STAT 209: Lab 10

Code

Data-Wrangling: Putting it All Together

Goal

Refresh your memory about the main data-wrangling building blocks, and work through an extended example that combines them in various ways to achieve a goal.

The Data

This lab is not about baby names! Woohoo!

Instead, we'll look at some baseball data from the `Lahman` package.

This lab is modified from section 4.4 in your textbook.

Our overarching goal is the following: for a few star major league hitters, create a plot that shows the hitting statistic OPS+ (On-Base-Plus-Slugging average for the player relative to league average in that year) for each player in each season that they played in the majors, as a function of their age. This will enable us to see at a glance at what age various star players "peaked" (at least, according to this one measure).

Load the packages and data:

Code

Peek at the data

Code

```
##    playerID yearID stint teamID lgID  G  AB  R  H X2B X3B HR RBI SB CS BB
## 1 abercda01   1871     1    TRO   NA  1   4  0  0   0   0  0   0  0  0  0
## 2  addybo01   1871     1    RC1   NA 25 118 30 32   6   0  0  13  8  1  4
## 3 allisar01   1871     1    CL1   NA 29 137 28 40   4   5  0  19  3  1  2
## 4 allisdo01   1871     1    WS3   NA 27 133 28 44  10   2  2  27  1  1  0
## 5 ansonca01   1871     1    RC1   NA 25 120 29 39  11   3  0  16  6  2  2
## 6 armstbo01   1871     1    FW1   NA 12  49  9 11   2   1  0   5  0  1  0
##   SO IBB HBP SH SF GIDP
## 1  0  NA  NA NA NA    0
## 2  0  NA  NA NA NA    0
## 3  5  NA  NA NA NA    1
## 4  2  NA  NA NA NA    0
## 5  1  NA  NA NA NA    0
## 6  1  NA  NA NA NA    0
```

Code

```
##    playerID birthYear birthMonth birthDay birthCountry birthState
## 1 aardsda01      1981         12       27          USA         CO
## 2 aaronha01      1934          2        5          USA         AL
## 3 aaronto01      1939          8        5          USA         AL
## 4  aasedo01      1954          9        8          USA         CA
## 5  abadan01      1972          8       25          USA         FL
## 6  abadfe01      1985         12       17         D.R.  La Romana
##    birthCity deathYear deathMonth deathDay deathCountry deathState
## 1     Denver        NA         NA       NA         <NA>       <NA>
## 2     Mobile        NA         NA       NA         <NA>       <NA>
## 3     Mobile      1984          8       16          USA         GA
## 4     Orange        NA         NA       NA         <NA>       <NA>
## 5 Palm Beach        NA         NA       NA         <NA>       <NA>
## 6  La Romana        NA         NA       NA         <NA>       <NA>
##   deathCity nameFirst nameLast        nameGiven weight height bats throws
## 1      <NA>     David  Aardsma      David Allan    215     75    R      R
## 2      <NA>      Hank    Aaron      Henry Louis    180     72    R      R
## 3   Atlanta    Tommie    Aaron       Tommie Lee    190     75    R      R
## 4      <NA>       Don     Aase   Donald William    190     75    R      R
## 5      <NA>      Andy     Abad    Fausto Andres    184     73    L      L
## 6      <NA>  Fernando     Abad Fernando Antonio    220     73    L      L
##        debut  finalGame  retroID   bbrefID  deathDate  birthDate
## 1 2004-04-06 2015-08-23 aardd001 aardsda01       <NA> 1981-12-27
## 2 1954-04-13 1976-10-03 aaroh101 aaronha01       <NA> 1934-02-05
## 3 1962-04-10 1971-09-26 aarot101 aaronto01 1984-08-16 1939-08-05
## 4 1977-07-26 1990-10-03 aased001  aasedo01       <NA> 1954-09-08
## 5 2001-09-10 2006-04-13 abada001  abadan01       <NA> 1972-08-25
## 6 2010-07-28 2017-10-01 abadf001  abadfe01       <NA> 1985-12-17
```

Focusing on a single player: Manny Ramirez

Notice that in both of these data tables, players are indexed not by their names, but by unique player IDs. Manny Ramirez, for example, is `"ramirma02"`.

How would we know this? We can get it from the `Master` data table which records players' first and last names along with their unique IDs.

Code:

Code

```
##    playerID birthYear birthMonth birthDay birthCountry       birthState
## 1 ramirma02      1972          5       30         D.R. Distrito Nacional
##       birthCity deathYear deathMonth deathDay deathCountry deathState
## 1 Santo Domingo        NA         NA       NA         <NA>       <NA>
##   deathCity nameFirst nameLast        nameGiven weight height bats throws
## 1      <NA>     Manny  Ramirez Manuel Aristides    225     72    R      R
##        debut  finalGame  retroID   bbrefID deathDate  birthDate
## 1 1993-09-02 2011-04-06 ramim002 ramirma02      <NA> 1972-05-30
```

(In this case, there were no other major leaguers with that name, so there's no ambiguity; but for other names there might be more than one player with that name, hence the need for a unique ID.)

Having done that, we can filter the batting data to look at Ramirez's season-by-season stats.

Code:

Code

```
##    playerID yearID stint teamID lgID   G  AB   R   H X2B X3B HR RBI SB CS
## 1 ramirma02   1993     1    CLE   AL  22  53   5   9   1   0  2   5  0  0
## 2 ramirma02   1994     1    CLE   AL  91 290  51  78  22   0 17  60  4  2
## 3 ramirma02   1995     1    CLE   AL 137 484  85 149  26   1 31 107  6  6
## 4 ramirma02   1996     1    CLE   AL 152 550  94 170  45   3 33 112  8  5
## 5 ramirma02   1997     1    CLE   AL 150 561  99 184  40   0 26  88  2  3
## 6 ramirma02   1998     1    CLE   AL 150 571 108 168  35   2 45 145  5  3
##   BB  SO IBB HBP SH SF GIDP
## 1  2   8   0   0  0  0    3
## 2 42  72   4   0  0  4    6
## 3 75 112   6   5  2  5   13
## 4 85 104   8   3  0  9   18
## 5 79 115   5   7  0  4   19
## 6 76 121   6   6  0 10   18
```

Code

```
## [1] 21
```

Which seasons was he active? (`summarize()`)

We can see from this data that each "case" in the `Batting` dataset appears to consist of a season's worth of hitting data for a single player. Manny Ramirez has 21 entries. Was he active for 21 seasons? Let's check.

Code

Code

```
##   rookie_year final_year num_seasons num_teams
## 1        1993       2011          19         5
```

Note that I used a new function, `n_distinct()` , to return the number of *distinct* values of a variable in a column.

Hmm… looks like he only played in 19 seasons (for a total of 5 different teams); not 21. What's happening here?

It turns out that the rows of the `Batting` table are not necessarily a full season's worth of data. If a player was traded during a season, then they played for two different teams that year, and so there are two different entries in the data. There must be two years when that happened for Ramirez. Let's find out which they are.

When was he traded? (`group_by()` and `summarize()`)

Code:

Code

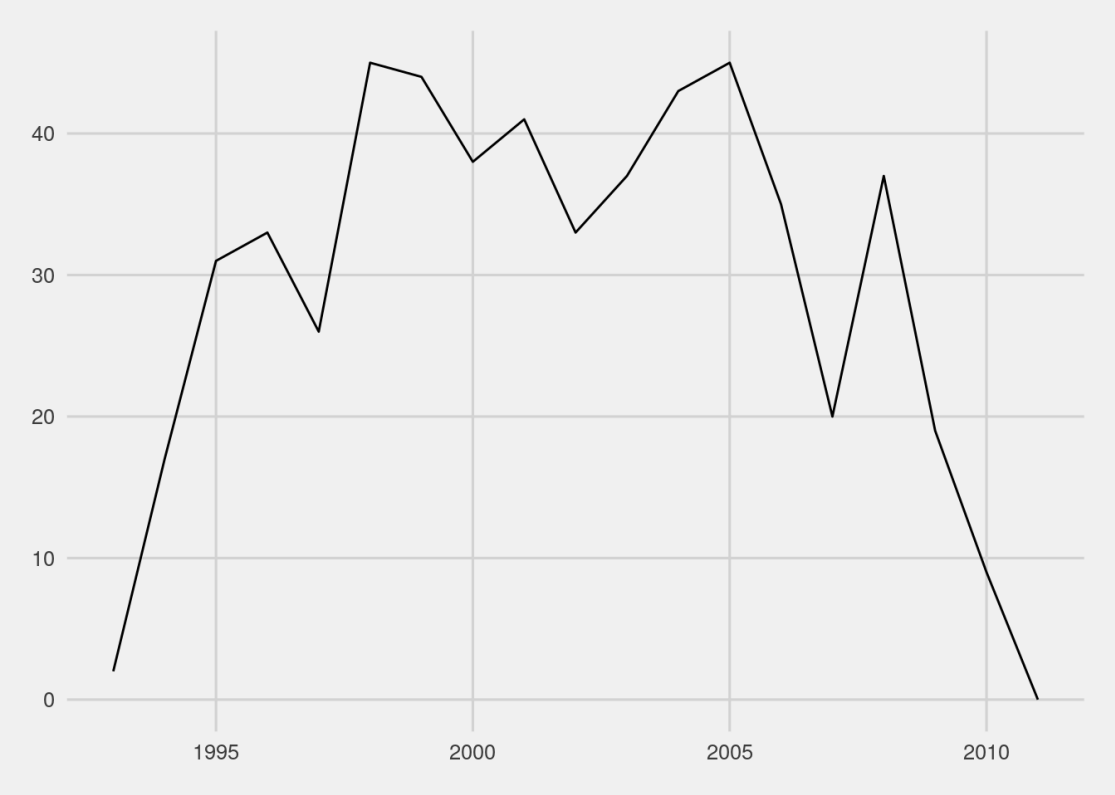
```
## # A tibble: 2 x 2
##   yearID num_teams
##    <int>     <int>
## 1   2008         2
## 2   2010         2
```

If we want to compute Manny's batting statistics by full season, we're going to have to do some aggregation to combine the two rows in these years.

Exercise 1 Compute and plot the number of home runs hit by Ramirez in each season.

Sample Solution:

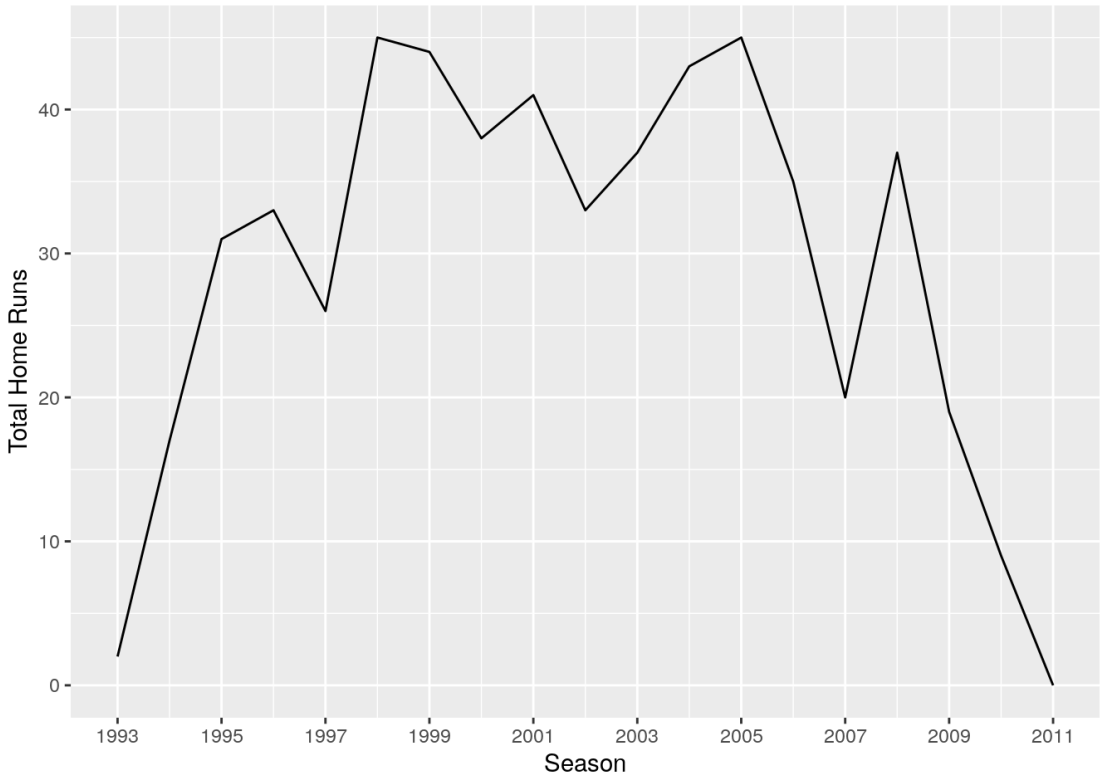
Code



It might be nice to have a few more years labeled on the x axis. Let's modify our plot using `scale_x_continuous()`:

Sample solution (modified)

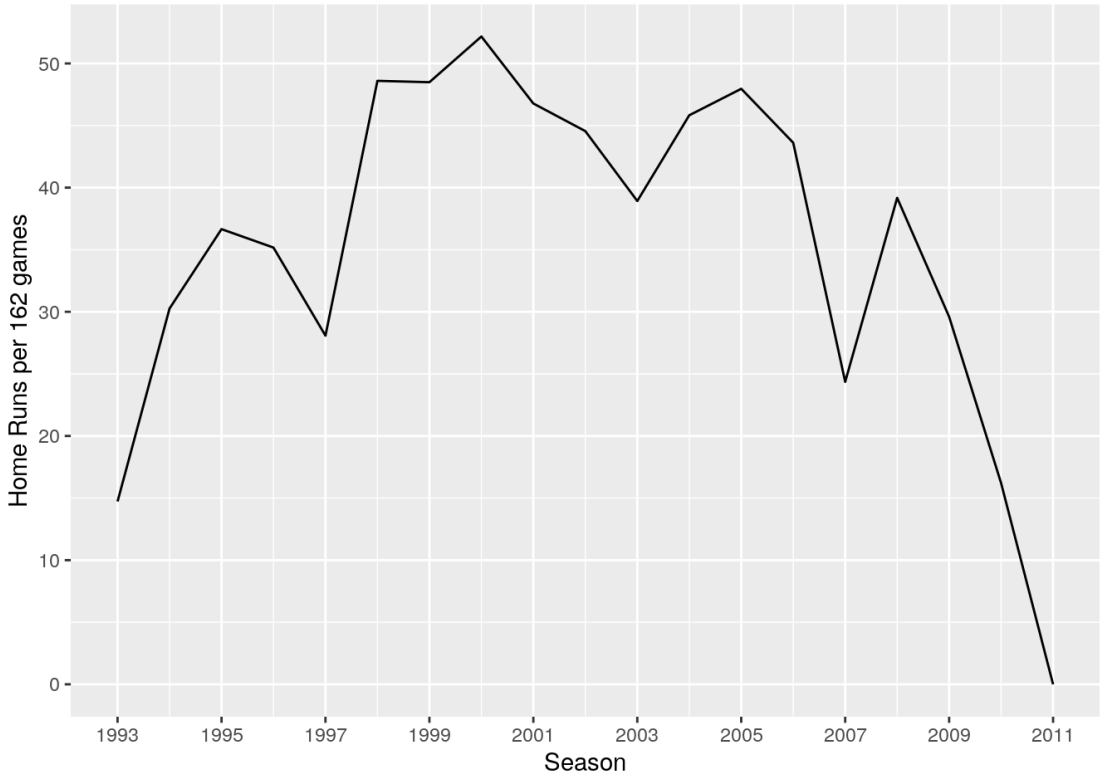
Code



Ramirez hit the most home runs in 1998 and 2005. Of course, new players may not play as often as players that have been around for a few years, so let's instead plot Ramirez's "projected" home runs extrapolating to 162 games.

Sample solution

Code



The shape is basically the same, but now we can see that although Ramirez hit the same number of home runs in 2000 and 2005, he did it in fewer games in 2000.

Adding Age to the Data (`join()`)

Our original goal was to construct a plot similar to this by the player's age, instead of by season. Can we find age in the `Batting` table?

Code

```
##  [1] "playerID" "yearID"   "stint"    "teamID"   "lgID"     "G"
##  [7] "AB"       "R"        "H"        "X2B"      "X3B"      "HR"
## [13] "RBI"      "SB"       "CS"       "BB"       "SO"       "IBB"
## [19] "HBP"      "SH"       "SF"       "GIDP"
```

Hmm… nope. Nor is there anything in this table that we can use to calculate age. What about in the biographical table, `Master`?

Code

```
##  [1] "playerID"     "birthYear"    "birthMonth"   "birthDay"
##  [5] "birthCountry" "birthState"   "birthCity"    "deathYear"
##  [9] "deathMonth"   "deathDay"     "deathCountry" "deathState"
## [13] "deathCity"    "nameFirst"    "nameLast"     "nameGiven"
## [17] "weight"       "height"       "bats"         "throws"
## [21] "debut"        "finalGame"    "retroID"      "bbrefID"
## [25] "deathDate"    "birthDate"
```

Aha, this table gives us players' birth years.

To a first approximation (that is, ignoring birth month and day), we can represent age using the formula `yearID - birthYear`. Only problem is, these two variables come from different data tables.

This sounds like a job for a `join`!

Exercise 2 Use a suitable `join` operation, together with whatever other verbs are needed, to add `Age` to the `Batting` table, and plot total HR/162 by Age.

Sample solution:

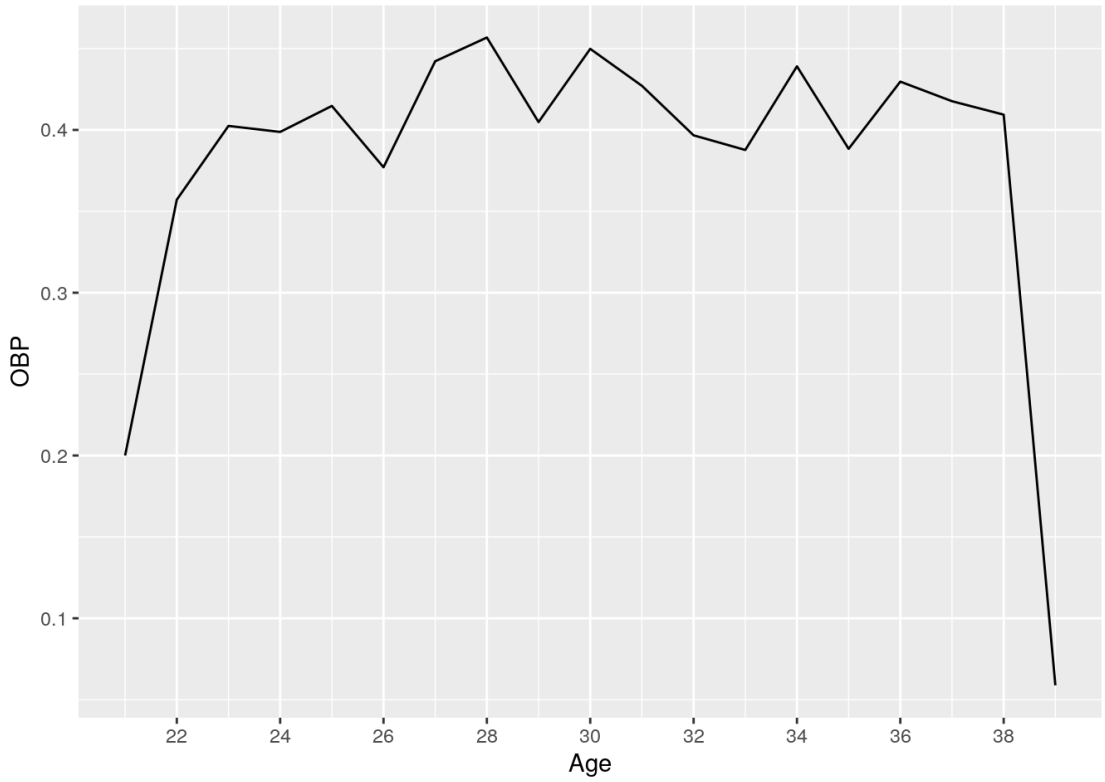
Code

Putting together some pieces

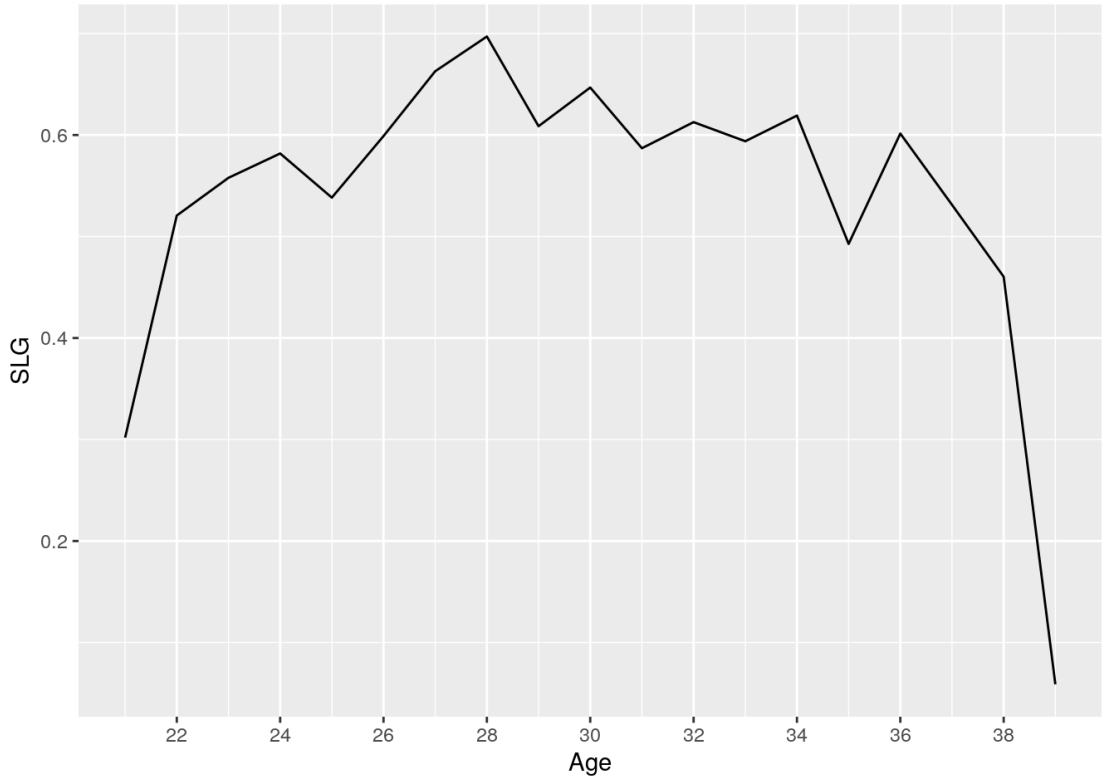
Exercise 3 Produce analogous plots using On-Base Percentage (OBP) and Slugging Average (SLG), and OPS (On-Base plus Slugging) instead of HR/162. If you are not familiar with baseball and/or baseball statistics, there's a quick run-down of definitions below the exercise. The formulas for OBP and SLG are as follows: $OBP = (H + BB + HBP)/(AB + BB + HBP + SF)$, where H stands for "hits" (an at-bat in which the player did not reach base safely after hitting the ball in play), BB stands for "base-on-balls" (a walk), HBP is a "hit by pitch". Slugging average is the average number of bases per at bat, and is calculated as a weighted sum of the number of singles, doubles, triples and home runs, divided by the number of at-bats: $SLG = (1 \times X1B + 2 \times X2B + 3 \times X3B + 4 \times HR)/AB$. Note that the data table does not include a separate column for singles, but every hit is either a single, a double, a triple or a home run, so it can be worked out. Finally, $OPS = OBP + SLG$.

Sample solution:

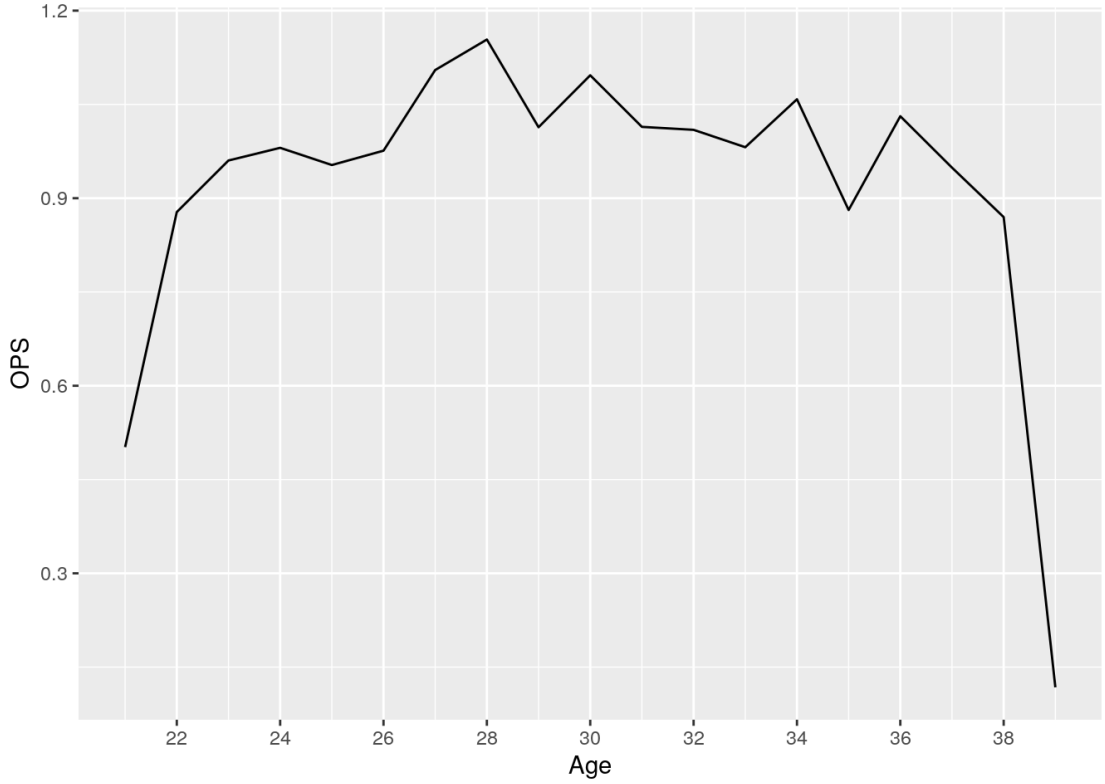
Code


OBP
0.4
0.3
0.2
0.1
22
24
26
28
30
32
34
36
38
Age

Code


SLG
0.6
0.4
0.2
22
24
26
28
30
32
34
36
38
Age

Code



For Reference: Baseball Hitting Statistics in a Nutshell

Each time a hitter appears at home plate to hit, the outcome is either

1. a **hit** (the player reached base safely after hitting the ball in play or out of the park as a home run)
2. a **walk** (the pitcher missed the strike zone four times without the batter swinging the bat)
3. a **"hit by pitch"** (the batter got hit by the pitch, and gets to go to first base)
4. **"sacrifice fly"** (the defending team catches the ball in the air so that the batter is out, but the ball is hit deep enough that a runner already on base can advance), or
5. some other kind of out.

Among hits * a **single** means the runner reached first base on the play * a **double** or a **triple** mean the batter reached second or third, respectively, and * a **home run** means the batter made it all the way around the bases (for a total of four bases on the play).

For the purposes of recording statistics, walks, being hit by a pitch, and sacrifice flies are not recorded as **"at bats"**, and so for the purposes of calculating the traditional "**batting average**", they don't factor in to either the numerator or the denominator: batting average (BA) is simply "hits" (H) divided by "at bats" (AB). Unlike batting average, "on-base percentage" counts any outcome in which the player ends up on the bases (hits, walks, hit by pitch) in the numerator, and counts every **plate appearance** (all of these things plus outs including sacrifice flies) in the denominator. If you've ever read the book or seen the movie *Moneyball* about the 2002 Oakland A's, one of the key insights that Billy Beane and his analysts had was that the league as a whole had been undervaluing outcomes in which the player reached base not via a hit, and so they tried to sign players with high OBPs relative to their batting average.

Comparing a hitter's stats to league average (wrangling together tables to be `join()`ed)

Around the turn of the millenium (coinciding with Manny Ramirez's peak), many hitters were putting up off the charts hitting numbers, which can at least in part be attributed to a high rate of steroid abuse during that time. If we want to know when a player (Ramirez, for example) provided the greatest "added offensive value" to his team, it would be instructive to know how well he was hitting in each season *relative to the rest of the league*.

The statistic OPS+ is defined as a player's OPS divided by the league average in that year, times 100 (so that 100 is league average, 150 means the player's OPS was 50% higher than league average, etc.). Like age, however, OPS+ depends on information from two different data tables; or at least two views of the `Batting` data: the numerator comes from data for a

specific player, and the denominator comes from data aggregated over players.

We can use a `join` to deal with this as well, but we first need to create the two tables (with the right type of aggregation) that we want to join.

We already know how to compute an individual player's OPS. To compute the average for the whole league, we can do the same thing but without filtering first.

Code:

Code

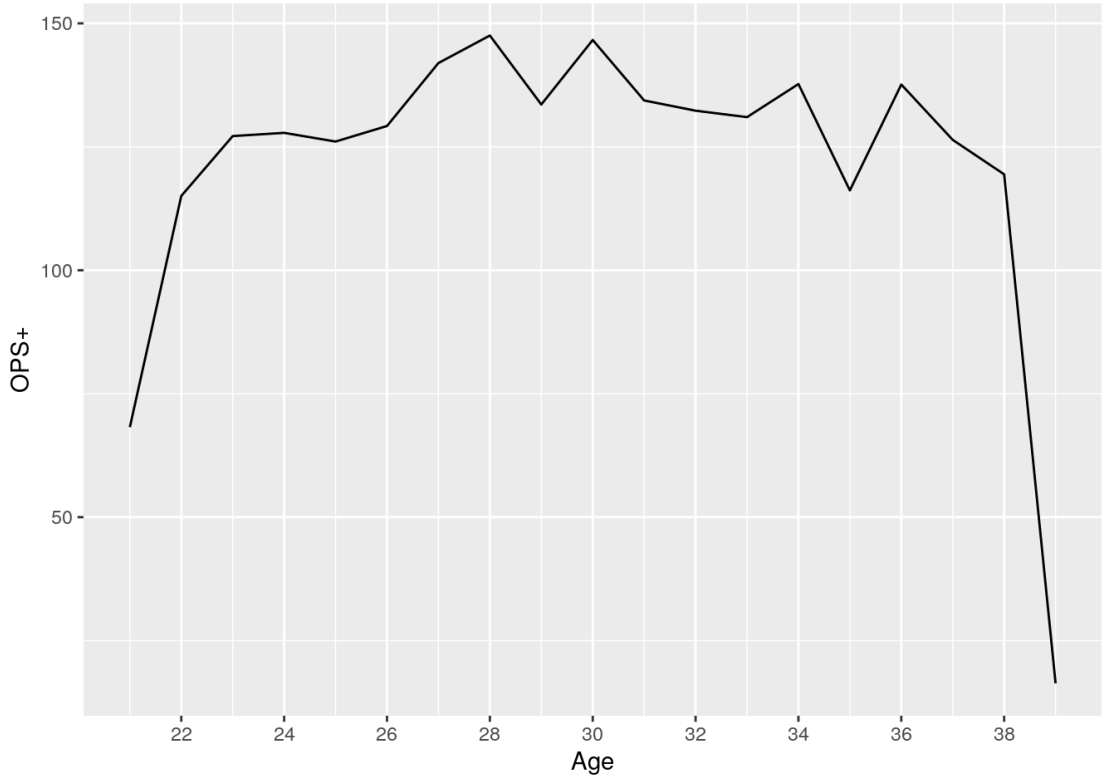
```
## # A tibble: 6 x 2
##   yearID lgOPS
##    <int> <dbl>
## 1   2013 0.714
## 2   2014 0.700
## 3   2015 0.721
## 4   2016 0.739
## 5   2017 0.750
## 6   2018 0.728
```

You might notice that very long-ago years are missing some of the components needed to compute OPS, and so we get "not a number" for league OPS since in some cases we end up trying to divide zero by zero. We could have filtered out these early years, but they won't cause a problem for what we want to do, since in `join` ing, they'll be left out anyway.

Exercise 4 Now that we have a dataset with league average OPS for each year, use a `join` to get Manny Ramirez's OPS+ for each season, and plot it as a time series.

Sample solution:

Code

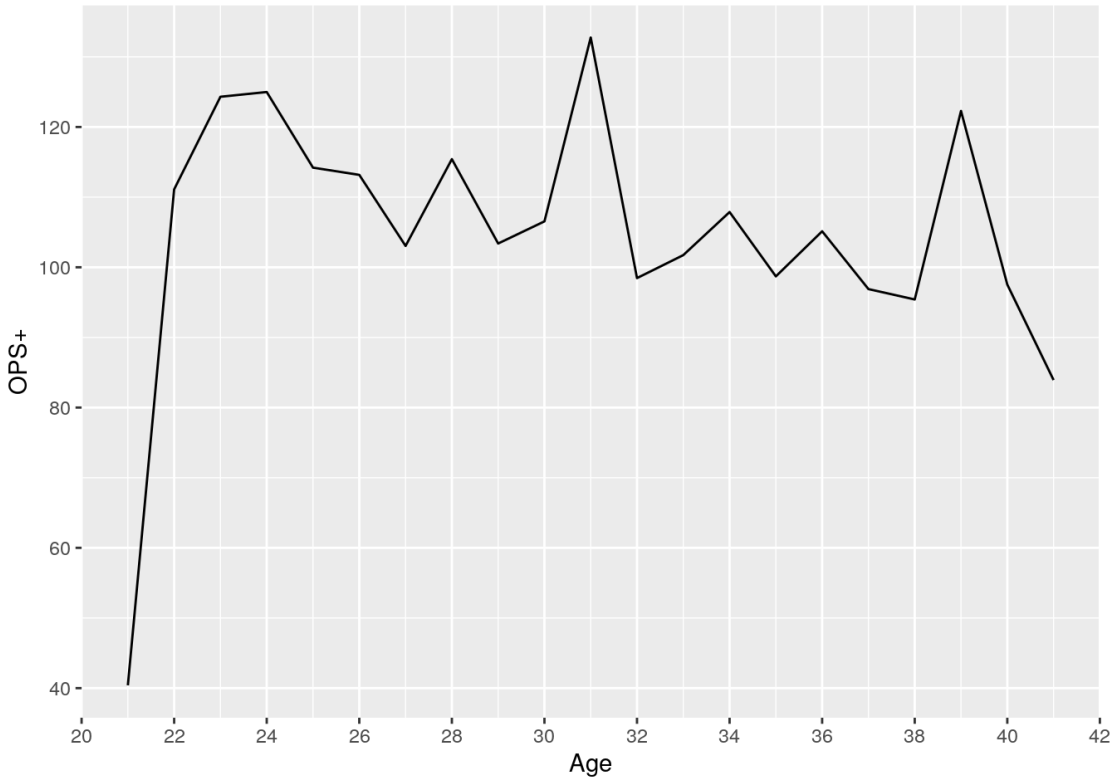


Beyond Manny Ramirez

Exercise 5 Produce a similar plot for another well known player of your choice (or if you're not familiar with baseball players, take, say, Cal Ripken). Does the career arc look similar to that of Manny Ramirez? **Note: Some names have more than one player in the database with that name. Cal Ripken is one (the most famous one is Cal Ripken, Jr.; his father also was an MLB player). To filter the data to include just one player, we'll want to extract the player ID for Cal Ripken, Jr. After filtering the Master table, to a table called, say, RipkenBioData, we can use the following syntax to get just the first entry in the "playerID" column:** `RipkenBioData %>% pull(playerID) %>% extract(1)` **, where** `extract()` **requires the** `magrittr` **package. It is of course possible to do this with base R syntax instead, but it's nice to keep as much as possible in pipeline form.**

Sample Solution:

Code



Overlaying Multiple Players (using a custom function)

Now let's try to overlay lines for several players on one graph, to facilitate comparison. Here's an arbitrary list of twelve very well known hitters spanning a wide range of decades, all of whom won the league Most Valuable Player award at least once: Ty Cobb, Babe Ruth, Lou Gehrig, Ted Williams, Jackie Robinson, Hank Aaron, Roberto Clemente, Reggie Jackson, Cal Ripken, Barry Bonds, Alex Rodriguez, Miguel Cabrera.

Exercise 6 Write a function that takes the player's first and last name as input, and produces a data frame that can be used for the plot. You'll need a way to return the `playerID` as a variable so that you can use it to filter the `Batting` data. You might need the `pull() %>% extract()` construction described above. For simplicity you can assume that, if there is more than one player with the first and last name given, that the one we are interested in is first in the list.

Sample Solution:

Code

```
## # A tibble: 22 x 17
## # Groups:   Age, playerID [22]
##      Age playerID yearID    AB     H    BB   HBP    SF   X2B   X3B    HR
##    <int> <chr>     <int> <int> <int> <int> <int> <int> <int> <int> <int>
##  1    19 ruthba01   1914    10     2     0     0    NA     1     0     0
##  2    20 ruthba01   1915    92    29     9     0    NA    10     1     4
##  3    21 ruthba01   1916   136    37    10     0    NA     5     3     3
##  4    22 ruthba01   1917   123    40    12     0    NA     6     3     2
##  5    23 ruthba01   1918   317    95    58     2    NA    26    11    11
##  6    24 ruthba01   1919   432   139   101     6    NA    34    12    29
##  7    25 ruthba01   1920   457   172   150     3    NA    36     9    54
##  8    26 ruthba01   1921   540   204   145     4    NA    44    16    59
##  9    27 ruthba01   1922   406   128    84     1    NA    24     8    35
## 10    28 ruthba01   1923   522   205   170     4    NA    45    13    41
## # … with 12 more rows, and 6 more variables: X1B <int>, OBP <dbl>,
## #   SLG <dbl>, OPS <dbl>, lgOPS <dbl>, OPSplus <dbl>
```

Applying our function to multiple players (`lapply()`)

Exercise 7 We already know how to use the `lapply()` function to call a particular function, varying the first argument. If we have a function with multiple arguments and we want to "walk" over more than one list, we can use the similar function `mapply()`, which has the following syntax: `mapply(FUN = <function_name>, <arg1name> = <list of arg1 values>, <arg2name> = <list of arg2values>, ...)`, and which, like `lapply()` returns a list of data frames that can be stacked with `bind_rows()` and the like. Use `mapply()` and `bind_rows()` with the function you wrote in the last exercise to produce a stacked data frame with the data for the players in the list above. Set `SIMPLIFY = FALSE` in `mapply()` to turn off the default behavior of "unlisting" the results so that `bind_rows()` can use them.

Sample Solution:

Code

```
## # A tibble: 6 x 17
## # Groups:   Age, playerID [6]
##     Age playerID yearID    AB     H    BB   HBP    SF   X2B   X3B    HR
##   <int> <chr>     <int> <int> <int> <int> <int> <int> <int> <int> <int>
## 1    19 cobbty01   1905   150    36    10     0    NA     6     0     1
## 2    20 cobbty01   1906   358   113    19     3    NA    15     5     1
## 3    21 cobbty01   1907   605   212    24     5    NA    28    14     5
## 4    22 cobbty01   1908   580   188    34     6    NA    36    20     4
## 5    23 cobbty01   1909   573   216    48     6    NA    33    10     9
## 6    24 cobbty01   1910   508   194    64     4    NA    35    13     8
## # … with 6 more variables: X1B <int>, OBP <dbl>, SLG <dbl>, OPS <dbl>,
## #   lgOPS <dbl>, OPSplus <dbl>
```

Putting your output to use (post `lapply()` wrangling)

Exercise 8 Use your "stacked" data frame to create the overlaid line graph that we set out to create in the first place. Include a legend that shows the player's last name (this might require revisiting your wrangling step).

Sample solution:

Code

```
## Warning: Removed 82 rows containing missing values (geom_path).
```

On Your Own

Exercise 9 Find each player's OPS+ peak season, and plot the age they were in that season against the year in which they made their major league debut. Use `geom_smooth()` to show a trendline. Has the age at which hitters "peak" changed over time?

Getting credit

1. Upload your plot from exercise 9 to the `#lab10` channel on Slack, along with the Honor Pledge and a brief comment on what problems you ran into and how you solved them.
2. In addition, save your `.Rmd` and Knitted HTML file in the `~/stat209/turnin/lab10/` folder on the RStudioPro server.