Lab #2, Part #2 – Exploring and Visualizing Baseball Data in R

Last week, we talked a bit about summarizing and visualizing data, including the use of measures of central tendency, measures of variability, histograms, scatterplots, bar plots, and time series plots. Today, we'll use apply those concepts in R to visualize some Major League Baseball Data.

These data come from Fangraphs and include team-level information on runs scored, runs allowed, home runs hit, stolen bases, batting average, and various other statistics from 2010 through 2019. There are 30 teams in MLB, and therefore your data should have 300 rows (observations).

Activity #1: Get the Data and Load It In.
The data are available in the Lab #2 Assignment (in addition to this lab prompt you're reading now). Download it onto your computer and save it in a new folder in your directory called Lab 2 (preferably in the same directory where the Lab 1 folder is). Name the data "`mlbdat`" and note that it should be in CSV format.

Once it's saved to the correct folder, go ahead and start a new script in R Studio. Put a title of the script at the top with a comment (#), set your working directory with `setwd()`, and load the data with `read.csv()`. Once it's loaded in, take a look at what you're working with using the `head()` function.

```
#Lab 2-1 R Script
#set working directory and load data
setwd("c:/Users/bmmil/…/Lab 2")
mlbdat <- read.csv(file = "mlbdat.csv", h = T)
head(mlbdat)
```

Activity #2: Fixing Up Column Names.
Notice a few things with the data. First, singles, doubles, and triples (1B, 2B, and 3B) have strange column (variable) names. These would be cumbersome to type, so it might be a good idea to rename these. R does this automatically because it does not allow a number to be the first character in a column name. To do this, we'll use the `names()` function for this, starting with singles, and renaming to "`Singles`".

```
#rename some columns
names(mlbdat)[names(mlbdat) == "X1B"] <- "singles"
```

Now do the same on your own for doubles and triples and take a look to make sure you have the new column names.

Activity #3: Missing Some Data.
Let's quickly try to take the average of home runs in the data with the `mean()` function like we did last week. You can use the code below:

```
#look at average home runs
mean(mlbdat$HR)
```

Did anything strange happen? What do you see?

It turns out that there are some missing values in the data! R automatically codes empty cells from the Excel or CSV file as `NA`. This is a special value that R recognizes as missing, rather than simply a character or factor variable. Let's try and figure out which values are missing by using the `subset()` function. This function allows you to grab portions of the data that are of interest to you and make it into a new object (data frame). For example, you could decide you only care about the Houston Astros data with:

```
#only look at Astros data
astros <- subset(mlbdat, Team == "Astros")
astros
```

Hopefully, you can see all of the Astros data printed out (just 10 rows). One nice thing about `subset()` is that we don't have to use the "`$`" symbol to refer to the columns. Since we've told the function that the data we are using is `mlbdat` it knows where to look for the column names already. This will come into handy later in the lab.

We can make use of this function alongside the `is.na()` function in order to grab all the rows that have missing data. This let's us see where the problem is (note that only some HR are missing in the data, and all other variables are complete, so we will just assume that's all we care about for now):

```
#subset with missing cells
missdat <- subset(mlbdat, is.na(HR))
missdat
```

It turns out I did this on purpose and know what values should be in those 3 rows you see. The 2019 Angels had 220 HR, the 2015 Royals had 139 HR, and the 2010 Red Sox had 211 HR. We can use logical statements to identify what needs replacing, and tell R to replace it with the correct values. For this, we'll make use of the `ifelse()` function. This function, in words, basically says, "If some condition is met, then do this, and otherwise, do this other thing." We'll use it to say, for example, "If the row is for the 2019 Angels, then replace the HR value with 220, otherwise leave it alone."

```
#replace missing HR values
missdat$HR <- ifelse(missdat$Season == 2019 & missdat$Team ==
     "Angels", 220, missdat$HR)
missdat
```

Notice here that we've replaced the value in the `missdat` data frame, and it seemed to work great. But we actually want to replace the values in the larger `mlbdat` data frame. Go ahead and try this with all three missing `HR` values. I'll get you started:

```
#replace missing HR values in mlbdat
mlbdat$HR <- ifelse(mlbdat$Season == 2019 & mlbdat$Team ==
     "Angels", 220, mlbdat$HR)
```

Activity #4: Adding Variables.

Now that things are cleaned up, let's add a couple useful variables to our data. Notice that we have both stolen bases (`SB`) and caught stealing (`CS`). It might be useful to know the total number of stolen base attempts and the success percentage for stolen bases at the team level. We'll name these `SBA` and `SB_rate` in the data. Start with `SBA` using the code below, noting that total attempts are the sum of successes and failures, by adding a new vector onto the end of `mlbdat`:

```
#add stolen base attempts
mlbdat$SBA <- mlbdat$SB + mlbdat$CS
head(mlbdat)
```

It's always good to check that things worked right by inspecting the data. Often the `head()` function is sufficient for something like this. Make sure the first rows of `SBA` are equivalent to the same row's `SB+CS`. Then, go ahead and add stolen base percentage to the data:

```
#add stolen base rate
mlbdat$SB_rate <- mlbdat$SB/mlbdat$SBA
head(mlbdat)
```

That's a lot of decimals. Those probably aren't necessary, so let's make use of the `round()` function, which tells R how many decimal places we want to round things. Let's round to 3 decimal places and make it a new variable called `SB_rateR`:

```
#round stolen base rate
mlbdat$SB_rateR <- round(mlbdat$SB_rate, 3)
head(mlbdat)
```

It's also possible to just write over `SB_rate` by naming it the same thing, which is usually preferable than having 2 of the same variable rounded to different decimal places. Let's subset our data again to remove the `SB_rateR` and `SB_rate` variables, but this time we'll remove a column:

```
#remove the columns
mlbdat <- subset(mlbdat, select = -c(SB_rate, SB_rateR))
head(mlbdat)
```

Notice we've written over our previous version of `mlbdat`, so that's gone forever. You should always be careful with this. Now just rewrite `SB_rate` with the function rounded in one swoop by wrapping the division inside our `round()` function:

```
#make a rounded SB_rate variable
mlbdat$SB_rate <- round(mlbdat$SB/mlbdat$SBA, 3)
head(mlbdat)
```

Activity #5: Describing the Data.

Now that things are cleaned up, let's begin by doing some descriptive analysis of our data. We'll start with mean, median, and standard deviation. Let's work specifically with `HR`, `R`, `SB_rate`, `AVG`, `OBP`, and `SLG`. Remember that the `summary()` function gives both the mean and median, so we can avoid using both the `mean()` and `median()` functions separately and save some lines of code:

```
#get some descriptives
summary(mlbdat$HR)
summary(mlbdat$R)

summary(mlbdat$SB_rate)
summary(mlbdat$AVG)
summary(mlbdat$OBP)
summary(mlbdat$SLG)

#get some standard deviations
sd(mlbdat$HR)
sd(mlbdat$R)

sd(mlbdat$SB_rate)
sd(mlbdat$AVG)
sd(mlbdat$OBP)
sd(mlbdat$SLG)
```

Note that we could also use the `round()` function for these summary statistics. Looking at the standard deviations, do you think `AVG`, `OBP`, or `SLG` has the most variability? Why do you think that is?

Now let's remind ourselves of the `aggregate()` function. Let's look at average home runs first by `Year`. From there, do the same for the standard deviation. Name the latter `yearHRsd` and remember the function for standard deviation is `sd()`. Print out the results of each in R and describe what you see. Which year averaged the most home runs per team? Which had the most variability among teams?

```
#use aggregate() to do year and team level HR averages
yearHR <- aggregate(HR ~ Season, data = mlbdat, FUN = "mean")
```

Activity #6: Making Histograms.
Now that we have some descriptions of the data, let's look at some visuals. We'll start with a histogram of `HR`. In R, we can easily make a histogram with the function `hist()`. Let's start with that below:

```
#make a HR histogram
hist(mlbdat$HR)
```

Notice that R automatically chose the bin size for us, and added a title and axis names in the figure. But the title is uninspiring, as is the x-axis name. We can rename these with additional arguments inside the `hist()` function. Let's call the figure "`Histogram of Team Home Runs`" and the x-axis "`Seasonal HR Total`." We will also rename the y-axis "`Count`" to exhibit how we change the y-axis name. We'll use the arguments `main`, `xlab`, and `ylab` for the title, x-axis, and y-axis, respectively. Remember to separate each argument with a comma and include the names in quotations.

```
#now add title and axis names
hist(mlbdat$HR, main = "Histogram of Team Home Runs", xlab =
     "Seasonal HR Total", ylab = "Count")
```

Things are looking better here. We can also change the color of the histogram to anything we want. Let's make our figure hot pink with the `col` argument. There are a lot of colors you can choose just by using color names. Try a few, including `green`, `steelblue`, and `grey`.

```
#let's play with colors
hist(mlbdat$HR, main = "Histogram of Team Home Runs", xlab =
     "Seasonal HR Total", ylab = "Count", col = "hotpink")
```

It might also be useful to include lines that show us where the mean and median are on the plot. For this, we'll wrap our `mean()` and `median()` functions inside a new function called `abline()` that we call after making the plot. This will make a vertical line on the histogram at each of these values. Inside the `abline()` function, we'll make sure we tell R that we want a vertical line with the `v=` argument. Start by drawing a line at 275 HR and make it red, plus use the line type argument, `lty = "dashed"` to make it a dashed line. Let's use the grey histogram for simplicity:

```
#draw a line at 275 home runs
hist(mlbdat$HR, main = "Histogram of Team Home Runs", xlab =
     "Seasonal HR Total", ylab = "Count", col = "grey")
abline(v = 275, col = "red", lty = "dashed")
```

Next add the mean and median. Make the mean a `green` line and the median a `blue` line. We can just add to the current plot instead of making it all over again:

```
#add mean and median to histogram
abline(v = mean(mlbdat$HR), col = "green", lty = "dashed")
```

```
abline(v = median(mlbdat$HR), col = "blue", lty = "dashed")
```

Given the location of the mean and median, and the shape of the histogram overall, do you think this is left-skewed, right-skewed, or symmetric?

Let's do one more thing with histograms. Here, we'll look at histograms side-by-side. We have to do this by first setting up the visualization window so that each histogram only takes up half of it. Let's compare the distributions of home runs with subsets of the data, one with the years 2010-2014 and another for 2015-2019. We'll call the year subsets using brackets and some logicals. Start by setting up the dual window with the `par()` function (parameters for the visualization) and `mfrow` argument:

```
#set up the visuals
par(mfrow(1,2))
```

Note that if we did `par(mfrow = c(2,1))` we would end up with stacked histograms instead of side-by-side. Now let's plot the data, and make the 2010-2014 data steel blue and the 2015-2019 data hot pink:

```
#make histograms
hist(mlbdat$HR[year < 2015], main = "Histogram of Team Home Runs
     2010-2014", xlab = "Seasonal HR Total", ylab = "Count", col
     = "steelblue")

hist(mlbdat$HR[year >= 2015], main = "Histogram of Team Home Runs
     2015-2019", xlab = "Seasonal HR Total", ylab = "Count", col
     = "hotpink")
```

What do you notice about home runs in each of these eras? Does there seem to be more or less variability? What about the average number of home runs?

Activity #7: Making Bar Plots.

We can also compare home run totals using bar plots. Remember that we've already aggregated our `HR` data by year with averages. Let's make a bar plot from that aggregation, which is now a data frame called `yearHR`. For this, we'll use the `barplot()` function and tell it that we want to use the `HR` column as the heights and the `Season` column as the x-axis labels. We'll also use the `main`, `ylab`, and `xlab` arguments. The default color is `grey`, so we won't mess with that.

```
#make a yearly HR barplot
barplot(HR ~ Season, data = yearHR, main = …)
```

Notice that we've returned to the `data =` argument, so we don't have to use the `$` to choose our variables. Did you run into any problems? It turns out that R saves the `par(mfrow = c(1,2))` parameters for visualization. To get rid of that, we have to clear the plots using the little broom icon button. Do that, then rerun the code above and things should look a bit better.

Now, on your own, aggregate `SB` by `Season` and call it `yearSB` just like we did with `HR`. Go ahead and create a bar plot of stolen bases by year. Do you see any trends?

Activity #8: Making Time Series Plots.

In the previous activity, note that we've essentially been using bar plots to track changes to values over time. Usually, it's good to use a time series plot with a line for this. For a small number of years, generally no more than 10, a bar plot suffices just fine. But things get hairy when using more years or time points. A line plot can be more suitable for that. We'll show how to make a line plot using the same data as above.

Instead of using the `barplot()` function, let's use the more general `plot()` function. It works in much the same way. Start simple with almost the same code as before, but with the new function:

```
#make a plot with the HR data
plot(HR ~ Season, data = yearHR, main = "Plot of Yearly Average
     Team HR Totals", xlab = "Season", ylab = "HR")
```

Hmm, well that didn't work. The `plot()` function defaults to making a scatter plot, something we'll do in the next activity. We have to tell R that we want a line plot here using the `type = "l"` argument. See below for this:

```
#make a line plot now
plot(HR ~ Season, data = yearHR, main = "Plot of Yearly Average
     Team HR Totals", xlab = "Season", ylab = "HR", type = "l")
```

Aha! That's better. Sometimes it might be useful to compare how `HR` progress across time for different teams. We can actually add lines to the line plot using the `lines()` function and subsets of the data. Let's take a look at the `Astros`, `Yankees`, `Dodgers`, and `Marlins` on the same line. We'll color their lines differently with `col` and make them a little thicker with `lwd`. We already subsetted the Astros data, so you'll first need to make `yankees`, `dodgers`, and `marlins` data. Then we'll plot them on a newly created plot. Make the league average a `darkred` line and very thick (`lwd = 4`), and use team colors for the individual teams.

```
#subset team data
yankees <- subset(mlbdat, Team == "Yankees")
dodgers <- subset(mlbdat, Team == "Dodgers")
marlins <- subset(mlbdat, Team == "Marlins")


#remake the plot
plot(HR ~ Season, data = yearHR, main = "Plot of Yearly Average
     Team HR Totals", xlab = "Season", ylab = "HR", type = "l",
     lwd = 4, col = "darkred")
lines(HR ~ Season, data = astros, col = "gold", lwd = 2)
```

```
lines(HR ~ Season, data = yankees, col = "black", lwd = 2)
lines(HR ~ Season, data = dodgers, col = "blue", lwd = 2)
lines(HR ~ Season, data = marlins, col = "turquoise", lwd = 2)
```

Something is wrong again. It turns out that, because we're adding lines to an already created plot, the axes may not go far enough out. Not that using the league average, there is less variability than across individual teams. R automatically chooses the y-axis based on the range of values in the `yearHR` data, and so limits the y-axis. We'll need to tell R that we want the y-axis to be larger using the `ylim = c(,)` argument. You can also do this with the x-axis with `xlim = c(,)`, but we don't need to use that here. First check the minimum and maximum values for each of your data sets

```
#check min and max
min(astros$HR)
max(astros$HR)

min(yankees$HR)
max(yankees$HR)

min(dodgers$HR)
max(dodgers$HR)

min(marlins$HR)
max(marlins$HR)
```

Looks like the minimum `HR` is `95` and maximum is `306` in these data frames. So let's set the minimum at 90 HR and the maximum at 310 HR and replot everything.

```
#remake the plot
plot(HR ~ Season, data = yearHR, main = "Plot of Yearly Average
     Team HR Totals", xlab = "Season", ylab = "HR", type = "l",
     lwd = 4, col = "darkred", ylim = c(90, 310))
lines(HR ~ Season, data = astros, col = "gold", lwd = 2)
lines(HR ~ Season, data = yankees, col = "black", lwd = 2)
lines(HR ~ Season, data = dodgers, col = "blue", lwd = 2)
lines(HR ~ Season, data = marlins, col = "turquoise", lwd = 2)
```

That seems much better. It looks like all teams are hitting more home runs for the most part. The Marlins might be the exception other than their 2017 season.

Activity #9: Making Scatter Plots.

The last thing we'll do today is show how to make a scatter plot. Remember in the last activity, the `plot()` function automatically wants to create a scatterplot, so we can just let it do that without telling it what `type` of plot to make. Let's plot home runs as a function of stolen bases.

After all, maybe teams that have big sluggers aren't going to be stealing lots of bases because they're too slow.

```
#make a SB-HR scatter plot
plot(HR ~ SB, data = mlbdat, main = "Home Runs and Stolen Bases",
     xlab = "Stolen Bases (SB)", ylab = "Home Runs (HR)")
```

What sort of relationship do you seem to see here? Are these two variables positively or negatively related to one another?

Although we can glean some useful information from this plot, we might not like the open circle dots. Instead, maybe we want them filled up. It turns out that `plot()` has an argument, `pch`, that allows us to choose various types of points. This includes both shape and whether they're filled. Unfortunately, they're given numbers, which isn't especially intuitive. The filled circles are `16`, so let's try those first. You can play around with various numbers here (for example, `17` will make filled triangle points).

```
#make a SB-HR scatter plot
plot(HR ~ SB, data = mlbdat, main = "Home Runs and Stolen Bases",
     xlab = "Stolen Bases (SB)", ylab = "Home Runs (HR)", pch =
     16)
```

We can also change the size of the points with the `cex` argument. Be careful with this one, especially with filled dots, as it can obscure points. Let's do an extreme example where we make the points 5 times as big with `cex = 5`.

```
#make giant points
plot(HR ~ SB, data = mlbdat, main = "Home Runs and Stolen Bases",
     xlab = "Stolen Bases (SB)", ylab = "Home Runs (HR)", pch =
     16, cex = 5)
```

Since there's a strong relationship, we can still see the downward trend, but the plot mostly looks like a blob now. You can also make the points very small by making `cex` less than `1`:

```
#make tiny points
plot(HR ~ SB, data = mlbdat, main = "Home Runs and Stolen Bases",
     xlab = "Stolen Bases (SB)", ylab = "Home Runs (HR)", pch =
     16, cex = 0.2)
```

I think the standard sized points are fine for our purposes. We can also change the color of the points, or add colored points to the plot based on team. Let's look at the `Astros` and `Dodgers` points. We'll overlay them on the main plot using the `points()` function, which works just like the `lines()` function, but for points. This time, leave the unfilled points for the main data, and then we'll fill the individual teams with colors with `cex = 1.5` to make them a little bigger.

```
#remake the plot with normal points and add team-specific ones
plot(HR ~ SB, data = mlbdat, main = "Home Runs and Stolen Bases",
     xlab = "Stolen Bases (SB)", ylab = "Home Runs (HR)")
points(HR ~ SB, data = astros, pch = 16, col = "gold", cex = 1.5)
points(HR ~ SB, data = dodgers, pch = 16, col = "blue", cex = 1.5)
points(HR ~ SB, data = marlins, pch = 16, col = "turquoise",
     cex = 1.5)
```

One problem here, though, is that if someone were to see our plot, they would not know what the colors mean! The same held for our line plot with multiple teams. We can remedy this with the `legend()` function. Here, we tell `legend()` that we want it in the top right corner (where there aren't any points to cover up), and tell it the colors and labels for each. See below.

```
#add a legend
plot(HR ~ SB, data = mlbdat, main = "Home Runs and Stolen Bases",
     xlab = "Stolen Bases (SB)", ylab = "Home Runs (HR)")
points(HR ~ SB, data = astros, pch = 16, col = "gold", cex = 1.5)
points(HR ~ SB, data = dodgers, pch = 16, col = "blue", cex = 1.5)
points(HR ~ SB, data = marlins, pch = 16, col = "turquoise",
     cex = 1.5)
legend("topright", legend = c("Other", "Astros", "Dodgers",
     "Marlins"), col = c("black", "gold", "blue", "turquoise"),
     pch = c(1, 16, 16, 16))
```

Now you can tell which points apply to which teams in the plot. We should *always* include a legend on a plot when there is information presented through shapes or colors so that anyone reading the figure knows what's going on.

Lab 2-1 Practice Questions. These questions should be completed prior to turning in your final lab. Please include all code (commented with the question number) in the same R Script that you used in class for each question.

1. In lab, we used the `aggregate()` function to aggregate `HR` and `SB` by `Year`. Go ahead and do the same here, but this time do it by `Team`. Name the new data frames `teamHR` and `teamSB` (you'll notice we actually did `teamHR` already in lab). With these data, make a bar plot for each of the two variables, with Team on the x-axis and `HR` or `SB` on the y-axis. Put them side-by-side using the `par(mfrow = ))` function. Note that you can copy and paste the plot here by using the **Export** button and copying it over.

2. Using the `teamSB` data, make a histogram of stolen bases. Do the same for `teamHR`. Do they look similar? Describe each distribution and compare them to one another.

3. Next, we'll merge together the `teamSB` and `teamHR` data sets with a new function, `merge()`. This function allows you to stick together 2 datasets based on a **key** variable. Here, we'll use `Team` as the key and the code below. Call this new data frame `teamBoth`.

   ```
   teamBoth <- merge(teamHR, teamSB, by = "Team", all = TRUE)
   teamBoth
   ```

   Note that the `by` argument tells R which column to use for the merge, and you want to merge `all` the data, so that's `TRUE`. Hopefully you'll see the data frame printed out now with one `Team` column, a `HR` column, and the `SB` column at the team level. Using this data, make a scatter plot of HR by SB. What do you see? Does the relationship look like the raw data's relationship?

4. You now have a number of tools at your disposal using R. For this last practice, I want you to choose 2 other continuous numeric variables from the data set and think about the distributions of those variables and how they relate to one another. Complete the following:
   a. State a useful research question about the relationship between these two variables.
   b. Make a histogram for each variable, side-by-side. Make sure to give the histograms a title and axis labels. Describe what you see and compare their distributions.
   c. Use R to calculate the mean, median, and standard deviation of each of your variables. Round to 3 decimal places. Which seems to have more variability?
   d. Make a bar plot of one of the variables by team. Make a bar plot of the other variable by year. Make sure to give the bar plot a title and axis labels. What do you notice about team-level differences? Any trends across time for your other variable?
   e. Make a scatterplot of your variables. Be careful to think about your research question and which might be the dependent vs. independent variable. Note that the dependent variable goes on the y-axis and the independent variable on the x-axis. Be sure to give your plot a title and label the axes appropriately. Describe any relationship you see here. Does it seem weak, strong, or moderate?
   f. In 3 or 4 sentences, summarize everything you've found in your data, and relate it back to your original research question.