The goal of this project was to learn about Elo Ranking systems, Monte Carlo simulations, and to improve my python coding skills. This project was inspired by and draws heavily on the FiveThirtyEight sports simulations.

There are multiple aspects the code I have written for this project. They are:

1. Data Scraping
2. Elo Rankings and Evaluating Elo Factors
3. Monte Carlo Simulations
4. **Data Scraping**

The first step in the process was to gather the data to base the simulations off. When I first started looking into this idea, I was going to try to keep track of all college volleyball teams. This proved difficult, as there are over 100 Division 1 teams that play 20+ games a season. Due to COVID-19, the Big Ten Conference has decided to only play games within the conference. This means that I would only have to keep track of 14 teams instead of 100+. There is also no great centralized place for college volleyball data. The NCAA website1 is good for glancing at score but was hard for me to parse. The next best option was a different version of the NCAA website2, but that proved difficult to parse as well. After realizing I only needed the Big Ten Data, I used the Big Ten archive3. This was very useful as all the data was formatted in the same way. First, I downloaded each page as an html file. Then I read in this page using beautifulSoup and read the table of games into python. I used this data to create a python list of all the games in this format:

1. [date, [team1,score1,team2,score2], location, court\_bias]

All the variables are self-explanatory except for court\_bias. This was a variable that captured which team was home and which team was away. For example, if Wisconsin Played Minnesota at home on 1-1-21, and won 3-2, the line would look like this:

1. [Jan. 1, 2021, [Wisconsin,3,Minnesota,2], “Madison, WI”, 1]

Since the data on the Big Ten Page included all the games the Big Ten Teams played, I only added the game line to the official list if both the teams were in the Big Ten. I repeated this scraping for the 2016-2019 seasons. Prior to 2016 there is no game-by-game data reported, only final standings. Maryland and Rutgers were also added to the Big Ten in 2014. For simplicity sake, I decided to start my predictions only using data from 2016 onward.

The other data the needed to be scraped was the 2021 season schedule. Each of the Big Ten teams had post their schedule, but I could not find a central location that had all the games. Luckily, I found a picture of a spreadsheet that had all the games on it4. I transferred the screenshot into excel by hand (the insert from picture did not work well for me). Then I read in this excel sheet to python and created a list of the games as described above. Instead of inserting a score, I used a placeholder of a “X”.

1. **Elo rankings and Evaluating Elo Factors**

I then created a function that I could call that would run an Elo calculation on the game data I gave it. The basics of that function looked like this:

1. elodiff = elo1\_pre - elo2\_pre
2. exp = elodiff/400
3. elo1\_prob=1/((10\*\*-exp)+1)
4. elo2\_prob=1/((10\*\*exp)+1)
6. elo1\_post = elo1\_pre+ k\*(s1-elo1\_prob)
7. elo2\_post = elo2\_pre+ k\*(s2-elo2\_prob)

elo1\_pre and elo2\_pre were the Elo scores of each team before the current game. Elo1\_post and elo2\_post are the new Elo scores after the game was ran through the function. K is the Elo factor. This determines how many Elo points a team stands to win or loose in a game. S1 and s2 are the actual outcomes of the game. S1 and S2 are set to 1 for a win, 0.5 for a draw, and 0 for a loss. This is a basic Elo setup, but can be improved slightly by tuning and adding some factors. The three factors I looked at were the home court advantage adjustment, and end of season reset, and the K value.

The home court advantage adjustment is the “edge” that home teams have due to playing at home. By reading through all the games we can see how many games the home team won and how many games the away team won. Of the 563 games played from 2016 to 2019 the home team won 317 of them. This means that in college volleyball, the home team wins 56.3% of the time. This ignores the Elo ranking of each of the teams. Assuming that two equally matched teams are playing at a biased site, the Elo equation predicts that each has a 50% chance of winning. This should be closer to the 56.3% change that was discovered. We can then use the Elo Equation to find out that an Elo difference of 44 points results in a 56.29% percent chance of winning. This means that we can use an adjustment of 44 Elo points towards the home team. Due to COVID-19, I am assuming most Big Ten games will have minimal attendance. For this reason, when simulating the 2020 season, I will reduce the home court advantage adjustment to 24. This results in the home team winning 53.4% of the time. Although the crowd is mostly silent while play is happening, there are still numerous other factors that affect playing on the road. This reduction is a guess, and is based off the similar reduction by FiveThirtyEight5.

The end of season adjustment factor (ESAF) is how much of the team’s Elo changes at the end of a season. In its FiveThirtyEight’s NFL simulations5, the NFL team Elo keeps 2/3rd of its score and reverts 1/3rd to the mean. They site the salary cap among other reasons that for this adjustment. In college sports, each team in theory might return to the mean, but there is not an overarching standard or rule to set this into practice. One team might keep 5 starting players whereas another might lose 5 starting players. For simplicity, I assumed that each team would revert to the mean by 25%, meaning it would keep 75% of its Elo score. This is presented in python as:

1. new\_elo = old\_elo\*(1-p) + (800\*(p))

where p is the ESAF. I first ran many simulations of the Elo system to find out which ESAF would be minimize the error. I isolated two sources of error I wanted evaluate. One was the mathematical error, and one was the error in the standings at the end of the year based on Elo.

The mathematical error of the game is Elo probability of winning minus the actual outcome. This error is squared for each team, and then summed. This gives the squared error over the simulation, which could then be divided by the number of games to get a mean squared error.

The error in the standings compared the final standings of the teams in the Elo simulation to the actual final standings. There are 14 teams in the Big Ten, so this error was a number 1-14, higher being better. A score of 14 meant that the Elo system perfectly ranked every team at the end of a season.

When analyzing these errors, as the reset percentage was increased, the mathematical error increased, and the standings error also increased. This meant that the Elo system was better on a seasonal scale, but worse on a per game basis. The results of those simulations with a K factor of 14 are shown here:

1. #ESAF, Standing Correct, Percent Standing Correct, Mathematical Error
2. [[0.0, 25, 0.45, 1.42],
3. [0.05, 26, 0.46, 1.42],
4. [0.1, 27, 0.48, 1.43],
5. [0.15, 28, 0.5, 1.44],
6. [0.2, 28, 0.5, 1.45],
7. [0.25, 29, 0.52, 1.46],
8. [0.3, 30, 0.54, 1.47],
9. [0.35, 30, 0.54, 1.49],
10. [0.4, 33, 0.59, 1.5],
11. [0.45, 33, 0.59, 1.51],
12. [0.5, 33, 0.59, 1.53],
13. [0.55, 33, 0.59, 1.54],
14. [0.6, 35, 0.62, 1.56],
15. [0.65, 34, 0.61, 1.57],
16. [0.7, 35, 0.62, 1.59],
17. [0.75, 35, 0.62, 1.6],
18. [0.8, 37, 0.66, 1.62],
19. [0.85, 39, 0.7, 1.63],
20. [0.9, 42, 0.75, 1.65],
21. [0.95, 44, 0.79, 1.66],
22. [1.0, 47, 0.84, 1.68]]

The first value is the ESAF in a percentage, the second value is the number of placements the Elo system got correct in 4 years, the third value is the number of placements the Elo system got correct in 4 years as a percent (56 total placements), and the fourth value is the total mathematical error the simulation. An ESAF of 1 means that all teams were set to the average Elo score at the beginning of the season. An ESAF of 0 means that all teams kept there same Elo Score from the previous year. From the results, as ESAF increases, the number of correct placements increased, but so did the mathematical error. For this reason, I assumed that 0.25 would be the best value, since it provided minimal per game error, and made intuitional sense as each team would lose about a quarter of their players to graduation. In the simulation above, the system only got 52% of the standings correct. If you let the system be off by one place in either direction (ex. A team finished 3rd but Elo predicted they would finish a close 4th) this percent increased to 80%, which seemed good enough for me. At the end of the day, minimizing the mathematical error would be better than getting rankings perfectly correct, as fluke games can happen, as well as teams being under or overrated. Another reason for this decision is that when an ESAF of 1 is chosen, the modeled system has zero predictive power at the beginning of the season. I figured it was better to sacrifice some end of season performance to allow for some predictive power earlier in the year. This is balanced out by the fact that Elo Ranks were designed to constantly update, and that sensitivity to change can be varied using the K factor.

Given a home court adjustment (44) and an end of season adjustment (0.25), I could then figure out a proper K value. I ran the system through all the data I had with different K values ranging from 2 to 50, and then tracked the number of correct standings as well as the mathematical error, as above. The results are:

1. #K Factor, Percent Placement Correct, Mathematical Error
2. [[2, 0.66, 1.47],
3. [3, 0.73, 1.52],
4. [4, 0.71, 1.49],
5. [5, 0.73, 1.45],
6. [6, 0.73, 1.42],
7. [7, 0.73, 1.38],
8. [8, 0.75, 1.36],
9. [9, 0.75, 1.34],
10. [10, 0.73, 1.32],
11. [11, 0.73, 1.3],
12. [12, 0.73, 1.29],
13. [13, 0.73, 1.28],
14. [14, 0.73, 1.27],
15. [15, 0.71, 1.26],
16. [16, 0.73, 1.26],
17. [17, 0.73, 1.25],
18. [18, 0.73, 1.24],
19. [19, 0.75, 1.24],
20. [20, 0.75, 1.24],
21. [21, 0.77, 1.23],
22. [22, 0.77, 1.23],
23. [23, 0.77, 1.23],
24. [24, 0.77, 1.23],
25. [25, 0.77, 1.22],
26. [26, 0.8, 1.22],
27. [27, 0.8, 1.22],
28. [28, 0.8, 1.22],
29. [29, 0.8, 1.22],
30. [30, 0.8, 1.22],
31. [31, 0.8, 1.22],
32. [32, 0.8, 1.23],
33. [33, 0.8, 1.23],
34. [34, 0.8, 1.23],
35. [35, 0.8, 1.23],
36. [36, 0.8, 1.23],
37. [37, 0.84, 1.23],
38. [38, 0.84, 1.24],
39. [39, 0.84, 1.24],
40. [40, 0.84, 1.24],
41. [41, 0.84, 1.25],
42. [42, 0.84, 1.25],
43. [43, 0.84, 1.25],
44. [44, 0.86, 1.26],
45. [45, 0.86, 1.26],
46. [46, 0.86, 1.26],
47. [47, 0.86, 1.27],
48. [48, 0.86, 1.27],
49. [49, 0.86, 1.27]]

From this, I noticed that the lowest mathematical error was 1.22 and the highest percentage correct was 86%. I still wanted to lean towards having the lowest mathematical error, so I could choose between 25 through 31. Extending the decimals out in that range, I found that the best K value was 28, as seen below:

1. #K Factor, Percent Placement Correct, Mathematical Error
2. [[25, 0.77, 1.211597],
3. [26, 0.8, 1.223255],
4. [27, 0.8, 1.223514],
5. [28, 0.8, 1.223447],
6. [29, 0.8, 1.223666],
7. [30, 0.8, 1.224185],
8. [31, 0.8, 1.224988]]

Finally, I could use the K value to verify that the trend in the ESAF still held. As seen below, the trend still held.

1. #ESAF, Standing Correct, Percent Standing Correct, Mathematical Error
2. [[0.0, 46, 0.82, 1.3],
3. [0.05, 46, 0.82, 1.31],
4. [0.1, 47, 0.84, 1.33],
5. [0.15, 47, 0.84, 1.34],
6. [0.2, 47, 0.84, 1.36],
7. [0.25, 47, 0.84, 1.38],
8. [0.3, 49, 0.88, 1.4],
9. [0.35, 49, 0.88, 1.42],
10. [0.4, 49, 0.88, 1.44],
11. [0.45, 50, 0.89, 1.46],
12. [0.5, 50, 0.89, 1.48],
13. [0.55, 49, 0.88, 1.5],
14. [0.6, 49, 0.88, 1.52],
15. [0.65, 50, 0.89, 1.54],
16. [0.7, 50, 0.89, 1.56],
17. [0.75, 50, 0.89, 1.58],
18. [0.8, 50, 0.89, 1.6],
19. [0.85, 52, 0.93, 1.62],
20. [0.9, 53, 0.95, 1.64],
21. [0.95, 53, 0.95, 1.66],
22. [1.0, 54, 0.96, 1.68]]

The percent standing correct increased slightly to 84% as well, which signals to me that I have made decent choices for the factors I have decided to use.

1. **Monte Carlo Simulations**

With the factors decided, I could then simulate the the 2016-2019 seasons, and then predict the 2020 season. The 2020 season will be played in Spring 2021. I will call it the 2020 season so that if there is the normal season in Fall 2021, that I do not have two seasons called 2021. The simulations started in 2016 and ran through the 2019 data using the factors that I have chosen. At the start of the 2016 season, the Elo score of each team was set to 800. After the 2019 Big Ten season, the Rankings were as follows:

1. [('Minnesota', 1032.9554274094298),
2. ('Wisconsin', 1020.1676350482704),
3. ('Nebraska', 1005.4841925665013),
4. ('Penn State', 994.2864711880087),
5. ('Purdue', 910.450647408784),
6. ('Illinois', 870.4751122484208),
7. ('Michigan', 860.8898754322137),
8. ('Ohio State', 694.9659007073808),
9. ('Michigan State', 686.6375155854157),
10. ('Maryland', 674.5880124091993),
11. ('Northwestern', 672.8316863037809),
12. ('Iowa', 650.4651897143746),
13. ('Indiana', 609.0105972586575),
14. ('Rutgers', 516.7917367195635)]

The Elo scores were then reset for and saved in a new dictionary (a tuple is used to show the scores in order) to be used for the following prediction simulations. After the reset, the Elo scores look like this:

1. [('Minnesota', 974.7165705570724),
2. ('Wisconsin', 965.1257262862028),
3. ('Nebraska', 954.1131444248759),
4. ('Penn State', 945.7148533910065),
5. ('Purdue', 882.837985556588),
6. ('Illinois', 852.8563341863156),
7. ('Michigan', 845.6674065741603),
8. ('Ohio State', 721.2244255305357),
9. ('Michigan State', 714.9781366890618),
10. ('Maryland', 705.9410093068996),
11. ('Northwestern', 704.6237647278357),
12. ('Iowa', 687.848892285781),
13. ('Indiana', 656.7579479439931),
14. ('Rutgers', 587.5938025396727)]

This represents the systems ranking of the teams going into the 2020 season.

I can then go through the 2020 season games and run an Elo function with one change. Instead of the Elo function deciding the winner by reading the score, it will generate a random number between 0 and 1. It will then compare this number to the win percentage of the home team. If the random number is below the win percentage, the home team wins, otherwise, the home team loses. For example. If Wisconsin has a win chance of 0.64 and the random number is 0.51, Wisconsin will win. If the random number was 0.84, then Wisconsin will lose. It does not matter which team is compared to the random number, as the win chance of team A is one minus the win chance of team B. If we wanted to use the away team, then we can also change the sign of the comparison. The Elo function will then update the Elo dictionary after each game. The simulation will then run through the whole season doing this. As this simulation runs, we can keep track of how many games each team wins and looses over the whole simulation. We can then divide the wins and loses by the number of simulations to get an average record for each team. As the number of simulations increases, so will the accuracy of the simulations. FiveThirtyEight reports they run 50,000 simulations5. I decided to run 50,000 simulations of the season each week as well. This is because above 50,000 simulations the expected records do not change as much. Here is an example using Wisconsin:

1. #Number of Simulations, Expected Wins, Expected Losses
2. [[1, 18.0, 4.0]
3. [10, 15.1, 6.9]
4. [100, 15.14, 6.86]
5. [1000, 15.427, 6.573]
6. [10000, 15.3603, 6.6397]
7. [100000, 15.3612, 6.6388]]

From this method, I can make predictions for the whole season, as well as game by game predictions for the duration of the 2020 season.

Links:

1. <https://www.ncaa.com/scoreboard/volleyball-women/d1>
2. <http://stats.ncaa.org/contests/scoreboards>
3. <https://bigten.org/sports/2018/6/6/sports-w-volley-archive.aspx>
4. <https://www.thedailygopher.com/2020/12/30/22206287/minnesota-volleyball-gophers-2021-spring-schedule-announced-big-ten>
5. <https://fivethirtyeight.com/methodology/how-our-nfl-predictions-work/>