play data to look over the course of a set and see how teams pull away to win. We investigate at what point in a set is a team far enough ahead that they can call the set. To do this, we gather the play-by-play data, and add a new variable to keep track of the point gap. For the purpose of analysis, we use the home team as "Team 1" and had the point gap be (Team 1 points – Team 2 points), so if Team 1 was winning, this point gap measure would be positive, and vise-versa for losing/negative. We then plot this data as a set goes on, so as Team 1 gets closer to 25, using a variable "Points until 25". With this, our X axis starts at 25, the start of the game, and ends at 0 (when there are 0 points left until 25). We disregard sets that go past 25 points. We plot the point gap variable on the y axis. Then, for each point gap and point in the game, we calculate the Win Rate as the percentage of times Team 1 won in that situation. We filter out any point-gap and point pair that did not have at least 3 occurrences. This plot can be seen in Figure 21.

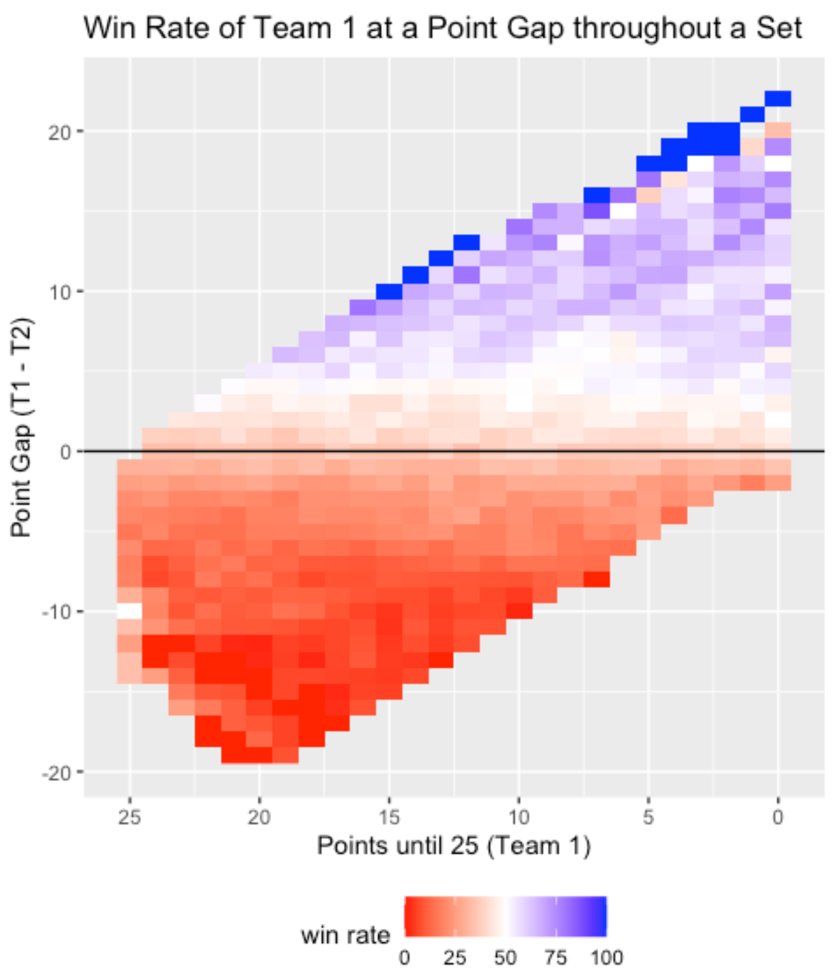


Figure 21. Win Rate of Team 1 at a point gap throughout a set

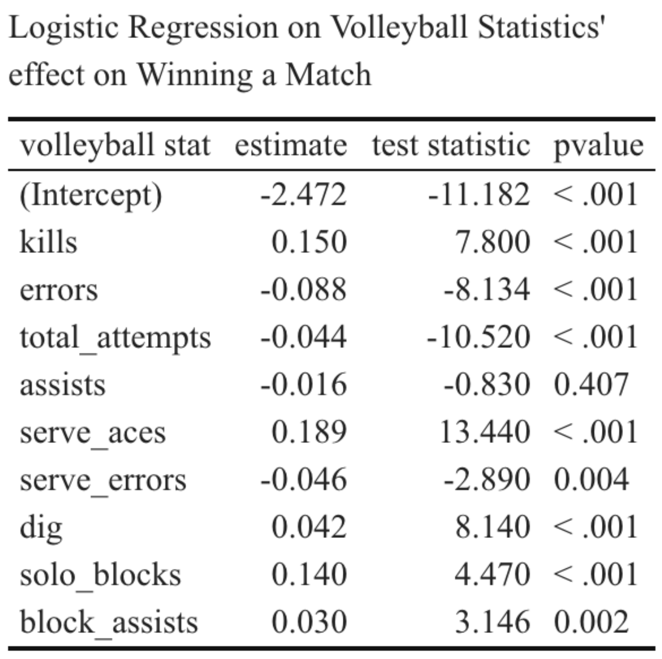
The first thing we see from Figure 21 is that in order to call a game (as a win or loss), meaning the win rate is blue/purple or dark red, the point gap has to be about 10 points. We can see this triangle form as the point gap reached 10 and goes beyond (or below -10), with the rectangle of squares in the -10 o 10 range being white/lighter colored. This is interesting as we see that the win rate of teams remains fairly consistent throughout the set up until the last couple points. It's also interesting to note that many of these games may have happened the minimum n=3 times, and we also are unsure how forfeits are recorded in the play-by-play data, so some of these win rates may be misleading or specific block may not tell a full story. We also see a line of white where Team 1 is up by 1 and a slightly less than 50% chance of winning when the point gap is a 0. This is notable because it appears that Team 1 being up by 1 has no advantage over Team 2 until very late in the game. This is something we could look at further in the future from a home team advantage standpoint. If we were to identify when Team 1 was actually at home rather than a neutral/tournament location, we could look further into this and see why win rate is not slightly above 50%.

5.2.3 Statistical Metrics effect on Winning a Match

The next analysis we perform is a logistic regression on common volleyball metrics to see what contributes the most to winning a match. For this we sum the stats per team and create a binary variable for whether or not that team won the match (best of 5 sets). The model we create was from this code and the summary of the model can be seen in Figure 22.

```
model <- glm(matchwon ~ kills + errors + total_attempts + assists +
serve_aces + serve_errors +dig + solo_blocks + block_assists, family =
binomial, data = statbox)

summary(model)
```



Logistic Regression on Volleyball Statistics' effect on Winning a Match

volleyball stat	estimate	test statistic	pvalue
(Intercept)	-2.472	-11.182	< .001
kills	0.150	7.800	< .001
errors	-0.088	-8.134	< .001
total_attempts	-0.044	-10.520	< .001
assists	-0.016	-0.830	0.407
serve_aces	0.189	13.440	< .001
serve_errors	-0.046	-2.890	0.004
dig	0.042	8.140	< .001
solo_blocks	0.140	4.470	< .001
block_assists	0.030	3.146	0.002

Figure 22. Logistic Regression Output

The regression output shows some expected values such as the positive coefficients for kills and solo blocks, as well as slightly lower but still positive coefficients for assisted blocks, and digs.

We also note that errors and total attempts had expected negative coefficients, meaning they would make it less likely of winning a match. One thing that is slightly unexpected is assists being negative and non-significant in contributing to wins; however, it is most likely due to assists being correlated with kills, causing some multicollinearity. The biggest take away from this regression is the coefficients on Serve Aces and Serve Errors. Serve Aces are the largest contributing factor to increasing a chance that the team wins the match, and serve errors are negative, but barely, meaning they do not cause as much of an impact as we would think. Volleyball has an ongoing debate for whether teams should go for aggressive serves and try for an ace, vs. just getting the serve over and playing good defense and offense after that. This regression would suggest that it is more beneficial to go for an Ace since they increase the likelihood of wining so much more than errors decrease that chance.

Future analysis could include looking into serving in more depth. We would have liked to see if the impact of a serve differs throughout a set, and whether the previously discussed strategy on serving would differ whether it was early in a set/match rather than later in a close game.

6 CONCLUSION

This project collects Liberty League Volleyball Data on matches from the 2013 through 2022 seasons. We clean the data into boxscores, per set statistics, individual player statistics, and play-by-play data, and store them in csv files as well as show how it can be a relational database. This puts the data in a more accessible form for pulling information and analyzing it. Since the NCAA has standardized stats pages, this also provides a blueprint for how we can pull different leagues data, or other divisions of volleyball and/or different NCAA sports. This makes the data more readily available for instructors and analysts and allows for replication.

Future work with these data could include updating the database with the earlier seasons we collected or scraping new data. In terms of further analysis on these data, there are many routes that could be taken, including looking into serving data, home team advantage, team compositions, or momentum throughout a set. These data set, and functions for scraping could also be put into an R package or library for universal access.

REFERENCES

Chen, Peter (March 1976). "The Entity-Relationship Model - Toward a Unified View of Data". *ACM Transactions on Database Systems*. **1** (1): 9–36. CiteSeerX 10.1.1.523.6679. doi:10.1145/320434.320440. S2CID 52801746

Galili, Tal; Benjamini, Yoav; Simpson, Gavin; Jefferis, Gregory (2021-10-28), dendextend: Extending 'dendrogram' Functionality in R, retrieved 2022-06-07

Morgulev, E., Azar, O.H. & Lidor, R. Sports analytics and the big-data era. *Int J Data Sci Anal* **5**, 213–222 (2018). https://doi.org/10.1007/s41060-017-0093-7

Nielsen, Frank (2016). "8. Hierarchical Clustering". Introduction to HPC with MPI for Data Science. *Springer. pp. 195–211. ISBN 978-3-319-21903-5.*

APPENDIX

The git hub repository at https://github.com/cjrich19/SYEvolleyball, contains all of the code files used for this project. This includes the R files for scraping and analysis, the data csv files, as well as the python interface and SQL queries used in that.