# Let's build a baseball projection system

Quinn Anhorn

29/12/2020

With the off-season upon us and with free agents hesitant to sign, I need something to keep myself occupied. And what better way to spend my time in my final term than to build a projection system for my favourite sport? Of course, I don't foresee it as being as robust or as thorough as industry favourites such as Steamer, Pecota, or ZiPS - but it should be a fun exercise. I have congregated a bunch of data in Excel as a learning exercise, now's the time to do the same in R and take it a step further.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(Lahman)
library(janitor)
```

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

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

Tidyverse and the Lahman database are the first two libraries I see us needing. I use the tidyverse and the libraries that come with it nearly every time I use R, and the Lahman database is useful when working with baseball statistics.

For now I will be focusing on batting statistics.

```
batting <- tibble(Batting)
View(batting)
```

As you can see, the Batting object contains basic batting statistics for every player from 1871 to 2019 (Stats for 2020 should be updated sometime before the 2021 season; maybe I'll scrape the data for 2020 before building models). In addition to the batting statistics, Batting also contains the year, along with the player id and the team id. These will be really useful.

Before we can build any models or projections, we will need to clean the data, as well as add on to it. While the Lahman database is excellent for basic stats such as batting average, home runs, and stolen bases, it is lacking in advanced statistics such as wOBA (weighted on base percentage) and WAR (wins above replacement).

The first decision I'm making is to disregard seasons before 2005. Why? For a few reasons. Baseball fans know that the offensive environment in baseball has varied greatly throughout the sport's history. It was harder to score a run in the dead ball era compared to modern day, for example. And the offensive era was high in the late 90s into the early 2000s as the steroid era was at its peak.

2005 is generally accepted as the end of the steroid era, and it gives us enough data to train our models on. In recent years the offensive environment has varied somewhat year-to-year as the "fly ball revolution" took off, along with the league altering the ball from season to season. However, this should not be too much of a factor.

(Note: AAA is about 80% as strong as MLB, AA about 70% as strong as MLB. Extrapolating A ball might be as 60% strong as MLB and perhaps rookie ball 50%. To keep things simple might use these ratios to convert minor league stats to major league.)

```
batting %>% filter(yearID >= 2005)
```

```
## # A tibble: 21,428 x 22
##    playerID yearID stint teamID lgID      G    AB     R     H   X2B   X3B    HR
##    <chr>     <int> <int> <fct>  <fct> <int> <int> <int> <int> <int> <int> <int>
##  1 abernbr~   2005     1 MIN    AL       24    67     5    16     1     0     1
##  2 abreubo~   2005     1 PHI    NL      162   588   104   168    37     1    24
##  3 accarje~   2005     1 SFN    NL       28     2     0     1     0     0     0
##  4 acevejo~   2005     1 COL    NL       36     8     0     1     0     0     0
##  5 adamsmi~   2005     1 MIL    NL       13     0     0     0     0     0     0
##  6 adamsru~   2005     1 TOR    AL      139   481    68   123    27     5     8
##  7 adamste~   2005     1 PHI    NL       16     0     0     0     0     0     0
##  8 adkinjo~   2005     1 CHA    AL        5     0     0     0     0     0     0
##  9 affelje~   2005     1 KCA    AL       49     0     0     0     0     0     0
## 10 aguilch~   2005     1 FLO    NL       65    78    11    19     3     0     0
## # ... with 21,418 more rows, and 10 more variables: RBI <int>, SB <int>,
## #   CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>, SH <int>, SF <int>,
## #   GIDP <int>
```

Now we can add in some statistics from the given data. Batting average, on base percentage, slugging percentage, and more can be added into the table.

```
batting_clean <- batting %>% filter(yearID >= 2005) %>%
  # I like lowercase variable names
  clean_names() %>%
  mutate("avg" = h / ab) %>%
  mutate("obp" = (h + bb + hbp) / (ab + bb + hbp + sf)) %>%
  mutate("slg" = (((h - x2b - x3b - hr) + x2b*2 + x3b*3 + hr*4)) / ab) %>%
  mutate("iso" = ((x2b + x3b*2 + hr*3) / (ab - avg))) %>%
  mutate("babip" = (h - hr) / (ab - so - hr + sf)) %>%
  mutate("pa" = ab + bb + hbp + sf + sh) %>%
```

```
  mutate("bb%" = bb / pa) %>%
  mutate("k%" = so / pa)
```

So we have some basic statistics. But now lets up our game and add in some more advanced batting statistics such as wOBA, WRC+, and WAR.For those unfamiliar with these statistics, Fangraphs has an excellent glossary at https://library.fangraphs.com/fangraphs-library-glossary/

All files can be found at Fangraphs.com.

```
guts <- read_csv("Fangraphs Leaderboard.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Season = col_double(),
##   wOBA = col_double(),
##   wOBAScale = col_double(),
##   wBB = col_double(),
##   wHBP = col_double(),
##   w1B = col_double(),
##   w2B = col_double(),
##   w3B = col_double(),
##   wHR = col_double(),
##   runSB = col_double(),
##   runCS = col_double(),
##   `R/PA` = col_double(),
##   `R/W` = col_double(),
##   cFIP = col_double()
## )
```

```
guts <- guts %>%
  rename(year_id = Season)
```

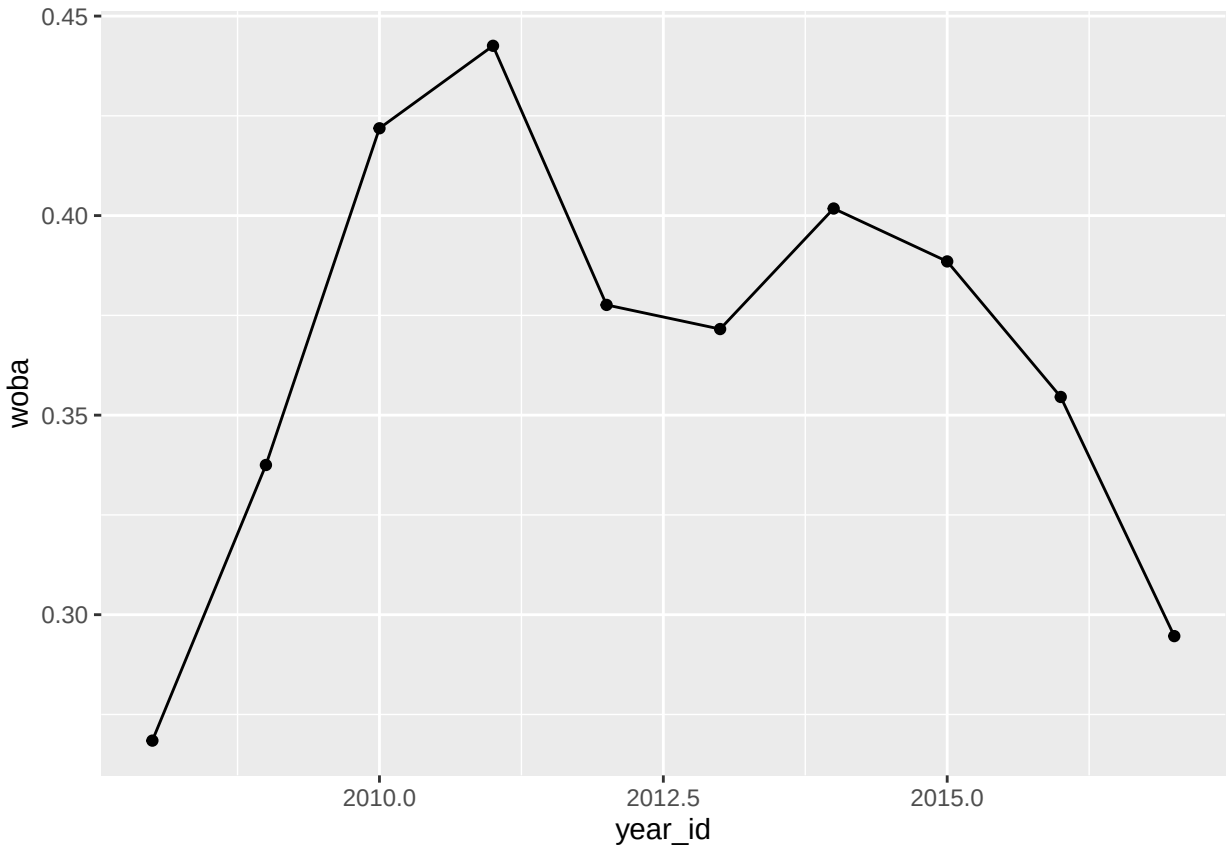
This file contains data I can use to calculate wOBA for each person. wOBA is a rate stat and average wOBA changes year per year as the offensive environment changes. The object guts now contains the linear weights for every year. Now all we have to do is join this data to batting_clean and run some calculations.

```
batting_clean <- batting_clean %>%
  left_join(guts, by = "year_id") %>%
  clean_names() %>%
  mutate("woba" =
           (((bb - ibb)*w_bb + hbp*w_hbp + (h - x2b - x3b - hr)*w1b + x2b*w2b +
           x3b*w3b + hr*w_hr)) / (ab + bb - ibb + sf + hbp))
```

Now we have each player's wOBA. As an illustration, let's chart Jose Bautista's Blue Jays career per season wOBA.

```
jose <- batting_clean %>%
  filter(player_id == "bautijo02", team_id == "TOR") %>%
  select(year_id, woba)
```

```
# quick and dirty graphs of Bautista's career wOBA in Toronto by season.
jose %>%
  ggplot(aes(year_id, woba)) +
  geom_line() +
  geom_point()
```




Now let's add in wRC+.

```
# First we find wRC (yes, there is a difference!).
batting_clean <- batting_clean %>%
  mutate("wrc" = (((woba - w_oba) / w_oba_scale) + r_pa) * pa)
```

Now we need to make a slight detour. wRC+ requires park factors, as well as the league runs per pa for each year.

```
factors <- read_csv("park_factors.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Season = col_double(),
##   Team = col_character(),
##   Basic = col_double(),
##   `1B` = col_double(),
##   `2B` = col_double(),
```

```
##   `3B` = col_double(),
##   HR = col_double(),
##   SO = col_double(),
##   BB = col_double(),
##   GB = col_double(),
##   FB = col_double(),
##   LD = col_double(),
##   IFFB = col_double(),
##   FIP = col_double()
## )
```

```
factors <- factors %>% select(Season, Team, Basic) %>%
  clean_names() %>%
  rename(year_id = season,
         team_id = team) %>%
  mutate("basic" = basic / 100)

# recode the values in the team_id column in factors
old <- c("Angels", "Orioles", "Red Sox", "White Sox", "Indians", "Tigers",
         "Royals", "Twins", "Yankees", "Athletics", "Mariners",
         "Devil Rays", "Rangers", "Blue Jays", "Diamondbacks", "Braves", "Cubs",
         "Reds", "Rockies", "Marlins", "Astros", "Dodgers", "Brewers",
         "Nationals", "Mets", "Phillies", "Pirates", "Cardinals", "Padres",
         "Giants")
new <- c("LAA", "BAL", "BOS", "CHA", "CLE", "DET", "KCA", "MIN", "NYA", "OAK",
         "SEA", "TBA", "TEX", "TOR", "ARI", "ATL", "CHN", "CIN", "COL", "FLO",
         "HOU", "LAN", "MIL", "WAS", "NYN", "PHI", "PIT", "SLN", "SDN", "SFN")

factors$team_id <- new[match(factors$team_id, old)]

factors[is.na(factors)] <- "TBA"
factors$team_id[factors$team_id == "FLO" & factors$year_id >= 2012] <- "MIA"

batting_clean <-batting_clean %>%
  left_join(factors, by = c("year_id", "team_id")) %>%
  rename(park_factor = basic)
```

Now that we have park factors included in our data, all we need now is the wrc per pa for each league per year to do some calculations.

```
totals <- read_csv("totals.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   `BB%` = col_character(),
##   `K%` = col_character(),
##   lg_id = col_character()
## )
## i Use `spec()` for the full column specifications.
```

```
totals <- totals %>%
  clean_names() %>%
  rename(year_id = season,
         pa_total = pa,
         `bb%_total` = bb_percent,
         `k%_total` = k_percent,
         avg_total = avg,
         obp_total = obp,
         slg_total = slg,
         iso_total = iso,
         babip_total = babip,
         w_oba_total = w_oba,
         )

batting_clean <- batting_clean %>%
  left_join(totals, by = c("lg_id", "year_id"))
```

Now we can calculate wRC+

```
batting_clean <- batting_clean %>%
  mutate("wrc+" = (((((woba - w_oba) / w_oba_scale) + r_pa) +
           (r_pa - (park_factor*r_pa))) / (w_rc / pa_total)) * 100)
```

Now we can begin the arduous task of calculating WAR. WAR, or wins above replacement, is an all inclusive statistic that attempts to qualify a player with a single number. As such, its calculation requires several steps, and more data than we have here.

I'll be following the example at https://library.fangraphs.com/calculating-position-player-war-a-complete-example/

From that source, the equation for WAR is WAR = (Batting Runs + Base Running Runs + Fielding Runs + Positional Adjustment + League Adjustment +Replacement Runs) / (Runs Per Win)

Let's start with batting runs.

```
batting_clean %>%
  mutate("wraa" = ((woba - w_oba) / w_oba_scale) * pa) %>%
  mutate("batting_runs" = wraa + ((r_pa - (park_factor*r_pa))*pa) +
           (r_pa - (w_rc / pa_total))*pa) %>% View()
```

Now we've run into a bit of a snag. Next we need data and stats from Fangraphs that we can't calculate on our own. And Fangraphs uses a differenct ID system compared to the Lahman database which uses baseball reference ID's. Luckily for me, I've found a baseball player ID excel file at (https://www.smartfantasybaseball.com/tools/) which I can use to let the data bases 'talk' to eachother. Should just be a matter of some left_joins and we will be back in business.

```
ids <- read_csv("SFBB-Player-ID-Map.csv")
```

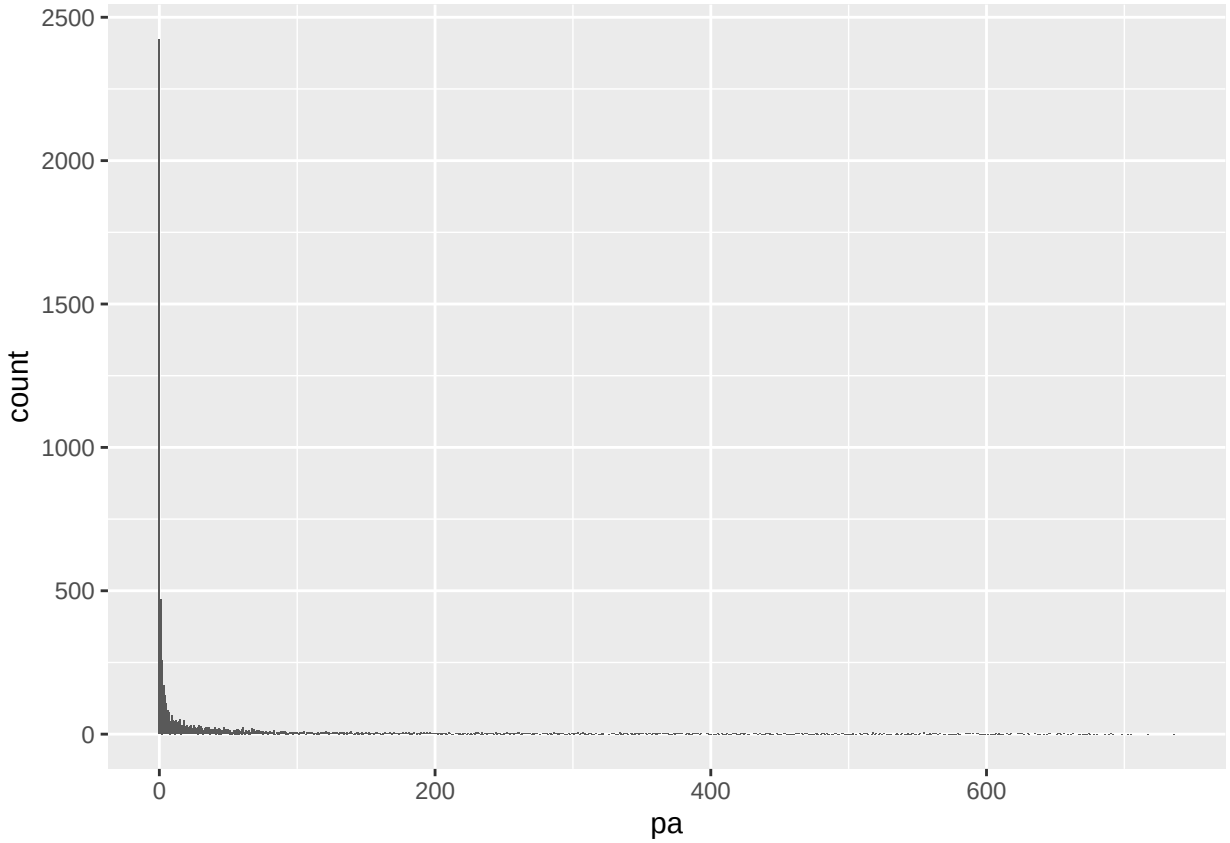
```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_character(),
##   BIRTHDATE = col_date(format = ""),
```

```
##   MLBID = col_double(),
##   CBSID = col_double(),
##   NFBCID = col_double(),
##   ESPNID = col_double(),
##   BPID = col_double(),
##   YAHOOID = col_double(),
##   ROTOWIREID = col_double(),
##   FANDUELID = col_double(),
##   OTTONEUID = col_double(),
##   HQID = col_double()
## )
## i Use `spec()` for the full column specifications.
```

```
ids <- ids %>% clean_names %>%
  select(idplayer, idfangraphs, playername) %>%
  rename(player_id = idplayer,
         player_name = playername,
         id_fangraphs = idfangraphs)
```

Now we've hit another snag. About 6500 of these players are missing from the ID database. But maybe that's not a problem. Let's take a look at the distribution of how many plate appearances these players had. If the majority had less than 200 or so plate appearances then they would have been removed when I set a threshold to start training the data. Let's take a peek.

```
batting_clean %>%
  left_join(ids, by = "player_id") %>%
  filter(is.na(player_name)) %>% ggplot(aes(pa)) +
  geom_bar()
```




From the bar plot, it looks like we we correct. Most of these players have no plate appearances at all, and the vast majority of them have less than 200.

```
batting_clean %>%
  left_join(ids, by = "player_id") %>%
  filter(pa >= 200) %>%
  filter(is.na(player_name))
```

```
## # A tibble: 763 x 67
##    player_id year_id stint team_id lg_id     g    ab     r     h   x2b   x3b
##    <chr>       <dbl> <int> <chr>   <chr> <int> <int> <int> <int> <int> <int>
##  1 adamsru01    2005     1 TOR     AL      139   481    68   123    27     5
##  2 alfoned01    2005     1 SFN     NL      109   368    36   102    17     1
##  3 aloumo01     2005     1 SFN     NL      123   427    67   137    21     3
##  4 anderga01    2005     1 LAA     AL      142   575    68   163    34     1
##  5 anderma02    2005     1 NYN     NL      123   235    31    62     9     0
##  6 ardoida01    2005     1 COL     NL       80   210    28    48    10     0
##  7 aurilri01    2005     1 CIN     NL      114   426    61   120    23     2
##  8 ausmubr01    2005     1 HOU     NL      134   387    35   100    19     0
##  9 barremi01    2005     1 CHN     NL      133   424    48   117    32     3
## 10 bellda01     2005     1 PHI     NL      150   557    53   138    31     1
## # ... with 753 more rows, and 56 more variables: hr <int>, rbi <int>, sb <int>,
## #   cs <int>, bb <int>, so <int>, ibb <int>, hbp <int>, sh <int>, sf <int>,
## #   gidp <int>, avg <dbl>, obp <dbl>, slg <dbl>, iso <dbl>, babip <dbl>,
## #   pa <int>, bb_percent <dbl>, k_percent <dbl>, w_oba <dbl>,
## #   w_oba_scale <dbl>, w_bb <dbl>, w_hbp <dbl>, w1b <dbl>, w2b <dbl>,
```

```
## #   w3b <dbl>, w_hr <dbl>, run_sb <dbl>, run_cs <dbl>, r_pa <dbl>, r_w <dbl>,
## #   c_fip <dbl>, woba <dbl>, wrc <dbl>, park_factor <dbl>, pa_total <dbl>,
## #   `bb%_total` <chr>, `k%_total` <chr>, bb_k <dbl>, avg_total <dbl>,
## #   obp_total <dbl>, slg_total <dbl>, ops <dbl>, iso_total <dbl>, spd <dbl>,
## #   babip_total <dbl>, ubr <dbl>, w_gdp <dbl>, w_sb <dbl>, w_rc <dbl>,
## #   w_raa <dbl>, w_oba_total <dbl>, w_rc_2 <dbl>, `wrc+` <dbl>,
## #   id_fangraphs <chr>, player_name <chr>
```

Only 763 player-seasons of our original sample of ~21 000 have greater than 200 plate appearances, or about 3.5%. This should not have much impact, if any at all later on.

```
batting_clean <- batting_clean %>%
  left_join(ids, by = "player_id") %>%
  filter(!is.na(player_name))
```

Of course, now that we have Fangraphs player IDs attached to the table, we can go ahead and save time by just attaching WAR itself to the table.

```
war <- read_csv("war.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Season = col_double(),
##   Name = col_character(),
##   Team = col_character(),
##   WAR = col_double(),
##   playerid = col_double()
## )
```

```
war <- war %>%
  clean_names() %>%
  rename(season_id = season,
         team_id = team,
         id_fangraphs = playerid)

war <- war %>%
  select(-name)
```

We're very close now to having our ideal data set. Next we're going to reuse our code to recode the team ids so we have no troubles joining this data to batting_clean.

```
war$team_id <- new[match(war$team_id, old)]

war[is.na(war)] <- "TBA"
war$team_id[war$team_id == "FLO" & war$season_id >= 2012] <- "MIA"

war <- war %>%
  rename(year_id = season_id)
```

Now we can join this to the main data set.

```
batting_clean$id_fangraphs <- as.numeric(as.character(batting_clean$id_fangraphs))
```

```
## Warning: NAs introduced by coercion
```

```
batting_clean <- batting_clean %>%
  left_join(war, by = c("id_fangraphs", "year_id", "team_id")) %>%
  filter(!is.na(id_fangraphs))
```

Now we have a table containing every stat we're interested in projecting for major league batters. Lets filter out some of the "helper" columns.

```
stats <- batting_clean %>%
  select(player_name, year_id, team_id, lg_id, g, ab, pa, r, h, x2b, x3b, hr,
         rbi, sb, cs, bb, so, ibb, hbp, sh, sf, gidp, avg, obp, slg, iso, babip,
         bb_percent, k_percent, woba, `wrc+`, war)

stats <-stats %>% filter(!is.na(war))
```

Here I've taken the stats we're interested in, filtered for at least 200 plate appearances per season, and filtered out players that didn't have at least 5 consecutive seasons of play. There are 367 such players.

```
stats <- stats %>%
  filter(pa > 199) %>%
  arrange(player_name) %>%
  group_by(player_name, grp = cumsum(c(1, diff(year_id)) != 1)) %>%
  filter(n() >= 5) %>%
  ungroup()
```

Many players have more than 5 years of data (in fact many have over 15!) but I still want to keep the data in five year increments. In this way a player's 6th year will count as year 1 again and we get to keep more data.

```
stats <- stats %>% group_by(player_name) %>% mutate(year = row_number()) %>%
  ungroup()

year_to_5 <- stats %>%
  filter(year <= 5)

year_to_10 <- stats %>%
  filter(year > 5 & year < 11) %>%
  group_by(player_name, grp = cumsum(c(1, diff(year_id)) != 1)) %>%
  filter(n() >= 5) %>%
  ungroup()

stats <- bind_rows(year_to_5, year_to_10)
```

This was a bit ugly, but now we have the ideal layout for our data to transform it even further. ACtually, it gets even uglier from here.

Now we need to convert each stat into a rate stat by dividing each stat by the number of plate appearances they had in each season, to account for playing time.

I'm going to write a function that aids in creating a new table. There's still a lot of typing and undoubtedly there's a better way to aggregate the data, but this will do for now.

```r
rate <- function(year1, year2, df, s){
  data <- df
  column <- s

yr1 <- data %>% filter(year == year1)
yr2 <- data %>% filter(year == year2)

vec_yr1 <- yr1[[column]] / yr1$pa
vec_yr2 <- yr2[[column]] / yr2$pa

vec <- c(vec_yr1, vec_yr2)

}

non_rate <- function(year1, year2, df, s){
  data <- df
  column <- s

  vec <- ifelse(data$year == year1 | data$year == year2,
                data[[column]], NA)

  vec <- vec[!is.na(vec)]
}

# could not work out how to do this in the previous function
# without breaking it
pa_rate <- function(year1, year2, df, s){
    data <- df
  column <- s

yr1 <- data %>% filter(year == year1)
yr2 <- data %>% filter(year == year2)

vec_yr1 <- yr1[[column]] / yr1$g
vec_yr2 <- yr2[[column]] / yr2$g

vec <- c(vec_yr1, vec_yr2)
}

reg <- tibble(hperpa_year1 = rate(1, 6, stats, "h"),
       hperpa_year2 = rate(2, 7, stats, "h"),
       hperpa_year3 = rate(3, 8, stats, "h"),
       hperpa_year4 = rate(4, 9, stats, "h"),
       hperpa_year5 = rate(5, 10, stats, "h"),
       x2bperpa_year1 = rate(1, 6, stats, "x2b"),
       x2bperpa_year2 = rate(2, 7, stats, "x2b"),
       x2bperpa_year3 = rate(3, 8, stats, "x2b"),
       x2bperpa_year4 = rate(4, 9, stats, "x2b"),
       x2bperpa_year5 = rate(5, 10, stats, "x2b"),
       x3bperpa_year1 = rate(1, 6, stats, "x3b"),
       x3bperpa_year2 = rate(2, 7, stats, "x3b"),
       x3bperpa_year3 = rate(3, 8, stats, "x3b"),
       x3bperpa_year4 = rate(4, 9, stats, "x3b"),
```

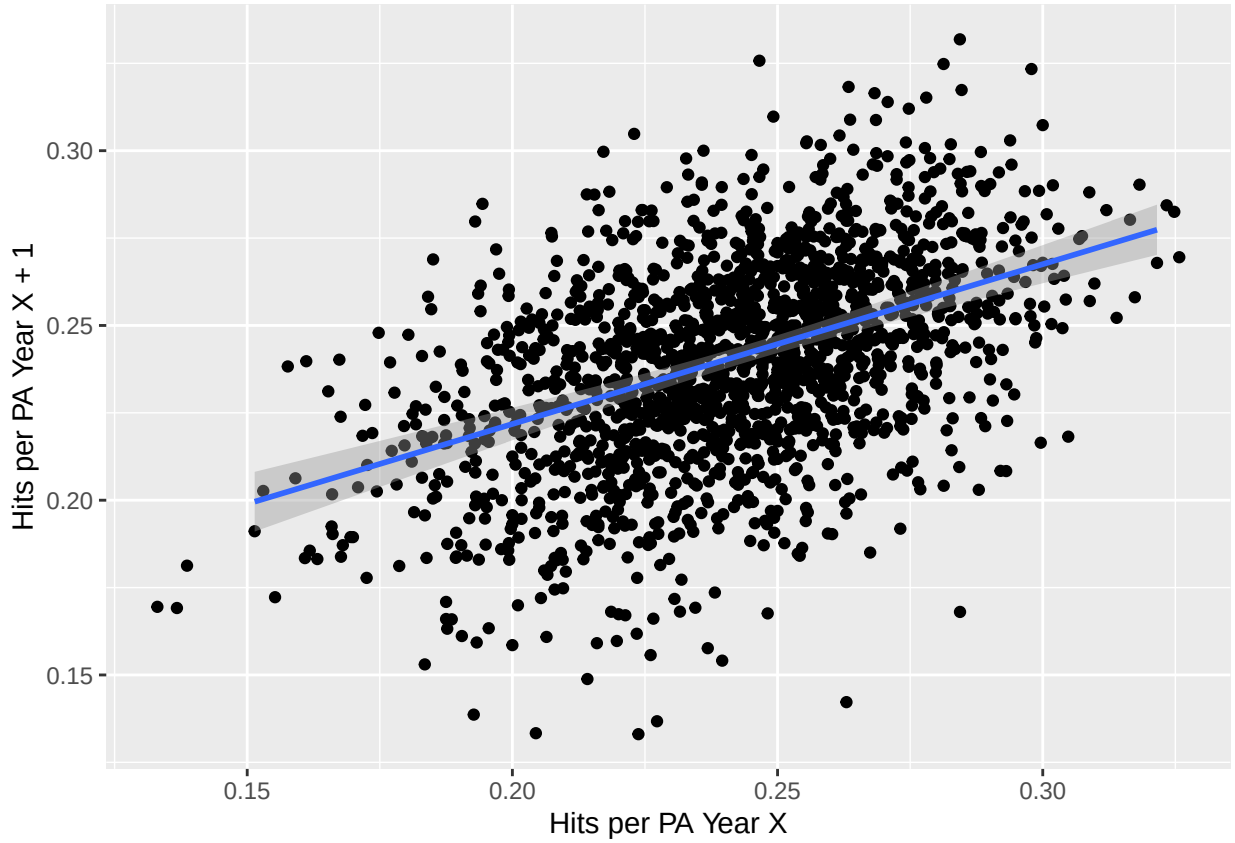
```
    x3bperpa_year5 = rate(5, 10, stats, "x3b"),
    hrperpa_year1 = rate(1, 6, stats, "hr"),
    hrperpa_year2 = rate(2, 7, stats, "hr"),
    hrperpa_year3 = rate(3, 8, stats, "hr"),
    hrperpa_year4 = rate(4, 9, stats, "hr"),
    hrperpa_year5 = rate(5, 10, stats, "hr"),
    rbiperpa_year1 = rate(1, 6, stats, "rbi"),
    rbiperpa_year2 = rate(2, 7, stats, "rbi"),
    rbiperpa_year3 = rate(3, 8, stats, "rbi"),
    rbiperpa_year4 = rate(4, 9, stats, "rbi"),
    rbiperpa_year5 = rate(5, 10, stats, "rbi"),
    sbperpa_year1 = rate(1, 6, stats, "sb"),
    sbperpa_year2 = rate(2, 7, stats, "sb"),
    sbperpa_year3 = rate(3, 8, stats, "sb"),
    sbperpa_year4 = rate(4, 9, stats, "sb"),
    sbperpa_year5 = rate(5, 10, stats, "sb"),
    csperpa_year1 = rate(1, 6, stats, "cs"),
    csperpa_year2 = rate(2, 7, stats, "cs"),
    csperpa_year3 = rate(3, 8, stats, "cs"),
    csperpa_year4 = rate(4, 9, stats, "cs"),
    csperpa_year5 = rate(5, 10, stats, "cs"),
    bbperpa_year1 = rate(1, 6, stats, "bb"),
    bbperpa_year2 = rate(2, 7, stats, "bb"),
    bbperpa_year3 = rate(3, 8, stats, "bb"),
    bbperpa_year4 = rate(4, 9, stats, "bb"),
    bbperpa_year5 = rate(5, 10, stats, "bb"),
    soperpa_year1 = rate(1, 6, stats, "so"),
    soperpa_year2 = rate(2, 7, stats, "so"),
    soperpa_year3 = rate(3, 8, stats, "so"),
    soperpa_year4 = rate(4, 9, stats, "so"),
    soperpa_year5 = rate(5, 10, stats, "so"),
    hbpperpa_year1 = rate(1, 6, stats, "hbp"),
    hbpperpa_year2 = rate(2, 7, stats, "hbp"),
    hbpperpa_year3 = rate(3, 8, stats, "hbp"),
    hbpperpa_year4 = rate(4, 9, stats, "hbp"),
    hbpperpa_year5 = rate(5, 10, stats, "hbp"),
    shperpa_year1 = rate(1, 6, stats, "sh"),
    shperpa_year2 = rate(2, 7, stats, "sh"),
    shperpa_year3 = rate(3, 8, stats, "sh"),
    shperpa_year4 = rate(4, 9, stats, "sh"),
    shperpa_year5 = rate(5, 10, stats, "sh"),
    sfperpa_year1 = rate(1, 6, stats, "sf"),
    sfperpa_year2 = rate(2, 7, stats, "sf"),
    sfperpa_year3 = rate(3, 8, stats, "sf"),
    sfperpa_year4 = rate(4, 9, stats, "sf"),
    sfperpa_year5 = rate(5, 10, stats, "sf"),
    gidpperpa_year1 = rate(1, 6, stats, "gidp"),
    gidpperpa_year2 = rate(2, 7, stats, "gidp"),
    gidpperpa_year3 = rate(3, 8, stats, "gidp"),
    gidpperpa_year4 = rate(4, 9, stats, "gidp"),
    gidpperpa_year5 = rate(5, 10, stats, "gidp"),
    woba_year1 = non_rate(1, 6, stats, "woba"),
    woba_year2 = non_rate(2, 7, stats, "woba"),
```

```
        woba_year3 = non_rate(3, 8, stats, "woba"),
        woba_year4 = non_rate(4, 9, stats, "woba"),
        woba_year5 = non_rate(5, 10, stats, "woba"),
        wrc_year1 = non_rate(1, 6, stats, "wrc+"),
        wrc_year2 = non_rate(2, 7, stats, "wrc+"),
        wrc_year3 = non_rate(3, 8, stats, "wrc+"),
        wrc_year4 = non_rate(4, 9, stats, "wrc+"),
        wrc_year5 = non_rate(5, 10, stats, "wrc+"),
        warperpa_year1 = rate(1, 6, stats, "war"),
        warperpa_year2 = rate(2, 7, stats, "war"),
        warperpa_year3 = rate(3, 8, stats, "war"),
        warperpa_year4 = rate(4, 9, stats, "war"),
        warperpa_year5 = rate(5, 10, stats, "war"),
        papergame_year1 = pa_rate(1, 6, stats, "pa"),
        papergame_year2 = pa_rate(2, 7, stats, "pa"),
        papergame_year3 = pa_rate(3, 8, stats, "pa"),
        papergame_year4 = pa_rate(4, 9, stats, "pa"),
        papergame_year5 = pa_rate(5, 10, stats, "pa"),
        g_year1 = non_rate(1, 6, stats, "g"),
        g_year2 = non_rate(2, 7, stats, "g"),
        g_year3 = non_rate(3, 8, stats, "g"),
        g_year4 = non_rate(4, 9, stats, "g"),
        g_year5 = non_rate(5, 10, stats, "g"))
```

Now we have data we can draw some models from. We have 438 observations and 85 total predictors. Of course, many of these predictors are highly correlated and most wouldn't make sense to use to predict another (predicting home runs from caught stealing is probably a futile attempt for example) - but we have more than enough data to get down to work.

```
reg %>%
  ggplot() +
  geom_point(aes(hperpa_year1, hperpa_year2)) +
  geom_point(aes(hperpa_year2, hperpa_year3)) +
  geom_point(aes(hperpa_year3, hperpa_year4)) +
  geom_point(aes(hperpa_year4, hperpa_year5)) +
  geom_smooth(aes(hperpa_year1, hperpa_year2), method = "lm") +
  xlab("Hits per PA Year X") +
  ylab("Hits per PA Year X + 1")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



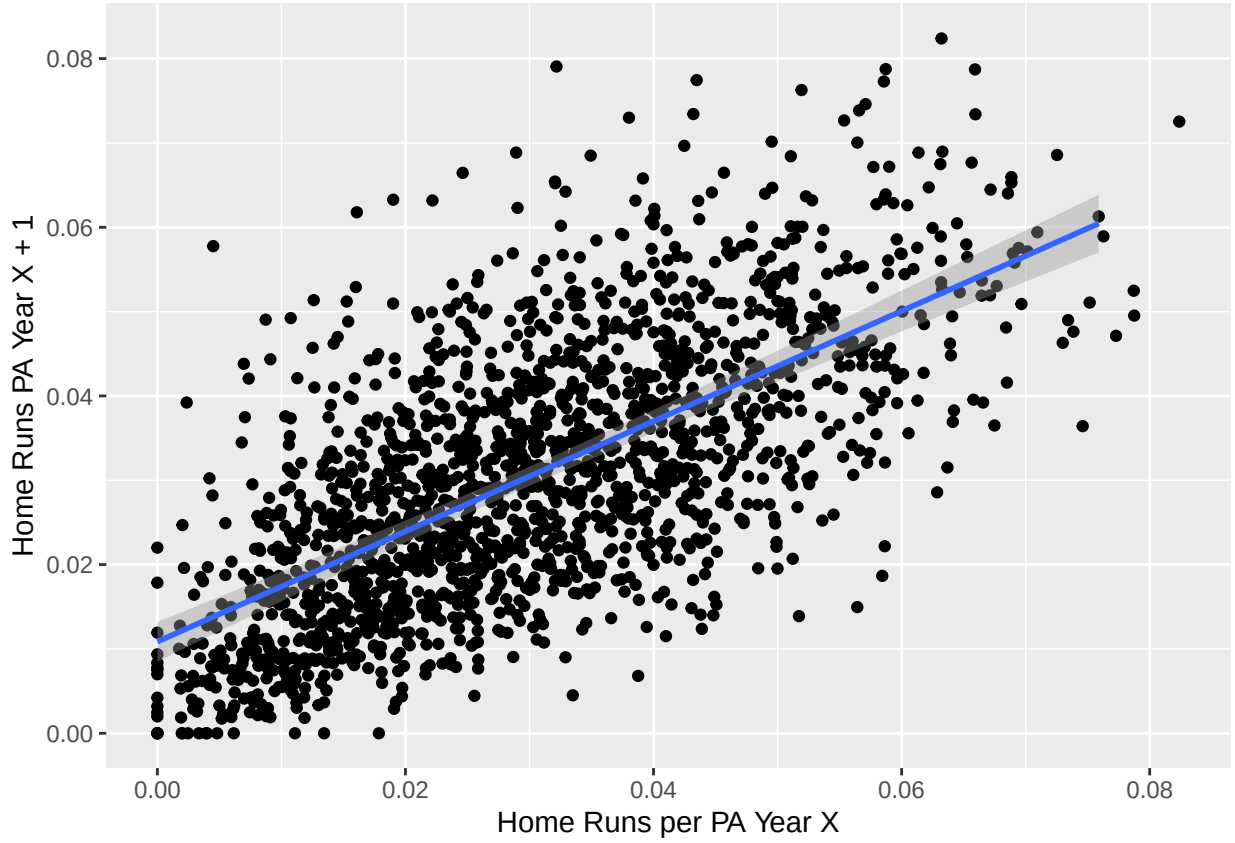

```
h_model <- lm(hperpa_year5 ~ hperpa_year4 + hperpa_year3 + hperpa_year2 +
                hperpa_year1, reg)

summary(h_model)
```

```
##
## Call:
## lm(formula = hperpa_year5 ~ hperpa_year4 + hperpa_year3 + hperpa_year2 +
##     hperpa_year1, data = reg)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.089445 -0.014810  0.000825  0.016659  0.066151
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.03632    0.01283   2.831  0.00485 **
## hperpa_year4  0.26551    0.04878   5.443 8.81e-08 ***
## hperpa_year3  0.25403    0.05141   4.941 1.11e-06 ***
## hperpa_year2  0.19417    0.04936   3.934 9.74e-05 ***
## hperpa_year1  0.11410    0.04799   2.378  0.01785 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02446 on 433 degrees of freedom
```

```
## Multiple R-squared:  0.3717, Adjusted R-squared:  0.3658
## F-statistic: 64.03 on 4 and 433 DF,  p-value: < 2.2e-16
```

```
reg %>%
  ggplot() +
  geom_point(aes(hrperpa_year1, hrperpa_year2)) +
  geom_point(aes(hrperpa_year2, hrperpa_year3)) +
  geom_point(aes(hrperpa_year3, hrperpa_year4)) +
  geom_point(aes(hrperpa_year4, hrperpa_year5)) +
  geom_smooth(aes(hrperpa_year1, hrperpa_year2), method = "lm") +
  xlab("Home Runs per PA Year X") +
  ylab("Home Runs PA Year X + 1")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



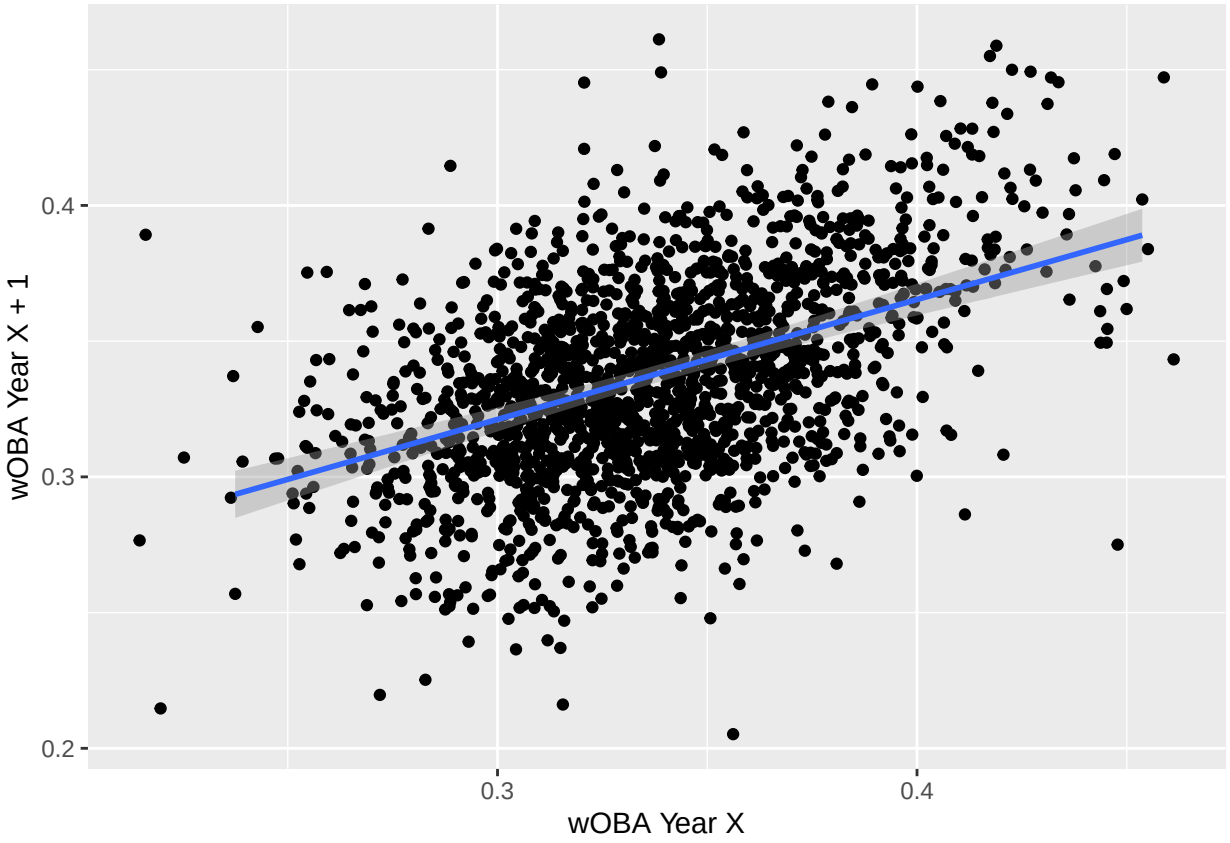

```
hr_model <- lm(hrperpa_year5 ~ hrperpa_year4 + hrperpa_year3 + hrperpa_year2 +
                hrperpa_year1, reg)

summary(hr_model)
```

```
##
## Call:
## lm(formula = hrperpa_year5 ~ hrperpa_year4 + hrperpa_year3 +
##     hrperpa_year2 + hrperpa_year1, data = reg)
```

```
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.028521 -0.008590 -0.001190  0.007633  0.046493
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.008146   0.001346   6.054 3.06e-09 ***
## hrperpa_year4 0.320633   0.053127   6.035 3.41e-09 ***
## hrperpa_year3 0.233782   0.057802   4.045 6.21e-05 ***
## hrperpa_year2 0.137647   0.052353   2.629  0.00886 **
## hrperpa_year1 0.088376   0.048862   1.809  0.07120 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01156 on 433 degrees of freedom
## Multiple R-squared:  0.4627, Adjusted R-squared:  0.4577
## F-statistic: 93.22 on 4 and 433 DF,  p-value: < 2.2e-16
```

```
reg %>%
  ggplot() +
  geom_point(aes(woba_year1, woba_year2)) +
  geom_point(aes(woba_year2, woba_year3)) +
  geom_point(aes(woba_year3, woba_year4)) +
  geom_point(aes(woba_year4, woba_year5)) +
  geom_smooth(aes(woba_year1, woba_year2), method = "lm") +
  xlab("wOBA Year X") +
  ylab("wOBA Year X + 1")
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
woba_model <- lm(woba_year5 ~ woba_year4 + woba_year3 + woba_year2 +
                 woba_year1, reg)

summary(woba_model)
```

```
##
## Call:
## lm(formula = woba_year5 ~ woba_year4 + woba_year3 + woba_year2 +
##     woba_year1, data = reg)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.123322 -0.019535  0.000833  0.022363  0.101324
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.08858    0.01765   5.019 7.58e-07 ***
## woba_year4   0.28232    0.05129   5.504 6.37e-08 ***
## woba_year3   0.20970    0.05475   3.830 0.000147 ***
## woba_year2   0.18882    0.05021   3.761 0.000193 ***
## woba_year1   0.04847    0.04549   1.066 0.287210
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03218 on 433 degrees of freedom
```

```
## Multiple R-squared:  0.319,  Adjusted R-squared:  0.3128
## F-statistic: 50.72 on 4 and 433 DF,  p-value: < 2.2e-16
```

I've made graphs of three statistics (hits, home runs, and woba) and plotted them against each other. As you can see, there is a linear relationship for all three, with home runs having the strongest relationship. However, even our linear model for home runs (predicting home runs in year 5 based on home runs in years 4, 3, 2, and 1) has an R^2 of just 0.46, meaning that this simple model explains 46% of the variability of the response data around its mean.However, this isn't unexpected; baseball is perhaps the most random of all the major sports, and it isn't unlikely that there are more predictors that can explain more variation. Furthermore, our p-values are very significant.

Lasso regression is a type of linear regression performs variable selection and regularization in order to produce a sparse model. In this way, it maximizes predictive accuracy. Lasso regression is a particularly good choice here where many of our predictors are correlated and we're not sure which ones to include, besides the obvious.

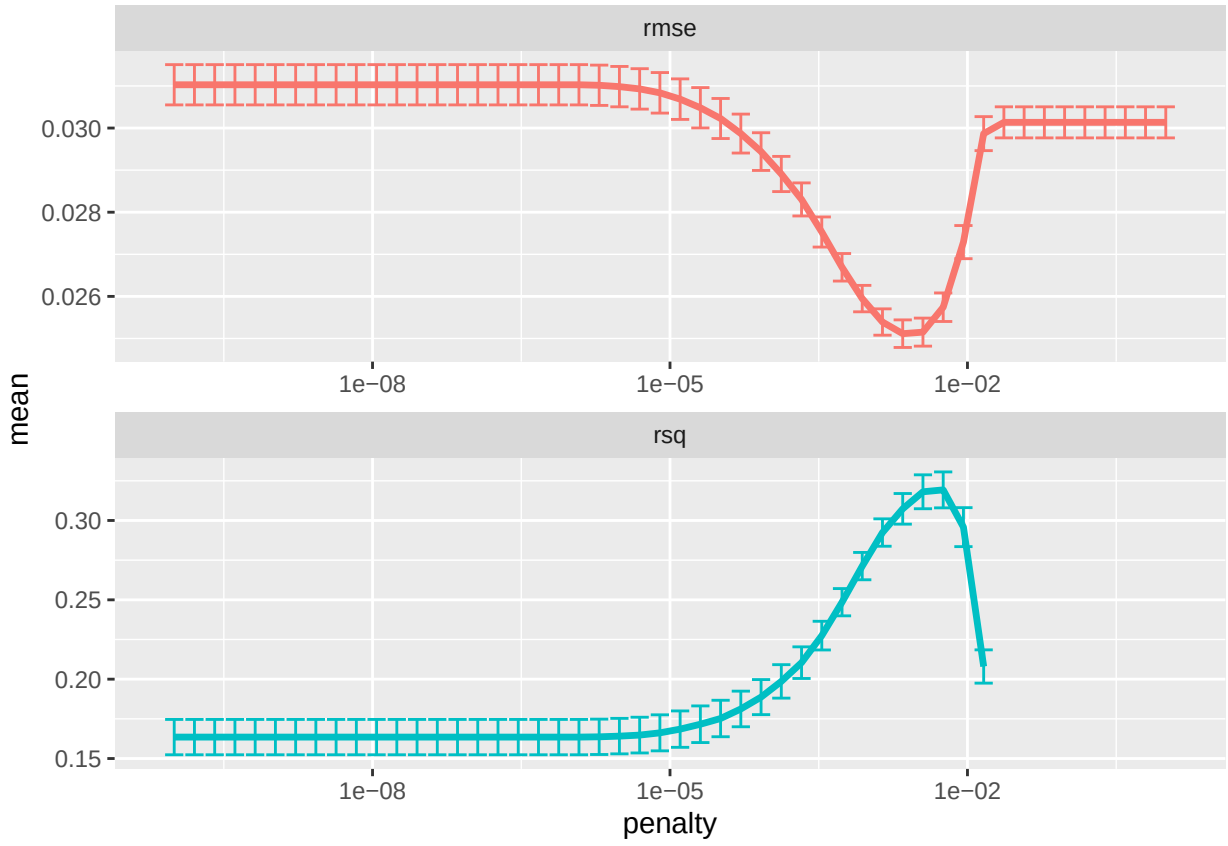
```
library(tidymodels)
```

```
## -- Attaching packages --------------------------------------- tidymodels 0.1.2 --

## v broom     0.7.2      v recipes   0.1.15
## v dials     0.0.9      v rsample   0.0.8
## v infer     0.5.3      v tune      0.1.2
## v modeldata 0.1.0      v workflows 0.2.1
## v parsnip   0.1.4      v yardstick 0.0.7

## -- Conflicts ------------------------------------------ tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
```

```
set.seed(2020)
# don't want to use "year5" stats for prediction
hits <- reg %>%
  select(-x2bperpa_year5, -x3bperpa_year5, -g_year5, -rbiperpa_year5,
         -soperpa_year5, -csperpa_year5, -sbperpa_year5, -wrc_year5,
         -hbpperpa_year5, -shperpa_year5, -sfperpa_year5, -gidpperpa_year5,
         -woba_year5, -warperpa_year5, -bbperpa_year5, -hrperpa_year5,
         -papergame_year5)

# Create the training and testing split for cross validation and
# to avoid over fitting the data
hits_split <- initial_split(hits)
hits_train <- training(hits_split)
hits_test <- testing(hits_split)

# lasso requires we center and scale our data as a pre-processing step
hits_rec <- recipe(hperpa_year5 ~ ., data = hits_train) %>%
  step_zv(all_numeric(), -all_outcomes()) %>%
  step_normalize(all_numeric(), -all_outcomes())
```

```
hits_prep <- hits_rec %>%
  prep()

# Set the model to lasso using glmnet
hits_spec <- linear_reg(penalty = 0.1, mixture = 1) %>%
  set_engine("glmnet")

hits_wf <- workflow() %>%
  add_recipe(hits_rec)

hits_fit <- hits_wf %>%
  add_model(hits_spec) %>%
  fit(data = hits_train)

# use re-sampling to tune the model
hits_boot <- bootstraps(hits_train)

hits_tune <- linear_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet")

hits_grid <- grid_regular(penalty(), levels = 50)

doParallel::registerDoParallel()


lasso_grid <- tune_grid(
  hits_wf %>% add_model(hits_tune),
  resamples = hits_boot,
  grid = hits_grid
)

# graph the rmse and rsq
lasso_grid %>%
  collect_metrics() %>%
  ggplot(aes(penalty, mean, color = .metric)) +
  geom_errorbar(aes(ymin = mean - std_err,
                    ymax = mean + std_err)) +
  geom_line(size = 1.2) +
  facet_wrap(~.metric, scales = "free", nrow = 2) +
  scale_x_log10() +
  theme(legend.position = "none")
```

```
## Warning: Removed 9 row(s) containing missing values (geom_path).
```




```
lowest_rmse <- lasso_grid %>%
  select_best("rmse")

final_hits <- finalize_workflow(hits_wf %>% add_model(hits_tune),
                  lowest_rmse)
library(vip)
```

```
##
## Attaching package: 'vip'

## The following object is masked from 'package:utils':
##
##     vi
```

```
importance <- final_hits %>%
  fit(hits_train) %>%
  pull_workflow_fit() %>%
  vi(lambda = lowest_rmse[["penalty"]])

final_fit <- final_hits %>%
  fit(hits_train)
```

Now we have the logic of our prediction model intact. All we have to do is run the same steps for each statistic we wish to predict, and we will have our complete model. However, let's shift gears for a minute.

Hitters age. They get old, and they become less productive. And young stars get more experience and become better as they age. We want to create aging curves for each statistic so we know how much to adjust up or down a player's stats depending on if we expect them to improve or decline as they age.

There are a few methods people have gone about this task in the past. The most common method is "The Delta Method", in which players are clumped together by season pairs grouped by age, and the collective differences in their stats between ages are found. This method has flaws, such as discarding any season that doesn't appear in a season-pair. Every player's final season is discarded, and player's who only appeared in one season aren't counted.

Rather than the delta method, we will use a general additive model (GAM) which doesn't throw out player seasons. The method is outlined at Baseball Prospectus (https://www.baseballprospectus.com/news/article/59972/the-delta-method-revisited/) and I follow their method closely. In fact, they even linked their R code which made recreating their findings much simpler.

```
library(mgcv)
```

```
## Loading required package: nlme
```

```
##
## Attaching package: 'nlme'
```

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

```
## This is mgcv 1.8-33. For overview type 'help("mgcv-package")'.
```

```
# Set defaults
season_min <- 1
pa_min <- 1


# Load data ; player stats from 2005 through 2019
players_raw <- read_csv("aging.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Season = col_double(),
##   Name = col_character(),
##   Team = col_character(),
##   Age = col_double(),
##   PA = col_double(),
##   H = col_double(),
##   `2B` = col_double(),
##   `3B` = col_double(),
##   HR = col_double(),
##   SB = col_double(),
##   CS = col_double(),
##   BB = col_double(),
##   SO = col_double(),
##   ISO = col_double(),
```

```
##   wOBA = col_double(),
##   `wRC+` = col_double(),
##   WAR = col_double(),
##   playerid = col_double()
## )
```

```
players_tidy <- players_raw %>%
  clean_names() %>%
  rename(woba = w_oba,
         wrc = w_rc) %>%
  mutate(hperpa = h/pa,
         x2bperpa = x2b/pa,
         x3bperpa = x3b/pa,
         hrperpa = hr/pa,
         sbperpa = sb/pa,
         csperpa = cs/pa,
         bbperpa = bb/pa,
         soperpa = so/pa,
         warperpa = war/pa) %>%
  select(-c(h, x2b, x3b, hr, sb, cs, bb, so, war))

### Centering around the mean ###
stat_sel <- function(df = players_tidy, stat, stat_denom = pa){

  stat <- enquo(stat)
  stat_denom <- enquo(stat_denom)

  df.1 <- df %>%
    group_by(season) %>%
    mutate(stat_sel = (!!stat) - weighted.mean((!!stat), (!!stat_denom))) %>%
    ungroup() %>%
    select(season, name, age, (!!stat), (!!stat_denom), stat_sel)
}

### Career stats functions ###
career_stats <- function(df, pa_min = 1, best = "high"){
  df.2 <- df %>%
    filter(pa >= pa_min[1])

  if(best == "high"){
    df.2 <- df.2 %>%
      group_by(name) %>%
      mutate(best_stat = max(stat_sel))

  } else {

    df.2 <- df.2 %>%
      group_by(name) %>%
      mutate(best_stat = min(stat_sel))
  }

  df.2 <- df.2 %>% mutate(
    first_year_mlb = min(season),
    first_year_age = ifelse(first_year_mlb == season, age, 0),
```

```
    last_year_mlb = max(season),
    last_year_age = ifelse(last_year_mlb == season, age, 0),
    stat_peak = ifelse(best_stat == stat_sel, 1, 0),
    peak_at_age = ifelse(stat_peak == 1, age, 0),
    lag_stat = lag(stat_sel),
    lag_pa = lag(pa),
    delta_stat = stat_sel - lag_stat) %>%
    summarise(seasons = n(),
              first_year_age = max(first_year_age),
              last_year_age = max(last_year_age),
              last_year_mlb = max(last_year_mlb),
              career_mean = weighted.mean(stat_sel, pa),
              career_pa = sum(pa),
              career_best_stat = max(best_stat),
              stat_age_peak = max(peak_at_age),
              mean_delta = mean(delta_stat, na.rm = TRUE), .groups = "drop")
  return(df.2)
}

### inter-player method ###
inter_player <- function(season_data, career_data, pa_min = 1, season_min = 1){

  df.3 <- career_data %>%
    left_join(season_data) %>%
    filter(pa >= pa_min[1]) %>%
    filter(seasons >= season_min[1]) %>%
    mutate(rel_age = age - stat_age_peak) %>%
    select(season, name:stat_age_peak, stat_sel:rel_age, age, pa)

  return(df.3)

}

get_aging_models <- function(stat, df = players_tidy, stat_denom = pa){
  stat <- enquo(stat)
  stat_denom <- enquo(stat_denom)

  seasons <- stat_sel(df, !!stat, !!stat_denom)
  careers <- career_stats(seasons)
  inter_player <- inter_player(seasons, careers)

  years <- tibble(age = seq.int(21, 41))
  model <- gam(stat_sel ~ s(age) + career_mean,
               data = inter_player %>% filter(age >= 20 & age <= 41),
               weights = pa)

  years$result <- predict(model, newdata = years %>% mutate(
    career_mean = weighted.mean(inter_player$career_mean, inter_player$pa)
  ))

  return(years)
}
```

```r
library(scales)

# Making some graphs
theme_set(theme_light())

woba <- get_aging_models(woba)
```

```
## Joining, by = "name"
```

```r
hperpa <- get_aging_models(hperpa)
```

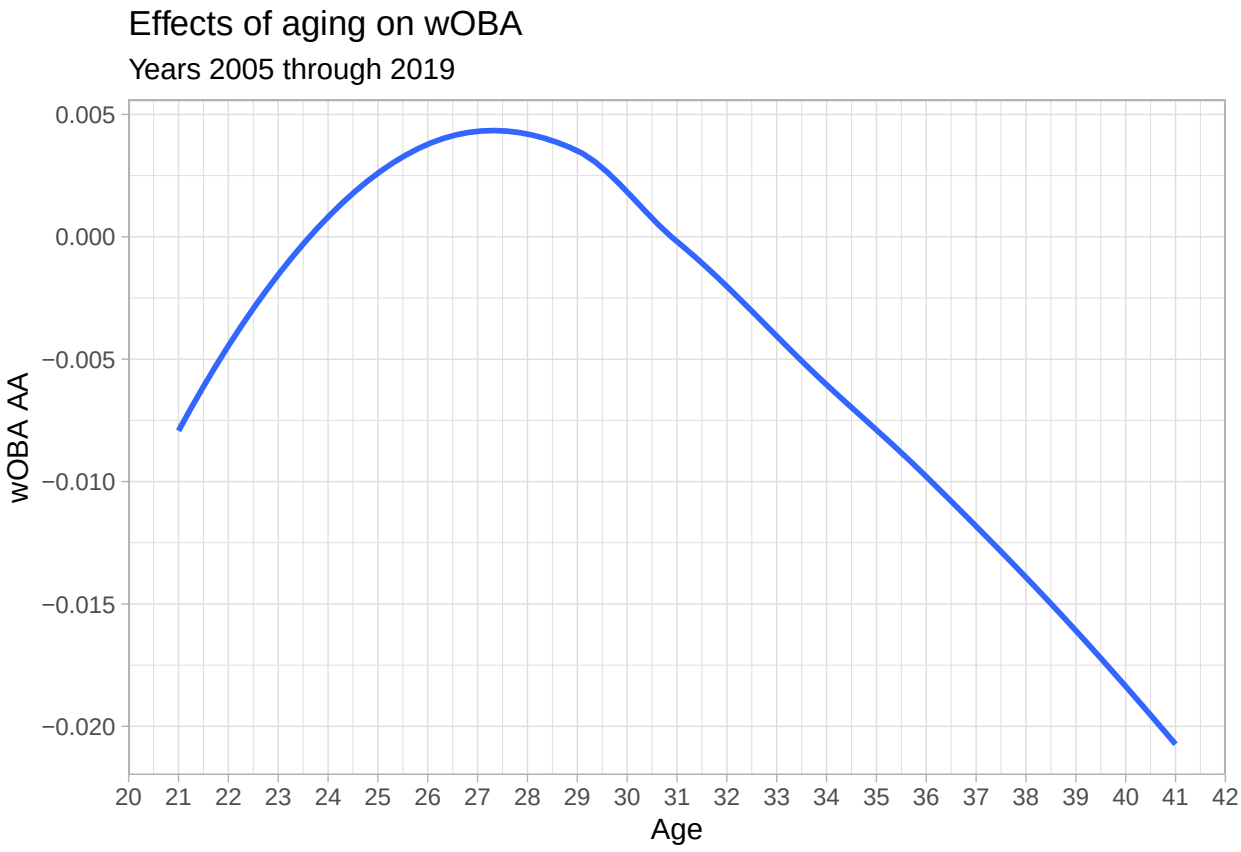
```
## Joining, by = "name"
```

```r
bb <- get_aging_models(bbperpa)
```

```
## Joining, by = "name"
```

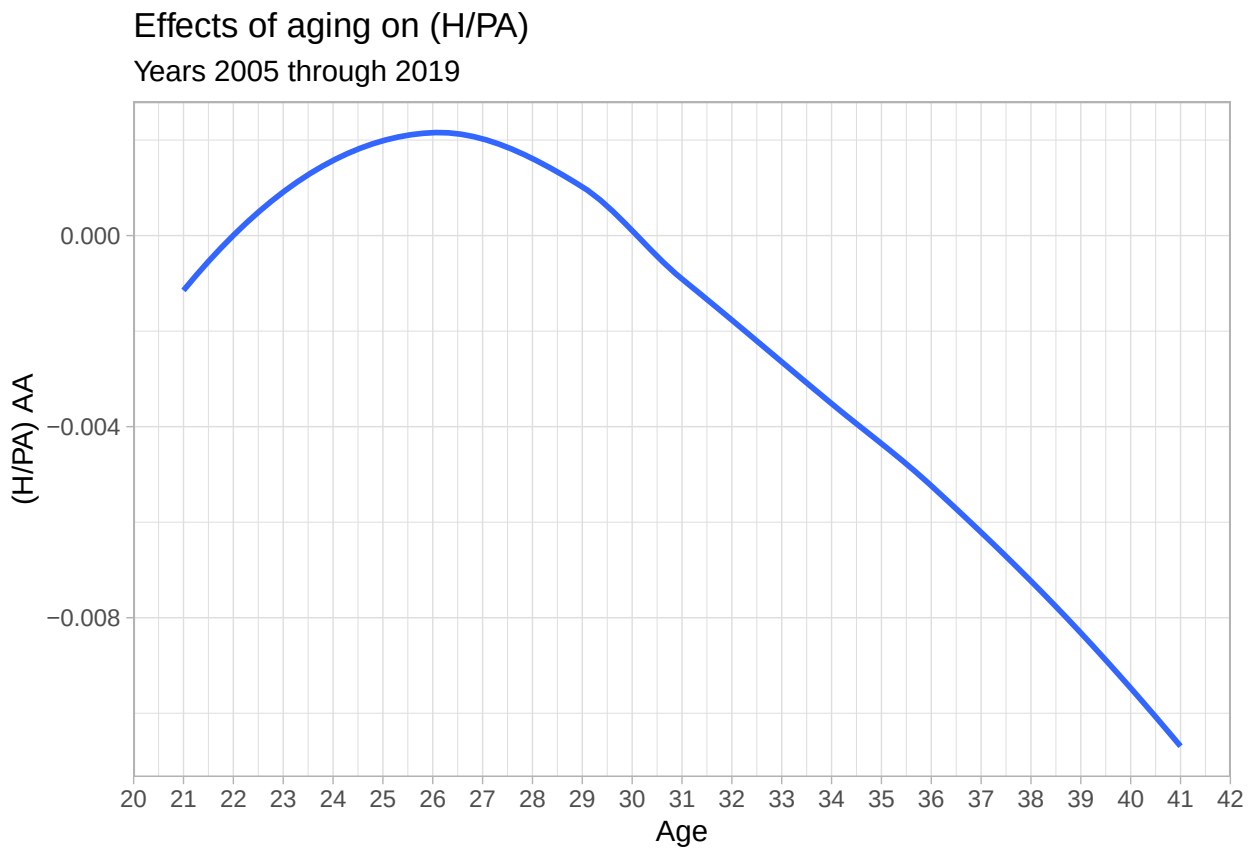
```r
woba %>%
  ggplot(aes(age, result)) +
  geom_smooth(se = FALSE) +
  labs(title = "Effects of aging on wOBA",
       subtitle = "Years 2005 through 2019",
       x = "Age",
       y = "wOBA AA") +
  scale_x_continuous(n.breaks = 20)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



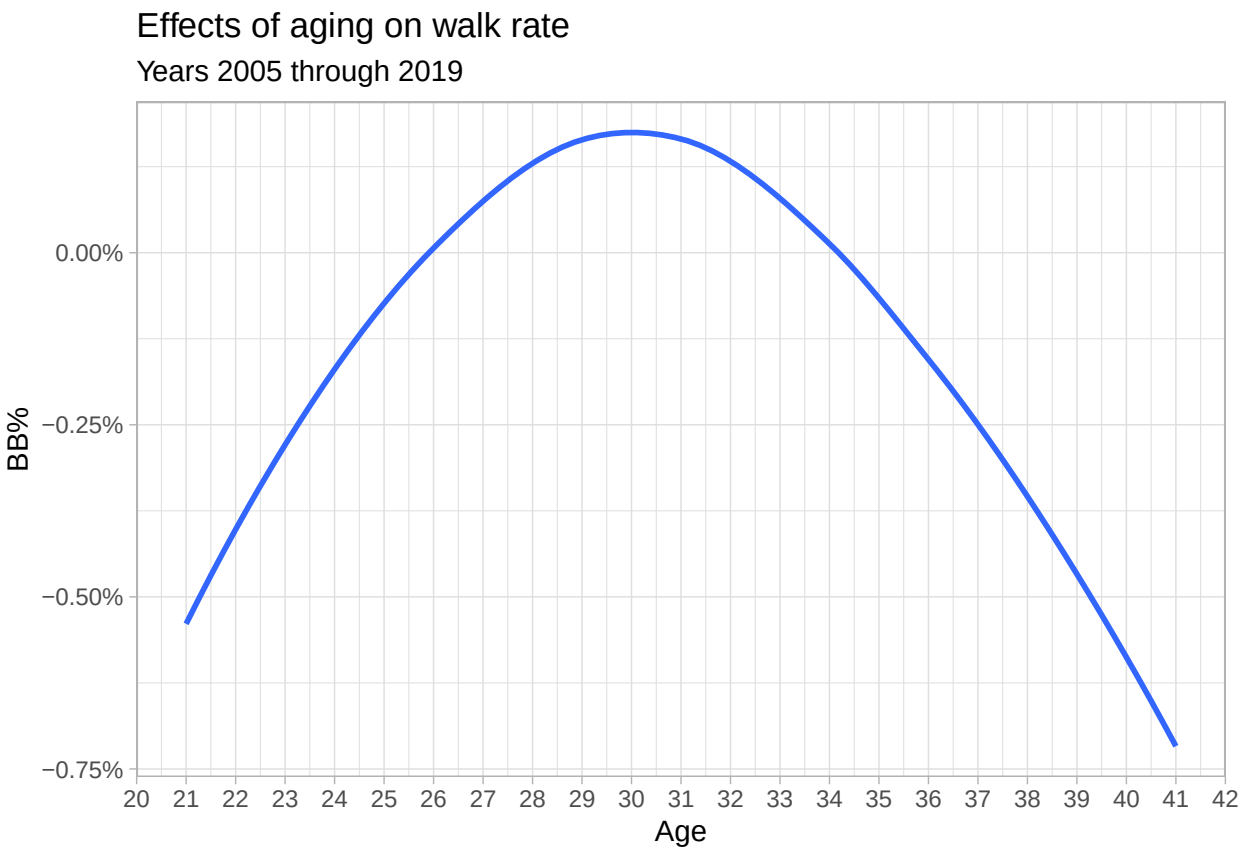

```
hperpa %>%
  ggplot(aes(age, result)) +
  geom_smooth(se = FALSE) +
  labs(title = "Effects of aging on (H/PA)",
       subtitle = "Years 2005 through 2019",
       x = "Age",
       y = "(H/PA) AA") +
  scale_x_continuous(n.breaks = 20)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
bb %>%
  ggplot(aes(age, result)) +
  geom_smooth(se = FALSE) +
  labs(title = "Effects of aging on walk rate",
       subtitle = "Years 2005 through 2019",
       x = "Age",
       y = "BB%") +
  scale_x_continuous(n.breaks = 20) +
  scale_y_continuous(labels = scales::percent)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




As we can see, wOBA rapidly improves between the ages 21-24 before peaking at the age of 28, and then steadily declining.

Hit rate gradually improves once the player enters the league and peaks a year earlier at 27 before beginning to decline.

BB% is a different story; walk rate does not peak until age 30 and doesn't become below league average until 35. This makes sense as batters become more patient as their contact rates decline. However, pitchers may also attack the batter more as those contact rates decline.

## Minor Leaguers

We're almost done with our projection system. However, you may have noticed that our models only work for batters with at least four years in the major leagues. We need to create models that project major league performance based on their performance in the minor leagues.

To do this I'll be using a combination of a hitters AA and A+ stats. I believe these stats to be the most predictive as good hitters often skip AAA entirely and what age a batter is in Rookie ball can be many years away from when they debut in the majors. AA and A+ stats offer a good middle ground.

Fangraphs only offers minor league stats dating back to 2006, so that's what we'll be using.

```
# Load in the data
# Players who never played in MLB have no player id
a_standard <- read_csv("a-standard-stats.csv") %>%
  filter(!is.na(PlayerId)) %>%
  mutate(Level = "A")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   Name = col_character(),
##   Team = col_character(),
##   Level = col_character()
## )
## i Use `spec()` for the full column specifications.

## Warning: 8450 parsing failures.
##  row      col expected    actual                   file
## 1474 PlayerId a double sa105000  'a-standard-stats.csv'
## 1475 PlayerId a double sa1052903 'a-standard-stats.csv'
## 1476 PlayerId a double sa1052944 'a-standard-stats.csv'
## 1477 PlayerId a double sa1115752 'a-standard-stats.csv'
## 1478 PlayerId a double sa1115767 'a-standard-stats.csv'
## .... ........ ........ ......... ......................
## See problems(...) for more details.
```

```
a_advanced <- read_csv("a-advanced-stats.csv") %>%
  filter(!is.na(PlayerId)) %>%
  mutate(Level = "A")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   Name = col_character(),
##   Team = col_character(),
##   Level = col_character()
## )
## i Use `spec()` for the full column specifications.

## Warning: 8450 parsing failures.
##  row      col expected    actual                   file
## 1474 PlayerId a double sa105000  'a-advanced-stats.csv'
## 1475 PlayerId a double sa1052903 'a-advanced-stats.csv'
## 1476 PlayerId a double sa1052944 'a-advanced-stats.csv'
## 1477 PlayerId a double sa1115752 'a-advanced-stats.csv'
## 1478 PlayerId a double sa1115767 'a-advanced-stats.csv'
## .... ........ ........ ......... ......................
## See problems(...) for more details.
```

```
high_a_standard <- read_csv("high-a-standard-stats.csv") %>%
  filter(!is.na(PlayerId)) %>%
  mutate(Level = "A+")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
```

```
##   Name = col_character(),
##   Team = col_character(),
##   Level = col_character()
## )
## i Use `spec()` for the full column specifications.

## Warning: 8364 parsing failures.
##  row      col expected    actual                         file
## 2663 PlayerId a double sa105000  'high-a-standard-stats.csv'
## 2664 PlayerId a double sa1115979 'high-a-standard-stats.csv'
## 2665 PlayerId a double sa1159952 'high-a-standard-stats.csv'
## 2666 PlayerId a double sa1170568 'high-a-standard-stats.csv'
## 2667 PlayerId a double sa183957  'high-a-standard-stats.csv'
## .... ........ ........ ......... ............................
## See problems(...) for more details.
```

```
high_a_advanced <- read_csv("high-a-advanced-stats.csv") %>%
  filter(!is.na(PlayerId)) %>%
  mutate(Level = "A+")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   Name = col_character(),
##   Team = col_character(),
##   Level = col_character()
## )
## i Use `spec()` for the full column specifications.

## Warning: 8364 parsing failures.
##  row      col expected    actual                         file
## 2663 PlayerId a double sa105000  'high-a-advanced-stats.csv'
## 2664 PlayerId a double sa1115979 'high-a-advanced-stats.csv'
## 2665 PlayerId a double sa1159952 'high-a-advanced-stats.csv'
## 2666 PlayerId a double sa1170568 'high-a-advanced-stats.csv'
## 2667 PlayerId a double sa183957  'high-a-advanced-stats.csv'
## .... ........ ........ ......... ............................
## See problems(...) for more details.
```

```
aa_standard <- read_csv("aa-standard-stats.csv") %>%
  filter(!is.na(PlayerId)) %>%
  mutate(Level = "AA")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   Name = col_character(),
##   Team = col_character(),
##   Level = col_character()
## )
## i Use `spec()` for the full column specifications.
```

```
## Warning: 9481 parsing failures.
##  row      col expected   actual                    file
## 6070 PlayerId a double sa168567 'aa-standard-stats.csv'
## 6071 PlayerId a double sa183957 'aa-standard-stats.csv'
## 6072 PlayerId a double sa183957 'aa-standard-stats.csv'
## 6073 PlayerId a double sa184229 'aa-standard-stats.csv'
## 6074 PlayerId a double sa184238 'aa-standard-stats.csv'
## .... ........ ........ ........ ......................
## See problems(...) for more details.
```

```
aa_advanced <- read_csv("aa-advanced-stats.csv") %>%
  filter(!is.na(PlayerId)) %>%
  mutate(Level = "AA")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   Name = col_character(),
##   Team = col_character(),
##   Level = col_character()
## )
## i Use `spec()` for the full column specifications.

## Warning: 9481 parsing failures.
##  row      col expected   actual                    file
## 6070 PlayerId a double sa168567 'aa-advanced-stats.csv'
## 6071 PlayerId a double sa183957 'aa-advanced-stats.csv'
## 6072 PlayerId a double sa183957 'aa-advanced-stats.csv'
## 6073 PlayerId a double sa184229 'aa-advanced-stats.csv'
## 6074 PlayerId a double sa184238 'aa-advanced-stats.csv'
## .... ........ ........ ........ ......................
## See problems(...) for more details.
```

```
# All we want from the "advanced" stats data frames is wOBA and wRC+
a_advanced_tidy <- a_advanced %>%
  select(wOBA, `wRC+`, PlayerId, Name)

high_a_advanced_tidy <- high_a_advanced %>%
  select(wOBA, `wRC+`, PlayerId, Name)

aa_advanced_tidy <- aa_advanced %>%
  select(wOBA, `wRC+`, PlayerId, Name)

# Join these into their respective standard stat data frames
a_standard %>%
left_join(a_advanced_tidy)
```

```
## Joining, by = c("Name", "PlayerId")

## # A tibble: 2,325 x 28
##    Season Name  Team  Level   Age     G    AB    PA     H  `1B`  `2B`  `3B`
```

```
##     <dbl> <chr> <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1   2010 Davi~ SDP   A        35     3     9    12     3     2     1     0
##  2   2009 Aram~ CHC   A        31     3     6     9     3     2     1     0
##  3   2009 Aram~ CHC   A        31     3     6     9     3     2     1     0
##  4   2009 Aram~ CHC   A        31     3     6     9     3     2     1     0
##  5   2010 Aram~ CHC   A        32     2     6     8     1     1     0     0
##  6   2010 Aram~ CHC   A        32     2     6     8     1     1     0     0
##  7   2010 Aram~ CHC   A        32     2     6     8     1     1     0     0
##  8   2014 Aram~ MIL   A        36     2     6     6     2     2     0     0
##  9   2014 Aram~ MIL   A        36     2     6     6     2     2     0     0
## 10   2014 Aram~ MIL   A        36     2     6     6     2     2     0     0
## # ... with 2,315 more rows, and 16 more variables: HR <dbl>, R <dbl>,
## #   RBI <dbl>, BB <dbl>, IBB <dbl>, SO <dbl>, HBP <dbl>, SF <dbl>, SH <dbl>,
## #   GDP <dbl>, SB <dbl>, CS <dbl>, AVG <dbl>, PlayerId <dbl>, wOBA <dbl>,
## #   `wRC+` <dbl>
```

```
# left_join() is giving us duplicate rows - there must be meta data in the
# file that is marking the rows as unique as even when using distinct() the rows
# remain. Base R to the rescue!
a_standard$woba <- a_advanced_tidy$wOBA
a_standard$`wrc+` <- a_advanced_tidy$`wRC+`

high_a_standard$woba <- high_a_advanced_tidy$wOBA
high_a_standard$`wrc+` <- high_a_advanced_tidy$`wRC+`

aa_standard$woba <- aa_advanced_tidy$wOBA
aa_standard$`wrc+` <- aa_advanced_tidy$`wRC+`

# Bind all the data frames into one master df
master_df <- a_standard %>%
  bind_rows(high_a_standard) %>%
  bind_rows(aa_standard)

# Now to condense stats from all three levels into one
combined_minors <- master_df %>%
  group_by(Name) %>%
  filter(Age <= 24) %>%
  mutate(final_age = max(Age),
         final_season = max(Season)) %>%
  summarise(final_age = max(Age),
            player_id = PlayerId,
            g = sum(G),
            pa = sum(PA),
            h = sum(H),
            x2b = sum(`2B`),
            x3b = sum(`3B`),
            hr = sum(HR),
            rbi = sum(RBI),
            bb = sum(BB),
            so = sum(SO),
            hbp = sum(HBP),
            sf = sum(SF),
            sh = sum(SH),
            gidp = sum(GDP),
```

```
            sb = sum(SB),
            cs = sum(CS),
            woba = weighted.mean(woba, PA),
            `wrc+` = weighted.mean(`wrc+`, PA), .groups = "drop") %>%
  distinct() %>% filter(pa >= 150) %>% arrange(desc(woba)) %>%
  mutate(level = "minor")

combined_minors %>% arrange(desc(woba))
```

```
## # A tibble: 1,334 x 21
##    Name  final_age player_id     g    pa     h   x2b   x3b    hr   rbi    bb
##    <chr>     <dbl>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Kris~        22     15429    84   359   107    25     1    27    72    46
##  2 Juan~        19     20123    62   278    86    15     4    17    66    39
##  3 Kevi~        24      5995    67   276    95    19     1    15    55    23
##  4 Bran~        23     10264   127   552   168    40    10    19   106    85
##  5 Josh~        24      2205    66   256    86    14     2    11    50    27
##  6 Matt~        22      4298   130   530   155    22     2    27    91    82
##  7 Kyle~        22     16478   125   530   141    27     2    27    82    79
##  8 Chri~        23      8267    44   185    50    10     2    11    26    24
##  9 Alex~        22      5209   130   577   158    39     1    29   101    72
## 10 Mark~        23      7619   143   604   164    34     4    37   120    72
## # ... with 1,324 more rows, and 10 more variables: so <dbl>, hbp <dbl>,
## #   sf <dbl>, sh <dbl>, gidp <dbl>, sb <dbl>, cs <dbl>, woba <dbl>,
## #   `wrc+` <dbl>, level <chr>
```

Take a look at the leader boards and you'll see some familiar faces. Kris Bryant and Juan Soto are number 1 and 2, respectively. Bryant has a mind-boggling .502 wOBA in those three minor league levels. Number 13 on the list is Vladimir Guerrero Jr, primed for a breakout season in 2021.

Next we need to grab the major league stats for these players and begin crafting our models.

```
major <- read_csv("batters-2006-fg.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   Name = col_character(),
##   Team = col_character()
## )
## i Use `spec()` for the full column specifications.
```

```
major_tidy <- major %>%
  group_by(Name) %>%
  arrange(Season) %>%
  mutate(year_in_mlb = row_number()) %>%
  ungroup() %>%
  clean_names() %>%
  rename(woba = w_oba,
         `wrc+` = w_rc,
         player_id = playerid,
         gidp = gdp)
```

We will be combining all ages into one group to have enough data, but we'll use age as a predictor if it's deemed significant enough by lasso.

```
milb_group1 <- combined_minors %>%
  arrange(player_id) %>% select(-c(Name, level))

id <- milb_group1$player_id

# Filter and set PA limit for the first year in MLB for these players
mlb_group1 <- major_tidy %>%
  filter(year_in_mlb == 1) %>%
  filter(player_id %in% id) %>%
  filter(pa >= 50) %>%
  arrange(player_id) %>% select(-c(season, name, team, age))

id <- mlb_group1$player_id
milb_group1 <- milb_group1 %>%
  filter(player_id %in% id)

# Create the table for regression
get_stat <- function(df, stat) {
  if (stat != "woba" & stat != "wrc+" & stat!= "g") {
    rate_stat <- df[[stat]] / df$pa
  } else {
    rate_stat <- df[[stat]]
  }
  return(rate_stat)
}

reg_group1 <- tibble(hperpa_milb = get_stat(milb_group1, "h"),
       x2bperpa_milb = get_stat(milb_group1, "x2b"),
       x3bperpa_milb = get_stat(milb_group1, "x3b"),
       hrperpa_milb = get_stat(milb_group1, "hr"),
       rbiperpa_milb = get_stat(milb_group1, "rbi"),
       bbperpa_milb = get_stat(milb_group1, "bb"),
       soperpa_milb = get_stat(milb_group1, "so"),
       hbpperpa_milb = get_stat(milb_group1, "hbp"),
       sfperpa_milb = get_stat(milb_group1, "sf"),
       shperpa_milb = get_stat(milb_group1, "sh"),
       gidpperpa_milb = get_stat(milb_group1, "gidp"),
       sbperpa_milb = get_stat(milb_group1, "sb"),
       csperpa_milb = get_stat(milb_group1, "cs"),
       woba_milb = get_stat(milb_group1, "woba"),
       `wrc+_milb` = get_stat(milb_group1, "wrc+"),
       g_milb = get_stat(milb_group1, "g"),
       age_milb = milb_group1$final_age,
       hperpa_year1 = get_stat(mlb_group1, "h"),
       x2bperpa_year1 = get_stat(mlb_group1, "x2b"),
       x3bperpa_year1 = get_stat(mlb_group1, "x3b"),
       hrperpa_year1 = get_stat(mlb_group1, "hr"),
       rbiperpa_year1 = get_stat(mlb_group1, "rbi"),
       bbperpa_year1 = get_stat(mlb_group1, "bb"),
       soperpa_year1 = get_stat(mlb_group1, "so"),
       hbpperpa_year1 = get_stat(mlb_group1, "hbp"),
```

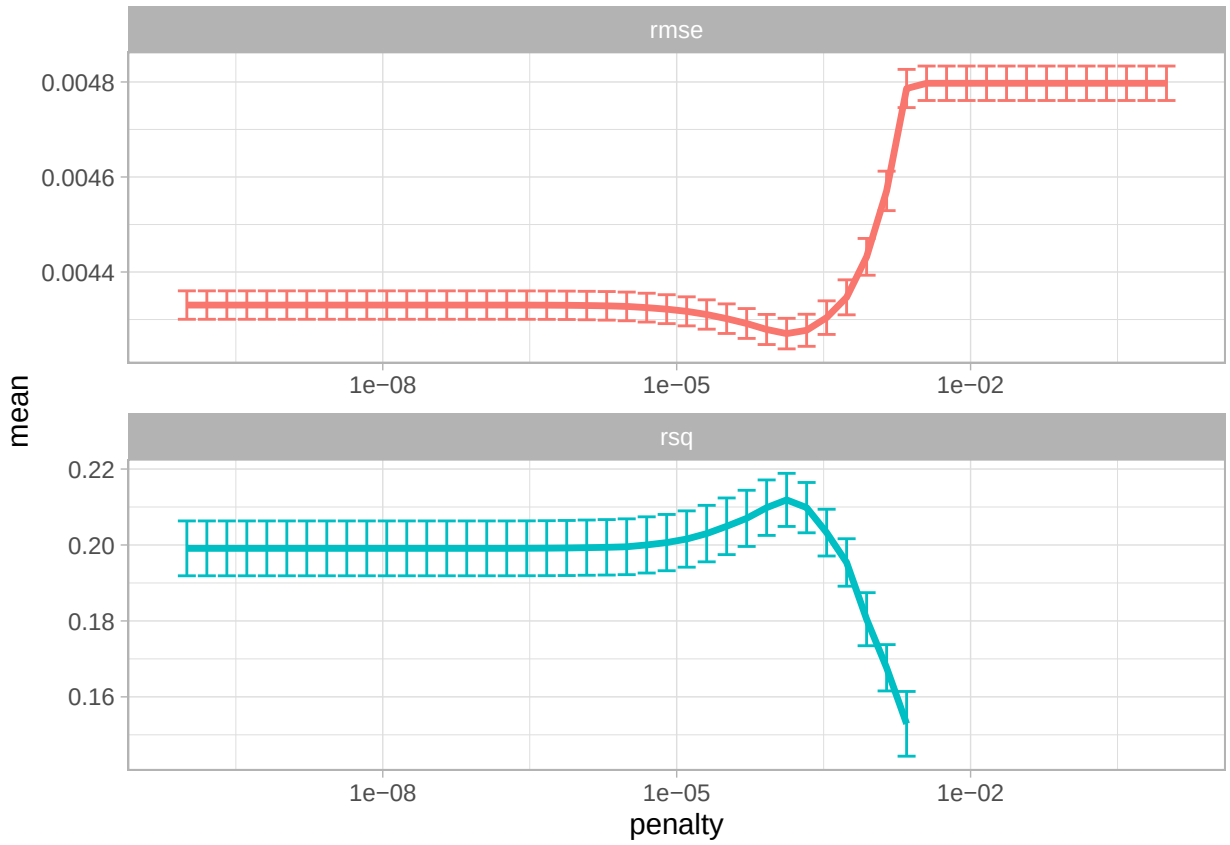
```
       sfperpa_year1 = get_stat(mlb_group1, "sf"),
       shperpa_year1 = get_stat(mlb_group1, "sh"),
       gidpperpa_year1 = get_stat(mlb_group1, "gidp"),
       sbperpa_year1 = get_stat(mlb_group1, "sb"),
       csperpa_year1 = get_stat(mlb_group1, "cs"),
       woba_year1 = get_stat(mlb_group1, "woba"),
       `wrc+_year1` = get_stat(mlb_group1, "wrc+"),
       warperpa_year1 = get_stat(mlb_group1, "war"),
       g_year1 = get_stat(mlb_group1, "g"))
```

Now we can follow our same lasso regression steps as before to predict major league performance based on minor league data.

```
library(tidymodels)
set.seed(2020)
# This time we will predict WAR from minor league performance
war <- reg_group1 %>%
  select(-c(hperpa_year1, x2bperpa_year1, x3bperpa_year1, `wrc+_year1`,
            rbiperpa_year1, bbperpa_year1, soperpa_year1, hbpperpa_year1,
            sfperpa_year1, shperpa_year1, gidpperpa_year1, sbperpa_year1,
            csperpa_year1, woba_year1, age_milb, hperpa_year1,
            -g_year1))

# Create the training and testing split for cross validation and
# to avoid over fitting the data
war_split <- initial_split(war)
war_train <- training(war_split)
war_test <- testing(war_split)

# lasso requires we center and scale our data as a pre-processing step
war_rec <- recipe(warperpa_year1 ~ ., data = war_train) %>%
  step_zv(all_numeric(), -all_outcomes()) %>%
  step_normalize(all_numeric(), -all_outcomes())

war_prep <- war_rec %>%
  prep()

# Set the model to lasso using glmnet
war_spec <- linear_reg(penalty = 0.1, mixture = 1) %>%
  set_engine("glmnet")

war_wf <- workflow() %>%
  add_recipe(war_rec)

war_fit <- war_wf %>%
  add_model(war_spec) %>%
  fit(data = war_train)

# use re-sampling to tune the model
war_boot <- bootstraps(war_train)

war_tune <- linear_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet")
```

```
war_grid <- grid_regular(penalty(), levels = 50)

doParallel::registerDoParallel()


lasso_grid <- tune_grid(
  war_wf %>% add_model(war_tune),
  resamples = war_boot,
  grid = war_grid
)

# graph the rmse and rsq
lasso_grid %>%
  collect_metrics() %>%
  ggplot(aes(penalty, mean, color = .metric)) +
  geom_errorbar(aes(ymin = mean - std_err,
                    ymax = mean + std_err)) +
  geom_line(size = 1.2) +
  facet_wrap(~.metric, scales = "free", nrow = 2) +
  scale_x_log10() +
  theme(legend.position = "none")
```

```
## Warning: Removed 13 row(s) containing missing values (geom_path).
```



```
lowest_rmse <- lasso_grid %>%
  select_best("rmse")

final_war <- finalize_workflow(war_wf %>% add_model(war_tune),
                  lowest_rmse)
library(vip)

importance <- final_war %>%
  fit(war_train) %>%
  pull_workflow_fit() %>%
  vi(lambda = lowest_rmse[["penalty"]])

final_fit <- final_war %>%
  fit(war_train)
```

Now all that's left to do is make a model for each stat and for each year the batter is in the majors, which I've done in a separate script which is available in my github.

```
proj <- read_csv("2021-advanced-projections.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   name = col_character(),
##   team = col_character(),
##   age = col_double(),
##   g = col_double(),
##   pa = col_double(),
##   hr = col_double(),
##   sb = col_double(),
##   cs = col_double(),
##   `bb%` = col_double(),
##   `k%` = col_double(),
##   iso = col_double(),
##   babip = col_double(),
##   avg = col_double(),
##   obp = col_double(),
##   slg = col_double(),
##   woba = col_double(),
##   `wrc+` = col_double(),
##   war = col_double()
## )
```

# Findings

And here's the end result. Not too shabby, I would say. One thing that stands out immediately is Mookie Betts surpasses Trout by 0.1 WAR. Trout was actually ahead of Betts before the aging adjustment.

Overall the model seems conservative, but a lot of the projections track with those on Fangraphs.com. Not all players are projected, but 468 them are, right around 75% of the batters that plays in a season.

Looking at the results, some things I would do differently include weighing the amount of plate appearances when building the models, a feature that is currently lacking in tidymodels; I would have to use individual

packages which is good to know for next time. Additionally, I would like to experiment by including a predictor that is the average of each statistic over those four years. Also, I would drop batters that had fewer than 100 plate appearances in a year and pretend that year didn't exist. If you scroll through the projections, you'll find players with not much playing time but who did well in that time projected for much loftier heights than they would actually hit.

Also, I would use the elasticnet penalization method next, a combination of ridge and lasso regression. Lasso can arbitrarily drop predictors if they're highly correlated and offer similar predictive value, which isn't always a good thing, whereas elasticnet will be a little more generous.

```
proj %>%
  group_by(team) %>%
  summarise(team_war = sum(war)) %>%
  arrange(desc(team_war))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 31 x 2
##    team  team_war
##    <chr>    <dbl>
##  1 LAD       31.2
##  2 NYM       30
##  3 TOR       26
##  4 NYY       25.6
##  5 LAA       25.1
##  6 SDP       24.8
##  7 CHW       24.6
##  8 PHI       23.6
##  9 ATL       23.3
## 10 SFG       22.2
## # ... with 21 more rows
```

Looking at the sum of projected war by team, we seem to do quite well. Of course, these are only projections for hitters, but good teams are at the top and bad teams are at the bottom. The 2020 World Series champions Dodgers are number one with 31.2 projected wins above replacement. The Mets are second which might come as a bit of a surprise considering people think of them as a pitching dominated team, but they do well by most projection systems, including Steamer. Toronto is number three here and number four if going by Steamer. The off-season acquisition of George Springer projects to be their best player. The Yankees are just below Toronto with most of their concerns coming with playing time as they've struggled with injuries the past couple of seasons. Next is the Angels, home to Mike Trout and Anthony Rendon; no surprise here. Then we have a few young upcoming teams such as the White Sox and Padres, who just signed their star Fernando Tatis Jr. to a massive 14 year, 340 Million dollar deal.

## Conclusion

Working on this project taught me a lot about coding in R while having fun doing it. Next I would like to tackle projecting pitchers with the lessons I've learned here, and use data.table and base R graphics in as opposed to dplyr and ggplot to gain more experience.