Baseball Value

Matthew Helbig

2019-09-28

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. It's definitely possible to run all your SQL commands inside the MySQL environment itself, but RStudio has packages available for installation that make it easy to work with your database right from R. This has a few advantages, most notably the fact that you don't have to keep hopping back and forth between programs.

Talking to our Database

The first thing you'll do is make sure that the RMySQL package is installed. This is as simple as calling **install.packages("RMySQL")** once.

Next we'll load up our RMySQL library and get to work.

```
library(RMySQL)
```

```
## Loading required package: DBI
```

You'll need to define your new database object, called "mydb" here, using the following template:

mydb = dbConnect(MySQL(), user='user', password='password', dbname='dbname', host='host').

Then we'll run a quick command to look at the different tables inside the dataset we've loaded up.

```
dbListTables(mydb)
```

```
##  [1] "AllstarFull"         "Appearances"         "AwardsManagers"
##  [4] "AwardsPlayers"       "AwardsShareManagers" "AwardsSharePlayers"
##  [7] "Batting"             "BattingPost"         "Batting_Salary"
## [10] "Batting_post_2006"   "CollegePlaying"      "Fielding"
## [13] "FieldingOF"          "FieldingOFsplit"     "FieldingPost"
## [16] "HallOfFame"          "HomeGames"           "Managers"
## [19] "ManagersHalf"        "Master"              "Parks"
## [22] "Pitching"            "PitchingPost"        "Player_Salary_Info"
## [25] "Salaries"            "Salaries_post_2006"  "Schools"
## [28] "SeriesPost"          "Teams"               "TeamsFranchises"
## [31] "TeamsHalf"           "test1"
```

So the three 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. We're going to be mostly concerned with salary information from 2006 to 2016, since a lot of salary data before 2006 is incomplete and we're going to be running some predictive models on 2017 and 2018 data later.

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).

```sql
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.

```sql
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.

```sql
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:

```sql
CREATE TABLE Salaries_post_2006
LIKE Salaries
```

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

```sql
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.

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.

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,
    IBB int,
    HBP int,
    SH int,
    SF int,
    GIDP 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
```

More SQL stuff It is possible to We should get the player value data from baseball-reference and convert it to CSV, then we should see if we can combine the baseball-reference data with our current data. That'd be a good task for Sunday or Monday