May 2023

Explorations in Baseball Analytics: Simulations, Predictions, and Evaluations for Games and Players

Katelyn Mongerson
University of Wisconsin-Milwaukee

EXPLORATIONS IN BASEBALL ANALYTICS: SIMULATIONS, PREDICTIONS, AND EVALUATIONS FOR GAMES AND PLAYERS

by

Katelyn Mongerson

A Thesis Submitted in
Partial Fulfillment of the
Requirements for the Degree of

Master of Science
in Mathematics

at
The University of Wisconsin-Milwaukee
May 2023

ABSTRACT

EXPLORATIONS IN BASEBALL ANALYTICS: SIMULATIONS, PREDICTIONS, AND EVALUATIONS FOR GAMES AND PLAYERS

by

Katelyn Mongerson

The University of Wisconsin-Milwaukee, 2023
Under the Supervision of Professor Gabriella Pinter

From statistics being reported in newspapers in the 1840s, to present day, baseball has always been one of the most data-driven sports. We make use of the endless publicly available baseball data to build models in R and Python that answer various baseball-related questions regarding predicting and optimizing run production, evaluating player effectiveness, and forecasting the postseason. To predict and optimize run production, we present three models. The first builds a common tool in baseball analysis called a Run Expectancy Matrix which is used to give a value (in terms of runs) to various in-game decisions. The second uses the batting statistics of a lineup of 9 players to predict the average number of runs per game that this team should score. The third gives the optimal position in the lineup of the 9th batter by calculating the average runs per game for each possible batting order placement and returns the lineup that produces the maximum runs. To evaluate player effectiveness, we built a model which calculates a player's WAR (Wins Above Replacement). To forecast the postseason, we follow an updated version of Bill James' "World Series Prediction System" developed by Rob Mains and present a code that models this system. Each of these models provides crucial data analysis that is useful in improving the performance of not just individual players, but entire teams.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

BIS Baseball Info Solutions

ERA Earned Run Average

GIDP Ground into Double Play

HBP Hit by Pitch

lgwOBA League Average wOBA

MLB Major League Baseball

PA Plate Appearances

SABR Society for American Baseball Research

UBR Ultimate Base Running

UZR Ultimate Zone Rating

WAR Wins Above Replacement

wGDP Weighted Ground into Double Plays

wOBA Weighted On Base Average

wRAA Weighted Runs Above Average

wSB Weighted Stolen Base Runs

ACKNOWLEDGEMENTS

I first want to acknowledge and thank each of my committee members, Dr. Gabriella Pinter, Dr. David Spade, and Dr. Istvan Lauko. I especially want to thank Dr. Pinter for advising me not just on this thesis, but throughout my education.

I also want to thank my family for always encouraging me to work harder and for pushing me to succeed. In particular, my brother Nathan, whose extensive knowledge of sports has been incredibly useful throughout my thesis. I want to thank him for his patience and his kindness whenever I had a sports question.

1 Introduction

1.1 History of Baseball Analytics

Statistics have been an integral part of Baseball since the very start of the game. All the way back in 1845, the first box score, which tallied only batters runs and outs, was reported in the newspaper. In the 1920s, Ferdinand Cole Lane, editor-in-chief of Baseball Magazine, developed one of the first ever Run Expectancy Models [26]. However, baseball analysis as it is today, wasn't developed until 1974 when Bill James, Dick Cramer, and Pete Palmer co-founded the Society for American Baseball Research (SABR)'s Statistical Analysis Committee. This is also where the term "Sabermetrics" originates [22]. Sabermetrics is, as Bill James defined, "the search for objective knowledge about Baseball" [12]. It is, essentially, an "umbrella term" for all data analysis in baseball.

Despite its extensive history, sabermetrics, and those who study it, spent decades being ignored and mocked by those who could most benefit from it, the front office executives of Major League teams. This is because it represents a huge diversion from baseball tradition, particularly when it comes to scouting. A scout, is "an observer sent to watch games and report on the performance of players he sees there," [27]. Sabermetrics challenges the necessity of scouts, who are so deeply ingrained in baseball culture. In fact, in 1979, Bill James, who is nicknamed the "godfather of sabermetrics," built the first version of his "Runs Created" formula, which he used to predict how many runs a team would score in a season. The formula looked like this:

$$RunsCreated = (Hits + Walks) * TotalBases/(AtBats + Walks)$$

His model ended up being much more accurate than anything even the baseball teams themselves could come up with. However, his model was still rejected because it implied that walks and extra base hits were undervalued by baseball teams, as opposed to batting average and stolen bases. At the time, batting average was considered the most accurate measure of a player's offensive skill, despite the fact that it doesn't include walks, hit-by-pitches, or hit

type, each of which is useful information to the team. That all changed in the 2002 season when the Oakland Athletics, nicknamed the "Moneyball A's" found success, despite their low budget, by favoring metrics like on-base percentage and slugging percentage, because of Bill James' work. Suddenly, sabermetricians were being taken seriously. Bill James was even hired by the Boston Red Sox just one year later [14]. From there, the world of baseball analysis and sabermetrics has only continued to grow.

1.2 Baseball Data

Even the collection of baseball data has an extensive history. For many years, baseball data was hard to come by, because baseball teams did not want to release them. The basics were not difficult to find, but for anyone wanting to do any extensive analysis, the data was not available. Bill James wrote in the 1978 edition of his *Baseball Abstract* that a few statistics he'd included had been easy to find but for the majority he had to calculate them himself by meticulously sorting through box scores [14]. However, as sabermetrics became more widespread so did the publication of data. Today, there are numerous sources for baseball data. The most common of which are Baseball-Reference.com, Retrosheet.org, and the Lahman Database [10].

Baseball-Reference.com, launched in 2000, provides baseball statistics for every player and team in major league history. The range of knowledge that can be gained from this website is extensive. One can find answers to simple questions like "what were the scores of yesterday's games?" to extremely specific questions like "which present-day player is most similar in batting ability to Hank Aaron?" (answer: Albert Pujols) [9]. However, Baseball-Reference.com only contains precalculated statistics and for those wanting to do more advanced analysis, the raw data is needed.

For raw data we look to the Lahman Database and Retrosheet.org. The Lahman Database contains pitching, batting, and fielding statistics from 1871 to present day. The only issue with the Lahman Database is that the data is only season-by-season, when we may need

game-by-game, or play-by-play data. That is where Retrosheet.org is more useful.

Retrosheet is a volunteer organization which works to collect play-by-play data from every single game in Major League Baseball history. The data these volunteers have published is truly remarkable. By combing through countless old newspapers and scanning microfilms, the workers have put together game-by-game summaries from all the way back to the beginning of the MLB. The website even has play-by-play data from nearly every game played since 1921 [18].

1.3 Rules of Baseball

A game of baseball is played between two teams. At the start of the game, the home team is fielding and the away team is batting. Each team consists of 9 players, a pitcher, catcher, first baseman, second baseman, third baseman, shortstop, and left, center, and right fielders. The pitcher pitches the ball to the opposing team's batter. If the batter swings and misses, or if they don't swing but the ball is thrown within a hittable range, called the strike zone, this is called a strike. If a batter gets 3 strikes during their plate appearance, this is a strikeout. If the batter doesn't swing, and the ball is thrown outside the strike zone, this is called a ball. If the batter gets 4 balls during their plate appearance, this is a walk, which means the batter can advance to first base. If the batter gets a hit, they run to advance as many bases as possible. Once on base, the runner can run to the next base at any time, assuming that the next base is empty. A runner scores a run if they are able to reach all four bases. The team with the most runs at the end of the game is the winner.

An "out" occurs when one of four events happen. As previously stated, a strikeout is one way to receive an out. A fly out occurs when a fielder catches a batted ball before it hits the ground. A force out is when a fielder (with the ball) reaches a base before the runner. Lastly, a tag out occurs when a runner is tagged with the ball while not on a base. Once three outs occur, the home and away teams switch, so the home team is now batting. After each team has had their turn batting, the game moves to the next inning, and the away

team is batting again. The game ends once 9 innings have been played.

1.4 States of a Half-Inning

The following table describes all 24 possible states of a half-inning. The corresponding codes are simpler, more efficient ways of stating what the current state of the game is, which will be useful for the following section. The first column in the codes is the "outs" column. It states how many outs there are, so if there is a 2 in the first column, that means there are 2 outs. The second column represents 1st base. If there is a 1 in the second column, there is a runner on 1st. If there is a 0 in the second column, there are no runners on 1st base. The same applies to the third and fourth columns, with 2nd and 3rd base, respectively. For example, if there is a runner on 3rd base with 2 outs, the code number would be 2001.

No runners, no outs	: 0000
No runners, 1 out	: 1000
No runners, 2 outs	: 2000
Runner on 1st, no outs	: 0100
Runner on 1st, 1 out	: 1100
Runner on 1st, 2 outs	: 2100
Runner on 1st, 2nd, no outs	: 0110
Runner on 1st, 2nd, 1 out	: 1110
Runner on 1st, 2nd, 2 outs	: 2110
Runner on 1st, 3rd, no outs	: 0101
Runner on 1st, 3rd, 1 out	: 1101
Runner on 1st, 3rd, 2 outs	: 2101
Runner on 1st, 2nd, 3rd, no outs	: 0111
Runner on 1st, 2nd, 3rd, 1 out	: 1111
Runner on 1st, 2nd, 3rd, 2 outs	: 2111
Runner on 2nd, no outs	: 0010
Runner on 2nd, 1 out	: 1010
Runner on 2nd, 2 outs	: 2010
Runner on 2nd, 3rd, no outs	: 0011
Runner on 2nd, 3rd, 1 out	: 1011
Runner on 2nd, 3rd, 2 outs	: 2011
Runner on 3rd, no outs	: 0001
Runner on 3rd, 1 out	: 1001
Runner on 3rd, 2 outs	: 2001

Table 1: States of a Half-Inning and their Corresponding Code Numbers

2 Run Expectancy Matrix

A run expectancy matrix is a very helpful tool when it comes to baseball decision making. The matrix tells us how much better one state is than another by giving the number of runs that the average team would score by the end of the inning given the current state of the game. For example, when there is a runner on first with 1 out, the average team would score .51 runs by the end of the inning. The following matrix is derived from Retrosheet.org's play-by-play data from the 2022 season. The entries are derived by averaging the number of runs scored in the remainder of an inning given the current state regardless of the team.

$$
\begin{array}{c} \\ 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{array}
\begin{array}{c}
\begin{array}{ccc} 0Outs & 1Out & 2Outs \end{array} \\
\left(\begin{array}{ccc}
.48 & .26 & .1 \\
1.23 & .97 & .39 \\
1.1 & .68 & .31 \\
2 & 1.41 & .56 \\
.87 & .51 & .21 \\
1.77 & 1.17 & .51 \\
1.45 & .91 & .44 \\
2.4 & 1.54 & .78
\end{array}\right)
\end{array}
$$

Figure 1: Run Expectancy Matrix

To understand the importance of a run expectancy matrix, consider the situation where we must decide whether a batter should bunt or not. (A bunt is defined as a batted ball that is not swung at, but intentionally met with the bat and tapped slowly within the infield [3].) To determine this, we calculate whether the average team would score more runs with a bunt or without a bunt. Let us assume that an "average" hitter is up to bat and that there are 0 outs and a runner on 1st base. The possible resulting states after a bunt and the probabilities that the team reaches that state (assuming that each runner does not advance more than one base) are:

State	**Probability**	**Expected Runs**
0110	p_1	1.45
1010	p_2	.68
2000	p_3	.1
1100	p_4	.51
1100	p_5	.51

Table 2: Resulting States after a Bunt and their Probabilities

Notice, the last two rows of this table are the same. This is because there are two separate ways of reaching this state. The batter could get out and the runner could stay on first base, or the batter could make it to first base and the runner could get out. By the law of conditional expectations, the expected number of runs scored after the bunt is given by: $1.45p_1 + .68p_2 + .1p_3 + .51p_4 + .51p_5$. If this value is greater than .87, then the team should

bunt. If it is not, the team shouldn't bunt. The values for $p_1, ..., p_5$ can be found using a similar method to the one we used to find the run expectancy matrix.

3 Nine Player Monte Carlo Simulation

A Monte Carlo Simulation is a technique used to analyze past data and use it to predict future outcomes. This is done by repeatedly playing out or "simulating" an uncertain event based on past data. In this case, each simulation is a game of baseball and the predicted outcome is the average number of runs per game that a team of nine players will score. The model first reads in batting data from a given list of 9 players as well as a transition matrix for the states of the game, a runs matrix, and an outs matrix.

The batting data we focus on is the average amount of each of the following events for each player during the 2022 season:

Event Code	**Event**
1	Strikeout
2	Walk
3	Hit by Pitch (HBP)
4	Error
5	Long Single
6	Medium Single
7	Short Single
8	Short Double
9	Long Double
10	Triple
11	Home Run
12	Ground into Double Play (GIDP)
13	Normal Ground Ball
14	Line drive or Infield Fly
15	Long Fly
16	Medium Fly
17	Short Fly

Table 3: Possible Batting Outcomes and their Event Codes

The data we found did not specify between the different types of singles, doubles, fly balls,

and does not give values for Grounds into Double Plays or Normal Ground Balls. Therefore, we work under the following assumptions, and collect the corresponding additional data [34]:

- 30% of singles are long singles, 50% are medium singles, and 20% are short singles.
- 80% of doubles are short doubles and 20% are long doubles.
- 53.8% of out in play are ground balls, 15.3% are infield flies or line drives, and 30.9% are fly balls.
- 50% of ground balls are GIDP and 50% are GOs.
- 20% of all fly balls are long fly balls, 50% are medium fly balls, and 30% are short fly balls.

To build the state transition, runs, and outs, matrices, we make the following assumptions about the outcomes of each event [34] :

- An error advances all base runners a single base.
- A long single advances each base runner two bases.
- A medium single scores a runner from second base but advances a runner on first one base.
- A short single advances all runners by one base.
- A short double advances each base runner two bases.
- A long double scores a runner from first.
- A GIDP is a ground ball double play if there is a runner on first, first and second, first and third, or bases loaded. In other situations, the batter is out and the other runners stay where they are.

- A normal GO is a ground out that results in a force out with a runner on first, first and second, first and third, or bases loaded. We assume that with runners on second and third, the runners stay put; with a runner on third, the runner scores; and a runner on second advances to third.

- A long fly ball advances (if there are fewer than two outs) a runner on second or third base.

- A medium fly ball (if there are fewer than two outs) scores a runner from third.

- A short fly or a line drive or an infield fly does not advance any runners.

The state transition matrix determines what the new state of the game is based on the current state and what batting event occurred. For example, if the current state is 0100 and a short single is hit, then according to the state transition matrix, the new state is 0110. The runs matrix determines how many runs were scored given the current state and the batting event that occurred. If the current state is 0100 and a triple is hit, the matrix determines that one run was scored. The outs matrix works in a similar way in that it determines how many outs occurred based on the current state and what batting event occurred. These matrices are all built based off of the above assumptions.

After the batting statistics and the matrices are read in, the model asks the user how many simulations they would like to run, (a larger number of simulations will produce a more accurate prediction). One simulation is one game so each simulation begins with 0 runs, 0 outs, and 0 runners. The game also starts with the first inning and the first batter in the lineup up to bat. The model generates a random number which determines what event occurred, based on the batter's statistics, and adjusts the inning state, inning runs, and inning outs, accordingly. The same is then repeated for the next batter and so on until there have been 3 outs. Once the third out occurs, we sum all the runs scored in that inning and then move to the next inning. This process repeats until the game is over (9 innings), when it sums the number of runs scored in all 9 innings. This is the number of runs scored

in that game. This whole process is then repeated until the desired number of simulations have been run. Finally, the model returns the average runs per game for this lineup.

3.1 Results

The following are the results of running this program for 100,000 games. The players studied come from one of the most common Brewer's lineups in 2022 [23].

```
How many simulations do you want: 100000
The average number of runs/game for this team: 4.66341
```

Figure 2: Nine Player Simulation Results

Therefore, this team should average about 4.66 runs per game. This is very close to 4.48, the actual number of runs per game for the 2022 Brewers.

4 Add Ninth Batter Simulation

This model works in a very similar way to the previous one. However, in this case, once the desired number of simulations have been run, the lineup position of the 9th player is changed. The 9th player starts at 1st in the lineup, the simulations are run, and then the average number of runs per game is stored in an array. The 9th player is then moved to 2nd in the lineup and this process repeats, and so on until the player has been in all positions. The model returns the index (plus one because Python indices start at 0) of the maximum value in the average runs per game array, which is the position of the player that gives the maximum runs per game, as well as what the maximum value is, which is the average runs scored with the optimal lineup.

4.1 Results

Here we present the results of simulating 100,000 games for each placement of Tyrone Taylor in the previously used lineup. The average runs per game for each placement are all fairly similar, however the optimal placement is 8th in the lineup, as opposed to his actual placement at 9th.

```
How many simulations do you want: 100000
The average runs per game for each placement of Tyrone Taylor are:
[4.66265, 4.65251, 4.67595, 4.65896, 4.66361, 4.67836, 4.67491, 4.68895, 4.68266]
and the maximum runs per game is:
4.68895
and the lineup where this average occurs is when Tyrone Taylor is number 8 in the lineup.
```

Figure 3: Add Ninth Player Simulation Results

5 Player Evaluation

To evaluate players, we present a code that calculates a player's WAR. Currently, WAR (Wins Above Replacement) is the best metric available to "estimate a player's total value relative to a free available player," for non-pitchers [29]. There are many different ways to calculate WAR. For example, FanGraphs.com, a popular baseball statistics website, adjusts WAR based on the player's home park and league. The method we use is a combination of those used in *Mathletics: How Gamblers, Managers, and Fans use Mathematics in Sports* and by FanGraphs.com:

$$WAR = (BattingRuns + BaseRunningRuns + FieldingRuns + PositionalAdjustment + ReplacementRuns)/RunsPerWin$$

To find the player's batting runs we calculate their wRAA as:

$$wRAA = ((wOBA - lgwOBA)/wOBAScale) * PA$$

Here, wOBA Scale normalizes wOBA for the relevant season. To calculate Base Running Runs we have:

$$BaseRunningRuns = UBR + wSB + wGDP$$

For non-catchers we use the batter's Ultimate Zone Rating (UZR) to measure fielding runs. UZR is an advanced defensive metric that uses play-by-play data recorded by Baseball Info Solutions (BIS) to estimate each fielder's defensive contribution in theoretical runs above or below an average fielder at his position in that player's league and year [15].

The positional adjustment ensures that all defensive positions are measured on the same scale. The fielding runs for a player are measured relative to the average for the position. However, it is harder to be average at some positions than it is for others. This is why it is necessary to include the positional adjustment for WAR. Table 4 lists the adjustments for each position.

Position	**Adjustment**
Catcher	+12.5 runs
First Base	-12.5 runs
Second Base	+2.5 runs
Third Base	+2.5 runs
Shortstop	+7.5 runs
Left Field	-7.5 runs
Center Field	+2.5 runs
Right Field	-7.5 runs
Designated Hitter	-17.5 runs

Table 4: Positional Adjustments for WAR Calculation

The full positional adjustment is calculated by:

$$PositionalAdjustment = ((InningsPlayed/9)/162) * specific\ adjustment\ from\ Table\ 4$$

If a player plays multiple positions the positional adjustment is calculated for each and then they're added together.

Replacement level players are players that would cost nothing but the league minimum salary to acquire. They are, essentially, the players that are freely available, like minor league free agents.

We compare players to replacement level instead of to the league average because there is value in being an average player and because Wins Above Average as opposed to Wins Above Replacement would not distinguish between a player who was average for 1 PA and a player who was average for 700 PA [28]. Let us assume that a replacement level player scores 20 less runs in a season than the average player. Thus, the formula for replacement level runs is:

$$ReplacementRuns = 20 * (fraction\ of\ team's\ plate\ appearances) * 9$$

The Runs Per Win value is the average number of runs a team needs to win. The Runs Per Win value is different every season, but it is typically around 9-10 runs. The formula for the Runs Per Win value, developed by Tom Tango, is:

$$RunsPerWin = 9 * (MLBRunsScored/MLBInningsPitched) * 1.5 + 3$$

5.1 Results

We developed a Python code that takes a list of player's necessary statistics and returns each of their WARs. We tested the code against the same list of players as we used for the lineup simulations. We set lgwOBA=.31, wOBAScale=1.259, MLB Runs Scored=20817, and MLB Innings Pitched=43075.1, because these are the values for the 2022 season.

```
Player: Yelich WAR: 2.9133306809063093
Player: Adames WAR: 4.1205539096041495
Player: Tellez WAR: 1.052046840070051
Player: Urias WAR: 2.0920912816185693
Player: Peterson WAR: 1.671548732346441
Player: Mitchell WAR: 0.8580179092289265
Player: Hiura WAR: 0.8951912287298417
Player: Taylor WAR: 1.6856670250320378
```

Figure 4: WAR Calculation Results

6 Bill James' Post Season Predictions

Bill James first created his "World Series Prediction System" in 1972 and published it in *Inside Sports* in 1982 and again in 1984 in the *Bill James Baseball Abstract*. The 1984 system was [17]:

- Give the team with the better record 1 point for each half-game difference in won-lost percentage.
- Give 3 points to the team that scored the most runs.
- Give 14 points to the team that hit fewer doubles.
- Give 12 points to the team that hit more triples.
- Give 10 points to the team that hit more home runs.
- Give 8 points to the team with the lower batting average.
- Give 8 points to the team with fewer errors.
- Give 7 points to the team that had more double plays.
- Give 7 points to the team that allowed more walks.
- Give 19 to the team that threw more shutouts.
- Give 15 points to the team whose ERA was more below the league average.
- Give 12 points to the team with most recent postseason experience. (In case of a tie, give the points to the team that had greater success)
- For intraleague series, give 12 points to the team with the better head-to-head record.

James calculated these weights by looking at data from all previous seasons and determining how often the winning team for each postseason exhibited these characteristics. For example, triples had a weight of 12 points because throughout World Series history, the team that had hit more triples won 12 more times than the team with less triples.

In 2017, BaseballProspectus.com writer, Rob Mains, published an updated version of this system. His new version not only takes into account decades more postseason games, but also separates them into division, championship, and world series games. Mains' system is as follows [17]:

- Division series
  - Give 7 points to the team that scored fewer runs.
  - Give 10 points to the team with fewer doubles.
  - Give 15 points to the team with fewer triples.
  - Give 4 points to the team with the most home runs.
  - Give 12 points to the team whose batters had fewer walks.
  - Give 17 points to the team whose batters had fewer strikeouts.
  - Give 4 points to the team that had a higher on base percentage.
  - Give 4 points to the team that had a lower slugging percentage.
  - Give 8 points to the team that had fewer errors.
  - Give 27 points to the team that pitched more shutouts.
  - Give 8 points to the team whose pitchers had more strikeouts.
  - Give 8 points to the team with the lower ERA.
  - Give 12 points to the team with more recent postseason experience.
- Championship Series

  - Give 12 points to the team that scored more runs.
  - Give 4 points to the team that hit fewer doubles.
  - Give 12 points to the team that hit fewer triples.
  - Give 13 points to the team that hit more home runs.
  - Give 14 points to the team with the higher batting average.
  - Give 10 points to the team with the higher on base percentage.
  - Give 10 points to the team with the higher slugging percentage.
  - Give 9 points to the team that turned fewer double plays.
  - Give 10 points to the team whose pitchers had fewer strikeouts.
  - Give 6 points to the team whose pitchers allowed fewer walks.
  - Give 10 points to the team with the lower ERA.
  - Give 14 points to the team with the better record.

- World Series
  - Give 10 points to the team that scored more runs.
  - Give 4 points to the team that hit more doubles.
  - Give 11 points to the team that hit more triples.
  - Give 3 points to the team that hit fewer home runs.
  - Give 4 points to the team with the higher batting average.
  - Give 4 points to the team with the higher slugging percentage.
  - Give 3 points to the team that committed fewer errors.
  - Give 10 points to team whose pitchers had more strikeouts.
  - Give 16 points to the team whose pitchers allowed more walks.
  - Give 3 points to the team with the worse overall record.
  - Give 11 points to the team with the more recent postseason experience.

6.1 Results

We built a Python code that models this system and tested it against the 2022 postseason. The model reads in three separate files, each with the necessary batting, fielding, and pitching statistics, respectively, for each team in the postseason, and converts each file into a numpy array. Beginning with the division series, the model contains a series of if-else statements, wherein if one element of an array is greater/less than another, then the corresponding team gains the given number of points. New arrays are then built containing only the statistics of the teams that won in the division series, and these are then used for the championship series section of the model. A similar process is followed for the championship series and then for the world series as well. At the beginning of the 2022 postseason the results of the division series were predicted to be:

```
The winner of the Houston Astros vs. the Seattle Mariners series is the Houston Astros
The winner of the New York Yankees vs. the Cleveland Guardians series is the New York Yankees
The winner of the Philadelphia Phillies vs. the Atlanta Braves series is the Philadelphia Phillies
The winner of the San Diego Padres vs. the Los Angeles Dodgers series is the San Diego Padres
```

Figure 5: Predicted Division Series Results

The results of the Championship series were predicted to be:

```
The winner of the  Houston Astros vs. the  New York Yankees  series is the  New York Yankees
The winner of the  Philadelphia Phillies  vs. the  San Diego Padres  series is the  Philadelphia Phillies
```

Figure 6: Predicted Championship Series Results

Lastly, at the beginning of the postseason, the results of the World Series were predicted to be:

```
The winner of the World Series is the Philadelphia Phillies
```

Figure 7: Predicted World Series Results

The actual results were, for the division series, the Astros, Yankees, Phillies, and Padres won. The Astros and the Phillies won in the championship series. The Astros won the World

Series. Thus, the division series predictions were 100% correct and the championship series predictions were 50% correct.

7 Future Work

An interesting way to add to the Nine Player Monte Carlo Simulation and the Add Ninth Batter Simulation would be to adjust the code so that instead of giving the optimal position in the lineup of the ninth player, it optimizes the entire lineup. A possible code for this would be similar to the Add Ninth Batter Simulation except instead of only changing the position of the ninth player, it adjusts all batters so that every possible lineup is tested. That was what the Add Ninth Batter Simulation was originally meant to be. However, the running time alone for the Nine Player Monte Carlo Simulation is about 88.631 seconds. That code is for only 1 possible order of these players. The running time for a code that optimizes the lineup would be about 88.631 seconds multiplied by 9!. In short, optimizing an entire lineup of batters is impractical and unnecessary for this thesis, but would be an exciting project in the future.

As said previously, WAR is the most effective metric at evaluating player performance. However, it is not without it's faults. For example, WAR only measures past performance and cannot be relied upon to predict future performance [30]. When it comes to future performance, there are other, much more reliable, metrics out there. For instance, ZiPS, a computer projection system developed by FanGraphs.com writer Dan Szymborski, predicts the future performance of a player by comparing them to past players [31].

The post season prediction system we used could be improved as well, especially for the championship and world series. The weights we used were based on data from 2017. If we were to adjust these weights to reflect the most recent data, we may find that the predictions are much more accurate.

References

[1] "2022 Major League Baseball Team Statistics." *Basebal-Reference*, Sports Reference, https://www.baseball-reference.com/leagues/majors/2022.shtml.

[2] "2022 Milwaukee Brewers Statistics." *Baseball-Reference*, Sports Reference, https://www.baseball-reference.com/teams/MIL/2022.shtml.

[3] "Bunt (Baseball)." *Wikipedia*, Wikimedia Foundation, 16 July 2022, https://en.wikipedia.org/wiki/Bunt_(baseball).

[4] "Chadwick Files." *SourceForge*, 12 Apr. 2019, https://sourceforge.net/projects/chadwick/files/.

[5] "Christian Yelich Stats, Height, Weight, Position, Rookie Status & More." *Baseball-Reference*, Sports Reference, https://www.baseball-reference.com/players/y/yelicch01.shtml.

[6] Friendly M, Dalzell C, Monkman M, Murphy D (2022). _Lahman: Sean 'Lahman' Baseball Database_. R package version 10.0-1,<https://CRAN.R-project.org/package=Lahman>.

[7] "Garrett Mitchell Stats, Height, Weight, Position, Rookie Status & More." *Baseball-Reference*, Sports Reference, https://www.baseball-reference.com/players/m/mitchga01.shtml.

[8] "Guts!: Fangraphs Baseball." *FanGraphs*, 29 Apr. 2023, https://www.fangraphs.com/guts.aspx?type=cn.

[9] "Henry Aaron." *Baseball-Reference*, Sports Reference, https://www.baseball-reference.com/players/a/aaronha01.shtml#all_ss_other.

[10] "How to Find Raw Data." *Society for American Baseball Research*, https://sabr.org/sabermetrics/data.

[11] "Jace Peterson Stats, Height, Weight, Position, Rookie Status & More." *Baseball-Reference*, Sports Reference, https://www.baseball-reference.com/players/p/peterja01.shtml.

[12] Kelly, Matt. "What Is Sabermetrics? Modern Analytics Impact Nearly Every Part of Today's Game." *MLB.com*, MLB, 27 May 2019, https://www.mlb.com/news/sabermetrics-in-baseball-a-casual-fans-guide.

[13] "Keston Hiura Stats, Height, Weight, Position, Rookie Status & More." *Baseball-Reference*, Sports Reference, https://www.baseball-reference.com/players/h/hiurake01.shtml.

[14] Lewis, Michael. *Moneyball: The Art of Winning an Unfair Game*. W.W. Norton, 2003.

[15] Lichtman, Mitchel. "The FanGraphs UZR Primer." *FanGraphs*, 19 May 2010, https://blogs.fangraphs.com/the-fangraphs-uzr-primer/.

[16] "Luis Urías Stats, Height, Weight, Position, Rookie Status & More." *Baseball-Reference*, Sports Reference, https://www.baseball-reference.com/players/u/uriaslu01.shtml.

[17] Mains, Rob. "Flu-like Symptoms: The Bill James World Series Prediction System, Updated." *Baseball Prospectus*. 5 Oct. 2017, https://www.baseballprospectus.com/news/article/32916/flu-like-symptoms-the-bill-james-world-series-prediction-system-updated/

[18] Marchi, Max, et al. *Analyzing Baseball Data with R*. CRC Press, Taylor & Francis Group, 2019.

[19] "Milwaukee Brewers Leaderboards " 2022 " All Positions " Fielding Statistics: Fangraphs Baseball." *FanGraphs*, https://www.fangraphs.com/leaders.aspx?pos=all&stats=fld&lg=all&qual=0&type=1&season=2022&month=0&season1=2022&ind=0&team=23&rost=0&age=0&filter=&players=0&startdate=&enddate=.

[20] "Milwaukee Brewers Leaderboards " 2022 " Batters " Advanced Statistics: Fangraphs Baseball." *FanGraphs*, 29 Apr. 2023, https://www.fangraphs.com/leaders.aspx?pos=all&stats=bat&lg=all&qual=0&type=1&season=2022&month=0&season1=2022&ind=0&team=23&rost=0&age=0&filter=&players=0&startdate=2022-01-01&enddate=2022-12-31.

[21] "Milwaukee Brewers Leaderboards " 2022 " Batters " Dashboard: Fangraphs Baseball." *FanGraphs*, 29 Apr. 2023, https://www.fangraphs.com/leaders.aspx?pos=all&stats=bat&lg=all&qual=0&type=8&season=2022&month=0&season1=2022&ind=0&team=23&rost=0&age=0&filter=&players=0&startdate=2022-01-01&enddate=2022-12-31.

[22] Mizels J, Erickson B, Chalmers P. Current State of Data and Analytics Research in Baseball. Curr Rev Musculoskelet Med. 2022 Aug;15(4):283-290. doi: 10.1007/s12178-022-09763-6. Epub 2022 Apr 29. PMID: 35486325; PMCID: PMC9276858.

[23] "Most Common Brewers Lineups." *Lineups*, 2023, https://www.lineups.com/mlb/lineups/milwaukee-brewers.

[24] "Play-by-Play Data Files (Event Files)." *Retrosheet*, 11 Dec. 2022, https://www.retrosheet.org/game.htm.

[25] "Rowdy Tellez Stats, Height, Weight, Position, Rookie Status & More." *Baseball-Reference*, Sports Reference, https://www.baseball-reference.com/players/t/tellero01.shtml.

[26] "Sabermetrics: Baseball Analytics and the Science of Winning [Infographic]." *Syracuse University*, Syracuse University, 27 May 2021, https://onlinegrad.syracuse.edu/blog/sabermetrics-baseball-analytics-the-science-of-winning/.

[27] "Scout." *Baseball-Reference*, Sports Reference, 1 Apr. 2023, https://www.baseball-reference.com/bullpen/Scout#:~:text=A%20scout%20is%20an%20observer,team%20may%20want%20to%20acquire.

[28] Slowinski, Piper. "Replacement Level." *FanGraphs*, 26 Feb. 2010, https://library.fangraphs.com/misc/war/replacement-level/.

[29] Slowinski, Piper. "WAR for Position Players." *FanGraphs*, 2 Apr. 2012, https://library.fangraphs.com/war/war-position-players/.

[30] Slowinski, Piper. "War Misconceptions." *FanGraphs*, 23 Mar. 2012, https://library.fangraphs.com/war/limitations-war/.

[31] Szymborski, Dan. "The 2023 ZiPS Projection Season Is Imminent." *FanGraphs*, 18 Nov. 2022, https://blogs.fangraphs.com/the-2023-zips-projection-season-is-imminent/.

[32] "Tyrone Taylor Stats, Height, Weight, Position, Rookie Status & More." *Baseball-Reference*, Sports Reference, https://www.baseball-reference.com/players/t/tayloty01.shtml.

[33] "Willy Adames Stats, Height, Weight, Position, Rookie Status & More." *Baseball-Reference*, Sports Reference, https://www.baseball-reference.com/players/a/adamewi01.shtml.

[34] Winston, Wayne L, et al. *Mathletics: How Gamblers, Managers, and Fans Use Mathematics in Sports, Second Edition*. PRINCETON UNIVERSITY PRESS, 2022.

[35] "World Series and MLB Playoffs." *Baseball-Reference*, Sports Reference, https://www.baseball-reference.com/postseason/.

Appendices

A Run Expectancy Matrix R Code

```
library(tidyverse)
fields <- read_csv("C://Users//katem//Downloads//baseball_R-master//
baseball_R-master//data//fields.csv")
data2022 <- read_csv("download.folder//unzipped//all2022.csv",
                     col_names = pull(fields, Header),
                     na = character())
data2022 %>%
  mutate(RUNS = AWAY_SCORE_CT + HOME_SCORE_CT,
         HALF.INNING = paste(GAME_ID, INN_CT, BAT_HOME_ID),
         RUNS.SCORED =
           (BAT_DEST_ID > 3) + (RUN1_DEST_ID > 3) +
           (RUN2_DEST_ID > 3) + (RUN3_DEST_ID > 3)) ->
  data2022
data2022 %>%
  group_by(HALF.INNING) %>%
  summarize(Outs.Inning = sum(EVENT_OUTS_CT),
            Runs.Inning = sum(RUNS.SCORED),
            Runs.Start = first(RUNS),
            MAX.RUNS = Runs.Inning + Runs.Start) ->
  half_innings
data2022 %>%
  inner_join(half_innings, by = "HALF.INNING") %>%
  mutate(RUNS.ROI = MAX.RUNS - RUNS) ->
  data2022
data2022 %>%
  mutate(BASES = paste(ifelse(BASE1_RUN_ID > '', 1, 0),
                       ifelse(BASE2_RUN_ID > '', 1, 0),
                       ifelse(BASE3_RUN_ID > '', 1, 0), sep = ""),
         STATE = paste(BASES, OUTS_CT)) ->
  data2022
data2022 %>%
  mutate(NRUNNER1 =
           as.numeric(RUN1_DEST_ID == 1 | BAT_DEST_ID == 1),
         NRUNNER2 =
           as.numeric(RUN1_DEST_ID == 2 | RUN2_DEST_ID == 2 |
                        BAT_DEST_ID == 2),
         NRUNNER3 =
           as.numeric(RUN1_DEST_ID == 3 | RUN2_DEST_ID == 3 |
                        RUN3_DEST_ID == 3 | BAT_DEST_ID == 3),
         NOUTS = OUTS_CT + EVENT_OUTS_CT,
         NEW.BASES = paste(NRUNNER1, NRUNNER2,
```

```
                            NRUNNER3, sep = ""),
         NEW.STATE = paste(NEW.BASES, NOUTS)) ->
  data2022
data2022 %>%
  filter((STATE != NEW.STATE) | (RUNS.SCORED > 0)) ->
  data2022
data2022 %>%
  filter(Outs.Inning == 3) -> data2022C
data2022C %>%
  group_by(STATE) %>%
  summarize(Mean = mean(RUNS.ROI)) %>%
  mutate(Outs = substr(STATE, 5, 5)) %>%
  arrange(Outs) -> RUNS
RUNS_out <- matrix(round(RUNS$Mean, 2), 8, 3)
dimnames(RUNS_out)[[2]] <- c("0 outs", "1 out", "2 outs")
dimnames(RUNS_out)[[1]] <- c("000", "001", "010", "011","100", "101", "110", "111")
RUNS_out
```

B Nine Player Monte Carlo Simulation

```
import pandas as pd
import numpy as np
from random import random

probs =
↪  pd.read_csv("C:\\Users\\katem\\Downloads\\Baseball-master\\Baseball-master
\\Chapter-4\\Book1.csv")


probs=probs.drop(probs.loc[:,'Name':'Outs'].columns, axis=1)
arr=probs.to_numpy()

probs_cumsum=np.cumsum(arr,axis=1)



runs =
↪  pd.read_csv("C:\\Users\\katem\\Downloads\\Baseball-master\\Baseball-master
\\Chapter-4\\runs.csv")
outs =
↪  pd.read_csv("C:\\Users\\katem\\Downloads\\Baseball-master\\Baseball-master
\\Chapter-4\\outs.csv")
state_transitions =
↪  pd.read_csv("C:\\Users\\katem\\Downloads\\Baseball-master
\\Baseball-master\\Chapter-4\\state_transitions.csv",dtype=object)
```

```
nsim = input("How many simulations do you want: ")

if (int(nsim) < 10000):
        print(" **** A higher number of simulations will provide a more
        ↪  accurate answer
        **** ")
        print(" **** Don't be afraid I can handle orders of magnitude higher
        ↪  **** ")


gruns = []

for i in range(int(nsim)):
    player= 0
    nruns=[]
    inning_num=0
    while (inning_num <9):
        inning_state = '000'
        inning_outs = 0
        inning_runs = 0
        while (inning_outs <3):
            coin_flip = random()
            if coin_flip < probs_cumsum[player][0]:
                event = 0
            else:
                event = [i for i in range(17) if coin_flip >=
                ↪  probs_cumsum[player][i]
                and coin_flip < probs_cumsum[player][i+1]][0]+1
            inning_runs += runs[inning_state+'-'+str(inning_outs)][event]
            inning_outs += outs[inning_state+'-'+str(inning_outs)][event]
            inning_state = str(state_transitions[inning_state][event])
            player=(player + 1)%9
        inning_num= inning_num + 1

        nruns.append(inning_runs)

    gruns.append(sum(nruns))
print("The average number of runs/game for this team:", np.mean(gruns))
```

C Add Ninth Batter Python Code

```
import pandas as pd
```

```
import numpy as np
from random import random

probs =
↪  pd.read_csv("C:\\Users\\katem\\Downloads\\Baseball-master\\Baseball-master\\
Chapter-4\\Book1.csv")

probs=probs.drop(probs.loc[:,'Name':'Outs'].columns, axis=1)
arr=probs.to_numpy()

probs_cumsum=np.cumsum(arr,axis=1)



runs =
↪  pd.read_csv("C:\\Users\\katem\\Downloads\\Baseball-master\\Baseball-master\\
Chapter-4\\runs.csv")
outs =
↪  pd.read_csv("C:\\Users\\katem\\Downloads\\Baseball-master\\Baseball-master\\
Chapter-4\\outs.csv")
state_transitions =
↪  pd.read_csv("C:\\Users\\katem\\Downloads\\Baseball-master\\
Baseball-master\\Chapter-4\\state_transitions.csv",dtype=object)

nsim = input("How many simulations do you want: ")

if (int(nsim) < 10000):
        print(" **** A higher number of simulations will provide a more
        ↪  accurate answer **** ")
        print(" **** Don't be afraid I can handle orders of magnitude higher
        ↪  **** ")


arr1=np.cumsum(arr,axis=1)
maxruns=[]
ave=[]
for i in range(0, 9):
    gruns=[]
    probs_cumsum=np.concatenate((arr1[:i],[arr1[8]],arr1[i:8]))
    for j in range(int(nsim)):
        player=0
        nruns=[]
        inning_num=0
        while (inning_num<9):
            inning_state='000'
```

```
            inning_outs=0
            inning_runs=0
            while (inning_outs<3):
                coin_flip=random()
                if coin_flip<probs_cumsum[player][0]:
                    event = 0
                else:
                    event = [m for m in range(17) if coin_flip >=
                    probs_cumsum[player][m] and
                    coin_flip < probs_cumsum[player][m+1]][0]+1
                inning_runs +=
                ↪  runs[inning_state+'-'+str(inning_outs)][event]
                inning_outs +=
                ↪  outs[inning_state+'-'+str(inning_outs)][event]
                inning_state = str(state_transitions[inning_state][event])
                player=(player+1)%9
            inning_num = inning_num + 1
            nruns.append(inning_runs)
        gruns.append(sum(nruns))
    ave=np.mean(gruns)
    maxruns.append(ave)

print("The average runs per game for each placement of Tyrone Taylor are:")
print(maxruns)
print("and the maximum runs per game is:")
print(np.max(maxruns))
print("and the lineup where this average occurs is when Tyrone Taylor is
↪  number",
maxruns.index(np.max(maxruns))+1, "in the lineup.")
```

D WAR Calculation

```
import pandas as pd
import numpy as np
from random import random

probs = pd.read_csv("C:\\Users\\katem\\Downloads\\Baseball-master\\
Baseball-master\\wardata.csv")



arr=probs.to_numpy()
```

```
lgWOBA=.31
wOBAScale=1.259
MLBRuns=20817
MLBInnings=43075.1
RPW=9*(MLBRuns/MLBInnings)*1.5+3
for i in range(0,8):
    WAR=[]
    wRAA=0
    baserunning=0
    pos_adjust=0
    wRAA=((arr[i,1]-lgWOBA)/wOBAScale)*arr[i,2]
    baserunning=arr[i,3]+arr[i,4]+arr[i,5]
    UZR=arr[i,6]
    pos_adjust=((arr[i,7])/1458)*-12.5 + ((arr[i,8])/1458)*2.5 +
    ↪  ((arr[i,9])/1458)*2.5 + ((arr[i,10])/1458)*7.5 +
    ↪  ((arr[i,11])/1458)*-7.5 + ((arr[i,12])/1458)*2.5 +
    ↪  ((arr[i,13])/1458)*-7.5 +((arr[i,14])/1458)*-17.5
    RLR=20*arr[i,15]*9
    TRAR=wRAA+baserunning+UZR+pos_adjust+RLR
    WAR=TRAR/RPW
    print("Player:", arr[i,0], "WAR:", WAR)
```

E Postseason Predictions Code

```
import pandas as pd
import numpy as np

probs1 =
↪  pd.read_csv("C:\\Users\\katem\\OneDrive\\Documents\\Baseball\\BattingWSData.csv")
arr1=probs1.to_numpy()
probs2 =
↪  pd.read_csv("C:\\Users\\katem\\OneDrive\\Documents\\Baseball\\FieldingWSData.csv")
arr2=probs2.to_numpy()
probs3 =
↪  pd.read_csv("C:\\Users\\katem\\OneDrive\\Documents\\Baseball\\PitchingWSData.csv")
arr3=probs3.to_numpy()

#Division Series
teama=0
teamb=0
newarr1=[]
newarr2=[]
newarr3=[]
list1=[0,2,4,6]
```

```
for i in list1:
    if arr1[i,7]<arr1[i+1,7]:
        teama=teama+7
    else:
        teamb=teamb+7
    if arr1[i,9]<arr1[i+1,9]:
        teama=teama+10
    else:
        teamb=teamb+10
    if arr1[i,10]<arr1[i+1,10]:
        teama=teama+15
    else:
        teamb=teamb+15
    if arr1[i,11]>arr1[i+1,11]:
        teama=teama+4
    else:
        teamb=teamb+4
    if arr1[i,15]<arr1[i+1,15]:
        teama=teama+12
    else:
        teamb=teamb+12
    if arr1[i,16]<arr1[i+1,16]:
        teama=teama+17
    else:
        teamb=teamb+17
    if arr1[i,18]>arr1[i+1,18]:
        teama=teama+4
    else:
        teamb=teamb+4
    if arr1[i,19]<arr1[i+1,19]:
        teama=teama+4
    else:
        teamb=teamb+4
    if arr2[i,11]<arr2[i+1,11]:
        teama=teama+8
    else:
        teamb=teamb+8
    if arr3[i,12]>arr3[i+1,12]:
        teama=teama+27
    else:
        teamb=teamb+27
    if arr3[i,22]>arr3[i+1,22]:
        teama=teama+8
    else:
        teamb=teamb+8
```

```
    if arr3[i,7]<arr3[i+1,7]:
        teama=teama+8
    else:
        teamb=teamb+8
    if arr2[i,19]>arr2[i+1,19]:
        teama=teama+12
    else:
        teamb=teamb+12
    if teama>teamb:
        print("The winner of the", arr1[i,0], "vs. the", arr1[i+1,0],
        ↪  "series is the", arr1[i,0])
        newarr1.append(arr1[i])
        newarr2.append(arr2[i])
        newarr3.append(arr3[i])
    else:
        print("The winner of the", arr1[i,0], "vs. the", arr1[i+1,0],
        ↪  "series is the", arr1[i+1,0])
        newarr1.append(arr1[i+1])
        newarr2.append(arr2[i+1])
        newarr3.append(arr3[i+1])

#Championship Series
teamc=0
teamd=0
new_arr1=[]
new_arr2=[]
new_arr3=[]
list2=[0,2]
for i in list2:
    if newarr1[i][7]>newarr1[i+1][7]:
        teamc=teamc+12
    else:
        teamd=teamd+12
    if newarr1[i][9]<newarr1[i+1][9]:
        teamc=teamc+4
    else:
        teamd=teamd+4
    if newarr1[i][10]<newarr1[i+1][10]:
        teamc=teamc+12
    else:
        teamd=teamd+12
    if newarr1[i][11]>newarr1[i+1][11]:
        teamc=teamc+13
    else:
        teamd=teamd+13
```

```
if newarr1[i][17]>newarr1[i+1][17]:
    teamc=teamc+14
else:
    teamd=teamd+14
if newarr1[i][18]>newarr1[i+1][18]:
    teamc=teamc+10
else:
    teamd=teamd+10
if newarr1[i][19]>newarr1[i+1][18]:
    teamc=teamc+10
else:
    teamd=teamd+10
if newarr2[i][12]<newarr2[i+1][12]:
    teamc=teamc+9
else:
    teamd=teamd+9
if newarr3[i][22]<newarr3[i+1][22]:
    teamc=teamc+10
else:
    teamd=teamd+10
if newarr3[i][20]<newarr3[i+1][20]:
    teamc=teamc+6
else:
    teamd=teamd+6
if newarr3[i][7]<newarr1[i+1][7]:
    teamc=teamc+10
else:
    teamd=teamd+10
if newarr3[i][4]>newarr3[i+1][4]:
    teamc=teamc+14
else:
    teamd=teamd+14
if teamc>teamd:
    print("The winner of the ", newarr1[i][0], " vs. the ",
    ↪  newarr1[i+1][0], " series is the ", newarr1[i][0])
    new_arr1.append(newarr1[i])
    new_arr2.append(newarr2[i])
    new_arr3.append(newarr3[i])
else:
    print("The winner of the ", newarr1[i][0], "vs. the ",
    ↪  newarr1[i+1][0], " series is the ", newarr1[i+1][0])
    new_arr1.append(newarr1[i+1])
    new_arr2.append(newarr2[i+1])
    new_arr3.append(newarr3[i+1])
```

```
#World Series
teame=0
teamf=0
if new_arr1[0][7]>new_arr1[1][7]:
    teame=teame+10
else:
    teamf=teamf+10
if new_arr1[0][9]>new_arr1[1][9]:
    teame=teame+4
else:
    teamf=teamf+4
if new_arr1[0][10]>new_arr1[1][10]:
    teame=teame+11
else:
    teamf=teamf+11
if new_arr1[0][11]<new_arr1[1][11]:
    teame=teame+3
else:
    teamf=teamf+3
if new_arr1[0][17]>new_arr1[1][17]:
    teame=teame+4
else:
    teamf=teamf+4
if new_arr1[0][19]>new_arr1[1][19]:
    teame=teame+4
else:
    teamf=teamf+4
if new_arr2[0][11]<new_arr2[1][11]:
    teame=teame+3
else:
    teamf=teamf+3
if new_arr3[0][22]>new_arr3[1][22]:
    teame=teame+10
else:
    teamf=teamf+10
if new_arr3[0][20]>new_arr3[1][20]:
    teame=teame+16
else:
    teamf=teamf+16
if new_arr3[0][4]<new_arr3[1][4]:
    teame=teame+3
else:
    teamf=teamf+3
if new_arr2[0][19]>new_arr2[1][19]:
    teame=teame+11
```

```
else:
    teamf=teamf+11
if teame>teamf:
    print("The winner of the World Series is the", new_arr1[0][0])
else:
    print("The winner of the World Series is the", new_arr1[1][0])
```