The new Major League Baseball season will soon begin, which means it’s time to look back and update my run scoring trends data visualization application, built using RStudio’s shiny package.  
This update gave me the opportunity to make some cosmetic tweaks to the front end, and more consequential changes to the code under the hood.

**1. retired reshape, using tidyr**

At one point in the app’s code, I had used the now-retired reshape package to melt a data table. Although this still works, I opted to update the code to use the gather() function in the package tidyr.

**2. feather instead of csv**

The app relied on some pre-wrangled csv files;

**3. wrangling: the calculation of team and league run scoring averages**

The goal is to create data tables that minimize the amount of processing the app has to do.  
In previous versions of the app, the filtering of rows (records or observations) and selecting of columns (variables), the calculation of each team’s average runs scored and runs allowed per game, the league average runs per game, and the comparison of the team to the league, was done first using base R’s apply family of functions.  
Then I switched to using dplyr, and although the steps were now in a pipe, this approach still required creating a separate data table with the league average, and then joining that table back into the main team table so that the team result could be compared to the league average.  
For this iteration, preparing the data for the app is now done using tidyr::nest() and purrr::map(). What follows is a detailed explanation of how I approached this.  
It’s always valuable to have your end-state in mind when working through a multi-step data wrangle like this. My goal is the values shown on the “team plot” tab of the app – an index value (i.e. a percentage) of a team’s average runs scored (and runs allowed) compared to the league run scoring rate, for a single season.

**a. Load packages and read the data**

First, load the necessary packages, the first four of which are part of the tidyverse.

# tidyverse packages  
library(dplyr)  
library(purrr)  
library(readr)  
library(tidyr)  
  
library(feather)

Then, read in the data.

* It comes from the “Lahman” baseball database, so-named after the originator and host, Sean Lahman. [The website www.seanlahman.com](http://www.seanlahman.com/baseball-archive/statistics/) has the current version of the database, as well as an archive of previous versions.

Teams <- read\_csv("Teams.csv",  
 col\_types = cols(  
 divID = col\_character(),  
 DivWin = col\_character(),  
 SF = col\_character(),  
 WCWin = col\_character()  
 ))  
  
head(Teams)

## # A tibble: 6 x 48  
## yearID lgID teamID franchID divID Rank G Ghome W L DivWin  
##   
## 1 1871 BS1 BNA 3 31 NA 20 10   
## 2 1871 CH1 CNA 2 28 NA 19 9   
## 3 1871 CL1 CFC 8 29 NA 10 19   
## 4 1871 FW1 KEK 7 19 NA 7 12   
## 5 1871 NY2 NNA 5 33 NA 16 17   
## 6 1871 PH1 PNA 1 28 NA 21 7   
## # ... with 37 more variables: WCWin , LgWin , WSWin ,  
## # R , AB , H , `2B` , `3B` , HR ,  
## # BB , SO , SB , CS , HBP , SF , RA ,  
## # ER , ERA , CG , SHO , SV , IPouts ,  
## # HA , HRA , BBA , SOA , E , DP ,  
## # FP , name , park , attendance , BPF ,  
## # PPF , teamIDBR , teamIDlahman45 , teamIDretro

The table above has far more variables than what we need, and some that we’ll have to calculate.

**b. Create league summary tables**

A short set of instructions that starts with the “Teams” table in the Lahman database .Thus rather than having the app do the work of

1. remove unnecessary records (rows) and fields (columns) and
2. run the calculations for the runs-per-game, runs-allowed-per-game, and indexed versions of those,

the calculations are conducted here. This will vastly improve the performance of the app.

**i. create nested table**

I started with the “Many Models”” chapter of Wickham and Grolemund. (And thanks to Dr. Charlotte Wickham, whose training course was invaluable in helping me wrap my head around this.)  
At this point, the code

* filters out the years prior to 1901.
* and then creates a nested data table, starting with the group\_by() year and league (lgID)

# select a sub-set of teams from 1901 [the establishment of the American League] forward to most recent year  
Teams\_lgyr <- Teams %>%  
 filter(yearID > 1900, lgID != "FL") %>%  
 group\_by(yearID, lgID) %>%  
 nest()  
  
Teams\_lgyr

## # A tibble: 236 x 3  
## yearID lgID data   
##   
## 1 1901 AL   
## 2 1901 NL   
## 3 1902 AL   
## 4 1902 NL   
## 5 1903 AL   
## 6 1903 NL   
## 7 1904 AL   
## 8 1904 NL   
## 9 1905 AL   
## 10 1905 NL   
## # ... with 226 more rows

Here’s a quick peek inside the first entry of the “data” column…the American League, 1901.

Teams\_lgyr$data[[1]]

## # A tibble: 8 x 46  
## teamID franchID divID Rank G Ghome W L DivWin WCWin LgWin  
##   
## 1 BLA NYY 5 134 66 68 65 N   
## 2 BOS BOS 2 138 69 79 57 N   
## 3 CHA CHW 1 137 71 83 53 Y   
## 4 CLE CLE 7 138 69 54 82 N   
## 5 DET DET 3 135 70 74 61 N   
## 6 MLA BAL 8 139 70 48 89 N   
## 7 PHA OAK 4 137 66 74 62 N   
## 8 WS1 MIN 6 138 68 61 72 N   
## # ... with 35 more variables: WSWin , R , AB , H ,  
## # `2B` , `3B` , HR , BB , SO , SB ,  
## # CS , HBP , SF , RA , ER , ERA ,  
## # CG , SHO , SV , IPouts , HA , HRA ,  
## # BBA , SOA , E , DP , FP , name ,  
## # park , attendance , BPF , PPF , teamIDBR ,  
## # teamIDlahman45 , teamIDretro

**ii – functional programming**

This step creates a league run scoring function, and then applies that using the purrr::map() function.  
**Note:**

* In the gapminder example in *R for Data Science*, the variables were called using their names. In this case, for a reason I have not yet determined, we have to specify the data object they are coming from; e.g. for the runs variable R, we have to use df$R (not just R).

First, a simple test, calculating runs scored, and checking to see if we got the right answer, b comparing that to the value calculated using dplyr:

# base R format  
leagueRuns\_fun <- function(df) {  
 sum(data = df$R)  
}  
  
league\_year\_runs <- map(Teams\_lgyr$data, leagueRuns\_fun)  
  
league\_year\_runs[[1]]

## [1] 5873

#check the answer by old school `dplyr` method  
Teams %>%  
 filter(yearID == 1901,  
 lgID == "AL") %>%  
 summarise(leagueruns = sum(R))

## # A tibble: 1 x 1  
## leagueruns  
##   
## 1 5873

Now we move on to the calculation of league averages.  
For the first approach, the sum calculation is part of the function.

* There are two functions, one for Runs and the other for Runs Allowed. This is because I have not yet figured out how to specify two different variables (i.e. the name of the data object and the variable to be used in the function) in the map\_() function and successfully have them carried into my calculation functions
* Also note that in order to be consistent with other sources, the number of games played is calculated using the sum of wins (W) and losses (L), rather than the number of games reported in the G variable.

# functions  
leagueRPG\_fun <- function(df) {  
 sum(data = df$R) / (sum(data = df$W) + sum(data = df$L))  
 }  
  
leagueRAPG\_fun <- function(df) {  
 sum(data = df$RA) / (sum(data = df$W) + sum(data = df$L))  
 }  
  
  
# simple `map` version  
league\_year\_RPG <- map(Teams\_lgyr$data, leagueRPG\_fun)  
  
# embed as new columns in nested data object  
Teams\_lgyr <- Teams\_lgyr %>%  
 mutate(lgRPG = map\_dbl(Teams\_lgyr$data, leagueRPG\_fun),  
 lgRAPG = map\_dbl(Teams\_lgyr$data, leagueRAPG\_fun))  
  
Teams\_lgyr

## # A tibble: 236 x 5  
## yearID lgID data lgRPG lgRAPG  
##   
## 1 1901 AL 5.43 5.43  
## 2 1901 NL 4.69 4.69  
## 3 1902 AL 4.97 4.97  
## 4 1902 NL 4.09 4.09  
## 5 1903 AL 4.15 4.15  
## 6 1903 NL 4.85 4.85  
## 7 1904 AL 3.65 3.65  
## 8 1904 NL 3.98 3.98  
## 9 1905 AL 3.75 3.75  
## 10 1905 NL 4.16 4.16  
## # ... with 226 more rows

In the second approach:

* the league and year total runs, runs allowed, and games are first calculated using separate functions
* RPG and RAPG for each league and year combination are then calculated outside the nested tibbles

# more functions - individual league by year totals  
leagueR\_fun <- function(df) {  
 sum(data = df$R)  
}  
  
leagueRA\_fun <- function(df) {  
 sum(data = df$RA)  
}  
  
leagueG\_fun <- function(df) {  
 (sum(data = df$W) + sum(data = df$L))  
}  
  
  
Teams\_lgyr <- Teams\_lgyr %>%  
 mutate(lgR = map\_dbl(Teams\_lgyr$data, leagueR\_fun),  
 lgRA = map\_dbl(Teams\_lgyr$data, leagueRA\_fun),  
 lgG = map\_dbl(Teams\_lgyr$data, leagueG\_fun))  
  
  
Teams\_lgyr <- Teams\_lgyr %>%  
 mutate(lgRPG = (lgR / lgG),  
 lgRAPG = (lgRA / lgG))  
  
Teams\_lgyr

## # A tibble: 236 x 8  
## yearID lgID data lgRPG lgRAPG lgR lgRA lgG  
##   
## 1 1901 AL 5.43 5.43 5873 5873 1082  
## 2 1901 NL 4.69 4.69 5194 5194 1108  
## 3 1902 AL 4.97 4.97 5407 5407 1088  
## 4 1902 NL 4.09 4.09 4494 4494 1098  
## 5 1903 AL 4.15 4.15 4543 4543 1096  
## 6 1903 NL 4.85 4.85 5349 5349 1102  
## 7 1904 AL 3.65 3.65 4433 4433 1216  
## 8 1904 NL 3.98 3.98 4872 4872 1224  
## 9 1905 AL 3.75 3.75 4547 4547 1212  
## 10 1905 NL 4.16 4.16 5092 5092 1224  
## # ... with 226 more rows

**iii. save LG\_RPG files**

And then write csv and feather versions. As noted above, the shiny app now uses the feather format.  
Notes:

* rounding of variables *after* all the calculations, simply to make the tables as viewed more legible.
* renaming of variables to correspond with shiny app names.

LG\_RPG <- Teams\_lgyr %>%  
 mutate(lgRPG = round(lgRPG, 2),  
 lgRAPG = round(lgRAPG, 2)) %>%  
 select(yearID, lgID, R = lgR, RA = lgRA, G = lgG,   
 leagueRPG = lgRPG, leagueRAPG = lgRAPG)  
  
LG\_RPG

## # A tibble: 236 x 7  
## yearID lgID R RA G leagueRPG leagueRAPG  
##   
## 1 1901 AL 5873 5873 1082 5.43 5.43  
## 2 1901 NL 5194 5194 1108 4.69 4.69  
## 3 1902 AL 5407 5407 1088 4.97 4.97  
## 4 1902 NL 4494 4494 1098 4.09 4.09  
## 5 1903 AL 4543 4543 1096 4.15 4.15  
## 6 1903 NL 5349 5349 1102 4.85 4.85  
## 7 1904 AL 4433 4433 1216 3.65 3.65  
## 8 1904 NL 4872 4872 1224 3.98 3.98  
## 9 1905 AL 4547 4547 1212 3.75 3.75  
## 10 1905 NL 5092 5092 1224 4.16 4.16  
## # ... with 226 more rows

write\_csv(LG\_RPG, "LG\_RPG.csv")  
write\_feather(LG\_RPG, "LG\_RPG.feather")

**c. Repeat for MLB total**

This only differs from the league summaries in the level of nesting; instead of grouping by year and league, it’s only year (yearID).

Teams\_lgyr <- Teams\_lgyr %>%  
 unnest() %>%  
 group\_by(yearID) %>%  
 nest()  
  
Teams\_lgyr

## # A tibble: 118 x 2  
## yearID data   
##   
## 1 1901   
## 2 1902   
## 3 1903   
## 4 1904   
## 5 1905   
## 6 1906   
## 7 1907   
## 8 1908   
## 9 1909   
## 10 1910   
## # ... with 108 more rows

Teams\_lgyr <- Teams\_lgyr %>%  
 mutate(mlbR = map\_dbl(Teams\_lgyr$data, leagueR\_fun),  
 mlbRA = map\_dbl(Teams\_lgyr$data, leagueRA\_fun),  
 mlbG = map\_dbl(Teams\_lgyr$data, leagueG\_fun),  
 mlbRPG = (mlbR / mlbG),  
 mlbRAPG = (mlbRA / mlbG))  
  
Teams\_lgyr

## # A tibble: 118 x 7  
## yearID data mlbR mlbRA mlbG mlbRPG mlbRAPG  
##   
## 1 1901 11067 11067 2190 5.05 5.05  
## 2 1902 9901 9901 2186 4.53 4.53  
## 3 1903 9892 9892 2198 4.50 4.50  
## 4 1904 9305 9305 2440 3.81 3.81  
## 5 1905 9639 9639 2436 3.96 3.96  
## 6 1906 8881 8878 2416 3.68 3.67  
## 7 1907 8703 8703 2406 3.62 3.62  
## 8 1908 8422 8422 2456 3.43 3.43  
## 9 1909 8810 8810 2436 3.62 3.62  
## 10 1910 9584 9584 2446 3.92 3.92  
## # ... with 108 more rows

And again, we save the files for use in the shiny app.

MLB\_RPG <- Teams\_lgyr %>%  
 mutate(mlbRPG = round(mlbRPG, 2),  
 mlbRAPG = round(mlbRAPG, 2)) %>%  
 select(yearID, R = mlbR, RA = mlbRA, G = mlbG,   
 leagueRPG = mlbRPG, leagueRAPG = mlbRAPG)  
  
write\_csv(MLB\_RPG, "MLB\_RPG.csv")  
write\_feather(MLB\_RPG, "MLB\_RPG.feather")

**d. Individual team values**

Calculate index of team run scoring against league average  
Note that we start with unnest() and create a new object, Teams\_append … a tibble with all of the variables exposed.

Teams\_append <- Teams\_lgyr %>%  
 unnest() %>%  
 mutate(teamRPG=(R / (W + L)),   
 teamRAPG=(RA / (W + L)),   
 WLpct=(W / (W + L))) %>%  
 # runs scored index where 100=the league average for that season  
 mutate(R\_index = (teamRPG / lgRPG) \* 100) %>%  
 mutate(R\_index.sd = sd(R\_index)) %>%  
 mutate(R\_z = (R\_index - 100) / R\_index.sd) %>%  
 # runs allowed  
 mutate(RA\_index = (teamRAPG / lgRAPG) \* 100) %>%  
 mutate(RA\_index.sd = sd(RA\_index)) %>%  
 mutate(RA\_z = (RA\_index - 100) / RA\_index.sd)   
  
  
Teams\_append

## # A tibble: 2,496 x 67  
## yearID mlbR mlbRA mlbG mlbRPG mlbRAPG lgID lgRPG lgRAPG lgR lgRA  
##   
## 1 1901 11067 11067 2190 5.05 5.05 AL 5.43 5.43 5873 5873  
## 2 1901 11067 11067 2190 5.05 5.05 AL 5.43 5.43 5873 5873  
## 3 1901 11067 11067 2190 5.05 5.05 AL 5.43 5.43 5873 5873  
## 4 1901 11067 11067 2190 5.05 5.05 AL 5.43 5.43 5873 5873  
## 5 1901 11067 11067 2190 5.05 5.05 AL 5.43 5.43 5873 5873  
## 6 1901 11067 11067 2190 5.05 5.05 AL 5.43 5.43 5873 5873  
## 7 1901 11067 11067 2190 5.05 5.05 AL 5.43 5.43 5873 5873  
## 8 1901 11067 11067 2190 5.05 5.05 AL 5.43 5.43 5873 5873  
## 9 1901 11067 11067 2190 5.05 5.05 NL 4.69 4.69 5194 5194  
## 10 1901 11067 11067 2190 5.05 5.05 NL 4.69 4.69 5194 5194  
## # ... with 2,486 more rows, and 56 more variables: lgG ,  
## # teamID , franchID , divID , Rank , G ,  
## # Ghome , W , L , DivWin , WCWin , LgWin ,  
## # WSWin , R , AB , H , `2B` , `3B` ,  
## # HR , BB , SO , SB , CS , HBP , SF ,  
## # RA , ER , ERA , CG , SHO , SV ,  
## # IPouts , HA , HRA , BBA , SOA , E ,  
## # DP , FP , name , park , attendance ,  
## # BPF , PPF , teamIDBR , teamIDlahman45 ,  
## # teamIDretro , teamRPG , teamRAPG , WLpct ,  
## # R\_index , R\_index.sd , R\_z , RA\_index ,  
## # RA\_index.sd , RA\_z

In this the final step, we first create a new data object Teams\_merge.  
Notes:

* rounding of a variety of the calculated variables, to address readability concerns.
* selection and renaming of variables to correspond with shiny app names.
* then write csv and feather versions.

Teams\_merge <- Teams\_append %>%  
 mutate(lgRPG = round(lgRPG, 2),  
 lgRAPG = round(lgRAPG, 2),  
 WLpct = round(WLpct, 3),  
 teamRPG = round(teamRPG, 2),  
 teamRAPG = round(teamRAPG, 2),  
 R\_index = round(R\_index, 1),  
 RA\_index = round(RA\_index, 1)  
 ) %>%  
 select(yearID, lgID, franchID, teamID, name,  
 W, L, WLpct, R.x = R, RA.x = RA,   
 teamRPG, leagueRPG = lgRPG, R\_index,  
 teamRAPG, leagueRAPG = lgRAPG, RA\_index)  
  
Teams\_merge

## # A tibble: 2,496 x 16  
## yearID lgID franchID teamID name W L WLpct R.x RA.x teamRPG  
##   
## 1 1901 AL NYY BLA Balt~ 68 65 0.511 760 750 5.71  
## 2 1901 AL BOS BOS Bost~ 79 57 0.581 759 608 5.58  
## 3 1901 AL CHW CHA Chic~ 83 53 0.61 819 631 6.02  
## 4 1901 AL CLE CLE Clev~ 54 82 0.397 666 831 4.9   
## 5 1901 AL DET DET Detr~ 74 61 0.548 741 694 5.49  
## 6 1901 AL BAL MLA Milw~ 48 89 0.35 641 828 4.68  
## 7 1901 AL OAK PHA Phil~ 74 62 0.544 805 760 5.92  
## 8 1901 AL MIN WS1 Wash~ 61 72 0.459 682 771 5.13  
## 9 1901 NL LAD BRO Broo~ 79 57 0.581 744 600 5.47  
## 10 1901 NL ATL BSN Bost~ 69 69 0.5 531 556 3.85  
## # ... with 2,486 more rows, and 5 more variables: leagueRPG ,  
## # R\_index , teamRAPG , leagueRAPG , RA\_index

write\_csv(Teams\_merge, "Teams\_merge.csv")  
write\_feather(Teams\_merge, "Teams\_merge.feather")

And the feather files can now be incorproated into the shiny app.  
-30-