Baseball Value

Matthew Helbig

2019-09-28

Contents

Introduction

The goal of this project is to identify the attributes of the players with the most value and the least value, as well as predicting which players will have the most and least value in the future.

Getting started

The first step we'll take is to identify which datasets we'll be working with. The Lahman Database has a lot of good information that we'll need, namely Player Value data, salary, and dates of birth.

We'll download that from http://www.seanlahman.com/baseball-archive/statistics/

It's possible to download the database as a .CSV file, and you could technically do everything needed for this project without using SQL, but it's a good practice to get used to working with databases, especially since a lot of the datasets you'd see in the real world will be bigger than the one we're currently working with.

Sean Lahman has posted the entire .sql file on his website, so importing it into MySQL is as simple as hitting 'Data Import' and loading that database into its own schema.

Once the Lahman database is all settled in its own schema, we want to start to work with that data. There are packages that let you run SQL commands inside R, and there is definitely some value in not having to hop back and forth between R and your SQL environments. However, I'm more comfortable working in MySQL directly, so we won't worry too much about those R-SQL packages (namely RMySQL).

Setting things up in SQL

The three tables that we're most interested in right now are "Batting," "Pitching," and "Salaries." The database has a ton of information that's interesting, but ultimately not super useful for our experiment. For our project, we're going to use data from 2006 to 2010 to find our "model" player, and then use data from 2011 to 2016 to test if our hypothesis is true.

"Subsetting" our Batting data

We'll need to create a new table in SQL based on our criteria of only needing 2006 and later (this database ends in 2016).

```
CREATE TABLE Batting_post_2006
LIKE Batting
```

Now we've created a table with the same containers as our Batting table, and now we need to load the data into those containers.

```
INSERT INTO Batting_post_2006
SELECT *
FROM Batting;
```

Now we're going to trim our data from our new table in order to limit ourselves to only 2006 to 2016. This is where SQL is beneficial, since our original dataset of "Batting" is complete safe and untouched.

```
DELETE FROM Batting_post_2006
WHERE yearID < 2006
```

We're going to do the same thing with our Salaries table, and eventually our Pitching table. We won't do the Pitching table just yet, but we can follow these same steps in order to set that table up for export to R. For now, we'll work on the Salaries table:

```
CREATE TABLE Salaries_post_2006
LIKE Salaries
```

```
INSERT INTO Salaries_post_2006
SELECT *
FROM Salaries;
```

```
DELETE FROM Salaries_post_2006
WHERE yearID < 2006
```

Here, we created a new table to put our data in, moved all of our data from the Salaries table into it, and then limited ourselves to data after 2016. Once again, the original Salaries table is untouched.

Joining Birth Year and Salary

One important step that hasn't happened yet is getting birth year statistics from the Master table in the database. This will allow us to create an Age variable in R later, which will be a potentially key attribute for determining who is overvalued and undervalued. In order to add birth year to our Salary data, we'll need to join the two tables.

First, we create a new container for our combined data:

```
CREATE TABLE Player_Salary_Info (
    AutoNum int NOT NULL AUTO_INCREMENT,
    playerID varchar(255),
    birthyear int,
    yearID int,
    salary int,
    PRIMARY KEY (AutoNum)
);
```

The AutoNum variable is important here, because we need something as the primary key, and if we were to use playerID then we'd have duplicate values between Salary and Master (since the playerIDs are the same between the two tables).

Next, we'll populate our new table with data:

```
INSERT INTO Player_Salary_Info
SELECT 0, Salaries_post_2006.playerID, Master.birthYear,
Salaries_post_2006.yearID, Salaries_post_2006.salary
FROM Salaries_post_2006
INNER JOIN Master ON Salaries_post_2006.playerID = Master.playerID
```

So now we know that we've managed to get playerID, birthyear, yearID, and salary in the same spot. Our next task will be combining this information with our Batting table, at which point we'll be able to move this new table over to R.

Joining Birth Year, Salary, and Batting

One hiccup we face is that several of our variables for some reason are indicated as strings despite being numeric values. I've encountered issues with this before, where a value of 0 would return an empty string (like " “) instead of actually being 0. To solve this, we can change the string columns into integer columns using the following lines of code:

```
ALTER TABLE example
ADD newCol int;

UPDATE example SET newCol = CAST(column_name AS UNSIGNED);

ALTER TABLE example
DROP COLUMN column_name;

ALTER TABLE example
CHANGE COLUMN newCol column_name int;
```

Now that our data is in a format we can work with that won't give us (hopefully) any errors, we can create and populate our Batting table with salary added. We create the parameters for the new table:

```
CREATE TABLE Batting_Salary (
    AutoNum int NOT NULL AUTO_INCREMENT,
    playerID varchar(255),
    birthyear int,
    yearID int,
    salary int,
    stint int,
    teamID varchar(255),
    lgID varchar(255),
    G int,
    AB int,
    R int,
    H int,
    2B int,
    3B int,
    HR int,
    RBI int,
    SB int,
    CS int,
    BB int,
    SO int,
    GIDP int,
    SF int,
    SH int,
    HBP int,
    IBB int,
    PRIMARY KEY (AutoNum)
);
```

Now we populate our new table:

```
INSERT INTO Batting_Salary
SELECT 0, Batting_post_2006.playerID, Player_Salary_Info.birthyear,
Player_Salary_Info.yearID, Player_Salary_Info.salary, Batting_post_2006.stint,
Batting_post_2006.teamID, Batting_post_2006.lgID, Batting_post_2006.G,
Batting_post_2006.AB, Batting_post_2006.R, Batting_post_2006.H,
Batting_post_2006.2B, Batting_post_2006.3B, Batting_post_2006.HR,
Batting_post_2006.RBI, Batting_post_2006.SB, Batting_post_2006.CS,
Batting_post_2006.BB, Batting_post_2006.SO, Batting_post_2006.GIDP,
Batting_post_2006.SF, Batting_post_2006.SH, Batting_post_2006.HBP,
Batting_post_2006.IBB
FROM Batting_post_2006
INNER JOIN Player_Salary_Info ON Player_Salary_Info.playerID = Batting_post_2006.playerID
AND Player_Salary_Info.yearID = Batting_post_2006.yearID
```

So now we've successfully combined our tables with salary information, birth year information, and batting statistics into one table called Batting_Salary. In the process, we also already shaped the data so that it's only dealing with the years we're concerned with (2006 to 2016). We're just one step away from having our Batter data ready for analysis in R. That step is getting our player value data from Baseball-Reference.

Getting player value data from Baseball-Reference

Our first step is to head to https://www.baseball-reference.com/data/, where they keep a daily log of player value data for every player in baseball history. Player value is measured in a lot of different ways, and a constant source of contention in the baseball community is which metrics provide the most accurate measurement of a player's true value.

The player value metric we'll be focusing on the most is Wins Above Replacement (abbreviated as WAR), which measures how many wins a player provided to his team compared to a random player that you could find in the minor leagues. As a rough translation, consider WAR to be the "stock price" of MLB players, as it gives a simple and quick approximation of the value of a player (or stock).

The information we're looking for is all the way at the bottom of the page, under "war_daily_bat.txt." We'll download that, and convert it to .csv using a spreadsheet program (I'm using Open Office). Just follow the steps for converting everything, and we'll save it to our active folder.

Now we need to load this data into SQL. However, if we look at this data in R:

```
Daily_bat <- read.csv("war_daily_bat.csv")
names(Daily_bat)
```

```
##  [1] "name_common"        "age"                "mlb_ID"
##  [4] "player_ID"          "year_ID"            "team_ID"
##  [7] "stint_ID"           "lg_ID"              "PA"
## [10] "G"                  "Inn"                "runs_bat"
## [13] "runs_br"            "runs_dp"            "runs_field"
## [16] "runs_infield"       "runs_outfield"      "runs_catcher"
## [19] "runs_good_plays"    "runs_defense"       "runs_position"
## [22] "runs_position_p"    "runs_replacement"   "runs_above_rep"
## [25] "runs_above_avg"     "runs_above_avg_off" "runs_above_avg_def"
## [28] "WAA"                "WAA_off"            "WAA_def"
## [31] "WAR"                "WAR_def"            "WAR_off"
## [34] "WAR_rep"            "salary"             "pitcher"
## [37] "teamRpG"            "oppRpG"             "oppRpPA_rep"
## [40] "oppRpG_rep"         "pyth_exponent"      "pyth_exponent_rep"
## [43] "waa_win_perc"       "waa_win_perc_off"   "waa_win_perc_def"
## [46] "waa_win_perc_rep"   "OPS_plus"           "TOB_lg"
## [49] "TB_lg"
```

We can see that we have a ton of variables that we don't necessarily want. Remember when we created a container in SQL above? If we were to try to read this into SQL now, we'd have to create a container for every one of these variables. We don't necessarily need "oppRpG," which measures how many runs per game that player's opponent scored. Similarly, we don't need a lot of other variables. We're going to limit ourselves to just the ones we want to keep by running the following (Note: you'll need the "dplyr" library for this function):

```
kept_Columns = select(Daily_bat, 1, 2, 4, 5, 6, 9, 31, 32, 33, 36, 47)
```

Which is going to keep only the columns we're interested in, namely:

```
names(kept_Columns)
```

```
##  [1] "name_common" "age"         "player_ID"   "year_ID"     "team_ID"
##  [6] "PA"          "WAR"         "WAR_def"     "WAR_off"     "pitcher"
## [11] "OPS_plus"
```

Next we're going to subset our data to keep it within the year range we're investigating (2006 to 2016), as well as limiting our data to be only batters, since we really aren't interested in how pitchers hit. Although the data we'll be analyzing is from 2006 to 2015, we'll also include 2016 as a potential year to see if our hypothesis is true. If a player is identified as a "pre-breakout" candidate in 2015, we can use 2016 to see if he did break out. Obviously, since the dataset doesn't include 2017, we won't be able to see if "pre-breakout" players in 2016 actually had good years in 2017. Pitchers aren't paid based on their ability to hit, so batting statistics for pitchers aren't relevant to what we're studying.

```
years_pitchers_Subset <- subset(kept_Columns, 2005 < year_ID & year_ID < 2017 & pitcher == "N")
```

Finally, we'll remove the pitcher column, since all of our kept players are going to be batters:

```
Batting_value <- select(years_pitchers_Subset, -10)
```

After all this, we're ready to export this data from R and into SQL so we can join it with our Batting_Salary table.

```
write.csv(Batting_value, "Batting_Value.csv")
```

We create our container in SQL by running the following:

```
CREATE TABLE Batting_Value (
    AutoNum int NOT NULL AUTO_INCREMENT,
    name_common varchar(255),
    age int,
    player_ID varchar(255),
    year_ID int,
    team_ID varchar(255),
    PA int,
    WAR DECIMAL(4,2),
    WAR_def DECIMAL(4,2),
    WAR_off DECIMAL(4,2),
    OPS_plus int,
    PRIMARY KEY (AutoNum)
);
```

Then we'll load our data into the container using the following command:

```
LOAD DATA LOCAL INFILE '/filepath/Batting_Value.csv'
INTO TABLE Batting_Value FIELDS TERMINATED BY ','
ENCLOSED BY '"' LINES TERMINATED BY '\n'
IGNORE 1 LINES
```

Now we want to combine this data with our Batting_Salary data so we can have our final dataset to work with in R.

We'll create the container we'll use for our combined data:

```
CREATE TABLE Value_Salary (
    AutoNum int NOT NULL AUTO_INCREMENT,
    playerID varchar(255),
    name_common varchar(255),
    yearID int,
    age int,
    salary int,
    stint int,
    teamID varchar(255),
    lgID varchar(255),
    G int,
    PA int,
    AB int,
    R int,
    H int,
    2B int,
    3B int,
    HR int,
    RBI int,
    SB int,
    CS int,
    BB int,
    SO int,
    GIDP int,
    SF int,
    SH int,
    HBP int,
    IBB int,
    WAR DECIMAL(4,2),
    WAR_def DECIMAL(4,2),
    WAR_Off DECIMAL(4,2),
    OPS_plus int,
    PRIMARY KEY (AutoNum)
);
```

Then we load in our data:

```
INSERT INTO Value_Salary
SELECT DISTINCT 0,
Batting_Salary.playerID,
Batting_Value.name_common,
Batting_Salary.yearID,
Batting_Value.age,
```

```sql
Batting_Salary.salary,
Batting_Salary.stint,
Batting_Salary.teamID,
Batting_Salary.lgID,
Batting_Salary.G,
Batting_Value.PA,
Batting_Salary.AB,
Batting_Salary.R,
Batting_Salary.H,
Batting_Salary.2B,
Batting_Salary.3B,
Batting_Salary.HR,
Batting_Salary.RBI,
Batting_Salary.SB,
Batting_Salary.CS,
Batting_Salary.BB,
Batting_Salary.SO,
Batting_Salary.GIDP,
Batting_Salary.SF,
Batting_Salary.SH,
Batting_Salary.HBP,
Batting_Salary.IBB,
Batting_Value.WAR,
Batting_Value.WAR_def,
Batting_Value.WAR_off,
Batting_Value.OPS_plus
FROM Batting_Salary
INNER JOIN Batting_Value ON Batting_Salary.playerID = Batting_Value.player_ID AND Batting_Salary.yearID
```

As a note, this table works very well until it comes to the players who were traded in the middle of the season. When a player hasn't been traded, there is one row per year. However, when a player has been traded, we would expect to see two columns that year (one for the old team, one for the new team). However, we see four:

```
##   AutoNum  playerID        name_common yearID age   salary stint
## 1       1  abadan01          Andy Abad   2006  33   327000     1
## 2       2 abercre01 Reggie Abercrombie   2006  25   327000     1
## 3       3 abreubo01        Bobby Abreu   2006  32 13600000     1
## 4       4 abreubo01        Bobby Abreu   2006  32 13600000     1
## 5       5 abreubo01        Bobby Abreu   2006  32 13600000     2
## 6       6 abreubo01        Bobby Abreu   2006  32 13600000     2
```

Split, apply, combine?

We need to remove those duplicates, and one very common way of doing so is via a process often used in data analysis called "Split, apply, combine."

This process involves splitting the data into subsets, applying a function to each subset, and then combining the data back together. Which is what we want to do here. We want to remove the duplicate players from every year, but we don't want to remove duplicates from the dataset as a whole. For example, there are multiple "Bobby Abreu" rows in our 2006 data, which we would like to reduce to a single row. However, we don't want to remove all of the "Bobby Abreu" rows from our master dataset, because that would remove Bobby Abreu's stats for 2007, 2008, and so on.

We could subset our data by year, write a function that removes duplicates for each year, and then join all of our subsets back together. Or, we could use a library that we've already called, the 'dplyr' library, and remove the duplicates by year from our main dataset.

If we execute the following command:

```
Batting_Value_Salary_2 <- Pre_Batting_Value_Salary %>%
  distinct(playerID, yearID, .keep_all = TRUE)
```

We can see that we've successfully removed the duplicates:

```
##   AutoNum  playerID        name_common yearID age   salary stint
## 1       1  abadan01           Andy Abad   2006  33   327000     1
## 2       2 abercre01 Reggie Abercrombie   2006  25   327000     1
## 3       3 abreubo01        Bobby Abreu   2006  32 13600000     1
## 4       7 adamsru01         Russ Adams   2006  25   343000     1
## 5       8 aguilch01       Chris Aguila   2006  27   327000     1
## 6       9 alfoned01    Edgardo Alfonzo   2006  32  8000000     1
```

This is a nice way of using pre-built libraries to limit the amount of code that you need to write. The dplyr library has a ton of functions like this, and we'll make use of them when we further analyze our data in R.

Additionally, we'll want to remove anyone who changed teams in the middle of the season, as the formatting of our database made the data for these players essentially impossible to work with. We'll do so with the following subset:

```
Batting_Value_Salary <- subset(Batting_Value_Salary_2, Batting_Value_Salary_2$PA >= Batting_Value_Salar
```

There we go! Our Batting Value data is all ready for analysis in R. It took a lot of behind the scenes work, but it left us with some very high quality data on which to test our hypotheses.

Working with our Batting Value Data

Now that we have everything loaded in and set up properly, we're going to begin the actual analysis step of this project. We spent a ton of time loading in the necessary variable in order to work with our data, but there are still more new variables that will help us draw even deeper conclusions.

Creating variables

Cost-per-win

Since we have wins above replacement data and salary data, we can create a new variable called "cost-per-win" that will let us know how expensive a win is on the open market. If a team were to go out and sign a player, the "cost-per-win" would be how much it would cost, on average, to buy 1 win above replacement. That data has been collected online, and we're going to use the data from "https://batflipsandnerds.com/2018/06/20/the-modern-myths-of-baseball-the-cost-of-a-win/"

These values change every year, so we'll need to make sure that the specific cost-per-win is applied to the appropriate year. In order to do this, we'll need to do another split-apply-combine on our data.

First we'll split the data by year:

```
year <- split(Batting_Value_Salary, Batting_Value_Salary$yearID)
```

Then we'll apply the appropriate cost-per-win to each year:

```
year$`2006`$cost_per_win <- 4600000
year$`2007`$cost_per_win <- 5300000
year$`2008`$cost_per_win <- 5600000
year$`2009`$cost_per_win <- 5700000
year$`2010`$cost_per_win <- 5800000
year$`2011`$cost_per_win <- 6400000
year$`2012`$cost_per_win <- 6900000
year$`2013`$cost_per_win <- 7500000
year$`2014`$cost_per_win <- 7700000
year$`2015`$cost_per_win <- 8700000
year$`2016`$cost_per_win <- 9600000
```

Finally, we'll combine our data back together now that we've split it by year and applied the appropriate cost-per-win:

```
Batting_Value_Salary <- rbind(year$`2006`, year$`2007`, year$`2008`, year$`2009`, year$`2010`, year$`20
```

Player Money Value

The next variable we're going to create is going to be "Player Money Value." This is going to capture how much value, in dollars, a player provided that season. We'll do this by multiplying a player's wins above replacement by the cost of a win above replacement. This number will essentially be how much a player was worth if they had been available on the open market and received fair market value. This is how we'll do it:

```
Batting_Value_Salary$player_money_value <- (Batting_Value_Salary$WAR * Batting_Value_Salary$cost_per_wi
```

Value Plus

It's important to de-contextualize a player's value. For example, if we said that Joe Random provided $6 million in value, we wouldn't know off the top of our heads if that was good or bad. We'd need to know how much value everyone else was providing. "Value Plus" is a statistic that allows us to do that. It normalizes the value a player provides, where 100 is average. If a player has a Value Plus of 120, it means that they provided 20% more value than average. We'll calculate this by doing the following:

```
Batting_Value_Salary$value_plus <- ((Batting_Value_Salary$player_money_value / mean(Batting_Value_Salar
```

Excess value

The next variable we'll need is "Excess value." This will measure the difference between how much a player was worth and how much they actually made. It's worth noting that for some players, especially good players with low salaries (i.e. players still on their rookie contracts), this number will be quite high. For some players, this number could be in the negatives, indicating that they cost their team money with their play. We'll calculate this as follows:

```
Batting_Value_Salary$excess_value <- (Batting_Value_Salary$player_money_value - Batting_Value_Salary$sal
```

Excess value plus

The final variable we'll be working with right now is "Excess Value Plus." This will be a normalized version of "Excess value" in order to give a more accurate and context-free representation of the excess value for each player. While knowing how valuable a player was with respect to other players, it's even more valuable to know how much excess value they provided relative to other players. This is very similar to our "Value Plus" statistic, except this new variable also incorporates a player's salary into it. Here's how we'll calculate it:

```
Batting_Value_Salary$excess_value_plus <- ((Batting_Value_Salary$excess_value / mean(Batting_Value_Sala
```

Batting average, On-Base Percentage, Slugging Percentage, OPS, and ISO

We've mostly created sabermetric statistics thus far, but in order to identify potential undervalued and overvalued players, it will help to use some traditional statistics as well.

Batting average measures how many hits a player gets per at-bat, on-base percentage measures how often a player gets on-base per at-bat, and slugging percentage measures how many bases a player obtains per at-bat (essentially, how hard does he hit the ball).

We'll calculate batting average as follows:

```
pre_batting_average <- (Batting_Value_Salary$H)/(Batting_Value_Salary$AB)
Batting_Value_Salary$batting_average <- round(pre_batting_average, digits = 3)
```

Next, we'll calculate on-base percentage:

```
pre_on_base_percentage <- (Batting_Value_Salary$H + Batting_Value_Salary$BB + Batting_Value_Salary$HBP)/
Batting_Value_Salary$on_base_percentage <- round(pre_on_base_percentage, digits = 3)
```

Then we'll calculate slugging percentage:

```
pre_slugging_percentage <- ((Batting_Value_Salary$H - Batting_Value_Salary$X2B - Batting_Value_Salary$X3
  (Batting_Value_Salary$AB)
Batting_Value_Salary$slugging_percentage <- round(pre_slugging_percentage, digits = 3)
```

OPS, or on-base plus slugging, is a statistic that, simply put, combines on-base-percentage and slugging percentage. It's a quick way to see how often a guy got on baseball, and how hard he hit the ball to get on base.

```
Batting_Value_Salary$OPS <- (Batting_Value_Salary$on_base_percentage) + (Batting_Value_Salary$slugging_p
```

ISO, or isolated power, is another quick metric to determine how powerful someone was at the plate. The higher the ISO, the more bases someone accumulated relative to their batting average. It's not super helpful as a predictive statistic, but it will give us another tool to work with when we're doing our player similarity evaluations.

```
Batting_Value_Salary$ISO <- (Batting_Value_Salary$slugging_percentage) - (Batting_Value_Salary$batting_a
```

Strikeout percentage and walk percentage

The final two statisics we'll create for the time being are going to measure plate discipline. Generally speaking, a low strikeout percentage and a high walk percentage are positive predictors of future success. In order to identify potential breakout players, we'll want to quantify these statistics to use for future evaluation.

```
pre_SO_percentage <- ((Batting_Value_Salary$SO)/(Batting_Value_Salary$PA)*100)
Batting_Value_Salary$SO_percentage <- round(pre_SO_percentage, digits = 2)
```

```
pre_BB_percentage <- ((Batting_Value_Salary$BB)/(Batting_Value_Salary$PA)*100)
Batting_Value_Salary$BB_percentage <- round(pre_BB_percentage, digits = 2)
```

Context-free Strikeout and walk percentages

One issue we'll run into with strikeout and walk percentages is that those totals vary by year in the league. From 2006 to 2016, strikeouts varied from 16.8% in 2006 to 21.1% in 2016. Similarly, walk rates ranged from 7.6% in 2014 to 8.9% in 2009.

If we didn't correct for context, then a strikeout rate of 19% could be either above-average or below-average depending on the year. In order to solve this, we'll once again do a split-apply-combine technique to create our variables.

Strikeout and walk percentages split-apply-combine

```
year <- split(Batting_Value_Salary, Batting_Value_Salary$yearID)

year$`2006`$league_SO_percentage <- 16.8
year$`2006`$league_BB_percentage <- 8.4
year$`2007`$league_SO_percentage <- 17.1
year$`2007`$league_BB_percentage <- 8.5
year$`2008`$league_SO_percentage <- 17.5
year$`2008`$league_BB_percentage <- 8.7
year$`2009`$league_SO_percentage <- 18.0
year$`2009`$league_BB_percentage <- 8.9
year$`2010`$league_SO_percentage <- 18.5
year$`2010`$league_BB_percentage <- 8.5
year$`2011`$league_SO_percentage <- 18.6
year$`2011`$league_BB_percentage <- 8.1
year$`2012`$league_SO_percentage <- 19.8
year$`2012`$league_BB_percentage <- 8.0
year$`2013`$league_SO_percentage <- 19.9
year$`2013`$league_BB_percentage <- 7.9
year$`2014`$league_SO_percentage <- 20.4
year$`2014`$league_BB_percentage <- 7.6
year$`2015`$league_SO_percentage <- 20.4
year$`2015`$league_BB_percentage <- 7.7
year$`2016`$league_SO_percentage <- 21.1
year$`2016`$league_BB_percentage <- 8.2
```

```
Batting_Value_Salary <- rbind(year$`2006`, year$`2007`, year$`2008`, year$`2009`, year$`2010`, year$`20
                year$`2012`, year$`2013`, year$`2014`, year$`2015`, year$`2016`)
```

Creating SO+ and BB+

```
pre_SO_plus <-
  ((Batting_Value_Salary$SO_percentage / Batting_Value_Salary$league_SO_percentage*100))
Batting_Value_Salary$SO_plus <- round(pre_SO_plus, digits = 2)

pre_BB_plus <-
  ((Batting_Value_Salary$BB_percentage / Batting_Value_Salary$league_BB_percentage*100))
Batting_Value_Salary$BB_plus <- round(pre_BB_plus, digits = 2)
```

Solving Issues by Subsetting

Now that we have all of our variables created, we seem like we're just about set to start getting to the meat of the project, and determining which players are the most undervalued and which are the most overvalued.

However, some quick graphs will highlight an issue that we face, and this issue isn't so much with our data so much as it is with an underlying fact about Major League Baseball.

Graphing Average Salary and Excess Value by Age

Since our main focus will be on the amount of excess value a player provided, and excess value is comprised of player value and salary, it will be key to focus on the average salaries for players in different age groups. This will demonstrate a bit where our problem lies.

Salary Plus

First, we'll create a Salary+ metric that will normalize salary data relative to the average salary.

```
Batting_Value_Salary$salary_plus <- ((Batting_Value_Salary$salary / mean(Batting_Value_Salary$salary)*1
```

Next, we're going to calculate the mean salary by age, and then plot that to get a graphical representation of how salary changes as a player gets older.

```
agg1 <- aggregate(x = Batting_Value_Salary$salary_plus, by = list(Batting_Value_Salary$age), FUN = mean
names(agg1)[1] <- "Age"
names(agg1)[2] <- "Average Salary"
```

Excess Value Plus

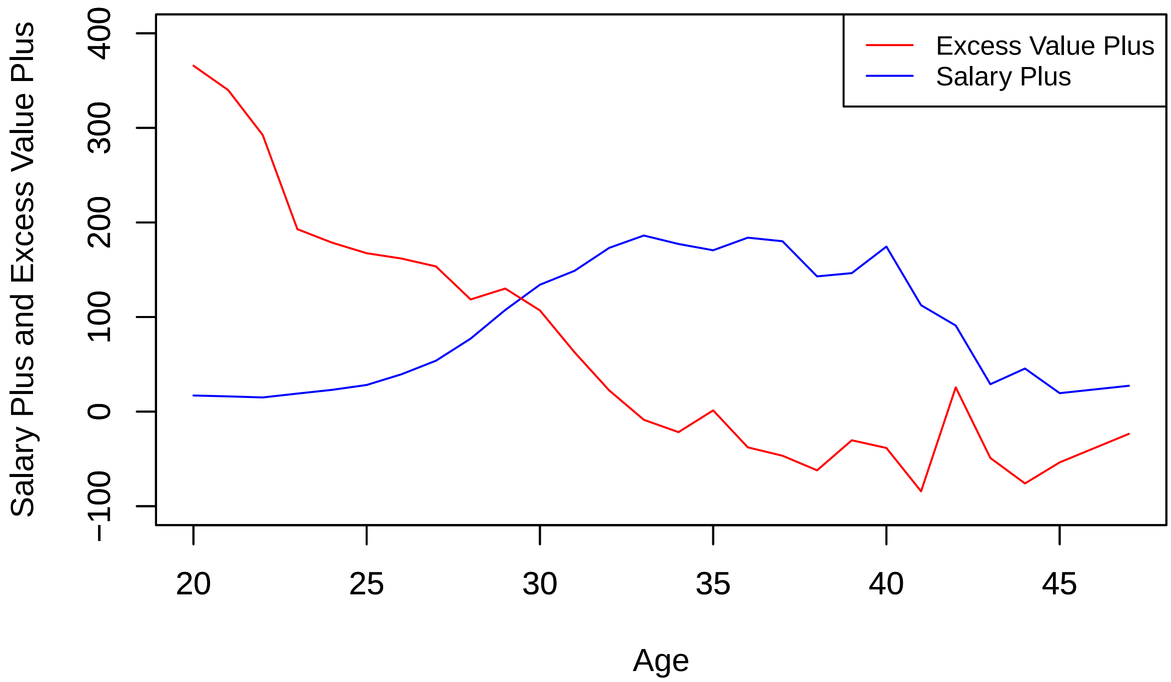
The second part of our graph will be Excess Value Plus by age, so we'll calculate the mean Excess Value Plus by Age as follows:

```
agg2 <- aggregate(x = Batting_Value_Salary$excess_value_plus, by = list(Batting_Value_Salary$age), FUN =
names(agg2)[1] <- "Age"
names(agg2)[2] <- "Excess Value Plus"
```

Our Graph

Finally, we'll graph the two sets of data and illuminate what our issue is.

```
plot1 <- plot(agg1$Age, agg1$`Average Salary`, xlab="Age", ylab="Salary Plus and Excess Value Plus", ty
              ylim = c(-100,400))
par(new=TRUE)
plot2 <- plot(agg2$Age, agg2$`Excess Value Plus`, xlab="", ylab="", type = 'l', col = 'red',
              ylim = c(-100,400))
legend("topright", legend=c("Excess Value Plus", "Salary Plus"),
       col=c("red", "blue"), lty=1:1, cex=0.8)
```



As we can see from our plot, the most excess value comes from players in their early 20s, which is also where salary is the lowest. This initially seems like a huge breakthrough, with the solution being to simply acquire as many talented young players as possible.

However, this is akin to advising someone to invest in Apple and Microsoft in 1980. Of course, it's easy to look back and say that drafting Mike Trout or Mookie Betts was the best course of action, and to an extent it was. However, that doesn't provide any valuable information going forward.

Further, by nature of the MLB's contract structure, players are underpaid for the first 6 years of their careers. They're even more severely underpaid for the first three years of their careers, as they're playing for league minimum. That has a lot to do with why the salary plus statistic is below average (below 100) until players are roughly 28 or 29 years old.

Once a player reaches six years of service time, they're eligible for free agency, in which they can sign with any team in the league. These are the players we're interested in evaluating. We want to find guys who were readily available to every team in the league, and provided the most excess value for the teams that took a risk on them.

For this reason, it makes sense to subset our data to only include players who are 27 years old and older. If you look at the graph, there's a small peak in Excess Value Plus at 27, before a steep decline into a player's 30s. This could indicate that there is a small group of players who slipped through the cracks initially that provided great value in their late 20s at a very low cost.

Subsetting our data

Since we want to limit our data to players who are 27 and older, we'll need to subset our data to only include those players. We'll also want to create an additional filter, limiting ourselves to players making less than 3 million dollars per year. The reason for this is that if a player is already making 3 million dollars, they aren't really a diamond in the rough. They've already been properly valued by their team, and aren't exactly readily available for any team to pick up. Since we want to limit ourselves to players that are readily available, it makes sense to create this salary limit.

We'll also want to limit ourselves to the years 2007 to 2010. We're going with 2007 because our dataset is from 2006 to 2016. If we identify a breakout player from 2007, we can use the 2006 data to see what they were like in their pre-breakout season. We wouldn't be able to do the same for a player who broke out in 2006, since we don't have 2005 data. We'll use the years 2007 to 2010 to find the players to populate our model, and then test that model on the years 2011 to 2015 to see how effective it was.

Our subsets will be created as follows:

```
oh_six_to_oh_ten <- subset(Batting_Value_Salary, 2006 < yearID & yearID < 2011)
age_salary_subset <- subset(oh_six_to_oh_ten, age > 26 & salary < 3000000)
```

Finding breakout players

Now that we've limited ourselves to the appropriate players, ones that aren't young phenoms or already established high-paid players, we can identify certain breakout players that we'll build our model breakout player off of. We'll identify the top 30 players in terms of "excess value plus." That is, these are the players who provided the most value relative to how much they made, adjusted to be context-independent. This reveals:

```
temp_top_excess_value <- age_salary_subset[ age_salary_subset$excess_value_plus >= age_salary_subset$exc
top_excess_value <- temp_top_excess_value[order(-temp_top_excess_value$excess_value_plus),]
head(top_excess_value[,1:7])
```

```
##      AutoNum  playerID    name_common yearID age  salary stint
## 3574    4836 zobribe01    Ben Zobrist   2009  28  415900     1
## 1159    1633 bautijo02  Jose Bautista   2010  29 2400000     1
## 3206    4390  penaca01    Carlos Pena   2007  29  800000     1
## 1211    1709  chooshi01  Shin-Soo Choo   2010  27  461100     1
## 886     1233 bartlja01 Jason Bartlett   2009  29 1981250     1
## 3598    4869 ramiral03 Alexei Ramirez   2010  28 1225000     1
```

While we identified fifty total players who provided considerable excess value, for the purpose of our model we'll be picking five. For various reasons, a majority of the players that were identified via the excess value plus leaderboards did not fit our criteria. In some instances, it was star players who simply had a rare bad season the year before. In other instances, it was a veteran who was signing for a cheap deal at the end of his career.

We also want to limit our model to players who were below average in value the year before. Teams are very adept already at determining value, so if a player was providing above-average value, it seems likely that they wouldn't be willing to easily part with that player.

The five players we'll use to create our "model" player are below:

```
post_breakout <- top_excess_value[ which(top_excess_value$AutoNum == 4836 |
                        top_excess_value$AutoNum == 4609 |
                        top_excess_value$AutoNum == 4692 |
                        top_excess_value$AutoNum == 4750 |
                        top_excess_value$AutoNum == 4749
                        ), ]
head(post_breakout[,1:7])
```

```
##      AutoNum  playerID   name_common yearID age  salary stint
## 3574    4836 zobribe01    Ben Zobrist   2009  28  415900     1
## 3500    4750  ryanbr01  Brendan Ryan   2009  27  403000     1
## 3383    4609 scutama01 Marco Scutaro   2008  32 1550000     1
## 3451    4692 paganan01   Angel Pagan   2009  27  575000     1
## 3499    4749  ruizca01   Carlos Ruiz   2009  30  475000     1
```

Since we're basing the most valuable players on how much excess value they provide, let's look at how much extra value these players provided in their breakout years than in the seasons the year before their breakout.

```
mean(post_breakout$excess_value_plus)
```

```
## [1] 509.4709
```

```
mean(pre_breakout$excess_value_plus)
```

```
## [1] 46.49784
```

```
mean(post_breakout$excess_value_plus) / mean(pre_breakout$excess_value_plus)
```

```
## [1] 10.95687
```

In their breakout seasons, this group of players provided excess value at a rate of five times average, compared to a rate of roughly half of average in their pre-breakout seasons. If a team could have successfully identified these players, they would have been able to see an almost eleven-fold increase in extra value, just from five players. In terms of dollars, these breakout players provided the below amount:

```
sum(post_breakout$excess_value)
```

```
## [1] 133791100
```

That's over $130 million in excess value, just from five players who were below average in value provided the season before. This is why this prediction model is so valuable. If we can identify the characteristics that led to these players breaking out, then we can attempt to recreate that level of extra value for ourselves.

Choosing the variables that make up our model (using a regression)

One of the most important aspects, if not the most important aspect, is determing which characteristics of our five model players played the biggest role in those players having breakout seasons the next year. If we look at the options we have, we can see the following:

```
##  [1] "AutoNum"              "playerID"             "name_common"
##  [4] "yearID"               "age"                  "salary"
##  [7] "stint"                "teamID"               "lgID"
## [10] "G"                    "PA"                   "AB"
## [13] "R"                    "H"                    "X2B"
## [16] "X3B"                  "HR"                   "RBI"
## [19] "SB"                   "CS"                   "BB"
## [22] "SO"                   "GIDP"                 "SF"
## [25] "SH"                   "HBP"                  "IBB"
## [28] "WAR"                  "WAR_def"              "WAR_Off"
## [31] "OPS_plus"             "cost_per_win"         "player_money_value"
## [34] "value_plus"           "excess_value"         "excess_value_plus"
## [37] "batting_average"      "on_base_percentage"   "slugging_percentage"
## [40] "OPS"                  "ISO"                  "SO_percentage"
## [43] "BB_percentage"        "league_SO_percentage" "league_BB_percentage"
## [46] "SO_plus"              "BB_plus"              "salary_plus"
```

We have close to fifty variables that could be predictive of future success. However, if we look at some of those variables more closely, we see that many of them are virtually the same as other variables, or they're subsumed under other variables. For example, strikeout percentage and walk percentage are both less-specific versions of Strikeouts+ and Walks+. Additionally, hits, doubles, triples, and home runs can all be quickly represented by ISO. Keeping this in mind, many of these variables can be excluded, and we'll have just a few that we'll need to test for statistical significance.

We'll be running a multivariable linear regression to identify if the variables we'd like to use to form our model are statistically significant. One important thing to note is that we'll be using Offensive WAR as our dependent variable, since all of the data we have is only for offense. It's unlikely that there would be any relationship (outside of random noise) between offensive statistics and Defensive WAR, so for our regression we'll be focusing on offense.

```r
library(jtools)
offensive_fit <- lm(WAR_Off ~ BB_percentage + SO_percentage + ISO + OPS, data = Batting_Value_Salary)
summ(offensive_fit)
```

```
## MODEL INFO:
## Observations: 4353 (5 missing obs. deleted)
## Dependent Variable: WAR_Off
## Type: OLS linear regression
##
## MODEL FIT:
## F(4,4348) = 1050.09, p = 0.00
## R² = 0.49
## Adj. R² = 0.49
##
## Standard errors: OLS
## --------------------------------------------------
##                         Est.   S.E.   t val.      p
## ------------------- ------- ------ -------- ------
## (Intercept)           -3.20   0.14   -22.63   0.00
## BB_percentage          0.03   0.01     5.40   0.00
## SO_percentage         -0.03   0.00    -9.30   0.00
## ISO                    5.16   0.52     9.99   0.00
## OPS                    5.80   0.24    24.40   0.00
## --------------------------------------------------
```

From this, we can see that the three variables we picked are all statistically significant due to the low p-values. Additionally, we see a positive relationship between walks and offensive value. This makes sense, as the more someone draws walks, the more times they're on base, and consequently the more runs they can score (and value is derived from runs).

On the opposite end, we see that strikeouts detract from value at roughly the same rate that walks add to value. Additionally, a one-unit increase in ISO leads to a 5.1 win increase in Offensive wins above replacement. This means that on average, going from an ISO of .100 to .200 will increased expected wins above replacement by 0.5. Similarly, a one-unit increase in OPS will lead to a 5.8 win increase (where going from a .700 OPS to a .800 OPS increases expected wins by 0.6)

Testing for collinearity

One important concept to pay attention to is the idea of multicollinearity. One assumption of linear regressions is that your independent variables aren't perfectly correlated. Essentially, you want to make sure that your explanatory variables aren't explaining part of each other in addition to the dependent variable. This is a key element of statistics.

In order to test for multicollinearity, we'll need the "car" library and the following code:

```
library(car)
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
car::vif(offensive_fit)
```

```
## BB_percentage SO_percentage           ISO           OPS
##      1.223398      1.299619      3.053792      3.475814
```

For the vif() function, the values start at 1 and can increase essentially forever. A value of 1 means that your explanatory variable is not at all correlated with other variables. A value greater than 5 means that you have some strong multicollinearity and your variables need to be reconsidered. Here, our highest value is 3.5, so we're good to proceed with the explanatory variables we've chosen.

Creating our model

Now we'll need to start adding in the components that will make up our model. Above, we noted that walk rate, strikeout rate, ISO, and OPS were all statistically significant factors and had meaningful coefficients. For our model, we'll need to incorporate the factors that make up ISO and OPS, and we'll also be using the context-free "BB+" and "SO+" variables we created.

Selecting our components will start like this:

```
pre_diff_score_subset <- pre_breakout%>%
                          select(name_common, PA, AB, H, X2B, X3B, HR, BB, SO, SF, HBP, SO_plus, BB_plus)
```

One quick housekeeping note is that we'll need to create separate variables for SO+ and BB+ so that we can properly average them for the players we chose. We'll need to weigh them by the amount of at-bats each player had, since the number of at-bats varies from 105 to 379. We'll do so as follows:

```
pre_diff_score_subset$adj_SO_plus <- pre_diff_score_subset$PA * pre_diff_score_subset$SO_plus
pre_diff_score_subset$adj_BB_plus <- pre_diff_score_subset$PA * pre_diff_score_subset$BB_plus
```

Now we'll create a new row in our data frame that is a sum of all the totals for each of the players. We'll need the "janitor" library for this.

```
library(janitor)
```

```
##
## Attaching package: 'janitor'
```

```
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
diff_score_subset <- pre_diff_score_subset %>%
  adorn_totals("row")
```

Creating variables again

Now we'll need to create the variables we'll actually use for our model based on the available data in our table. We already created them in our "Batting_Value_Salary" dataset, so we'll just quickly create them again for our new dataset.

True SO+ and BB+:

```
test_SO_plus <- diff_score_subset$adj_SO_plus / diff_score_subset$PA
diff_score_subset$True_SO_plus <- round(test_SO_plus, digits = 2)

test_BB_plus <- diff_score_subset$adj_BB_plus / diff_score_subset$PA
diff_score_subset$True_BB_plus <- round(test_BB_plus, digits = 2)
```

Batting average:

```
test_BA <- diff_score_subset$H / diff_score_subset$AB
diff_score_subset$batting_average <- round(test_BA, digits = 3)
```

On-base percentage:

```
test_OBP <- (diff_score_subset$H + diff_score_subset$BB + diff_score_subset$HBP) /
  (diff_score_subset$AB + diff_score_subset$BB + diff_score_subset$HBP + diff_score_subset$SF)
diff_score_subset$on_base_percentage <- round(test_OBP, digits = 3)
```

Slugging percentage:

```
test_SLG <- ((diff_score_subset$H - diff_score_subset$X2B - diff_score_subset$X3B - diff_score_subset$H
               (diff_score_subset$X2B * 2) + (diff_score_subset$X3B * 3) + (diff_score_subset$HR * 4))/
    (diff_score_subset$AB)
diff_score_subset$slugging_percentage <- round(test_SLG, digits = 3)
```

ISO:

```
diff_score_subset$ISO <- (diff_score_subset$slugging_percentage) - (diff_score_subset$batting_average)
```

Finally, we'll create a subset that just has our "Total" values. This will help us with working on our difference score calculations.

```
total_sub <- subset(diff_score_subset, name_common == "Total")
View(total_sub)
```

Creating our difference score formula

In order to identify which players are best represented by our determinative statistics, we can develop a difference score formula. A player with a score closer to 0 will be closer to the average for our chosen statistics and would theoretically be a player identified by the model as an undervalued asset. It looks as follows:

```
temp_DiffSc <- ((total_sub$True_BB_plus - diff_score_subset$True_BB_plus)/total_sub$True_BB_plus) +
  ((total_sub$True_SO_plus - diff_score_subset$True_SO_plus)/total_sub$True_SO_plus) +
  ((total_sub$ISO - diff_score_subset$ISO)/total_sub$ISO) +
  ((total_sub$batting_average - diff_score_subset$batting_average)/total_sub$batting_average) +
  ((total_sub$on_base_percentage - diff_score_subset$on_base_percentage)/total_sub$on_base_percentage)
  ((total_sub$slugging_percentage - diff_score_subset$slugging_percentage)/total_sub$slugging_percentag
```

Now that we have the formula down, we can apply that formula to our data frame.

```
temp_DiffSc2 <- ((total_sub$True_BB_plus - Batting_Value_Salary$BB_plus)/total_sub$True_BB_plus) +
  ((total_sub$True_SO_plus - Batting_Value_Salary$SO_plus)/total_sub$True_SO_plus) +
  ((total_sub$ISO - Batting_Value_Salary$ISO)/total_sub$ISO) +
  ((total_sub$batting_average - Batting_Value_Salary$batting_average)/total_sub$batting_average) +
  ((total_sub$on_base_percentage - Batting_Value_Salary$on_base_percentage)/total_sub$on_base_percentag
  ((total_sub$slugging_percentage - Batting_Value_Salary$slugging_percentage)/total_sub$slugging_percen

Batting_Value_Salary$DiffSc <- abs(temp_DiffSc2 * 100)
```

Once our formula who has been applied to our dataframe, we'll need to subset the data to only include the explanatory years (2011 to 2015) and only the players we're interested in identifying:

```
eleven_to_fifteen <- subset(Batting_Value_Salary, 2010 < yearID & yearID < 2016)
Older_than_25 <- subset(eleven_to_fifteen, salary < 3000000 & age > 25 & excess_value_plus < 100)
View(Older_than_25)
```

From this, we've identified 602 players that meet our criteria, however some additional narrowing down needs to take place to avoid players who have characteristics that cause them to not be eligible. Instances would include too small of a sample size or not playing the season after our explanatory year.

The first 5 players that meet our full criteria are as follows:

```
pre_breakout_two <- Older_than_25[ which(Older_than_25$AutoNum == 3785 |
                                           Older_than_25$AutoNum == 3715 |
                                           Older_than_25$AutoNum == 5242 |
                                           Older_than_25$AutoNum == 3243 |
                                           Older_than_25$AutoNum == 2218
),]
View(pre_breakout_two)
```

Now that we've seen their pre-breakout stats, we can take a look at their stats the next season.

```
post_breakout_two <- Batting_Value_Salary[ which(Batting_Value_Salary$AutoNum == 4155 |
                                                   Batting_Value_Salary$AutoNum == 4090 |
                                                   Batting_Value_Salary$AutoNum == 5433 |
                                                   Batting_Value_Salary$AutoNum == 3630 |
                                                   Batting_Value_Salary$AutoNum == 2581
), ]
View(post_breakout_two)
```

In order to see how well our model worked, we can use the same criteria that we used for the players we built our model with. We'll take the mean excess value before breakout and then compare it to the excess value after the players break out.

```
mean(pre_breakout_two$excess_value_plus)
```

```
## [1] 23.2811
```

```
mean(post_breakout_two$excess_value_plus)
```

```
## [1] 151.2403
```

```
mean(post_breakout_two$excess_value_plus) / mean(pre_breakout_two$excess_value_plus)
```

```
## [1] 6.496271
```

Notes to self

On 9/3, we'll want to do the following things:

1. Need to display the pre-breakout and post-breakout average stats in a table (line 785 for pre and line 797 for post)
2. Create a third table that shows the improvement between post-breakout and pre-breakout
3. Clean everything up and edit
4. Create R file that just has the code (with comments) without any of the R Markdown stuff