Analyzing Baseball Data with R

A Senior Project

Presented to

The Faculty of the Statistics Department

California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science

By

Claudia Sison

June 2017

Table of Contents

Introduction

Baseball and statistics go hand-in-hand. In fact, my love for baseball is what got me interested in statistics in the first place. When I took AP Statistics in high school, I would take what I had just learned and try to apply it to some aspect of baseball. I am a Yankees fan so I always get a lot of flak for them "buying their rings." When I first learned about correlation, I wanted to see if the Yankees' payroll had a positive relationship with their wins over the years. Sadly, I did not find any sort of relationship between the two variables. However, this exploration into baseball and statistics sparked a big interest in me to use numbers to defend my love for the most hated team in baseball. After calculating something as simple as a correlation between payroll and wins, my statistical analysis became more advanced. Once I learned about hypothesis testing, I tested to see if the Yankees' winning percentage was significantly better than the league average, or even the average of their rival, the Boston Red Sox. I have never had a subject interest me enough to use it in a nonacademic setting before. After I took Statistics in high school, I knew that that was what I wanted to focus my college and life career on.

Coming into my first year at Cal Poly as a Statistics major, I knew I was going to have to do a senior project. I also knew that I wanted to do that senior project on baseball. However, I could never narrow down a specific question or aspect about baseball that I wanted to explore because so many different features of baseball interest me. For this reason, I decided to do my project on just that. The book Analyzing Baseball Data with R by Max Marchi and Jim Albert explores different ways to analyze baseball data with one of my favorite statistical software packages. Each chapter focuses on a different part of baseball analytics including, but not limited to, graphics, ball and strike effects, and valuing plays. For my Senior Project, I went through each chapter of the book, completed every example problem as well as the end-of-chapter exercises, and applied the concepts I liked most to data I found on my own. The following pages are divided up by chapter and within each of those sections, I detail the most interesting analyses I found from that chapter. I include the code as well as explain what the code is doing so that readers, with or without a background in R, can go through and understand the analysis.

Chapters 1 and 2: The Baseball Datasets and an Introduction to R

Analyzing Baseball Data with R uses 4 main different types of data. Chapter 1 describes the different data the reader will be using and its applications. A brief summary of each of the four types of data is listed below.

1. The Lahman Database: Season-by-Season Data

The Lahman Database was created by journalist and author Sean Lahman who over the years has accumulated a database containing pitching, hitting, and fielding statistics for the entire history of baseball (1871 - Present).

2. Retrosheet Game-by-Game Data

Retrosheet is an organization founded by professor David Smith. It is a collection of accounts of almost every game ever played in the history of Major League Baseball. They have indivudual game data available dating back to 1871 that can answer questions such as **"In which ballparks are homeruns most likely to occur?"** or **"What was the average attendance for a New York Yankees home game in 1996?"**

3. Retrosheet Play-by-Play Data

Retrosheet also has information for play-by-play accounts starting in the 1945 season. This information includes what pitches were thrown, who was on base at the time, what kind of out the play resulted in, etc. This database can answer questions such as **"What is the American League average when the ball/strike count is 3-2?"** or **"Which player is most successful when hitting with men on base?"**

4. Pitch-by-Pitch Data

MLB.com has an application called Major League Baseball Advanced Media (MLBAM) Gameday that has data on every pitch starting from the 2008 season. This data includes information on pitch velocity, location, and ball trajectory. This data is useful for looking at a pitcher's average speed, their average location with respect to the strike zone, etc. It can be used to analyze pitches in regards to not only pitchers, but batters and umpires as well.

After the reader is familiar with the datasets that will be used throughout the book, Chapter 2 details introductory commands and functions that are useful for those who do not have a background in R. The main topics this chapter covers are defining and using vectors, creating matrices, writing simple functions, importing and exporting data, and different packages that are available in R.

Both Chapters 1 and 2 are great because they provide a good foundation for the subsequent chapters. I never knew that these types of data, especially the play-by-play data, were available for anyone to download. However, in order to download it in a cleanly-formatted way, I had to utilize functions in R in order to do so. The entire process is detailed in Appendix A of the book.

Chapter 3: Traditional Graphics

Chapter 3 focuses on using R's built-in graphics capabilities to display different types of baseball statistics. The measurement scale of the variables dictates what type of graph is most appropriate. Not only does R make it easy to produce these graphs by the call of a fairly simple function, but it is also possible to customize the graphs. I will be going through each of the end-of-chapter exercises to show what graphs work best for what types of data.

How many batters have Hall of Fame Pitchers faced?

For ease of visulizing this data, we want to create a new variable that uses intervals to express the number of batters each pitcher has faced rather than the actual number. For example, 10,251 batters faced (BF) will be in the interval (10,000, 15,000).

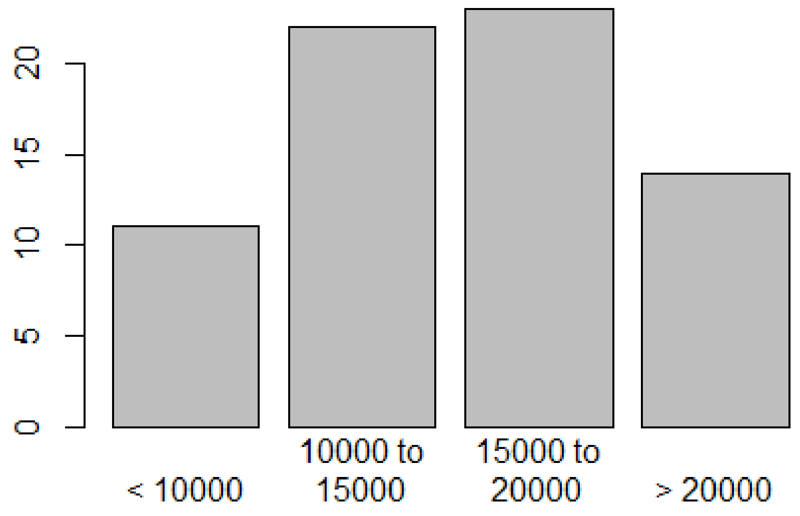
```
hofpitching$BF.group =with(hofpitching, cut(BF, c(0, 10000, 15000, 200
00, 30000), labels=c("< 10000", "10000 to\n15000", "15000 to\n20000",
"> 20000")))
```

Frequency tables can be used to see how many pitchers are in each BF interval.

```
tab.BF.grp =table(hofpitching$BF.group)
tab.BF.grp

##
##          <10000  10000 to 15000  15000 to 20000           > 20000
##              11              22              23                14
```

Similarly, a bar graph is graphic way to view this table.

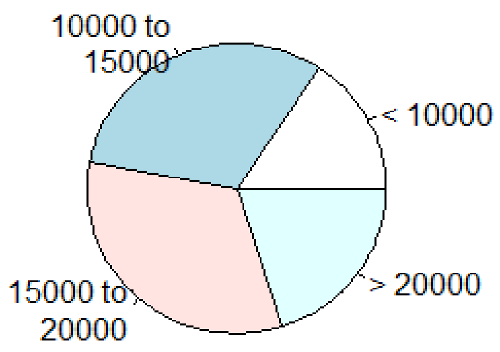
```
barplot(tab.BF.grp)
```



As we can see, **14** Hall of Fame pitchers have faced more than 20,000 batters.

Pie charts are another way to visualize this data, but instead of displaying the number of pitchers in each interval, the relative proportion of total pitchers is displayed.

```r
pie(tab.BF.grp)
```

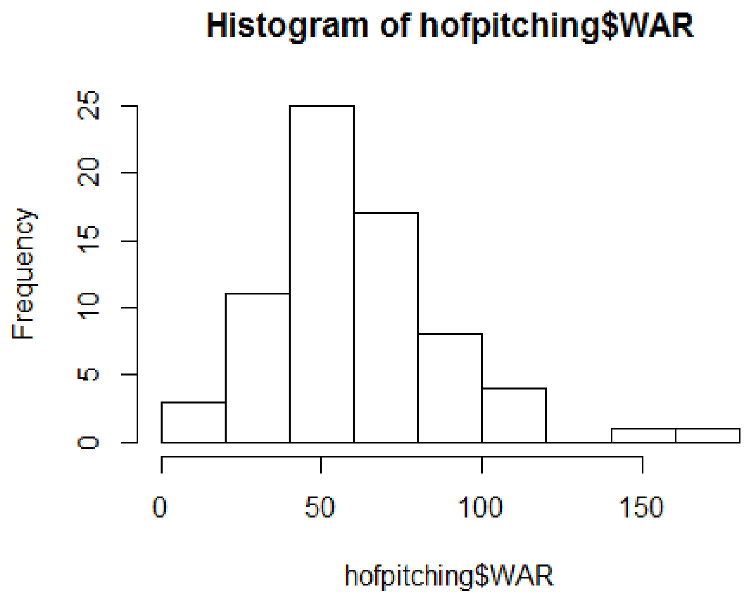


Unless we already know the total number of Hall of Fame pitchers, the pie chart is a little less helpful in comparing the different intervals.

WAR (or Wins Above Replacement) is a useful statistic to gauge how well a pitcher performs. Without going into too much detail, it quantifies how much "better" a player is than the average player that would be available to replace them.

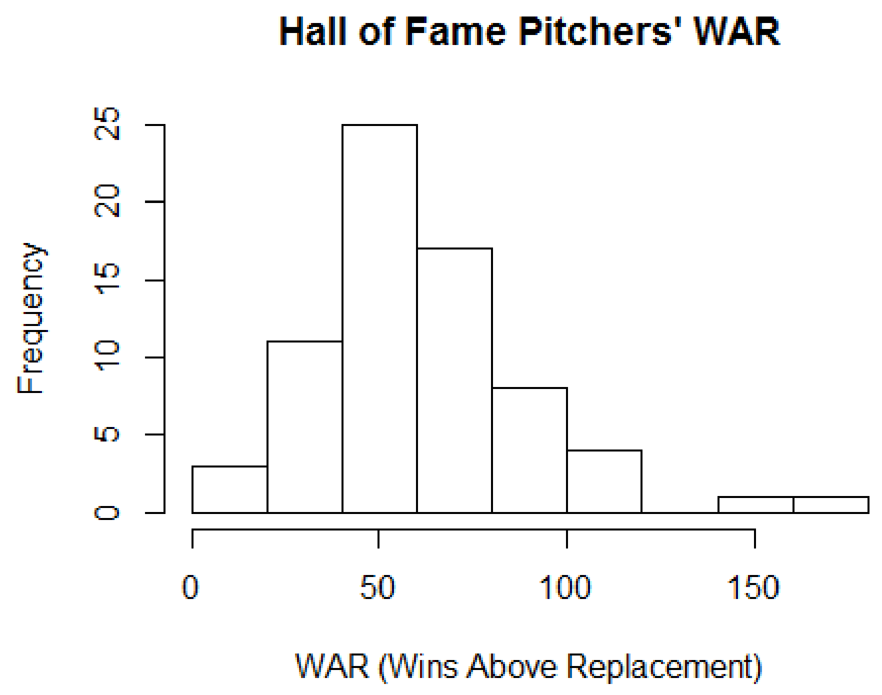
Because WAR is a continuous variable, a histogram works best at displaying the distribution of all Hall of Fame Pithcers' WAR statistic.

```r
hist(hofpitching$WAR)
```



I prefer to have the axes and title labels look more clean so by using the code below, it is possible to customize these labels.

```
hist(hofpitching$WAR, xlab="WAR (Wins Above Replacement)", main="Hall
of Fame Pitchers' WAR")
```

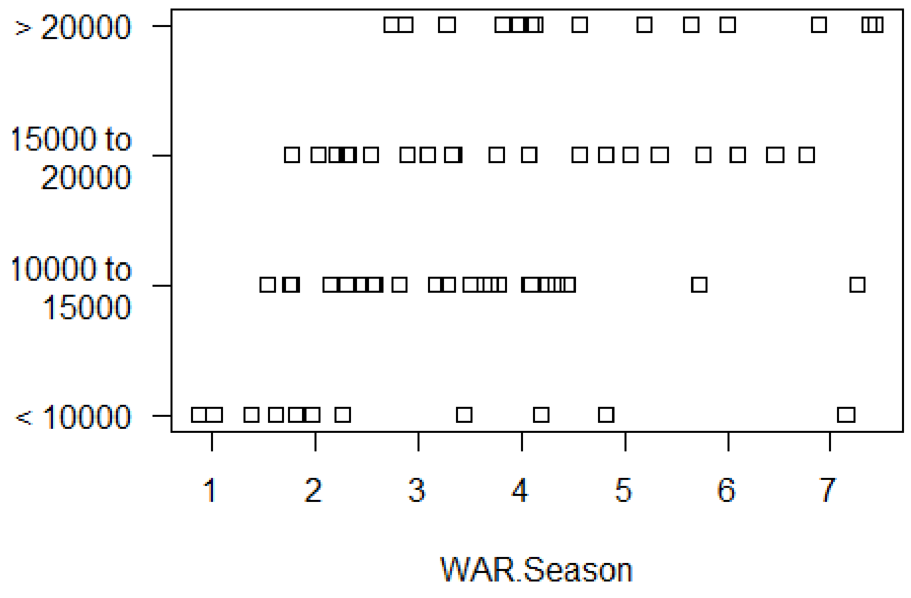


How can we gauge a pitcher's season contribution?

The Hall of Fame Pitching data set provides us with a pitcher's career WAR, but if we want to see their average WAR, we can divide their **WAR** with the number of **years** they played.

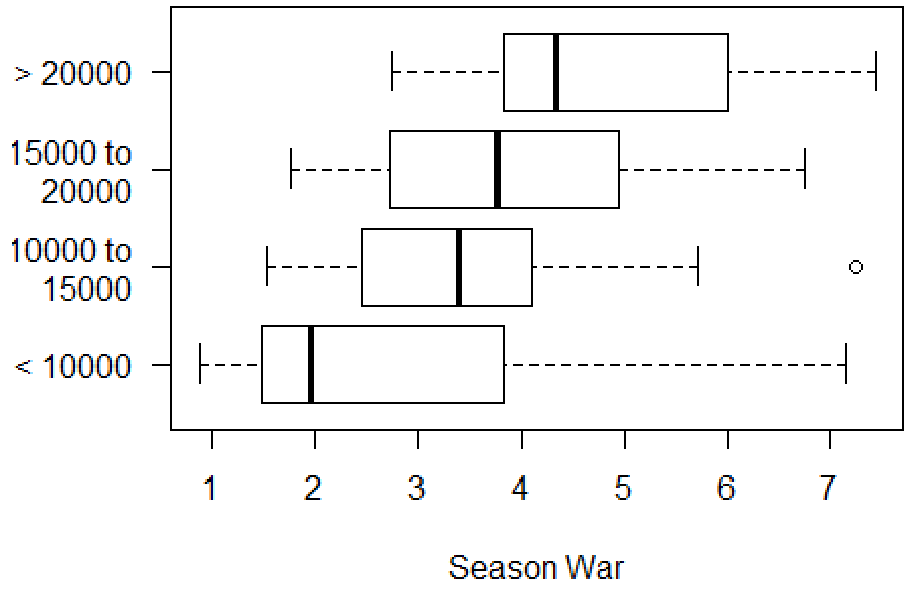
A **stripchart** is a good way to display **continuous** data when it is **grouped** by another variable. In this case, we want to group each pitcher's WAR by BF.

```
stripchart(WAR.Season ~BF.group, data=hofpitching, las=1)
```



A **stripchart** graphs each individual observation which can make the data a little hard to interpret. If one would rather see a summary of the data, **parallel boxplots** are more pleasing to the eye and easier to interpret.

```
boxplot(WAR.Season ~BF.group, data=hofpitching, horizontal=T, xlab="Se
ason War", las=1)
```



Because the center line of the boxplots increases as you move along the horizontal axis and as number of BF increases, we can see that the number of batters faced is **positively associated** with WAR per season.

What if we want to only look at modern era pitchers?

By subsetting our pitching data to those whose mid-career was 1960 or later, we can create a new data frame for recent pitchers.

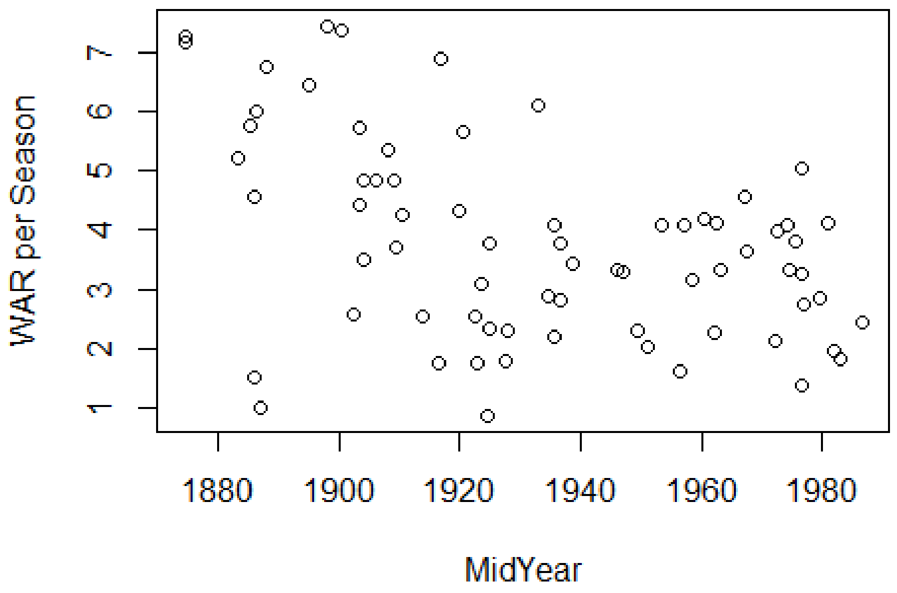
```
hofpitching$MidYear =with(hofpitching, (From +To)/2)
hofpitching.recent =subset(hofpitching, MidYear >=1960)
order.WAR =order(hofpitching.recent$WAR.Season)
head(hofpitching.recent)
```

```
##     Rk                   X Inducted Yrs From   To ASG  WAR   W   L
 W.L.
## 3    3     Bert Blyleven HOF     2011  22 1970 1992   2 90.7 287 250
0.534
## 5    5       Jim Bunning HOF     1996  17 1955 1971   9 56.7 224 184
0.549
## 6    6     Steve Carlton HOF     1994  24 1965 1988  10 78.6 329 244
0.574
## 12 12      Don Drysdale HOF     1984  14 1956 1969   9 57.4 209 166
0.557
## 13 13 Dennis Eckersley HOF     2004  24 1975 1998   6 58.6 197 171
0.535
## 16 16   Rollie Fingers HOF     1992  17 1968 1985   7 23.3 114 118
0.491
##     ERA    G  GS  GF  CG SHO  SV     IP    H    R   ER  HR   BB IBB
   SO
## 3  3.31  692 685   3 242  60   0 4970.0 4632 2029 1830 430 1322  71
 3701
## 5  3.27  591 519  39 151  40  16 3760.1 3433 1527 1366 372 1000  98
 2855
## 6  3.22  741 709  13 254  55   2 5217.2 4672 2130 1864 414 1833 150
 4136
## 12 2.95  518 465  34 167  49   6 3432.0 3084 1292 1124 280  855 123
 2486
## 13 3.50 1071 361 577 100  20 390 3285.2 3076 1382 1278 347  738  91
 2401
## 16 2.90  944  37 709   4   2 341 1701.1 1474  615  549 123  492 109
 1299
##     HBP BK  WP    BF        BF.group WAR.Season MidYear
## 3   155 19 114 20491         > 20000   4.122727  1981.0
## 5   160  8  47 15618 15000 to\n20000   3.335294  1963.0
## 6    53 90 183 21683         > 20000   3.275000  1976.5
## 12 154 10  82 14097 10000 to\n15000   4.100000  1962.5
## 13  75 16  28 13534 10000 to\n15000   2.441667  1986.5
## 16  39  7  40  6942         < 10000   1.370588  1976.5
```

Mid-career Year vs. seasonal WAR average

Because both **MidYear** and **WAR.Season** are quantitative variables, graphing a **scatterplot** makes the most to display the relationship between these two variables.

```
plot(hofpitching$MidYear, hofpitching$WAR.Season, xlab="MidYear", ylab
="WAR per Season")
```



There seems to be a general **decrease** in WAR per season as mid-career year increases.

Lahman Batting dataset

In order to extract the player ids and birth years from the **Master.csv**, we need to create a function **getinfo** that takes a first and last names and inputs and returns player id and birth year.

```
getinfo =function(firstname, lastname){
  playerline =subset(Master,
                    nameFirst ==firstname &nameLast ==lastname)
  name.code =as.character(playerline$playerID)
  birthyear =playerline$birthYear
  birthmonth =playerline$birthMonth
  birthday =playerline$birthDay
  bYear =ifelse(birthmonth <=6, birthyear, birthyear+1)
list(name.code=name.code, byear=bYear)
}
```

From here, we can get the info for players **Ty Cobb**, **Ted Williams**, and **Pete Rose** and then create data frames for each respective player.

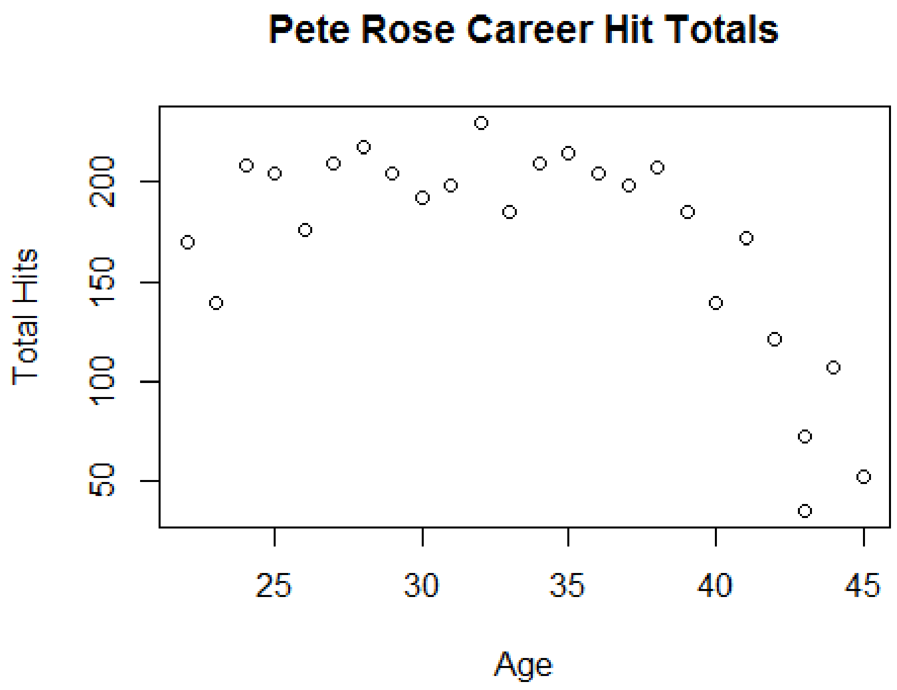
```
cobb.info =getinfo("Ty", "Cobb")
williams.info =getinfo("Ted", "Williams")
rose.info =getinfo("Pete", "Rose")

cobb.data =subset(Batting, playerID ==cobb.info$name.code)
williams.data =subset(Batting, playerID ==williams.info$name.code)
rose.data =subset(Batting, playerID ==rose.info$name.code[1])
```

Because the data frame does not have an **age** variable, we can create it by subtracting birth year (**bYear**) from **yearID**.

```
cobb.data$Age =cobb.data$yearID -cobb.info$byear
williams.data$Age =williams.data$yearID -williams.info$byear
rose.data$Age =rose.data$yearID -rose.info$byear[1]
```

We are interested in seeing the relationship between **hits** and **age** for Pete Rose, and because these variables are both continuous, we can use a **scatterplot**.

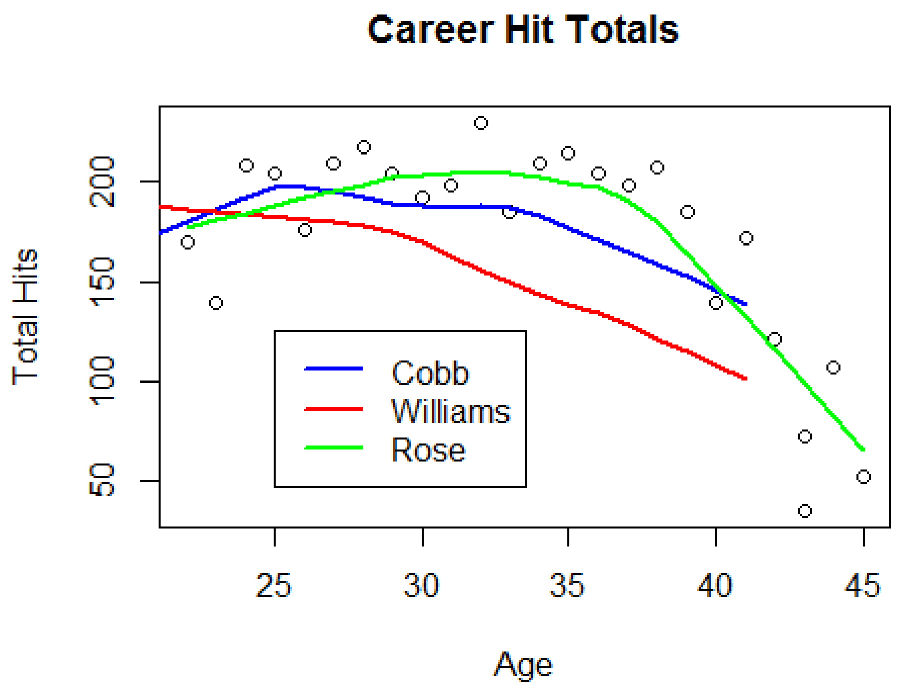
```
plot(rose.data$Age, rose.data$H, ylab="Total Hits", xlab="Age",
main="Pete Rose Career Hit Totals")
```



In order to compare Ty Cobb's and Ted Williams' season batting statistics to Pete Rose, we can overlay line plots of their statistics onto the plot we just created.

```
plot(rose.data$Age, rose.data$H, ylab="Total Hits", xlab="Age",
main="Career Hit Totals")
```

```
lines(lowess(cobb.data$Age, cobb.data$H), lwd=2, col="blue")
lines(lowess(williams.data$Age, williams.data$H), lwd=2, col="red")
lines(lowess(rose.data$Age, rose.data$H), lwd=2, col="green")
legend(25, 125, legend=c("Cobb", "Williams", "Rose"), lwd=2, col=c("bl
ue", "red", "green"))
```



Retrosheet Play-by-Play Data

One way to compare the patterns of homerun hitting is to look at the number of **plate appearances (PAs)** between each homerun.

Because **plate appearance** is not a variable in Retrosheet's **Play-by-Play** data, we can create it by restricting the data frames to plays where a **batting event** occurred.

Let's compare **Mark McGwire** and **Sammy Sosa**; two of baseball's most infamous home run hitters.

But first: we have to create a home run (**HR**) variable for their data frames by using an **ifelse** function. More specifically, if the event of the at-bat (**EVENT_CD**) is equal to **23** (the number code associated with **HR**), then the HR variable gets a value of 1, if not, it gets a 0.

```
sosa.id =as.character(subset(retro.ids,
                              FIRST=="Sammy"&LAST=="Sosa")$ID)
mac.id =as.character(subset(retro.ids,
                             FIRST=="Mark"&LAST=="McGwire")$ID)
```

```
sosa.data =subset(data1998, BAT_ID ==sosa.id)
mac.data =subset(data1998, BAT_ID ==mac.id)

createdata =function(d){
  d$Date =as.Date(substr(d$GAME_ID, 4, 11),
format="%Y%m%d")
  d =d[order(d$Date), ]
  d$HR =ifelse(d$EVENT_CD ==23, 1, 0)
  d$cumHR =cumsum(d$HR)
  d[, c("Date", "cumHR")]
}

mac.hr =createdata(mac.data)
sosa.hr =createdata(sosa.data)
```

Again, we want to restrict McGwire's and Sosa's data frames to where a batting event occurred. From there, we can create the **PA** variable by equating it to the observation it is.

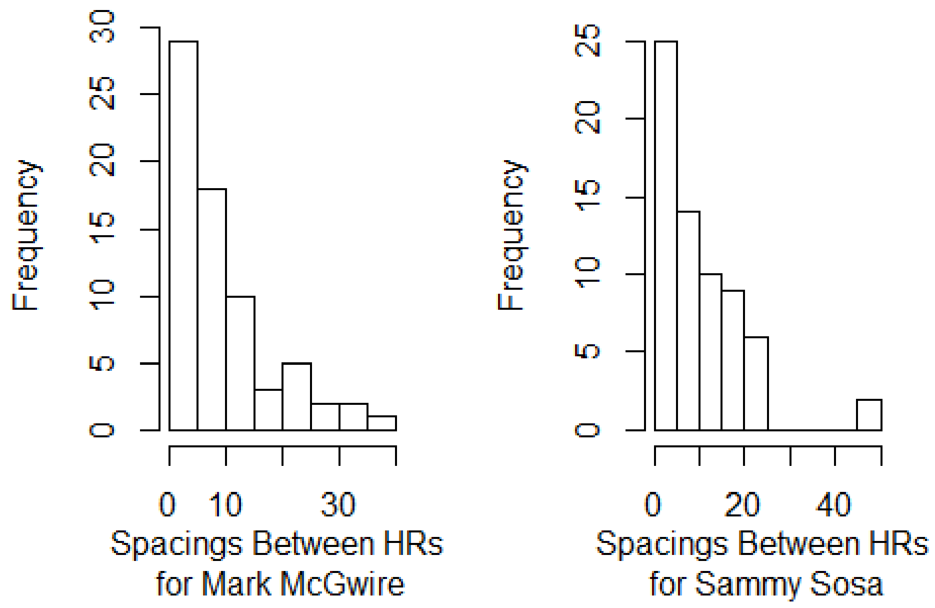
```
mac.data =subset(mac.data, BAT_EVENT_FL ==TRUE)
sosa.data =subset(sosa.data, BAT_EVENT_FL ==TRUE)

mac.data$PA =1:nrow(mac.data)
sosa.data$PA =1:nrow(sosa.data)
```

Since we are interested in the number of **PAs** between **HRs**, we can create a new variable that displays the number of the plate appearance where a home run occurs.

```
mac.HR.PA =mac.data$PA[mac.data$EVENT_CD==23]
sosa.HR.PA =sosa.data$PA[sosa.data$EVENT_CD==23]
```

Now we can compute the spacings between the occurrences of **HRs** and create a hisogram to display them.

```
mac.spacings =diff(c(0, mac.HR.PA))
sosa.spacings =diff(c(0,sosa.HR.PA))

par(mfrow=c(1,2))
hist(mac.spacings, xlab="Spacings Between HRs \nfor Mark McGwire", mai
n="")
hist(sosa.spacings, xlab="Spacings Between HRs \nfor Sammy Sosa", main
="")
```



Chapter 3 did a great job in showing how to graph different types of baseball data. I especially liked how it had me create multiple plots for the same data so I could see and decide for myself which was more effective. Some other skills that were used in this chapter were creating new variables using functions such as **subset** and **ifelse** as well as requiring functions to be written from scratch. Overall, Chapter 3 does well in utilizing functions built-in to R as well as accustoming the user to data frame manipulation.

Chapter 4: The Relation Between Runs and Wins

Chapter 4 explores the relationship between how many runs a team scores and the number of the wins they have. Once we can understand this relationship, it will help us in determining the value an individual player has on their team's wins.

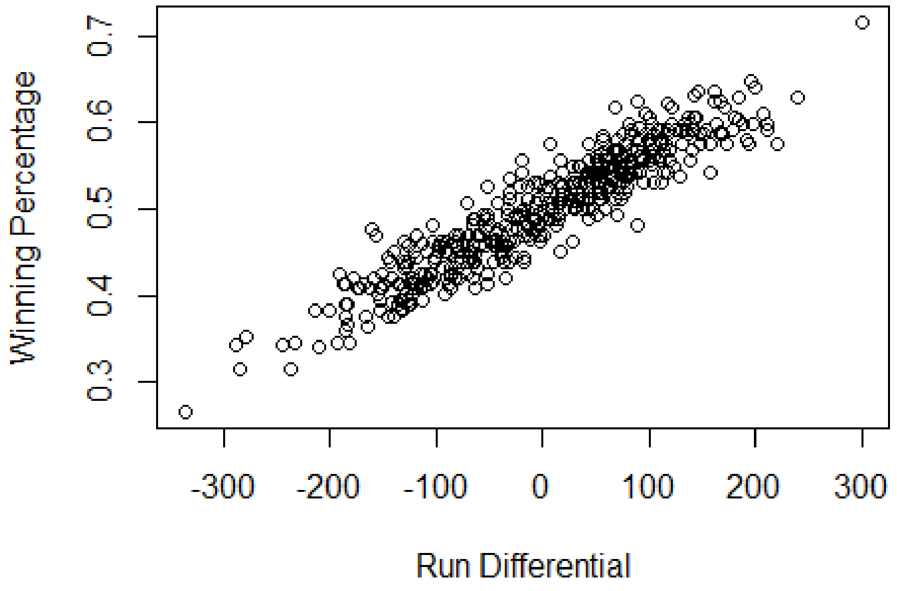
Let's start with looking at the relationship between runs and wins for the most recent seasons. The variables that we will need to accomplish this is**overall Run Differential** and **Winning Percentage**, or **RD** and **Wpct** respectively. Because these variables are not automatically included in the teams data, we must create them ourselves.

For reference, let's denote the variables by the following letters: Runs (**R**), Runs Allowed (**RA**), Wins (**W**), and Losses (**L**).

```
myteams$RD =with(myteams, R -RA)
myteams$Wpct =with(myteams, W/(W +L))
```

A scatterplot can be created to visually display the relationship between **Run Differential** and **Winning Percentage**.

```
plot(myteams$RD, myteams$Wpct, xlab ="Run Differential", ylab ="Winnin
g Percentage")
```



As we can see, there is a strong, positive linear association, or correlation, between Run Differential and Winning Percentage.

It is rather intuitive that in order to win more games, a team should limit the amount of runs they allow. If we want to see how good of a predictor **Run Differential** is of **Winning Percentage**, we can run a **linear regression** of these two variables.

```
linfit =lm(Wpct~RD, data=myteams)
linfit

##
## Call:
## lm(formula = Wpct ~ RD, data = myteams)
##
## Coefficients:
## (Intercept)           RD
##   0.4999918    0.0006287
```

Thus, a team's **Winning Percentage** can be estimated from its **Run Differential** by the equation:

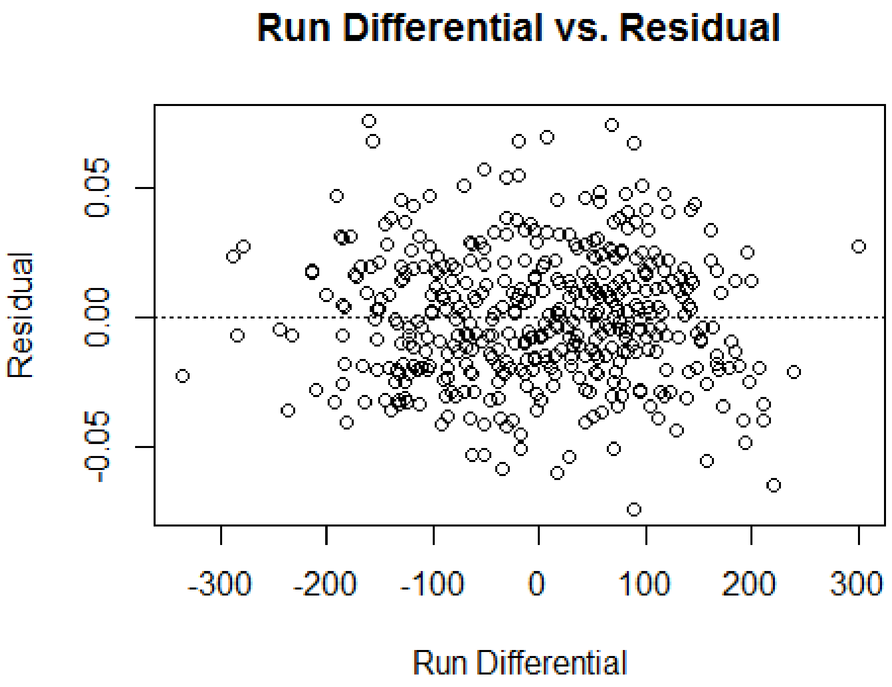
Wpct = 0.4999918 + 0.0006287 X RD

The **0.4999918** value from this equation means that if a team has a run differential of **0**, our model predicts that, **on average**, they will win about **49.99%** of their games.

Likewise, the **0.0006287** value suggests that for every **1** additional run in run differential, our model predicts a 0.0006287 **increase** in winning percentage **on average**.

We can see how well this equation predicts values by looking at its residuals.

```
myteams$linResiduals =residuals(linfit)
plot(myteams$RD, myteams$linResiduals, xlab="Run Differential", ylab="
Residual",
main="Run Differential vs. Residual")
abline(h=0, lty=3)
```



Because the residuals are randomly dispersed around 0, we have evidence that using a linear model for this data is appropriate.

The Pythagorean Formula for Winning Percentage

Bill James, the father of Sabermetrics, derived the following non-linear formula to estimate winning percentage, called the **Pythagorean expectation**:

Wpct = $\frac{R^2}{R^2+RA^2}$

The purpose of using a non-linear function rather than the linear model is to avoid exceeding our bounds. For example, let's say that in a 162-game season, a team scores 9 runs per game and allows 4 runs per game. Thus, the run differential would be (162)(9) - (162)(4) = 810. Using our linear model:

Wpct = 0.4999918 + 0.0006287(810) = 1.009

A winning percentage over 1 is impossible. Using the non-linear model:

Wpct = $\frac{1458^2}{1458^2+648^2}$ = 0.835

which is much more realistic.

After Bill James applied his Pythagorean formula, other analysts wondered if an exponent different from 2 was more fitting.

By rewriting the Pythagorean formula to be more general, they came up with:

Wpct = $\frac{\mathrm{R}^{\mathrm{k}}}{R^k + RA^{\mathrm{k}}}$

where**k** is some unknown variable.

With some algebra, and keeping in mind that winning percentage is the ratio of wins to the sum of wins and losses, it can be rewritten as:

$\frac{W}{L} = \frac{\mathrm{R}^{\mathrm{k}}}{\mathrm{RA}^{\mathrm{k}}}$

By taking the logarithm of both sides, we can obtain a linear relationship and estimate **k** using linear regression:

$\log(\frac{W}{L}) = \mathrm{k} \cdot \log(\frac{R}{RA})$

In this case, **log(W/L)** is our response and **log(R/RA)** is our predictor.

```
myteams$logWratio =log(myteams$W/myteams$L)
myteams$logRratio =log(myteams$R/myteams$RA)
pytFit =lm(logWratio ~0 +logRratio, data=myteams)
pytFit

##
## Call:
## lm(formula = logWratio ~ 0 + logRratio, data = myteams)
##
## Coefficients:
## logRratio
##      1.88
```

As it turns out, the R output suggests that the formula use an exponent of 1.903, which is smaller than James' choice of 2.

Chapter 4 addresses the question every baseball player, manager, and fan wants to know: what does it take to win a game? It is intuitive that the more runs a team scores, the more likely it is that they wil win games. What I liked most about this chapter is the use of the Pythagorean theorem. Throughout school, I never encountered a way that this theorem would apply to my interests outside of academics, but now that I have, it does make a lot of sense.

Chapter 5: Value of Plays Using Run Expectancy

Chapter 5 explores one of the most important aspects of sabermetrics: run expectancy. Intuitively, run expectancy changes depending on how many runners are on base and how many outs there are. More specifically, first, second, and third base can either be empty or occupied (8 possibilities), and there can be anywhere from 0 to 2 outs (3 possibilities). This gives us 8 x 3 = 24 possible arrangements of runner position and number of outs. Chapter 5 details how to create a matrix for these possibilities and use it to determine run expectancy for each case.

The data that we will be using is the play-by-play data from the 2011 season denoted as **data2011**.

At any particular plate appearance, there is a potential to score. However, the potential is not always the same. For instance, 3 runners on base and 0 outs is likely to have a higher potential than 0 runners on base and 2 outs. To calculate a measurement for run contribution at any given plate appearance, we must know how many total runs have already been scored in the game and how many total runs are scored at the end of the half inning. By taking the difference of the two we will then have the runs scored in the remainder of the inning.

First, we need to create a variable **RUNS** that is equal to the sum of the home team's score and the visitor's score at each plate appearance. For example, let's say that in the top of the first inning the first 3 batters get out. The corresponding **RUNS** score for each of those 3 plate appearances would be 0 because no runs have been scored in the game yet.

```
data2011$RUNS =with(data2011, AWAY_SCORE_CT +HOME_SCORE_CT)
```

Because **data2011** does not contain data that splits up innings into halves (the home team's half and the visitor's half), we must create that ourselves. The information we need to do so is the game id, the inning, and the team at bat. By pasting all of this information into one variable, **HALF.INNING** displays what game is being played, what inning the game is in, and if the home team or the visitor is batting (denoted by 1 or 0 respectively).

```
data2011$HALF.INNING =with(data2011,
paste(GAME_ID, INN_CT, BAT_HOME_ID))
head(data2011$HALF.INNING)

## [1] "ANA201104080 1 0" "ANA201104080 1 0" "ANA201104080 1 0"
## [4] "ANA201104080 1 1" "ANA201104080 1 1" "ANA201104080 1 1"
```

The first row says that the game is in the first inning with the visiting team batting in Anaheim on April 8, 2011. The second row says the same information except that the home team (Los Angeles Angels of Anaheim) is batting.

Next, we need to know the maximum total score for each half inning. To do so, we must check if the batter's destination or any of the runners' destinations exceed 3 (or go past 3rd base).

```
data2011$RUNS.SCORED =with(data2011, (BAT_DEST_ID >3) +(RUN1_DEST_ID >
3) +(RUN2_DEST_ID >3) +(RUN3_DEST_ID >3))
```

The **aggregate** function then lets us compute the total runs scored in each half inning and store it in **RUNS.SCORED.INNING**.

```
RUNS.SCORED.INNING =aggregate(data2011$RUNS.SCORED, list(HALF.INNING =
 data2011$HALF.INNING), sum)
```

By using the **aggregate** function again, now with the **[** function, we set **RUNS.SCORED.START** to be equal to the total number of runs scored in the game at the beginning of each half inning. (the **[** function and the **1** tells R to only look at the first observation for each half inning and calculate the number of runs scored up to that point)

```
RUNS.SCORED.START =aggregate(data2011$RUNS, list(HALF.INNING = data201
1$HALF.INNING), "[", 1)
```

Now, the maximum total score in a half inning is equal to the sum of initial total runs and the runs scored. **MAX** is a new data frame that assigns maximum runs scored to a new variable **x**. We then merge this data frame to **data2011** where the maximum runs scored is called **MAX.RUNS**.

```
MAX =data.frame(HALF.INNING=RUNS.SCORED.START$HALF.INNING)
MAX$x =RUNS.SCORED.INNING$x +RUNS.SCORED.START$x
data2011 =merge(data2011, MAX)
N =ncol(data2011)
names(data2011)[N] = "MAX.RUNS"
```

Finally, we can compute the runs scored in the remainder of the inning by subtractng **RUNS** from **MAX.RUNS**.

```
data2011$RUNS.ROI =with(data2011, MAX.RUNS -RUNS)
```

Now that the remainder-of-the-inning runs scored are calculated, we can now create the matrix for the run expectancy.

Like previously mentioned, we must know how and if the bases are occupied and the number of outs at any given plate appearance. The following variables are assigned either a 1 or a 0 if the specified base is occupied.

```
RUNNER1 =ifelse(as.character(data2011[,"BASE1_RUN_ID"])=="", 0, 1)
RUNNER2 =ifelse(as.character(data2011[,"BASE2_RUN_ID"])=="", 0, 1)
RUNNER3 =ifelse(as.character(data2011[,"BASE3_RUN_ID"])=="", 0, 1)
```

The number of outs is already a variable in **data2011** named **OUTS_CT**.

Because we want to know both of these pieces of information at once, the function **get.state** helps us create a variable by combining the runner indicators and the number of outs. **State** is the term we will use to describe the runners/outs situation. For example, 010 1 means that there is a runner on second with one out. Likewise, 111 2 means that the bases are loaded with two outs.

```
get.state =function(runner1, runner2, runner3, outs){
  runners =paste(runner1, runner2, runner3, sep="")
paste(runners, outs)
}
```

We can use this function to create the variable **STATE**.

```
data2011$STATE =get.state(RUNNER1, RUNNER2, RUNNER3, data2011$OUTS_CT)
head(data2011$STATE)

## [1] "000 0" "000 1" "000 2" "000 0" "100 0" "100 1"
```

We are really only interested in game plays where there is change. More specifically, we are interested in changes of runner position, number of outs, or runs scored. We can't just look at changes of state because let's say that the state is 010 1. If the player at bat hits a double, the state stays at 010 1 but the score has increased by 1.

Three new runner variables indicate if the specified base is occupied after the play, either by an already present runner or the batter. The variable **NOUTS** is also the number of outs following the play.

```
NRUNNER1 =with(data2011, as.numeric(RUN1_DEST_ID==1 |BAT_DEST_ID==1))
NRUNNER2 =with(data2011, as.numeric(RUN1_DEST_ID==2 |RUN2_DEST_ID==2 |
BAT_DEST_ID==2))
NRUNNER3 =with(data2011, as.numeric(RUN1_DEST_ID==3 |RUN2_DEST_ID==3 |
RUN3_DEST_ID==3 |BAT_DEST_ID==3))
NOUTS =with(data2011, OUTS_CT +EVENT_OUTS_CT)
```

Again, we can use **get.state** to create a new variable **NEW.STATE**.

```
data2011$NEW.STATE =get.state(NRUNNER1, NRUNNER2, NRUNNER3, NOUTS)
```

Now, we can restrict **data2011** to plays where either a change of state happened or run(s) scored.

```
data2011 =subset(data2011, (STATE!=NEW.STATE) |(RUNS.SCORED>0))
```

Before we can compute the run expectancies, we must first make sure that our calculations are based on full half innings. Because teams can have walk-off wins, we need to limit our data set to full 3 out half innings. A walk-off win is when the home team scores the game-winning run in the 9th inning or later, automatically ending the game before recording 3 outs.

The **ddply** function in the **plyr** package helps us compute the number of outs in each half inning. Once the outs are computed, we can subset **data2011** by half innings that have 3 outs and create a complete half inning version called **data2011C**. It is important to note that by not including these run-scoring incomplete half innings, we are introducing a small bias to our run expectancies.

```
library(plyr)
data.outs =ddply(data2011, .(HALF.INNING), summarize,
Outs.Inning =sum(EVENT_OUTS_CT))
data2011 =merge(data2011, data.outs)
data2011C =subset(data2011, Outs.Inning ==3)
```

By taking the average of remainder-of-the-inning runs scored by state, we can now compute the run expectancies!

```
RUNS =with(data2011C, aggregate(RUNS.ROI, list(STATE), mean))
```

In order to display the expectancies as a matrix, a variable **Outs** must be created by taking the "outs" portion of state in the **Group** variable in **RUNS**.

```
RUNS$Outs =substr(RUNS$Group, 5, 5)
RUNS =RUNS[order(RUNS$Outs), ]
```

Now, we can create an 8 x 3 matrix and label the rows with runner positions and columns with outs.

```
RUNS.out =matrix(round(RUNS$x, 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

##      0 outs 1 out 2 outs
## 000   0.47  0.25   0.10
## 001   1.45  0.94   0.32
## 010   1.06  0.65   0.31
## 011   1.93  1.34   0.54
## 100   0.84  0.50   0.22
## 101   1.75  1.15   0.49
## 110   1.41  0.87   0.42
## 111   2.17  1.47   0.76
```

As we expected, the run expectancy for a state that has more runners and fewer outs is greater than a state of fewer runners and more outs. With the bases loaded and no outs, we expect 2.17 runs to be scored on average while empty bases with two outs, one-tenth of a run is expected to be scored on average.

Measuring Success of a Batting Play

When a batter comes to the plate with a certain number of runners on and outs recorded, the matrix we created tells us the average number of runs we can expect to be scored as a result of their plate appearance.

But how does this factor into how good of a player they are?

Depending on the outcome of the batter's plate appearance, there will be a change in either state or runs scored, thus resulting in an updated run expectancy. We can put a value to a specific batter's plate appearance by taking into account the expected runs of the state in which they came up to bat (old state), the expected runs of the state after the play (new state), and the number of runs scored. We will call this value a **RUNS VALUE**.

RUNS VALUE = $\text{RUNS}_{\text{New State}}$ - $\text{RUNS}_{\text{Old State}}$ + $\text{RUNS}_{\text{Scored on Play}}$

In order to compute this RUNS VALUE equation, we first create a 32 x 1 matrix that include the 24 states previously defined plus 8 states that represent every runner situation at 3 outs because the result of a play can result in 3 outs. Clearly, the expected number of runs for states with 3 outs is 0 because the half inning is over.

```
RUNS.POTENTIAL =matrix(c(RUNS$x, rep(0, 8)), 32, 1)
dimnames(RUNS.POTENTIAL)[[1]] =c(RUNS$Group, "000 3","001 3",
"010 3","011 3","100 3","101 3","110 3","111 3")
```

We then compute the run expectancy for both the old state and new state, named **RUNS.STATE** and **RUNS.NEW.STATE** respectively.

```
data2011$RUNS.STATE =RUNS.POTENTIAL[data2011$STATE,]
data2011$RUNS.NEW.STATE =RUNS.POTENTIAL[data2011$NEW.STATE,]
```

Now we can create the variable **RUNS.VALUE**.

```
data2011$RUNS.VALUE =data2011$RUNS.NEW.STATE -data2011$RUNS.STATE +dat
a2011$RUNS.SCORED
```

Runs Values of Different Base Hits?

Intuitively, there should be a different runs value for a home run versus a single. But how different are they?

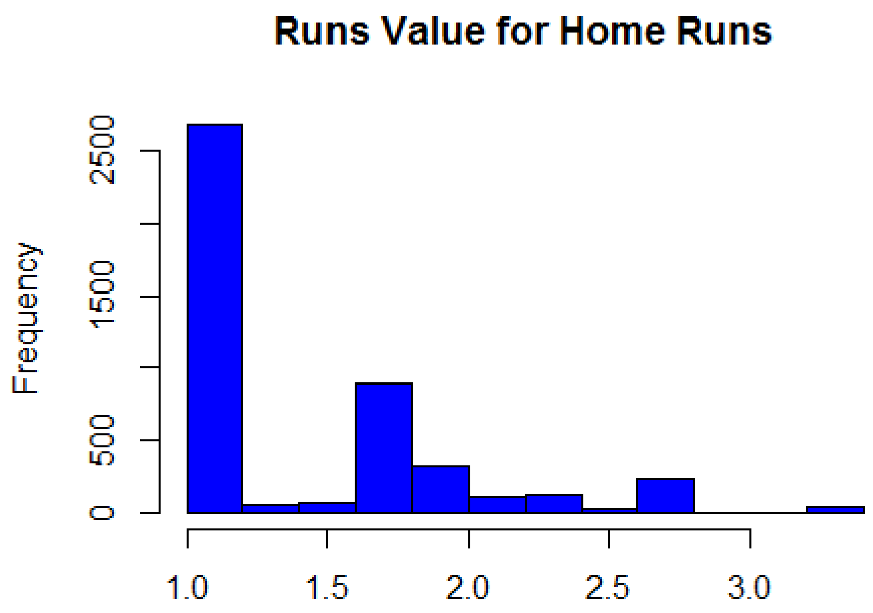
In order to compute these runs values, we must first subset **data2011** by home runs and singles.

Let's do home runs first.

```
d.homerun =subset(data2011, EVENT_CD ==23)
```

We can now create a histogram to look at the overall distribution of runs values when a homerun is hit.

```
hist(d.homerun$RUNS.VALUE, xlab="", main="Runs Value for Home Runs", c
ol="blue")
```



From the histogram, we see that most home runs have a runs value of 1. This means that most home runs come when the bases are empty because the bases start out empty and are still empty as the result of the home run; the only change is in score. We also see a small amount of home runs that result in a runs value over 3. Let's find out in what situation this occurs.

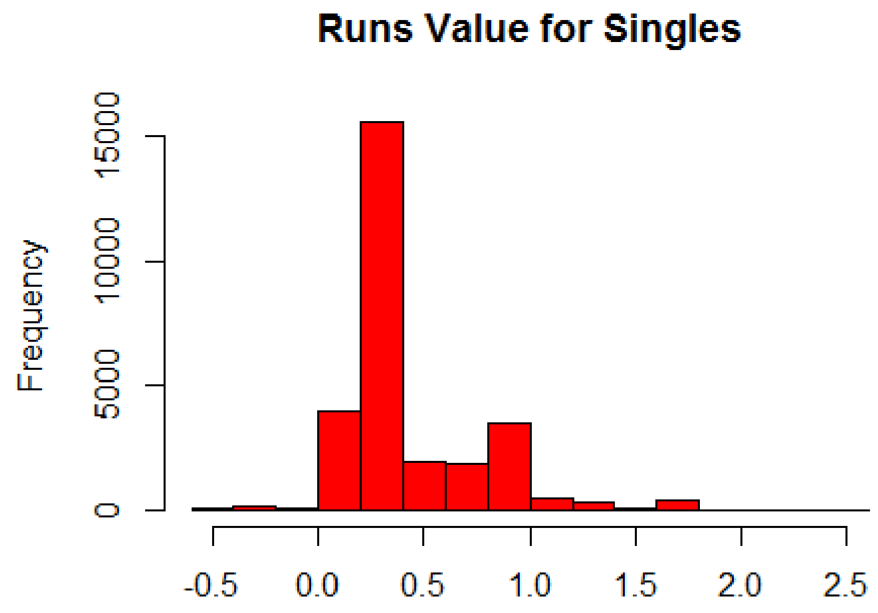
By subsetting the home run runs values by the maximun, we can find which state to new state situation this happens. Intuitively, the highest runs value should come from a grand slam (home run with the bases loaded) because that results in 4 runs.

```
subset(d.homerun, RUNS.VALUE ==max(RUNS.VALUE))[1, c("STATE", "NEW.STA
TE", "RUNS.VALUE")]

##      STATE NEW.STATE RUNS.VALUE
## 6747 111 2     000 2   3.336175
```

As we predicted, the highest runs value comes from a grand slam when there were 2 outs. The state of having 2 outs makes sense because it seems most likely to get the bases loaded with a couple outs mixed in. For instance, getting the first 3 batters on is not as likely as it is for a batter to get on, then maybe the next batter to fly out, etc.

As previously mentioned, we want to be able to compare the runs values from home runs versus those of singles. The best way to do so is by finding the average runs value for both. But first, let's follow the previous steps that we did for home runs and apply it to singles.

```
d.single =subset(data2011, EVENT_CD ==20)
library(MASS)
hist(d.single$RUNS.VALUE, xlab="", main="Runs Value for Singles", col=
"red")
```



```
subset(d.single, RUNS.VALUE ==max(RUNS.VALUE))[1, c("STATE", "NEW.STAT
E", "RUNS.VALUE")]

##        STATE NEW.STATE RUNS.VALUE
## 105749 111 2     001 2   2.556382
```

Interestingly, we find that the maximum runs value for both home runs and singles comes when the bases are loaded with 2 outs.

What is even more interesting is that the runs value can possibly be negative for singles. How can that be?

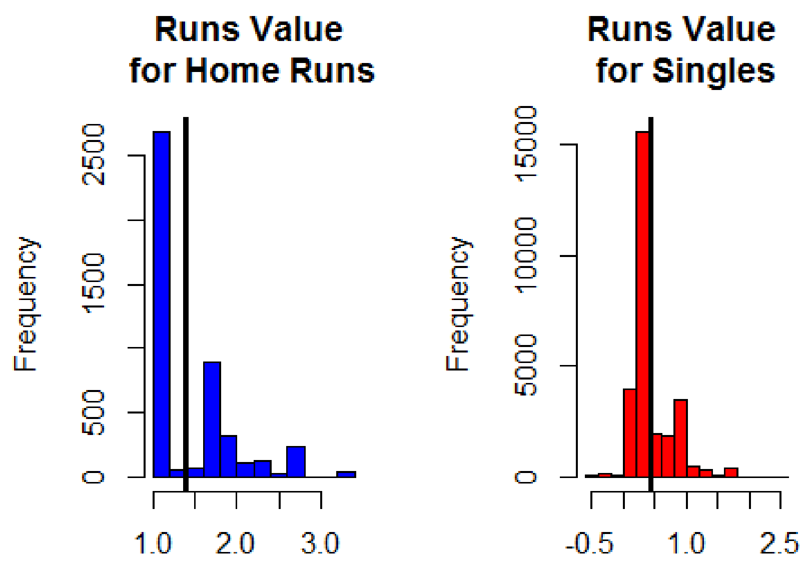
```
subset(d.single, RUNS.VALUE ==min(RUNS.VALUE))[1, c("STATE", "NEW.STAT
E", "RUNS.VALUE")]

##       STATE NEW.STATE RUNS.VALUE
## 69618 010 0     100 1 -0.5622312
```

It turns out that going from a runner on second with no outs to a runner on first with one out results in a negative runs value. This makes sense because with a runner on second and no outs, we expect about 1.06 runs to be scored on average. However, the result of the play is a runner on first with one out which means that the batter made it safely to first, but the runner on second got out. A runner on first with one out is expected to result in about 0.50 runs on average. Thus, resulting in a negative runs value of about -0.56.

```
mean.HR =mean(d.homerun$RUNS.VALUE)
mean.single =mean(d.single$RUNS.VALUE)

mean.HR

## [1] 1.392393

mean.single

## [1] 0.4424186
```

The mean runs value for home runs is about 3 times greater than that of singles.

Looking at the respective histograms side-by-side with the means plotted is a great way to really compare and contrast the two hits.



The Runs Value is really helpful for understanding the benefits of different batting plays, but its uses do not stop there. We can compare career Runs Values for different players or how the position in the batting lineup plays a role. In Chapter 7, we will explore how Runs Values differ depending on pitch count, or how many balls and strikes a batter has during a plate appearance.

Chapter 6: Advanced Graphics

The *lattice* package

Graphics in the **lattice** package display multivariate data in a regular gridlike framework. It automatically produces color-coded figures so the user does not have to implement that themselves.

To help show the utility of **lattice**, let's use five years worth of pitch-by-pitch data for 2011 Cy Young Winner **Justin Verlander**.

To show the type of data that is in the **verlander** data set, we can display a random sample of the data.

```
sampleRows =sample(1:nrow(verlander), 20)
verlander[sampleRows,]

##       season   gamedate pitch_type balls strikes pitches speed    p
x   pz
## 4115    2010 2010-04-17         FF     0       0      29  92.7  0.8
4 0.94
## 12399   2012 2012-08-23         FF     3       2      67  97.8 -1.0
6 1.96
## 12501   2012 2012-08-11         SL     0       0      75  86.4  0.6
4 1.43
## 10338   2011 2011-07-31         CH     0       1      39  87.3 -0.9
6 2.70
## 8728    2011 2011-05-19         FT     0       2      92  95.6 -0.8
3 2.89
## 6937    2010 2010-09-02         CU     1       0      15  79.5  0.1
0 1.15
## 3508    2009 2009-09-19         FF     0       0     108  96.4  0.6
3 2.82
## 2394    2009 2009-08-03         SL     0       0      13  85.5  0.0
6 2.03
## 15058   2012 2012-04-16         CH     0       0      40  87.6 -1.4
2 2.84
## 919     2009 2009-05-20         CU     0       1      67  79.9  0.0
6 1.06
## 2668    2009 2009-08-13         FF     3       2      81 100.1  1.7
0 3.45
## 14082   2012 2012-05-29         FF     1       2     104  98.6 -1.1
4 2.46
## 1541    2009 2009-06-21         FF     2       1      73  96.6  0.2
5 3.10
## 3142    2009 2009-09-04         FT     2       2      83  96.2 -0.9
```

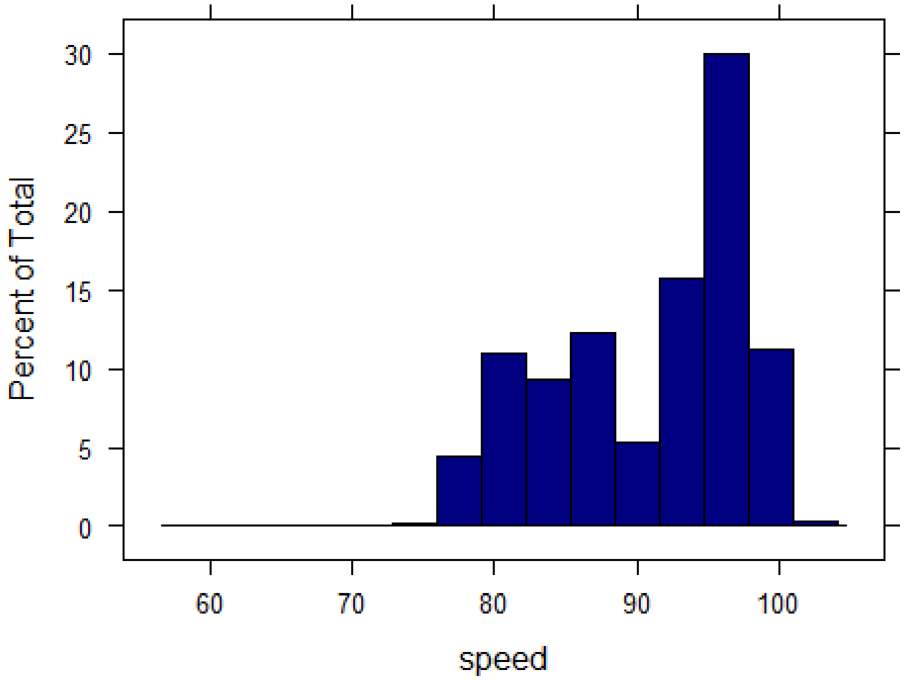
```
0 2.30
## 14302   2012 2012-05-24         FF      0       2      108 100.2  1.3
6 3.72
## 9950    2011 2011-07-15         FT      2       1       12  93.4  0.2
8 1.33
## 7044    2010 2010-09-07         CU      0       1       18  79.8  0.2
5 1.10
## 11019   2011 2011-09-02         FT      0       0       31  92.8 -0.6
3 1.68
## 15237   2012 2012-04-05         FF      0       0       70  92.6 -0.1
8 3.39
## 2795    2009 2009-08-19         CH      1       2       85  86.5 -1.3
0 2.22
##         pfx_x pfx_z batter_hand
## 4115    -6.93 15.44           R
## 12399   -4.35 11.23           L
## 12501    1.57  3.16           R
## 10338   -7.84  4.56           L
## 8728   -11.29  5.10           L
## 6937     8.25 -7.36           R
## 3508    -7.76 13.27           L
## 2394     2.61  2.89           R
## 15058   -9.54  6.35           L
## 919      3.98 -9.59           R
## 2668    -7.55  9.26           R
## 14082   -6.78  6.61           L
## 1541    -5.71  9.61           R
## 3142    -8.98 10.38           L
## 14302   -6.46 11.63           L
## 9950    -9.57  6.76           R
## 7044     5.29 -5.28           R
## 11019   -9.76  9.63           L
## 15237   -5.59  9.77           L
## 2795    -9.00  3.24           L
```

The variable **pitch_type** includes 5 pitch types: four-seam fast ball (**FF**), two-seam fast ball (**FT**), curveball (**CU**), changeup (**CH**), and slider (**SL**). The variable **pitches** contains the number of pitches that Verlander has already thrown. Lastly, variables **px** and **py** indicate the vertical and horizontal location of the pitch respectively.

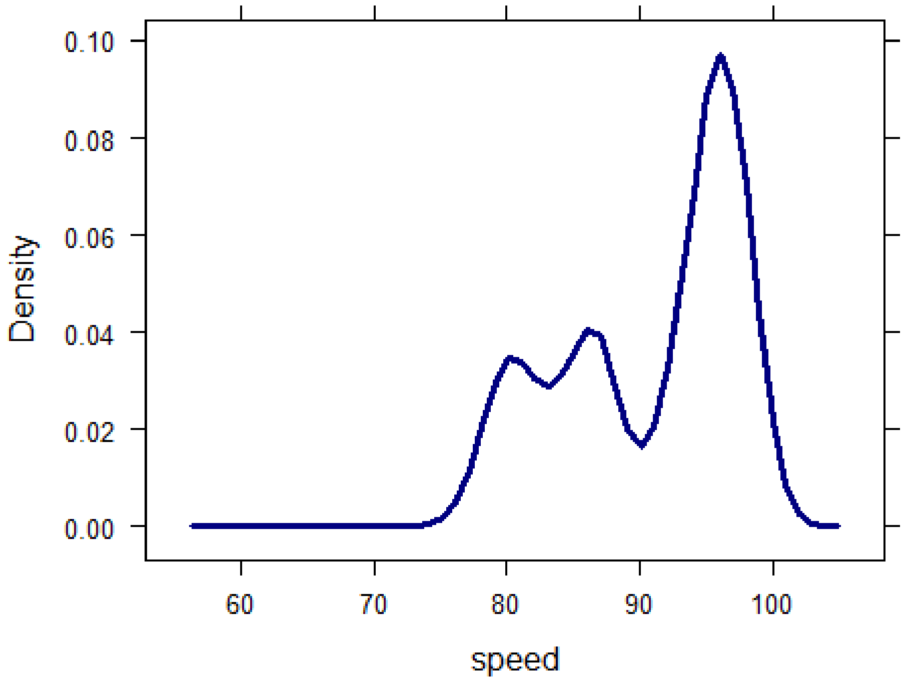
Let's start with some **basic plotting**.

We can use the **histogram** and **densityplot** functions, aptly named, to easily display the distribution of the speed of Verlander's pitches.

```
histogram(~speed, data=verlander, col="navy")
```



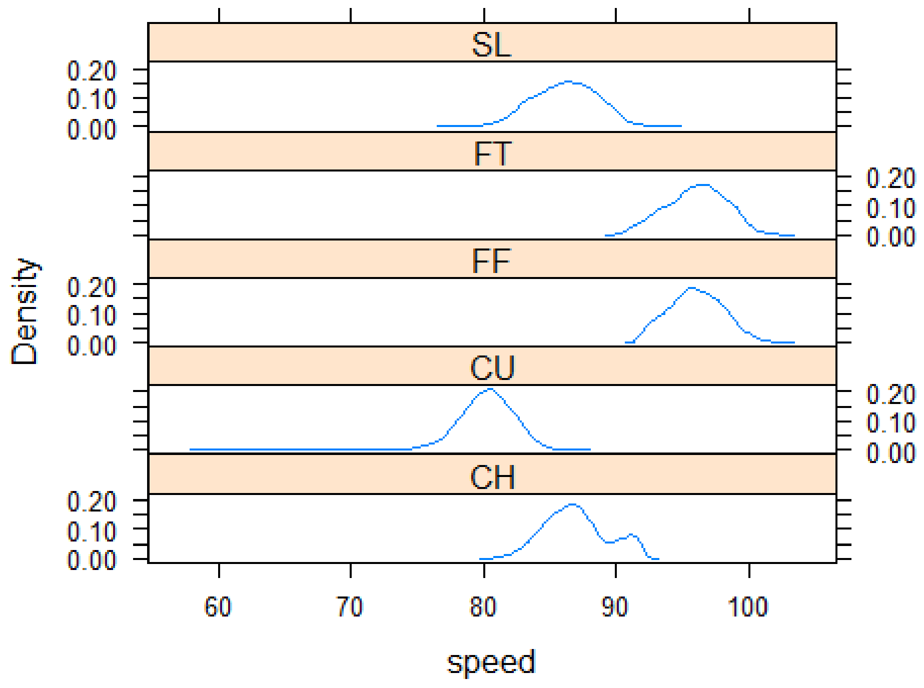
```
densityplot(~speed, data=verlander, plot.points=F, lwd=3, col="navy")
```



An upside of the **lattice** package over the general **graphics** package is that it allows **mulitpanle conditioning**.

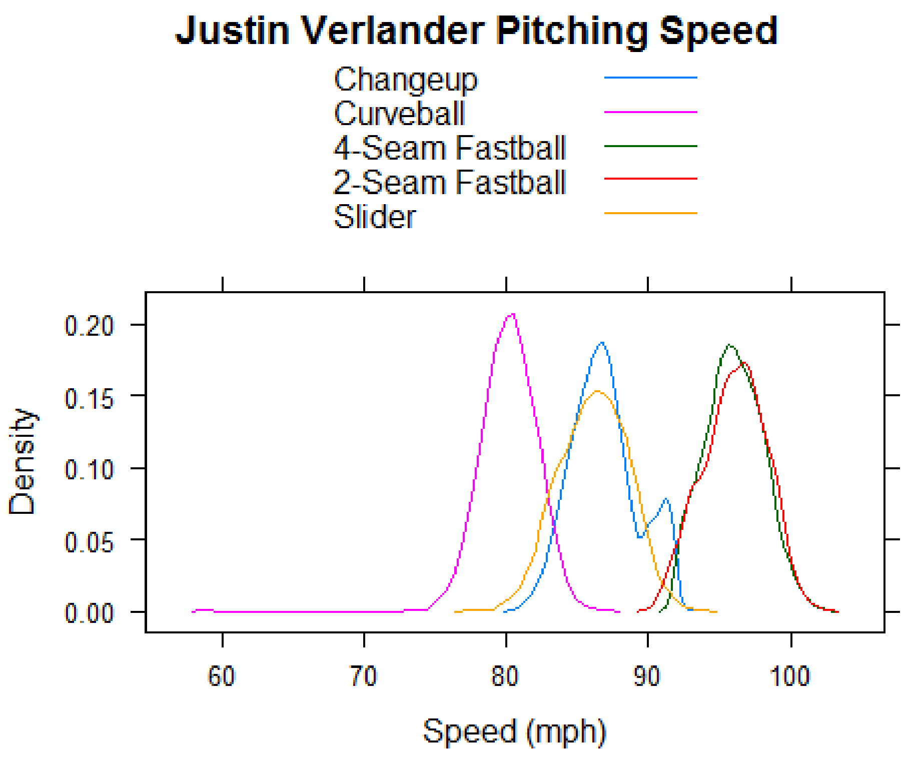
For example, **pitch speed** is highly dependent on the **type** of pitch being thrown, so we can plot pitch speed grouped by pitch type in panel form.

```
densityplot(~speed |pitch_type, data=verlander, layout=c(1,5), plot.po
ints=F)
```



lattice also lets you plot each speed density plot in one panel on top of each other.

```
densityplot(~speed, data=verlander, groups=pitch_type, plot.points=F,
auto.key=list(text=c("Changeup", "Curveball", "4-Seam Fastball",
"2-Seam Fastball", "Slider")),
main="Justin Verlander Pitching Speed", xlab ="Speed (mph)")
```

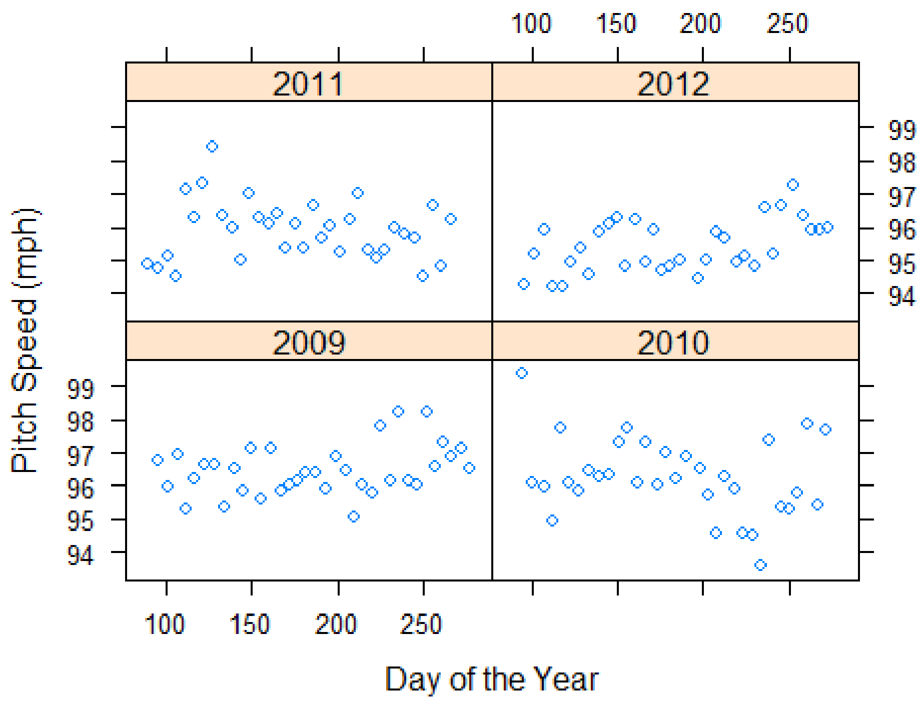


With an overlapping density plot, we can directly compare the speed distributions between the pitch types.

Another useful function is **xyplot**, which produces a scatterplot. Say we want to display the trend of Verlander's four-seam fastball pitch speed throughout the season, **xyplot** can help us do that.

In order to have the correct data to see this trend, we must subset the **verlander** data to only include **FF**, format **gamedate** as an integer that represents the day of the year, and average **speed** by day of the year and season.

```
F4verl =subset(verlander, pitch_type == "FF")
F4verl$gameDay =as.integer(format(F4verl$gamedate, format="%j"))
dailySpeed =aggregate(speed ~gameDay +season, data=F4verl, FUN=mean)

xyplot(speed ~gameDay |factor(season), data=dailySpeed,
xlab="Day of the Year", ylab="Pitch Speed (mph)")
```



One interesting note is that in 2012, Verlander's pitching speed seems to get faster towards the end of the season, while in 2010, some of Verlander's slowest pitches are at the end of the season. COuld pitch type be the factor here?

Suppose we want to compare the speeds of four-seam fastballs and changeups only. As we saw earlier in the densityplots, there seemed to be no overlap between these two pitch types.

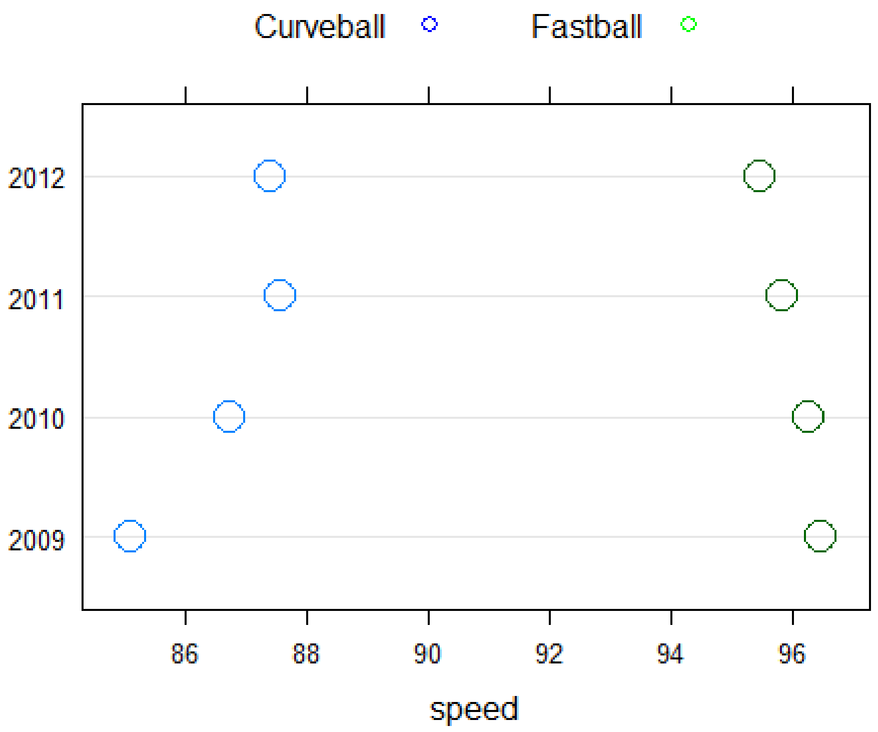
By finding the average for each of these pitches over the course of each season, we can see how they have changed and compare.

```
speedFC =subset(verlander, pitch_type %in%c("FF", "CH"))
avgspeedFC =aggregate(speed ~pitch_type +season, data=speedFC, FUN=mea
n)
avgspeedFC
```

```
##   pitch_type season    speed
## 1         CH   2009 85.06900
## 2         FF   2009 96.46576
## 3         CH   2010 86.72249
## 4         FF   2010 96.23772
## 5         CH   2011 87.56312
## 6         FF   2011 95.83171
## 7         CH   2012 87.38355
## 8         FF   2012 95.46240
```

A **dotplot** can be used to plot these pitch type averages.

```r
dotplot(factor(season) ~speed, groups=pitch_type, data=avgspeedFC, cex
=2, key=list(columns=2, text=list(lab=c("Curveball","Fastball")), poin
ts=list(pch=c(1,1), col=c("blue", "green"))))
```

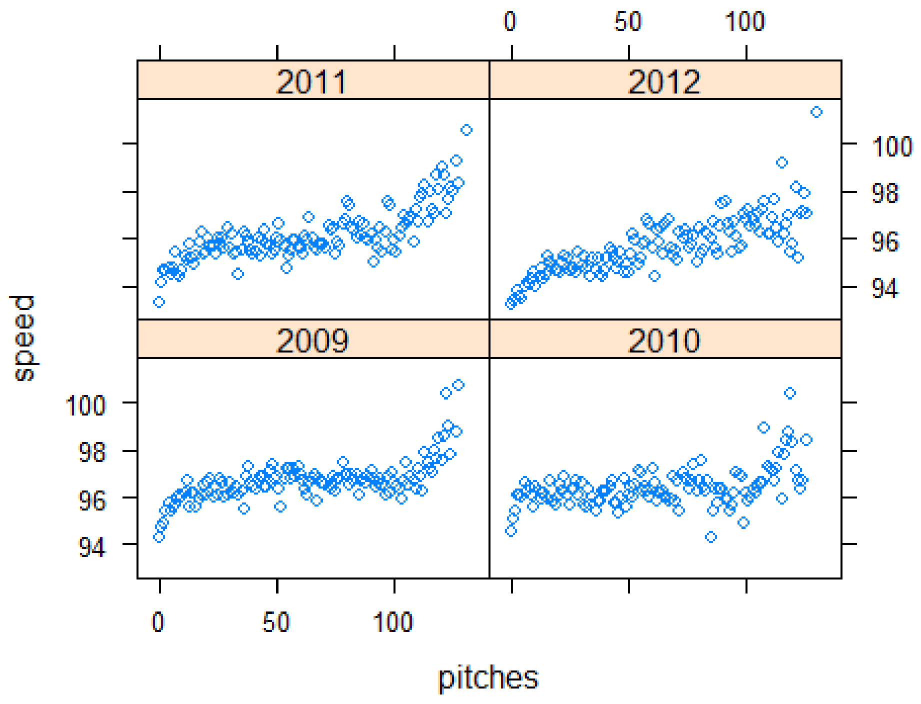


Justin Verlander's speed differential between his fastball and change-up has **decreased** from 2009 to 2012.

Can Verlander maintain the speed of his fastball throughout a game?

We can analyze his pitch speed against his pitch count by aggregating his pitch speeds at every pitch count across the four seasons.

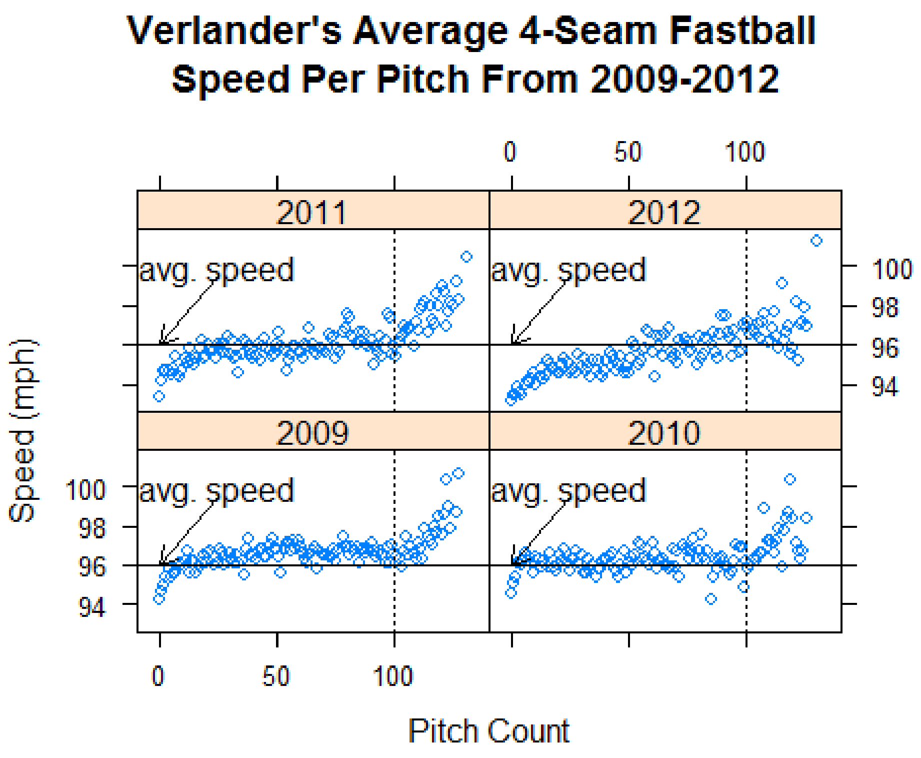
```r
avgSpeed =aggregate(speed ~pitches +season, data=F4verl, FUN=mean)
xyplot(speed ~pitches |factor(season), data=avgSpeed)
```



```
avgSpeedComb =mean(F4verl$speed)
```

Because we are not only considering this relationship over one season, we can use the **panel** function to plot **pitch speed** by **pitch count** for all 4 aforementioned seasons.

```
xyplot(speed ~pitches |factor(season)
       , data = avgSpeed
       , panel = function(...){
panel.xyplot(...)
panel.abline(v =100, lty ="dotted")
panel.abline(h = avgSpeedComb)
panel.text(25, 100, "avg. speed")
panel.arrows(25, 99.5, 0, avgSpeedComb
                              , length = .1)
       }
       ,main="Verlander's Average 4-Seam Fastball Speed Per Pitch From
 2009-2012",
xlab="Pitch Count", ylab="Speed (mph)"
)
```

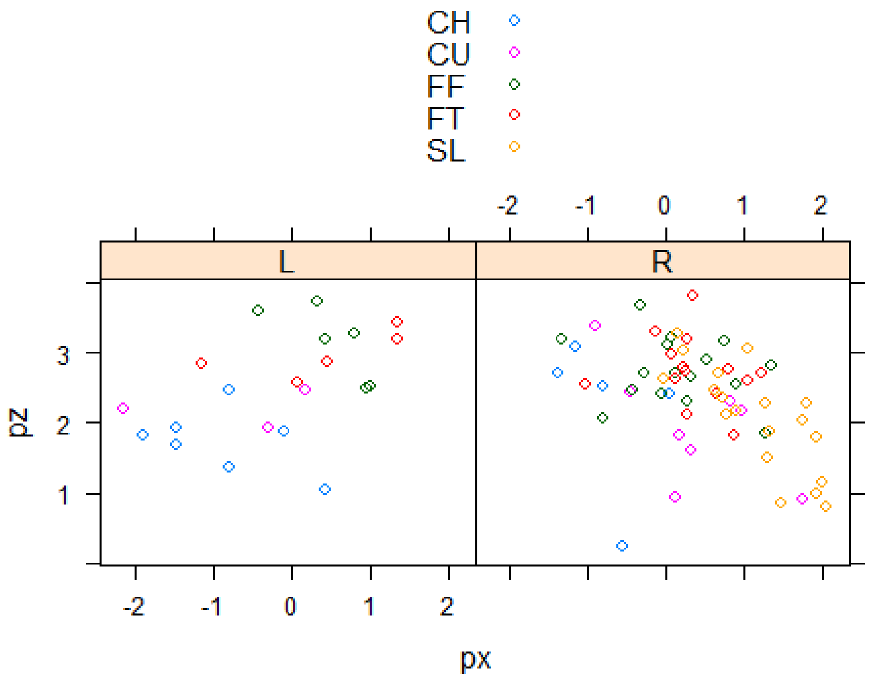


Verlander's pitch speed tends to increase over the duration of a game, even after the 100-pitch count, a typical benchmark period that a pitcher is taken out of a game.

The **lattice** package has great built-in graphics functions, but it is also possible to build a graph step-by-step.

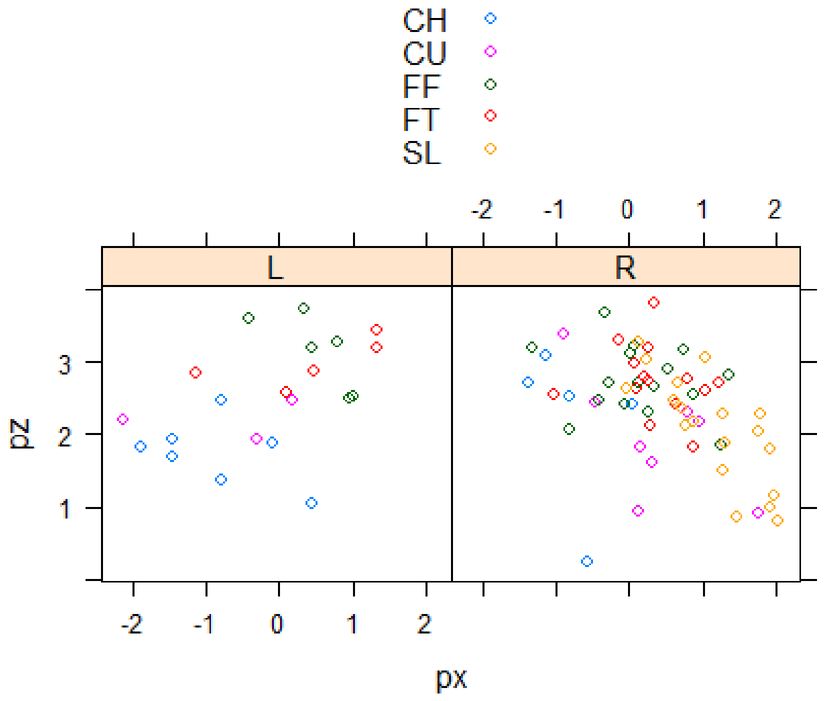
Suppose we are interested in Verlander's pitch location during his no-hitter against the Blue Jays on May 7, 2011. Our **verlander** data frame gives us the x and y coordinates of his pitches. We can use this data, along with pitch type and what side the hitter bats from, to clearly show what type of pitches he throws and where they end up. We are also able to trace an outline of where the limits for the strikezone are to compare Verlander's accuracy. The following code and plots show step-by-step how this desired plot is created.

```
NoHit =subset(verlander, gamedate=="2011-05-07")

xyplot(pz ~px |batter_hand, data=NoHit, groups=pitch_type,
auto.key=TRUE)
```



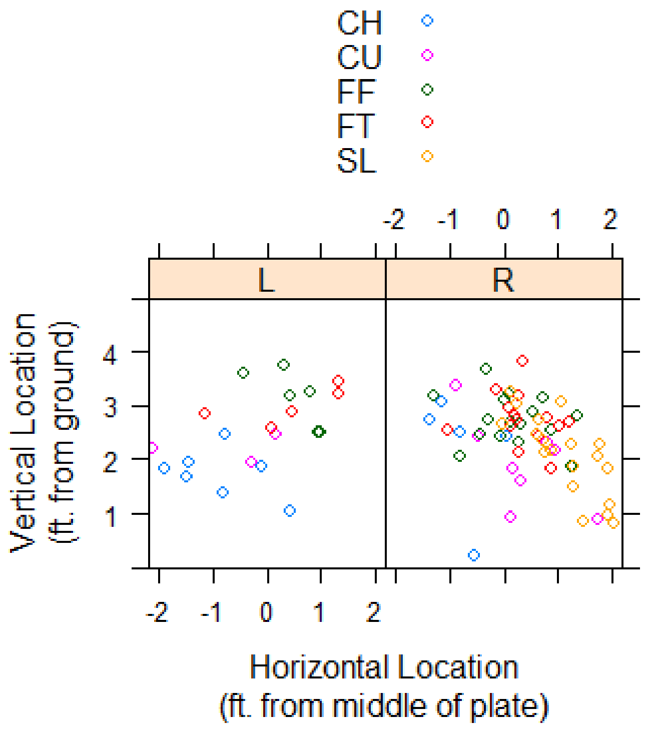
By implementing **aspect="iso"**, we are ensuring that the units of the x and y axes are the same.

```
xyplot(pz ~px |batter_hand, data=NoHit, groups=pitch_type,
auto.key=TRUE, aspect="iso")
```



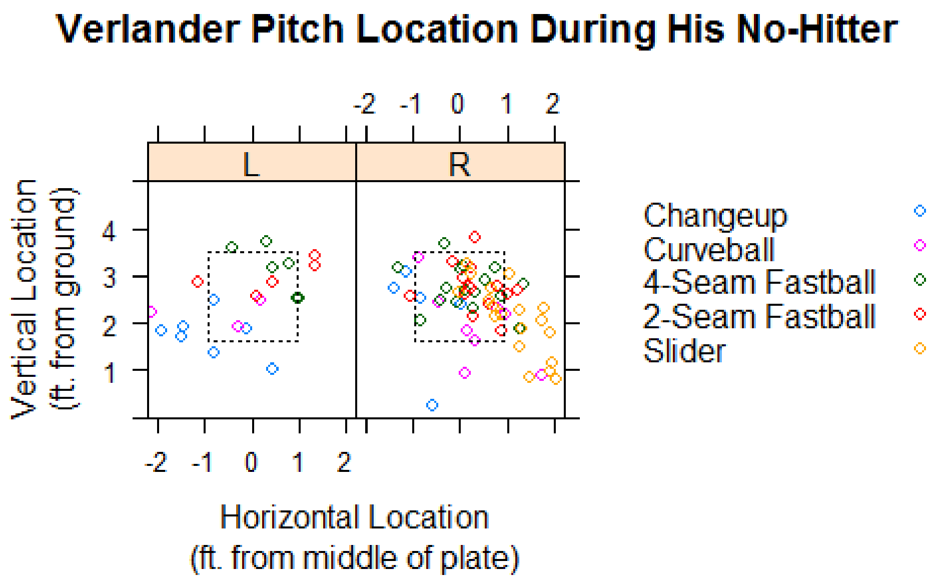
Adding labels to the x and y axes aids in the readability of the graph.

```
xyplot(pz ~px |batter_hand, data=NoHit, groups=pitch_type,
auto.key=TRUE,
aspect="iso",
xlim =c(-2.2, 2.2),
ylim =c(0, 5),
xlab ="Horizontal Location\n(ft. from middle of plate)",
ylab ="Vertical Location\n(ft. from ground)")
```



And finally, by sketching out where the strike zone is located, we can assess the accuracy of Verlander's pitches throughout his no-hitter.

```
topKzone =3.5
botKzone =1.6
inKzone =-0.95
outKzone =0.95

xyplot(pz ~px |batter_hand, data=NoHit, groups=pitch_type,
auto.key=list(space="right", text=c("Changeup", "Curveball",
"4-Seam Fastball",
"2-Seam Fastball", "Slider")),
aspect="iso",
xlim =c(-2.2, 2.2),
ylim =c(0, 5),
xlab ="Horizontal Location\n(ft. from middle of plate)",
ylab ="Vertical Location\n(ft. from ground)",
main ="Verlander Pitch Location During His No-Hitter",
panel = function(...){
panel.xyplot(...)
panel.rect(inKzone, botKzone, outKzone, topKzone,
border ="black", lty =3)
       }
)
```



The great thing about the **lattice** package is that its graphics functions are very easy to use and intuitive. Not only that, but the ability to build your own plots makes this package highly customizable as well. In the following section, another graphics package will be introduced that focuses more on building a plot from scratch.

The *ggplot2* package

Graphics in the **ggplot2** library are constructed by adding layers, starting with the raw data, and then adding strata of statistical summaries. Here, I will illustrate the layer building of **ggplot2** graphics.

To help show the utility of **ggplot2**, let's use Miguel Cabrera's pitch-by-pitch batting data from the 2012 season where he became the first Triple Crown winner since Carl Yastrzemski in 1967.

To show the type of data that is in the **cabrera** data set, we can display a random sample of the data.

```
sampleRows =sample(1:nrow(cabrera), 20)
cabrera[sampleRows,]

##      season   gamedate pitch_type balls strikes speed    px    pz s
wung
## 4285   2011 2011-09-02         CH     1       1  83.3  0.49  3.05
   1
## 1704   2010 2010-04-20         CH     1       0  81.6 -0.79  2.47
   1
## 4021   2011 2011-08-05         SL     0       0  87.2  0.76  1.15
   0
## 790    2009 2009-07-07         SI     1       0  92.2  0.66  2.04
   0
## 513    2009 2009-06-08         FT     1       1  94.1  0.34  2.90
   1
## 3790   2011 2011-07-09         SL     2       2  88.4  0.22  1.15
   1
## 1490   2009 2009-09-29         CH     0       1  85.5 -0.38  0.54
   0
## 2956   2010 2010-09-24         FT     2       0  90.3 -1.09  3.46
   1
## 1982   2010 2010-05-28         FA     0       0  92.5 -0.45  1.20
   0
## 3061   2011 2011-04-09         CH     1       0  76.2 -0.73  0.88
   0
## 4598   2012 2012-09-26         FF     0       2  88.5  1.90 -0.09
   0
## 1908   2010 2010-05-15         FF     0       1  94.6  0.35  2.19
   0
## 4658   2012 2012-09-12         CU     0       0  75.8  0.42  1.57
   1
## 1672   2010 2010-04-17         FT     1       0  89.9  0.82  2.43
   1
## 5956   2012 2012-05-07         CU     1       2  79.7 -0.37  2.02
   1
```

```
## 22      2009 2009-04-08         SL      3       1  86.8 -0.36  2.98
   1
## 611     2009 2009-06-18         FC      0       0  93.6 -0.14  1.34
   1
## 2437    2010 2010-07-24         SI      1       0  91.9  0.10  2.08
   1
## 5519    2012 2012-06-16         FF      0       0  92.5 -0.78  2.23
   1
## 2514    2010 2010-08-03         FF      2       0  87.8 -0.20  1.95
   1
##          hitx   hity hit_outcome
## 4285    94.80  56.19           O
## 1704   -67.25 320.86           O
## 4021       NA     NA        <NA>
## 790        NA     NA        <NA>
## 513        NA     NA        <NA>
## 3790   -45.65 129.09           O
## 1490       NA     NA        <NA>
## 2956       NA     NA        <NA>
## 1982       NA     NA        <NA>
## 3061       NA     NA        <NA>
## 4598       NA     NA        <NA>
## 1908       NA     NA        <NA>
## 4658       NA     NA        <NA>
## 1672   183.94 283.04           O
## 5956       NA     NA        <NA>
## 22      24.59 210.12           O
## 611    -32.15  53.48           O
## 2437       NA     NA        <NA>
## 5519   -61.84 323.55           H
## 2514 -129.36 409.98           O
```

px and **py** correspond to the vertical and horizontal location of the pitch which **hitx** and **hity** correspond to the location of the batted ball. Lastly, **hit_outcome** has on **O** for out, **H** for hit, and **E** for error.

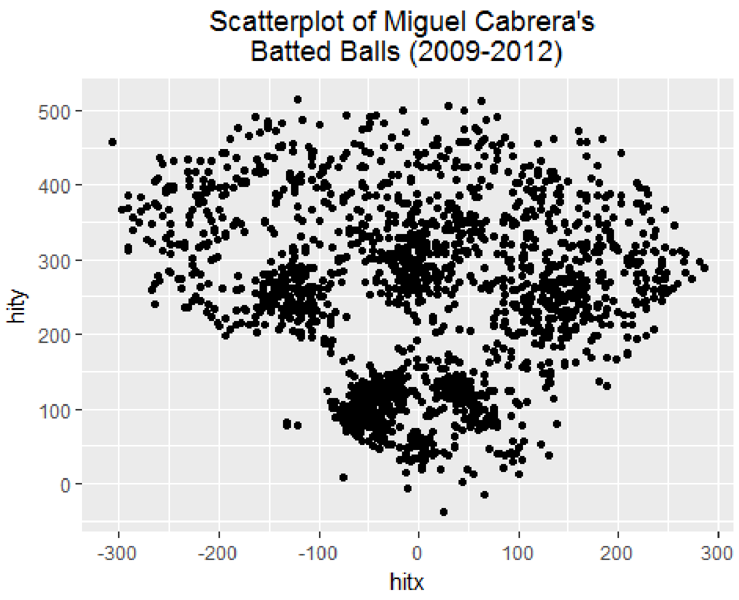
Spray charts are a great way to display the locations of balls put into play by a batter. To begin this process, the **hit** locations of Cabrera's batted balls need to be assigned the position of the x and y axes in the ggplot function.

```
p0 =ggplot(data=cabrera, aes(x=hitx, y=hity))
```

In order to actually produce a graph, we need to add a **points** layer by using the function **geom_point**. We leave the input of this function blank because it uses the inputs set by **p0**.

```
p1 =p0 +geom_point() +
ggtitle("Scatterplot of Miguel Cabrera's \nBatted Balls (2009-2012)")
```

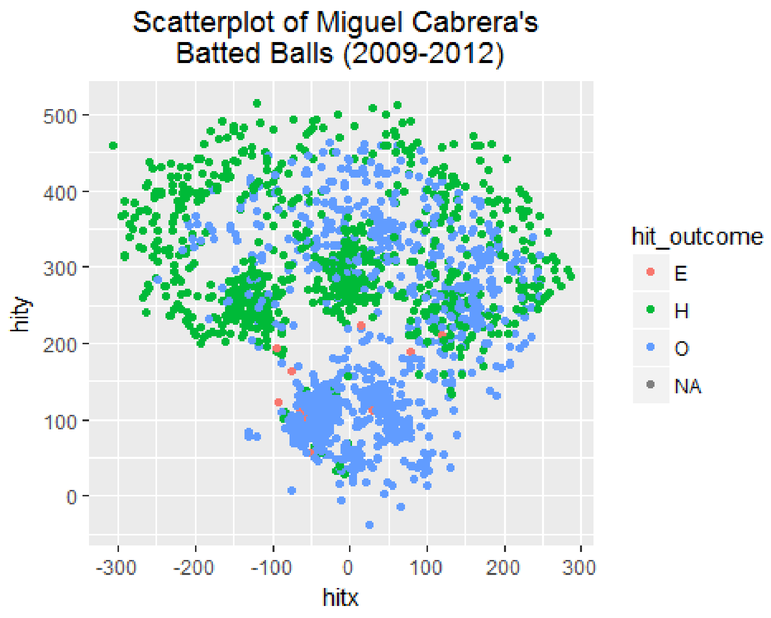
```
+
theme(plot.title =element_text(hjust =0.5))
p1
```



As we can see, the scatterplot roughly takes the form of a baseball field.

It is useful to see what types of batted balls each point is. We can categorize these points by coding each batted ball outcome type (**hit_outcome**) to a different color.

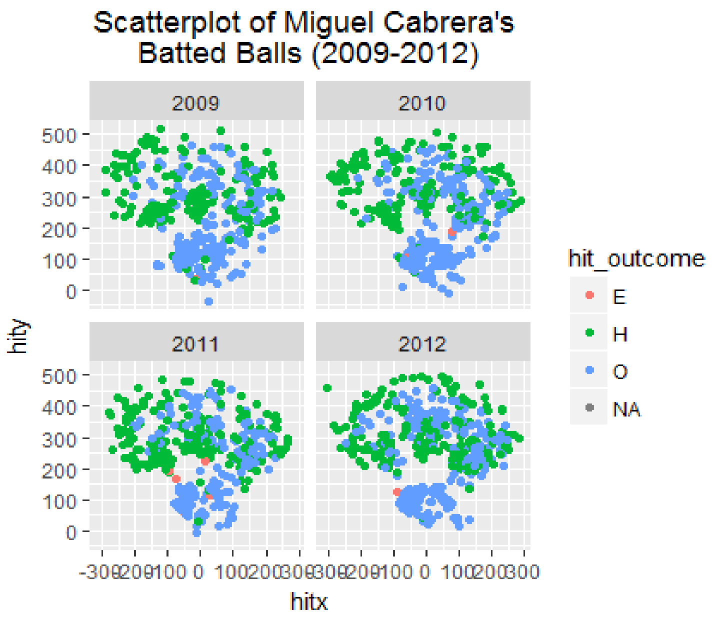
```
p1 =p0 +geom_point(aes(color=hit_outcome)) +
ggtitle("Scatterplot of Miguel Cabrera's \nBatted Balls (2009-2012)")
+
theme(plot.title =element_text(hjust =0.5))
p2 =p1 +coord_equal()
p2
```



coord_equal is a function that ensures the units on the x and y axes are scaled equally (i.e. both in feet).

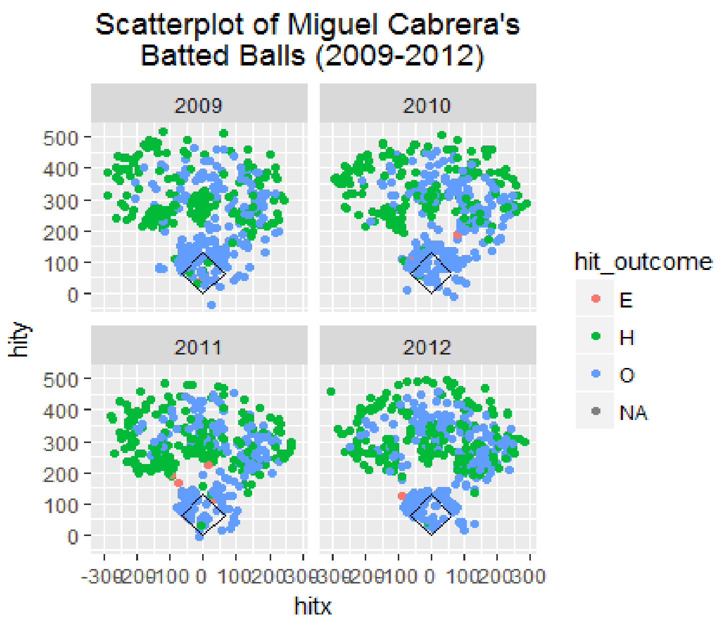
Suppose we want to view the spray chart for all seasons that we have data on fro Miguel Cabrera. To break up the plots into different panels, or **facets**, the **facet_wrap** function is added, taking **season** as an argument.

```
p3 =p2 +facet_wrap(~season)
p3
```



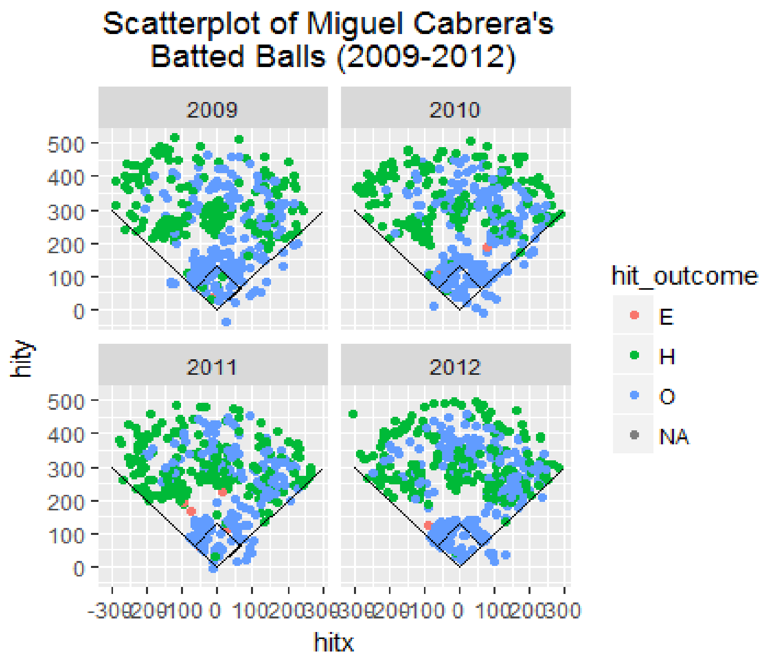
To give the batted ball locations some reference lines, we can create a data frame that defines that location of the base lines in (x,y) coordinates.

```
bases =data.frame(x=c(0, 90/sqrt(2), 0, -90/sqrt(2), 0),
y=c(0, 90/sqrt(2), 2*90/sqrt(2), 90/sqrt(2), 0)
                   )
p4 =p3 +geom_path(aes(x=x, y=y), data=bases)
p4
```



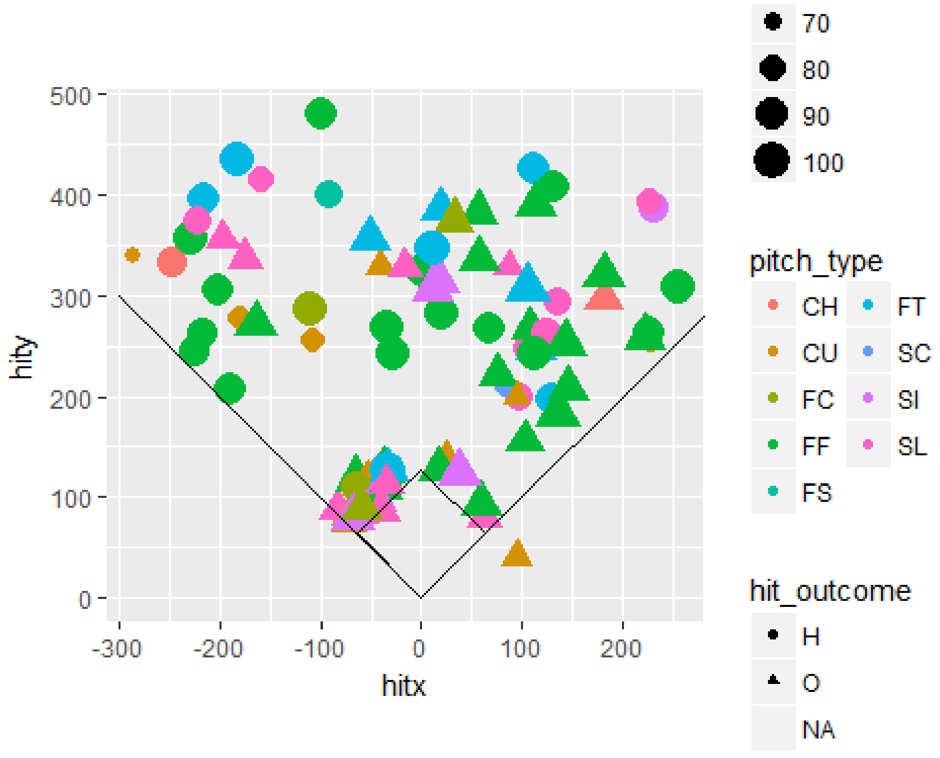
We can add foul lines to the plots as well.

```
p4 +geom_segment(x=0, xend=300, y=0, yend=300) +geom_segment(x=0, xend
=-300, y=0, yend=300)
```



ggplot2 can also combine multiple information on a single plot. For example, let's graph Miguel Cabrera's hits for the months of September and October in the 2012 season. We can assign the shape of the point to be categorized by **hit_outcome**, the color of the point to be categorized by **pitch_type**, and the size of the point to be categorized by **pitch speed**.

```
cabreraStretch =subset(cabrera, gamedate > "2012-08-31")

p0 =ggplot(data = cabreraStretch, aes(hitx, hity))
p1 =p0 +geom_point(aes(shape = hit_outcome, colour = pitch_type
                          , size = speed))
p2 =p1 +coord_equal()
p3 =p2 +geom_path(aes(x = x, y = y), data = bases)
p4 =p3 +guides(col =guide_legend(ncol =2))
p4 +geom_segment(x =0, xend =300, y =0, yend =300) +geom_segment(x =0,
 xend = -300, y =0, yend =300)
```



Being able to build graphs from scratch really makes them customizable. I have seen many of these types of plots, mainly spraycharts, but never knew how they were created. Miguel Cabrera is an atypical hitter in that he can hit to all sides of the field. As a follow-up to this chapter, it would be interesting to get this data for players, such as Jose Bautista, who usually hit to one side of the field or mainly pull the ball.

Chapter 7: Balls and Strikes Effects

Chapter 7 explores the effect of the ball and strike count on the behavior of players, umpires, and the outcome of a plate appearance. By using functions and variables previously created in the prior chapters, Chapter 7 shows more applications for variables, such as, RUNS.VALUE, and packages, such as, lattice.

Hitter's Counts and Pitcher's Counts

In order to see how Runs Values differ amongst different pitch counts, we must first create binary variables for each possible count that indicates if a plate appearance went through that count.

```
pbp2011$pseq =gsub("[.>123N+*]", "", pbp2011$PITCH_SEQ_TX)

pbp2011$c10 =grepl("^[BIPV]", pbp2011$pseq)

pbp2011$c01 =grepl("^[CFKLMOQRST]", pbp2011$pseq)

pbp2011[1:10, c("PITCH_SEQ_TX", "c10", "c01")]
```

```
##    PITCH_SEQ_TX   c10   c01
## 1          FBSX FALSE  TRUE
## 2             X FALSE FALSE
## 3          CBCS FALSE  TRUE
## 4         CBBBB FALSE  TRUE
## 5         BCSBS  TRUE FALSE
## 6        CBB1>S FALSE  TRUE
## 7   CBB1>S.FBFB FALSE  TRUE
## 8           CCX FALSE  TRUE
## 9            BX  TRUE FALSE
## 10        CBBFX FALSE  TRUE
```

Creating these variables is complicated and tedious, although I will go through those thoroughly in a later example, using the data set **pbp11rc** already provides these variables as well as the **RUNS.VALUE** for us.

We want to view these average runs values across counts by using a matrix. More specifically, the number of balls will be denoted by rowand the number of strikes will be denoted by column. Thus, we need to initialize this 4-by-3matrix.

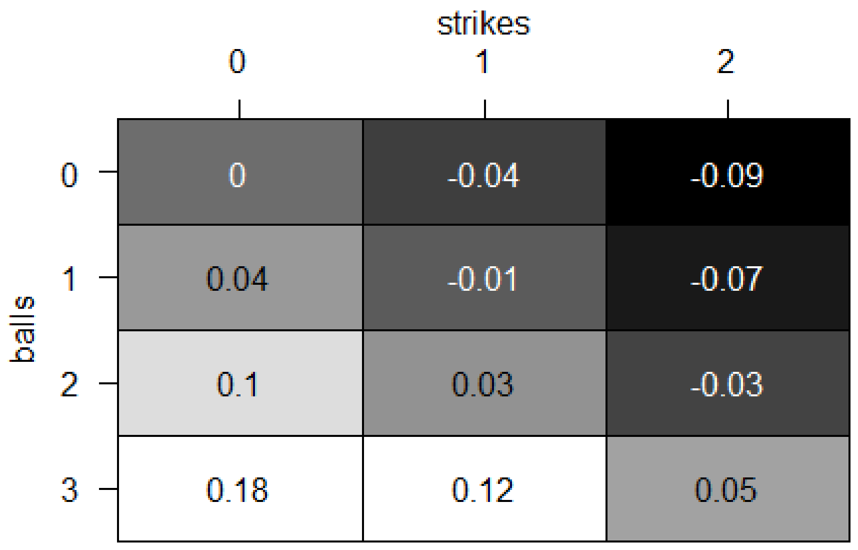
```
runs.by.count =expand.grid(balls =0 :3, strikes =0 :2)
runs.by.count$value =0
```

This next function allows us to calculate the average runs value dependent on count. By taking number of balls and number of strikes as inputs, the function finds every instance that a plate appearance went through that specific pitch count and then averages the corresponding runs values, creating a new variable with those averages.

```
bs.count.run.value =function(b, s){
  column.name =paste("c", b, s, sep="")
mean(pbp11rc[pbp11rc[, column.name] ==1, "RUNS.VALUE"])
}

runs.by.count$value =mapply(FUN=bs.count.run.value,
b=runs.by.count$balls,
s=runs.by.count$strikes
)
```

For visual purposes, the function **countmap** displays this matrix as a 3-by-4 rectangle, with shading corresponding to how negative or how positive the runs value is.

```
countmap =function(data){
require(plotrix)
  data =xtabs(value ~., data)
color2D.matplot(data, show.values=2, axes=FALSE
                  , xlab="", ylab="")
axis(side=2, at=3.5:0.5, labels=rownames(data), las=1)
axis(side=3, at=0.5:2.5, labels=colnames(data))
mtext(text="balls", side=2, line=2, cex.lab=1)
mtext(text="strikes", side=3, line=2, cex.lab=1)
}

countmap(runs.by.count)
```



Miguel Cabrera's Swing Propensity by Count

We can also use the effect of balls and strikes to see how it affects a batter's propensity to swing. For example, we can look at a few seasons from Miguel Cabrera and see how pitch count has affected whether or not he swung. The **balls_strikes_count** data frame contains pitch information on Cabrera such as, swing or no swing, pitch location, and balls and strikes.

Another key factor of whether or not Carbrera swings at a pitch is pitch location. We can presume that pitch location affects if Cabrera swings so running a **loess** or local regression on these variables can help us predict the likelihood of a swing from Cabrera.

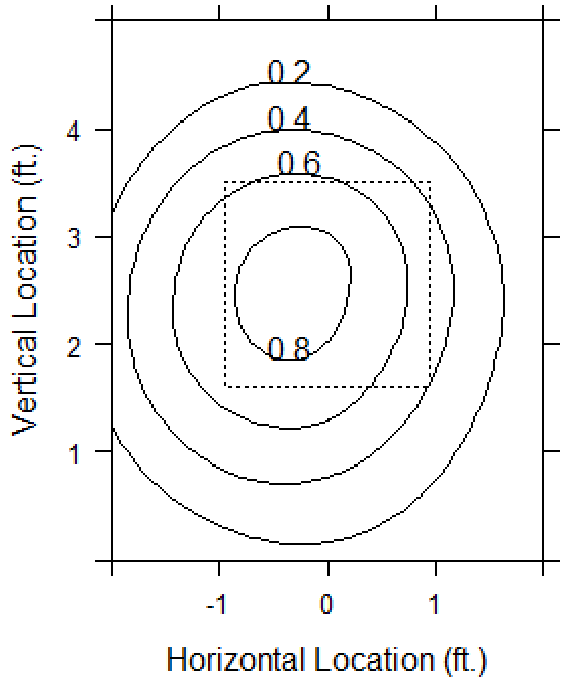
```
miggy.loess =loess(swung ~px +pz, data=cabrera,
control=loess.control(surface="direct"))
```

When we finally graph these probabilities, we want to make sure that we include enough area to capture every pitch. Thus, **pred.area** creates a space that is 2 feet wide and 6 feet tall. The new variable **fit** contains the predicted likelihood of Cabrera's swing.

```
pred.area =expand.grid(px=seq(-2, 2, 0.1), pz=seq(0, 6, 0.1))
pred.area$fit =c(predict(miggy.loess, pred.area))
```

The function **contourplot** overlays these swing likelihoods onto the dimensions previously discussed as well as the established MLB strikezone.

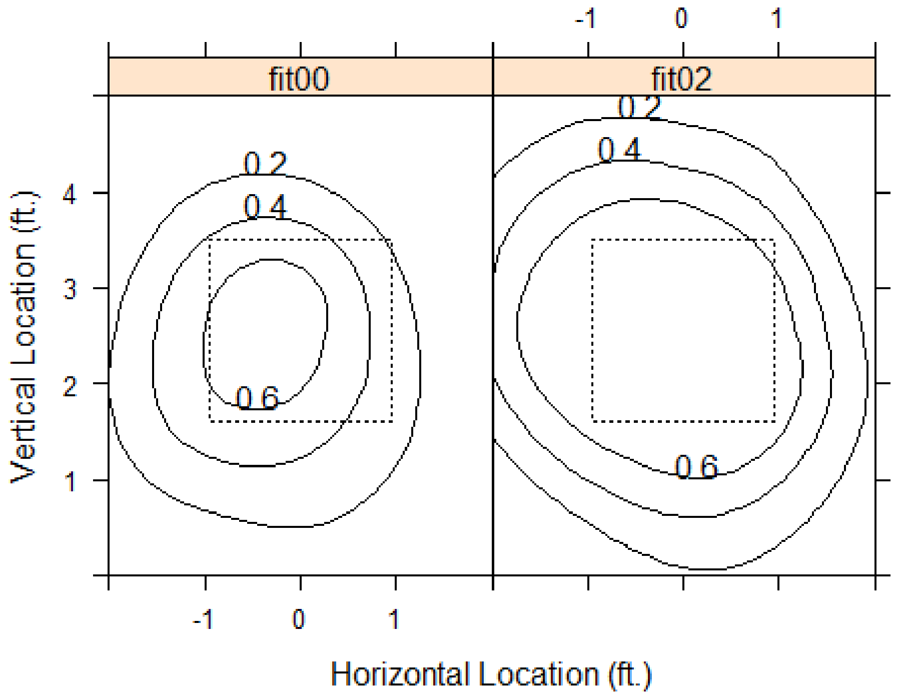
```
topKzone =3.5
botKzone =1.6
inKzone =-0.95
outKzone =0.95

contourplot(fit ~px *pz, data=pred.area,
at=c(.2, .4, .6, .8),
aspect="iso",
xlim=c(-2, 2), ylim=c(0, 5),
xlab="Horizontal Location (ft.)",
ylab="Vertical Location (ft.)",
panel=function(...){
panel.contourplot(...)
panel.rect(inKzone, botKzone, outKzone, topKzone,
border="black", lty="dotted")
            })
```



As we can see, the propensity to swing decreases as the pitch gets further from the strike zone. For reference, the contour plots are in the perspective of the catcher.

Going back to our originial interest of how pitch count affects Cabrera's propensity to swing, we must first decide in which pitch counts we are interested. Let's choose a 0-0 count and an 0-2 count.

```
cabrera$bscount =paste(cabrera$balls, cabrera$strikes, sep="-")

miggy00 =subset(cabrera, bscount == "0-0")
miggy00loess =loess(swung ~px +pz, data=miggy00, control=
loess.control(surface="direct"))
pred.area$fit00 =c(predict(miggy00loess, pred.area))

miggy02 =subset(cabrera, bscount == "0-2")
miggy02loess =loess(swung ~px +pz, data=miggy02, control=
loess.control(surface="direct"))
pred.area$fit02 =c(predict(miggy02loess, pred.area))

contourplot(fit00 +fit02 ~px *pz, data=pred.area,
at=c(.2, .4, .6),
aspect="iso",
xlim=c(-2, 2), ylim=c(0, 5),
xlab="Horizontal Location (ft.)",
ylab="Vertical Location (ft.)",
panel=function(...){
panel.contourplot(...)
panel.rect(inKzone, botKzone, outKzone, topKzone,
border="black", lty="dotted")
              })
```



At a 0-0 count, Miguel Cabrera tends to only swing when the pitch is in or near the strike zone, but when the count is 0-2, he fears being called out on strikes, thus he expands his own strike zone and swings more freely even if the ball is not in the strike zone.

Do Umpires' Strike Zones Change Based on Pitch Count

We just looked at how different pitch counts can affect the way a batter approaches the plate. Now let's look at how pitch count affects the way an umpire calls a pitch, whether it be a ball or a strike.

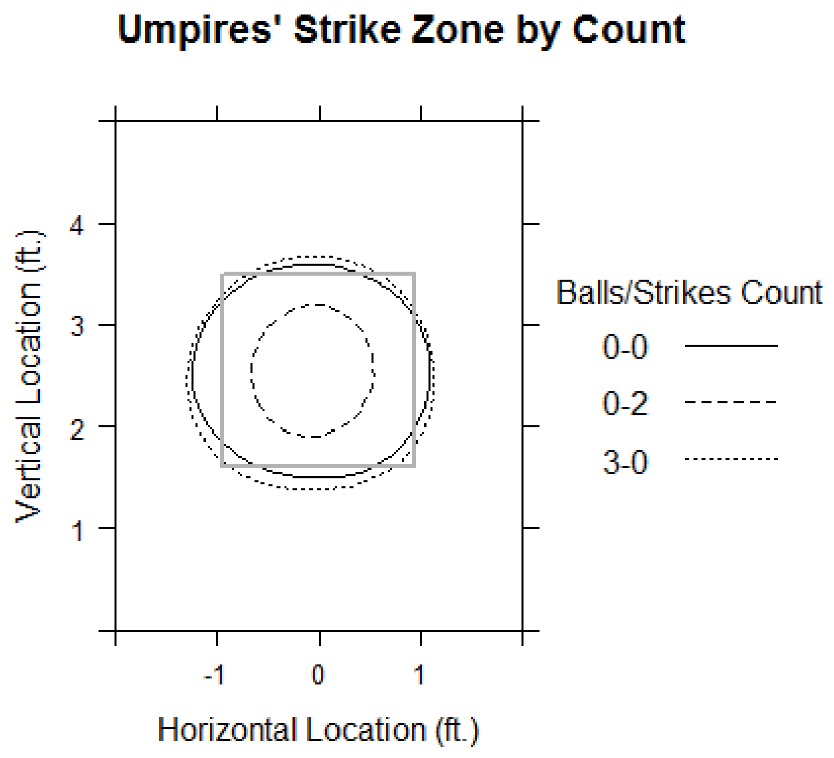
Just from experience I have noticed that when a batter has a 3-0 count, the next pitch is almost always a strike. Likewise, when a batter has a 0-2 count, the next pitch is almost always a ball. Are these pitches actually strikes and balls respectively, or does the count influence how an umpire calls that pitch?

By doing the same process we did for Miguel Cabrera's data, we can see how the propensity for an umpire to call a strike changes based on pitch location and pitch count.

```
umpiresRHB =subset(umpires, batter_hand == "R")
```

```
ump00 =subset(umpiresRHB, balls ==0&strikes ==0)
ump00samp =ump00[sample(1:nrow(ump00), 3000),]
ump00.loess =loess(called_strike ~px +pz, data=ump00samp,
control=loess.control(surface="direct"))

ump30 =subset(umpiresRHB, balls ==3&strikes ==0)
ump30samp =ump30[sample(1:nrow(ump30), 3000),]
ump30.loess =loess(called_strike ~px +pz, data=ump30samp,
control=loess.control(surface="direct"))

ump02 =subset(umpiresRHB, balls ==0&strikes ==2)
ump02samp =ump02[sample(1:nrow(ump02), 3000),]
ump02.loess =loess(called_strike ~px +pz, data=ump02samp,
control=loess.control(surface="direct"))

ump00contour =contourLines(x=seq(-2, 2, 0.1),
y=seq(0, 6, 0.1),
z=predict(ump00.loess, pred.area),
levels=c(0.5))
ump00df =as.data.frame(ump00contour)
ump00df$bscount = "0-0"

ump30contour =contourLines(x=seq(-2, 2, 0.1),
y=seq(0, 6, 0.1),
z=predict(ump30.loess, pred.area),
levels=c(0.5))
ump30df =as.data.frame(ump30contour)
ump30df$bscount = "3-0"

ump02contour =contourLines(x=seq(-2, 2, 0.1),
y=seq(0, 6, 0.1),
z=predict(ump02.loess, pred.area),
levels=c(0.5))
ump02df =as.data.frame(ump02contour)
ump02df$bscount = "0-2"

umpireContours =rbind(ump00df, ump02df, ump30df)

trellis.par.set(theme=canonical.theme(color=F))

myKey =list(lines=T
             , points=F
             , space="right"
             , title="Balls/Strikes Count"
             , cex.title=1
             , padding=4)
```

```
xyplot(y ~x, data=umpireContours
       , groups=bscount
       , type="l", aspect="iso"
       , col="black"
       , xlim=c(-2, 2), ylim=c(0, 5)
       , xlab="Horizontal Location (ft.)"
       , ylab="Vertical Location (ft.)"
       , main ="Umpires' Strike Zone by Count"
       , auto.key=myKey
       , panel=function(...){
panel.xyplot(...)
panel.rect(inKzone, botKzone, outKzone, topKzone,
border="grey70", lwd=2)
       })
```



The circles in the graph represent the average area where umpires call strikes for each respective pitch count. As I had predicted, umpires are less likely to call strikes when the count is 0-2, and they are more likely to call strikes when the count is 3-0. When it is a fresh count, 0-0, umpires tend to call strikes pretty near to where the actual strike zone is.

Chapter 7 was one of my personal favorites because it introduced new concepts, built off of old ones, and really gave context to the different analyses it had been going through thus far.

Chapter 8: Career Trajectories

While it is interesting and informative to analyze a player's statistics over the course of a season, looking at the rise and fall of a player's hitting, fielding, or pitching statistics over the course of their career can really describe the type of player they were. Usually, players peak in their 20s and then steadily decline. We can use R to plot a past player's career trajectory or predict a current player's trajectory.

Let's start off by looking at one of the greatest hitters of all time.

Mickey Mantle's Batting Trajectory

Before we can plot Mickey Mantle's trajectory, we must first know what age he was in each of the seasons he played. The function **get.birthyear** calculates the year a player was born which can then be subtracted from the year id of the season in order to calculate the age.

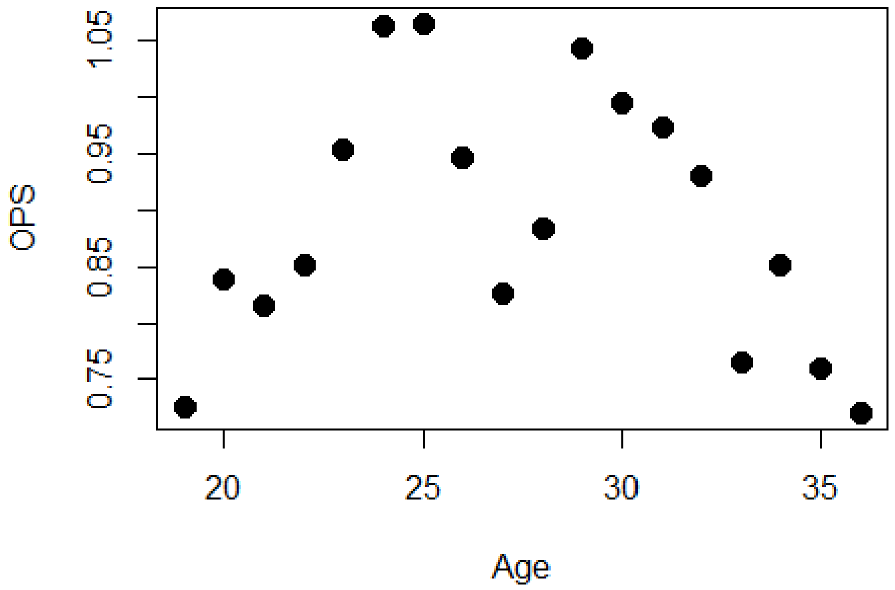
```
mantle.info =subset(Master, nameFirst == "Mickey"&nameLast == "Mantle"
)
mantle.id =as.character(mantle.info$playerID)

get.birthyear =function(player.id){
  playerline =subset(Master, playerID ==player.id)
  birthyear =playerline$birthYear
  birthmonth =playerline$birthMonth
ifelse(birthmonth >=7, birthyear +1, birthyear)
}

get.birthyear(mantle.id)

## [1] 1932
```

The function **get.stats** takes in a player's id and outputs different batting statistics as a data frame. In this case, we are interested in age, slugging (**SLG**), on-base percentage (**OBP**), and on-base plus slugging (**OPS**). **SLG** is the total number of bases (single is one base, double is two bases, etc.) divided by the total number of at-bats, **OBP** is the total number of times a player has reached base divided by the total number of plate appearances, and **OPS** is slugging plus on-base percentage.

```
get.stats =function(player.id){
  d =subset(Batting, playerID==player.id)
  byear =get.birthyear(player.id)
  d$Age =d$yearID -byear
  d$SLG =with(d, (H -X2B -X3B -HR +
2*X2B +3*X3B +4*HR) /AB)
  d$OBP =with(d, (H +BB) /(H +AB +BB +SF))
  d$OPS =with(d, SLG +OBP)
  d
}

Mantle =get.stats(mantle.id)
```

To see how Mantle's OPS changes over time we can plot it against age.

```
with(Mantle, plot(Age, OPS, cex=1.5, pch=19))
```



Here, we can see a general up-and-down relationship, with OPS peaking at around 25 years old.

In order to correctly understand and summarize Mantle's career batting trajectory, we must fit the curve as best as possible. Quadratic functions do a good job of fitting curves, so the function we will use is:

A + B(Age - 30) + C$(\mathbf{Age} - \mathbf{30})^2$

The function **fit.model** fits the quadratic curve to a player's batting data, which we can then apply to Mantle.

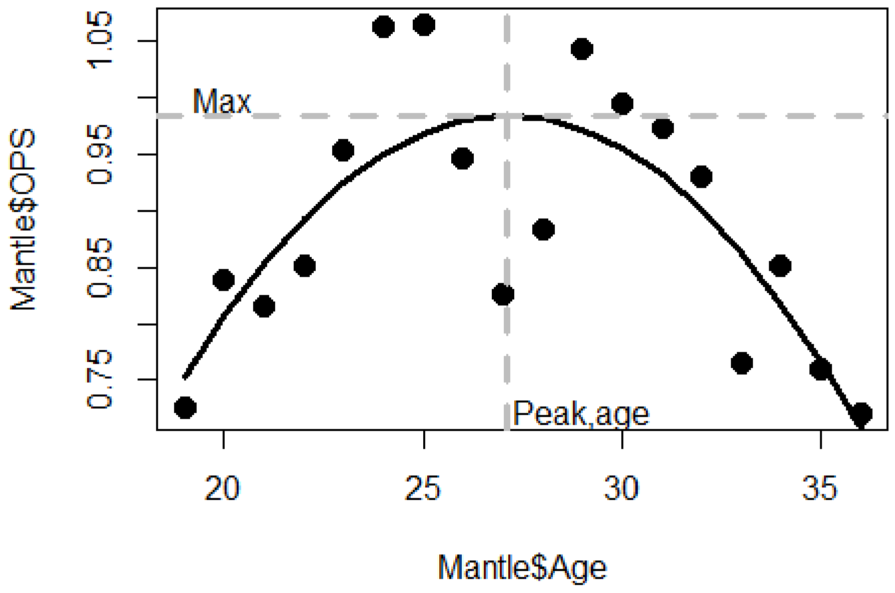
```
fit.model =function(d){
  fit =lm(OPS ~I(Age -30) +I((Age -30)^2), data=d)
  b =coef(fit)
  Age.max =30 -b[2] /b[3] /2
  Max =b[1] -b[2] ^2 /b[3] /4
list(fit=fit, Age.max=Age.max, Max=Max)
}

F2 =fit.model(Mantle)
coef(F2$fit)
```

```
##     (Intercept)     I(Age - 30) I((Age - 30)^2)
##     0.955433417    -0.020289562    -0.003520738
```

```
c(F2$Age.max, F2$Max)
```

```
## I(Age - 30) (Intercept)
##   27.118564    0.984665
```

```
plot(Mantle$Age, Mantle$OPS, cex=1.5, pch=19)
lines(Mantle$Age, predict(F2$fit, Age=Mantle$Age), lwd=3)
abline(v=F2$Age.max, lwd=3, lty=2, col="grey")
abline(h=F2$Max, lwd=3, lty=2, col="grey")
text(29, 0.72, "Peak,age", cex=1)
text(20, 1, "Max", cex=1)
```



This curve helps us pinpoint the age in which Mantle peaked. Earlier, I guessed he peaked arround 25, but in actuality, the value is closer to 27.

Not only can we look at the trajectory for one player, but we can also compare trajectories across many players. For instance, the catching position is considered to be associated with good hitting, so we can use trajectories to compare multiple catchers. Thus, comparing trajectories within fielding position is a great way to look at the careers of like players.

Comparing Trajectories Within Fielding Positions

The function **find.position** takes in a player's id, totals the number of games played at every possible position, and returns the position where the most games were played. **find.position** is applied to every players in the data frame and then merged with the Batting data frame that contains data for players who have had over 2000 at-bats.

```
find.position =function(p){
  positions =c("OF", "1B", "2B", "SS", "3B", "C", "P", "DH")
  d =subset(Fielding, playerID ==p)
  count.games =function(po)
sum(subset(d, POS ==po)$G)
  FLD =sapply(positions, count.games)
  positions[FLD ==max(FLD)][1]
}

PLAYER =as.character(unique(Batting.2000$playerID))

POSITIONS =sapply(PLAYER, find.position)

Fielding.2000 =data.frame(playerID=names(POSITIONS), POS=POSITIONS)

Batting.2000 =merge(Batting.2000, Fielding.2000)

head(Batting.2000)

##    playerID yearID stint teamID lgID   G  AB   R   H X2B X3B HR RBI
 SB CS
## 1 aaronha01   1965     1    ML1   NL 150 570 109 181  40   1 32  89
 24  4
## 2 aaronha01   1968     1    ATL   NL 160 606  84 174  33   4 29  86
 28  5
## 3 aaronha01   1963     1    ML1   NL 161 631 121 201  29   4 44 130
 31  5
## 4 aaronha01   1954     1    ML1   NL 122 468  58 131  27   6 13  69
  2  2
## 5 aaronha01   1967     1    ATL   NL 155 600 113 184  37   3 39 109
 17  6
## 6 aaronha01   1972     1    ATL   NL 129 449  75 119  10   0 34  77
  4  0
##   BB SO IBB HBP SH SF GIDP Career.AB POS
## 1 60 81  10   1  0  8   15     12364  OF
## 2 64 62  23   1  0  5   21     12364  OF
## 3 78 94  18   0  0  5   11     12364  OF
## 4 28 39  NA   3  6  4   13     12364  OF
## 5 63 97  19   0  0  6   11     12364  OF
## 6 92 55  15   1  0  2   17     12364  OF
```

Here, we can see that the Position Hank Aaron played is in the outfield.

As previously stated, we are not interested in a player's season statistics, but rather, statistics for their entire career. To do so, we want to look at their career averages. First, we must total all relevant statistics, including runs, hits, home runs, strikeouts, etc. This process is done with the function **C.totals**. Then, we can compute the averages by dividing their career totals by total at-bats.

```
C.totals =ddply(Batting.2000, .(playerID),
                 summarize,
C.G=sum(G, na.rm=T),
C.AB=sum(AB, na.rm=T),
C.R=sum(R, na.rm=T),
C.H=sum(H, na.rm=T),
C.2B=sum(X2B, na.rm=T),
C.3B=sum(X3B, na.rm=T),
C.HR=sum(HR, na.rm=T),
C.RBI=sum(RBI, na.rm=T),
C.BB=sum(BB, na.rm=T),
C.SO=sum(SO, na.rm=T),
C.SB=sum(SB, na.rm=T))

C.totals$C.AVG =with(C.totals, C.H /C.AB)
C.totals$C.SLG =with(C.totals, (C.H -C.2B -C.3B -C.HR +2*C.2B +
3*C.3B +4*C.HR) /C.AB)

C.totals =merge(C.totals, Fielding.2000)
```

Each position has an associated value so we define the variable **Value.POS** as the value corresponding to the player's position.

```
C.totals$Value.POS =with(C.totals,
ifelse(POS == "C", 240,
ifelse(POS == "SS", 168,
ifelse(POS == "2B", 132,
ifelse(POS == "3B", 84,
ifelse(POS == "OF", 48,
ifelse(POS == "1B", 12, 0)))))))
```

Now that we have all of the positions defined and career statistics calculated, we must now discuss how to quantify similarities between players.

Bill James, a baseball writer and statistician, came up with a way to compare hitters.

Starting at 1000 points, one point is then subtracted for each of the following differences:

1. 20 games played
2. 75 at-bats
3. 10 runs scored
4. 15 hits
5. 5 doubles
6. 4 triples

7. 2 home runs
8. 10 RBI (runs batted in)
9. 25 walks
10. 150 strikeouts
11. 20 stolen bases
12. 0.001 in BA (batting average)
13. 0.002 is slugging percentage

Thus, the closer the score is to 1000, the more similar two players are. Intuitively, comparing a hitter with themselves would result in a score of 1000.

The function **similar** computes these similarity scores by taking into account all of the differences mentioned above. Its inputs are a player id and the number of similar players we'd like to see. However, because the player that is most similar to a player is themselves, we actually need to put in a value equivalent to (players + 1). More specifically, if we want to see 5 similar players, we must input the value 6.

```
similar =function(p, number=10){
  P =subset(C.totals, playerID ==p)
  C.totals$SS =with(C.totals,
1000 -
floor(abs(C.G -P$C.G) /20) -
floor(abs(C.AB -P$C.AB) /75) -
floor(abs(C.R -P$C.R) /10) -
floor(abs(C.H -P$C.H) /15) -
floor(abs(C.2B -P$C.2B) /5) -
floor(abs(C.3B -P$C.3B) /4) -
floor(abs(C.HR -P$C.HR) /2) -
floor(abs(C.RBI -P$C.RBI) /10) -
floor(abs(C.BB -P$C.BB) /25) -
floor(abs(C.SO -P$C.SO) /150) -
floor(abs(C.SB -P$C.SB) /20) -
floor(abs(C.AVG -P$C.AVG) /0.001) -
floor(abs(C.SLG -P$C.SLG) /0.002) -
abs(Value.POS -P$Value.POS))
C.totals =C.totals[order(C.totals$SS, decreasing=T), ]
C.totals[1:number, ]
}
```

To illustrate the use of the function, let's look at the 5 players who are most similar to Mickey Mantle.

```
similar(mantle.id, 6)

##        playerID  C.G C.AB  C.R  C.H C.2B C.3B C.HR C.RBI C.BB C.SO C
.SB
```

```
## 1337 mantlmi01 2401 8102 1677 2415  344   72  536  1509 1733 1710
153
## 2112 thomafr04 2322 8199 1494 2468  495   12  521  1704 1667 1397
 32
## 1363 matheed01 2391 8537 1509 2315  354   72  512  1453 1444 1487
 68
## 1895 schmimi01 2404 8352 1506 2234  408   59  548  1595 1507 1883
174
## 1935 sheffga01 2576 9217 1636 2689  467   27  509  1676 1475 1171
253
## 1994  sosasa01 2354 8813 1475 2408  379   45  609  1667  929 2306
234
##          C.AVG     C.SLG POS Value.POS   SS
## 1337 0.2980745 0.5567761  OF        48 1000
## 2112 0.3010123 0.5549457  1B        12  856
## 1363 0.2711725 0.5094295  3B        84  853
## 1895 0.2674808 0.5272989  3B        84  848
## 1935 0.2917435 0.5139416  OF        48  847
## 1994 0.2732327 0.5337569  OF        48  831
```

Mickey Mantle was one of the best hitters in baseball, so it makes sense that we see the similar players were also great hitters in the likes of Frank Thomas, Eddie Matthews, Mike Schmidt, Gary Sheffield, and Sammy Sosa.

Using the **similar** function is great to see how similar certain players were when using the method defined by Bill James. But did one player do better earlier in his career than another? Did one player have a bigger drop off towards the end of his career? By plotting the trajectories, we can visualize just how similar these players really are.

Fitting and Plotting Trajectories for Similar Players

The **Batting** dataset from Lahman's database is great because it has batting information for every player. However, the only problem is that if a player played for multiple teams during one season, there are multiple data lines for them rather than one. The function **collapse.stint** not only calculates career batting statistics, but also summarizes this information into one line per season per player.

```
collapse.stint =function(d){
  G =sum(d$G); AB =sum(d$AB); R =sum(d$R)
  H =sum(d$H); X2B =sum(d$X2B); X3B =sum(d$X3B)
  HR =sum(d$HR); RBI =sum(d$RBI); SB =sum(d$SB)
  CS =sum(d$CS); BB =sum(d$BB); SH =sum(d$SH)
  SF =sum(d$SF); HBP =sum(d$HBP)
  SLG =(H -X2B -X3B -HR +2*X2B +3*X3B +4*HR) /AB
  OBP =(H +BB +HBP) /(AB +BB +HBP +SF)
  OPS =SLG +OBP
data.frame(G=G, AB=AB, R=R, H=H, X2B=X2B, X3B=X3B, HR=HR, RBI=RBI, SB=
SB,
CS=CS, BB=BB, HBP=HBP, SH=SH, SF=SF, SLG=SLG, OBP=OBP, OPS=OPS,
Career.AB=d$Career.AB[1], POS=d$POS[1])
}

Batting.2000 =ddply(Batting.2000, .(playerID, yearID), collapse.stint)
```

It is also important to have the corresponding age of the player during any given season so that we can accurately compare trajectories. This is accomplished by subtracting the players birth year from the year id of the season.

```
player.list =as.character(unique(Batting.2000$playerID))
birthyears =sapply(player.list, get.birthyear)
Batting.2000 =merge(Batting.2000, data.frame(playerID=player.list,
Birthyear=birthyears))
Batting.2000$Age =with(Batting.2000, yearID -Birthyear)

Batting.2000 =Batting.2000[complete.cases(Batting.2000$Age), ]
```

By expanding on the function **fit.model**, **fit.trajectory** fits the career OPS for any player as best as possible.

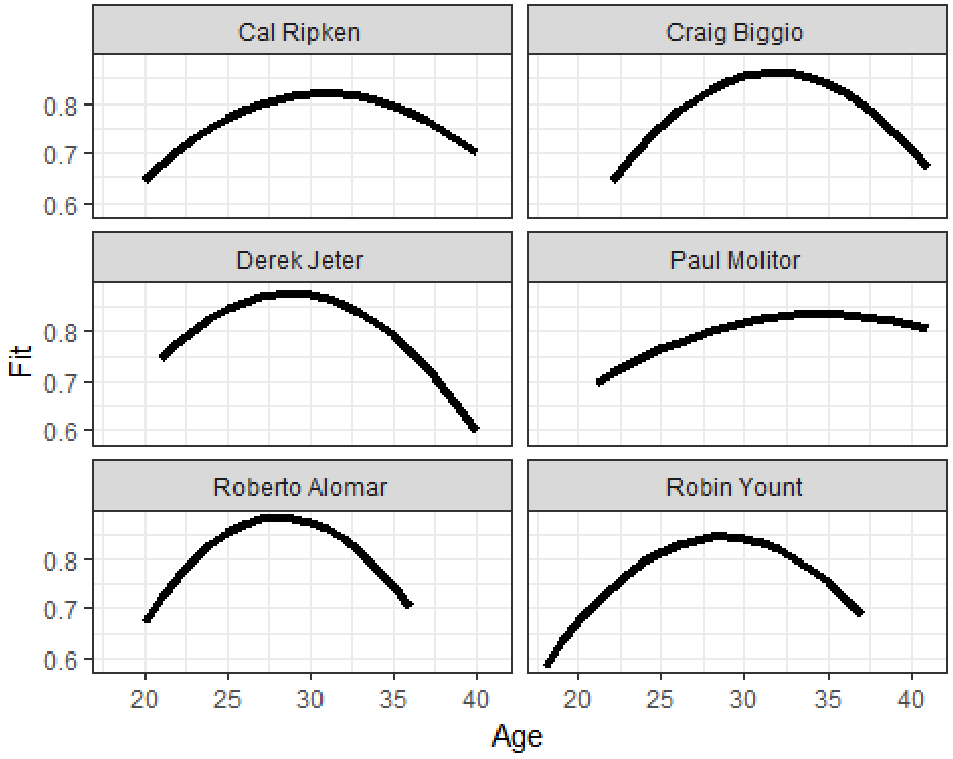
```
fit.traj =function(d){
  fit =lm(OPS~I(Age -30) +I((Age -30)^2), data=d)
data.frame(Age=d$Age, Fit =predict(fit, Age=d$Age))
}
```

Finally, the function **plot.trajectories** graphs the trajectories for a given player as well as the indicated number of similar players.

```
plot.trajectories =function(first, last, n.similar=5, ncol){
require(plyr)
require(ggplot2)
  get.name =function(playerid){
    d1 =subset(Master, playerID ==playerid)
with(d1, paste(nameFirst, nameLast))
  }
  player.id =subset(Master, nameFirst ==first &nameLast ==last)$player
ID
  player.id =as.character(player.id)
  player.list =as.character(similar(player.id, n.similar)$playerID)
  Batting.new =subset(Batting.2000, playerID %in%player.list)

  F2 =ddply(Batting.new, .(playerID), fit.traj)
  F2 =merge(F2, data.frame(playerID=player.list,
Name=sapply(as.character(player.list), get.name)))
print(ggplot(F2, aes(Age, Fit)) +geom_line(size=1.5) +
facet_wrap(~Name, ncol=ncol) +theme_bw())
}
```

My all-time favorite baseball player is New York Yankee great, Derek Jeter. Let's see the trajectory of his OPS compared to 5 similar players. *Note, the final input of 2 just tells the function that I want to see the 6 trajectories divided into 2 columns.

```
plot.trajectories("Derek", "Jeter", 6, 2)
```



Derek Jeter's OPS peak was higher than the other similar players, aside from Roberto Alomar. However, his decline was sharper and his OPS in his final season was lower than his OPS when he first started, unlike all other similar players.

Plotting similar trajectories is a great way to show that even if players have very similar career statistics, their season statistics can differ drastically.

Chapter 8 is a very relevant chapter to baseball fans because we constantly hear baseball analysts and announcers comparing today's hot hitter to the hot hitter of yesterday. I remember when Manny Machado first emerged onto the scene, people were constantly saying he was going to be the next Alex Rodriguez. Or Ivan Rodriguez is the next Johnny Bench.

This chapter now gives me the ability to quantify and plot these supposed similarities for myself.

Chapter 9: Simulation

Because every baseball season is made up of 162 games, every typical baseball game is made up of 9 innings (sometimes 8-and-a-half), and each inning is divided into 2 half-innings, representing this sport with probability models is fairly simple. The most common way to simulate a baseball game is with Markov chains. Markov chains, in this case, describe the movements between states until three outs are accomplished. States are defined the same way here as they were in Chapter 5 and Chapter 7, a description of the number of outs and runners on base.

Simulating the Markov Chain

Defining the states and probabiliy matrices will be the same for this section as Chapter 5. Therefore, only the new functions will be introduced here.

Because we are using Markov chains, we need to compute the frequencies for all possible transitions between old states and new states. From there, we can convert this frequency matrix to a probability matrix.

```
T.matrix =with(data2011C, table(STATE, NEW.STATE))

P.matrix =prop.table(T.matrix, 1)

P.matrix =rbind(P.matrix, c(rep(0, 24), 1))
```

Before we can run the simulation, we must first know the number of runs that can be scored in all possible state-to-state transitions. For instance, say we start off with runners on second and third with one out and the next play results in a state of a runner on third with two outs. Thus, one run was scored. We can set this up as the equation:

RUNS = (initial # of runners + initial # of outs + 1) - (new # of runners + new # of outs)

Going back to our example, RUNS = (2 + 1 + 1) - (1 + 2) = 1

The function **count.runners.outs** takes **state** as an input and returns the total number of runners and outs. The function is then applied to all possible states and then the RUNS calculation is performed and stored in the variable R.

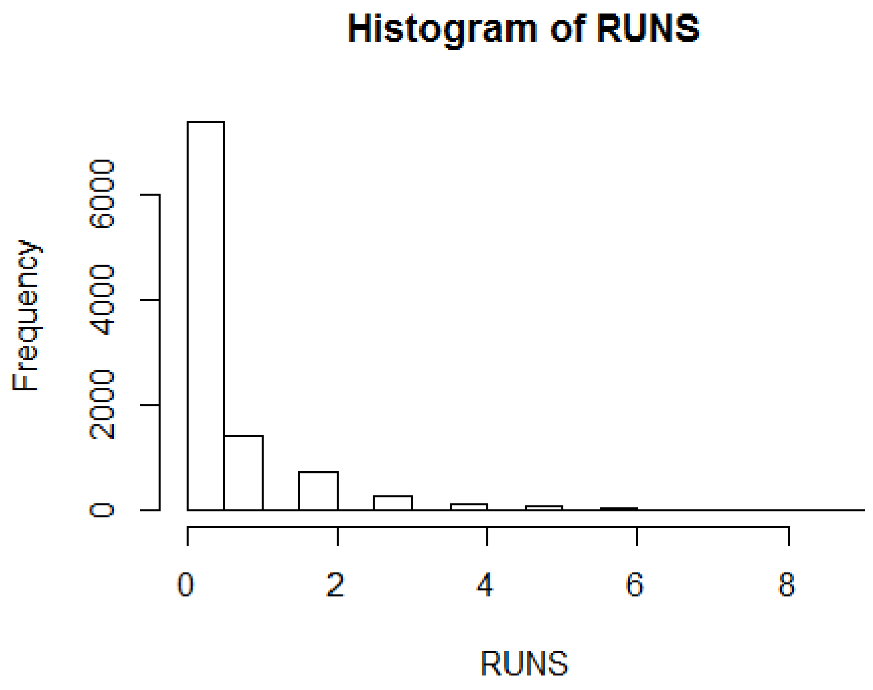
```
count.runners.outs =function(s)
sum(as.numeric(strsplit(s,"")[[1]]), na.rm =TRUE)
runners.outs =sapply(dimnames(T.matrix)[[1]], count.runners.outs)[-25]

R =outer(runners.outs +1, runners.outs, FUN="-")
dimnames(R)[[1]] =dimnames(T.matrix)[[1]][-25]
dimnames(R)[[2]] =dimnames(T.matrix)[[1]][-25]
R =cbind(R, rep(0, 24))
```

We can now simulate a half inning. **simulate.half.inning** takes in the transition matrix, the run matrix, and the starting state, denoted by a number 1-24 because there are 24 possible states, and returns the number of runs scored in that half inning. We begin with **start=1** because every half inning begins in the starting state of **000 0**.

```
simulate.half.inning =function(P, R, start=1){
  s =start; path =NULL; runs =0
  while(s <25){
    s.new =sample(1:25, 1, prob = P[s, ])
    path =c(path, s.new)
    runs =runs +R[s, s.new]
    s =s.new
  }
  runs
}

RUNS =replicate(10000, simulate.half.inning(T.matrix, R))
table(RUNS)
```

```
## RUNS
##    0    1    2    3    4    5    6    7    8    9
## 7386 1411  717  269  120   61   18   13    2    3
```

```
hist(RUNS)
```



```
mean(RUNS)

## [1] 0.4679
```

The RUNS variable stores 10000 replications of **simulate.half.inning**. In these 10,000 simulations, we find that scoring one or less runs is very likely, while scoring greater than 2 runs is not very likely. In fact, the average runs scored per half inning is about 0.45.

It is intuitive that runs potential would be different depending on the starting state. We can write a function that calculates the run expectancy for each possible state by applying **simulate.half.inning** to each of the 24 states.

```
RUNS.j =function(j){
mean(replicate(10000, simulate.half.inning(T.matrix, R, j)))
}

Runs.Expectancy =sapply(1:24, RUNS.j)
Runs.Expectancy =t(round(matrix(Runs.Expectancy, 3, 8), 2))
dimnames(Runs.Expectancy)[[2]] =c("0 outs", "1 out", "2 outs")
dimnames(Runs.Expectancy)[[1]] =c("000", "001", "010", "011", "100", "
101",
"110", "111")

Runs.Expectancy

##     0 outs 1 out 2 outs
## 000   0.47  0.24   0.09
## 001   1.35  0.91   0.33
## 010   1.05  0.61   0.30
## 011   1.87  1.32   0.52
## 100   0.82  0.51   0.20
## 101   1.70  1.13   0.46
## 110   1.44  0.86   0.40
## 111   2.21  1.52   0.68
```

When there are more runners than outs, the runs expectancy tends to be greater than 1, and vice versa. These run expectancies are very similar to those we calculated in Chapter 5, with no difference being greater than 0.08.

Running simulations is a great way to get an understanding of run-scoring patterns. This chapter, while it does delve into more in-depth analyses, is a good introductory chapter for simulations. My biggest takeaway from this chapter is that because the game of baseball is so structured, one can simulate a portion as small as one half-inning, all the way up to simulating an entire season.

Chapter 10: Exploring Streaky Performances

One of the greatest batting accomplishments in the history of baseball came from one of the greatest hitters, Joe DiMaggio. DiMaggio's 56-game hitting streak in 1941 was unprecedented and has yet to be matched. Chapter 10 explored this type of streakiness, whether it be a hot streak or a cold one, a hitting streak or a hitting slump. There are two ways in which we can analyze streakiness: find the longest hitting slumps or hitting streaks for a player, or simulate a random pattern of his hits and outs to predict a streak.

Game Hitting Streaks

Let's explore all of the hitting streaks DiMaggio had during is record-breaking season in 1941. The function **streaks** totals up every consecutive game a batter had a hit until he reached a game where he did not get one. For example, if a batter's streak was 8, that means in that span, they hit in 8 straight games but did not get a hit in that 9th game.

```
## [1] 0.3567468

streaks =function(y){
  n =length(y)
  where =c(0, y, 0) ==0
  location.zeros =(0 :(n+1))[where]
  streak.lengths =diff(location.zeros) -1
  streak.lengths[streak.lengths >0]
}

streaks(joe$HIT)

##  [1]  8  3  2  1  3 56 16  4  2  4  7  1  5  2
```

Here, we can see that most of DiMaggio's hit streaks in 1941 did not exceed 10 games, save for his 56-game streak.

We can also explore how many and how long of batting slumps DiMaggio had by looking at the number of consecutive games he did not get a hit. Again, we can use **streaks** to help us find these numbers.

```
joe$NO.HIT =1 -joe$HIT

streaks(joe$NO.HIT)

##  [1] 3 1 2 3 2 1 2 2 3 3 1 1 1
```

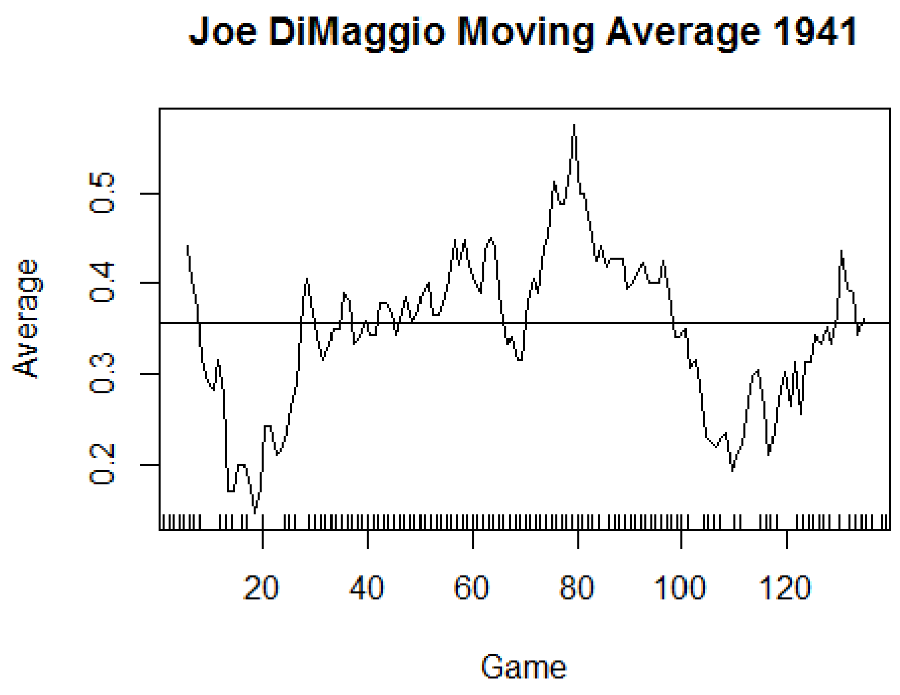
With 3 being the longest streak of no-hit games, DiMaggio rarely was in a batting slump.

Moving Batting Averages

We just analyzed hitting performances on a game-to-game basis, but we can also divide the season into smaller portions, calculate a player's batting average for that span of time, and see how it changes as the season goes on. In other words, we can compute moving batting averages.

The function **moving.average** takes total hits, total at-bats, and number of games as inputs, and returns the moving batting averages.

```
moving.average =function(H, AB, width){
require(TTR)
  game =1:length(H)
  P =data.frame(Game=SMA(game, width),
Average =SMA(H, width) /SMA(AB, width))
  P[complete.cases(P), ]
}

plot(moving.average(joe$H, joe$AB, 10), type="l", main="Joe DiMaggio M
oving Average 1941")

abline(h=sum(joe$H)/sum(joe$AB))
game.hits =(1:nrow(joe))[as.logical(joe$HIT)]
rug(game.hits)
```



Applying the **moving.average** function to DiMaggio's data, we can see that his moving averages were most consistently above his season average in the middle bulk of the season while they tapered off in the beginning and end of the season. It also makes sense that his highest averages came during his 56-game hitting streak.

Hitting Slumps for All Players

The previous functions only consider one player, but what if we want to see the hitting slumps for all players? The function **longest.ofer** calculates the longest hitting slump for a player. We can then apply this function to a data frame of all players.

```
longest.ofer <-function(batter, d){
  d.AB =subset(d, BAT_ID ==batter &AB_FL==TRUE)
  d.AB$HIT =ifelse(d.AB$H_FL >0, 1, 0)
  d.AB$DATE =substr(d.AB$GAME_ID, 4, 12)
  d.AB =d.AB[order(d.AB$DATE), ]
max(streaks(1 -d.AB$HIT))
}

A =aggregate(data2011$AB_FL, list(Player = data2011$BAT_ID), sum)
players.400 =A$Player[A$x >=400]
S =sapply(players.400, longest.ofer, data2011)
S =data.frame(Player=players.400, Streak=S)
rownames(S) =NULL

roster2011 <-read.csv("C:/Users/Claudia/Desktop/Baseball Data/roster20
11.csv")

S1 =merge(S, roster2011, by.x="Player", by.y="Player.ID")
S.ordered =S1[order(S1$Streak, decreasing=TRUE), ]
head(S.ordered)

##        Player Streak    X Last.Name First.Name Bats Pitches Team V7
## 80  ibanr001     35  941    Ibanez       Raul    L       R  PHI OF
## 6   aybae001     30    3     Aybar      Erick    B       R  ANA SS
## 113 mcgec001     27  726   McGehee      Casey    R       R  MIL 3B
## 122 olivm001     27 1095     Olivo     Miguel    R       R  SEA  C
## 152 ruizc001     26  958      Ruiz     Carlos    R       R  PHI  C
## 48  ellim001     25  896     Ellis       Mark    R       R  OAK SS
```

Raul Ibanez had the longest hitting slump in 2011. But was this just an anomaly or does Ibanez typically go many games without a hit?

We can address this question by running a simulation. A good measure of streakiness, or clumpiness, is the sum of squares of the gaps between successive hits.

For example, let's consider a player who bats 15 times with the outcomes:

0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1

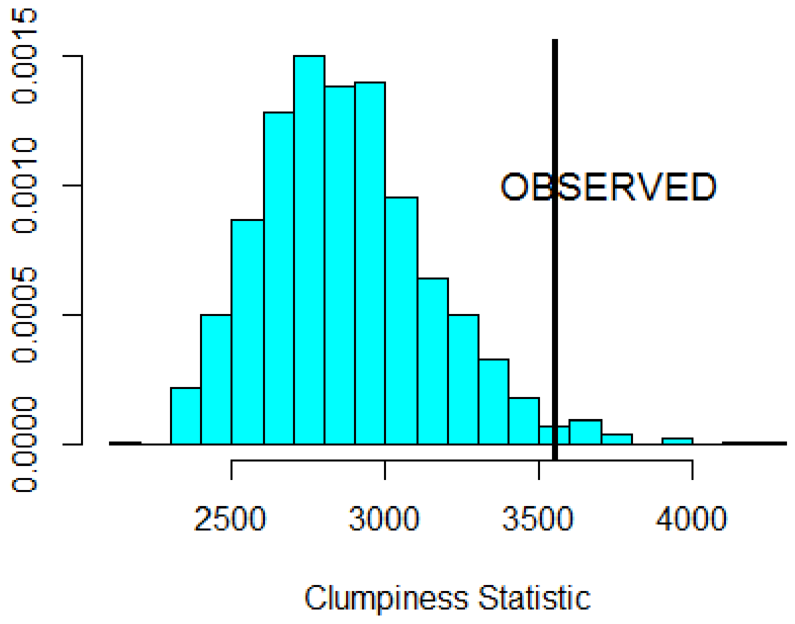
The gaps between hits are 2, 1, 3, and 2 which means that the sum of the squares of the gaps (**S**) is:

$S = 2^2 + 1^2 + 3^2 + 2^2 = 18$

Is a clumpiness statistic of 18 out of the ordinary? If we randomly arrange this player's 7 hits and 8 outs and calculate the clumpiness statistic say, 1000 times, and we find that 18 falls in the middle of the 1000 simulations, then we would not have evidence to believe that 18 is unexpected based on a random model. However, if it is in the right tail of the distribution of clumpiness statistics, then we have evidence to believe that the player is streaky.

By looking at all of Ibanez's hit outcomes, we can run 1000 replications of his clumpiness statistics to see if his 35-game hitting slump was unusual or not.

```
random.mix =function(y){
  clump.stat =function(sp) sum(sp^2)
  mixed =sample(y)
clump.stat(streaks(1 -mixed))
}

clump.test =function(playerid, data){
  player.AB =subset(data, BAT_ID ==playerid &AB_FL==TRUE)
  player.AB$HIT =ifelse(player.AB$H_FL >0, 1, 0)
  player.AB$DATE =substr(player.AB$GAME_ID, 4, 12)
  player.AB =player.AB[order(player.AB$DATE), ]
  ST =replicate(1000, random.mix(player.AB$HIT))
truehist(ST, xlab="Clumpiness Statistic")
  stat =sum((streaks(1 -player.AB$HIT))^2)
abline(v = stat, lwd=3)
text(stat *1.05, 0.0010, "OBSERVED", cex=1.2)
}

clump.test("ibanr001", data2011)
```



As we can see, Ibanez's 35-game hitting slump is in the right tail of the distribution of replications. Based on this, we can conclude that since this observation is not in the bulk of the data, the hitting slump was an anomaly.

This clump.test process can also be done to analyze hitting streaks, such as DiMaggio's 56-game streak.

I really enjoyed this chapter because streakiness is always present in baseball. From my own experience as a baseball fan, I would always rather my team have consistent hitters rather than streaky hitters. With the functions created in Chapter 10, I can now analyze if "streaks" that I observe throughout the season are in fact streaks. Not only am I gaining more R skills with this chapter, but I am gaining a better understanding of the game as well.