Introduction to R and RStudio using baseball stats

Michael Lopez

The movie Moneyball focuses on the "quest for the secret of success in baseball". It follows a low-budget team, the Oakland Athletics, who believed that underused statistics, such as a player's ability to get on base, betterpredict the ability to score runs than typical statistics like home runs, RBIs (runs batted in), and batting average. Obtaining players who excelled in these underused statistics turned out to be much more affordable for the team.

In this lab we'll be looking at data from all 30 Major League Baseball teams and examining the relationships between in-game statistics. Our primary aim, for today, is a familiarity with R and RStudio, which you'll be using throughout the course both to learn the statistical concepts discussed in class and also to analyze real data and come to informed conclusions.

To straighten out which is which: R is the name of the programming language itself and RStudio is a convenient interface.

As the labs progress, you are encouraged to explore beyond what the labs dictate; a willingness to experiment will make you a much better programmer. Before we get to that stage, however, you need to build some basic fluency in R. Today we begin with the fundamental building blocks of R and RStudio: the interface, reading in data, and basic commands.

The panel in the upper right contains your *workspace* as well as a history of the commands that you've previously entered. Any plots that you generate will show up in the panel in the lower right corner.

The panel on the left is where the action happens. It's called the *console*. Everytime you launch RStudio, it will have the same text at the top of the console telling you the version of R that you're running. Below that information is the *prompt*. As its name suggests, this prompt is really a request, a request for a command. Initially, interacting with R is all about typing commands and interpreting the output. These commands and their syntax have evolved over decades (literally) and now provide what many users feel is a fairly natural way to access data and organize, describe, and invoke statistical computations.

To get you started, enter the following command at the R prompt (i.e. right after `>` on the console). You can either type it in manually or copy and paste it from this document. This command instructs R to access the OpenIntro website and fetch some data: the `mlb11` data set.

```
download.file("http://www.openintro.org/stat/data/mlb11.RData", destfile = "mlb11.RData")
load("mlb11.RData")
```

Let's load up the data for the 2011 season. In addition to runs scored, there are seven traditionally used variables in the data set: at-bats, hits, home runs, batting average, strikeouts, stolen bases, and wins. There are also three newer variables: on-base percentage, slugging percentage, and on-base plus slugging. For the first portion of the analysis we'll consider the seven traditional variables. At the end of the lab, you'll work with the newer variables on your own.

You should see that the workspace area in the upper righthand corner of the RStudio window now lists a data set called `mlb11` that has 30 observations on 12 variables. As you interact with R, you will create a series of objects. Sometimes you load them as we have done here, and sometimes you create them yourself as the byproduct of a computation or some analysis you have performed. Note that because you are accessing data from the web, this command (and the entire assignment) will work in a computer lab, in the library, or in your dorm room; anywhere you have access to the Internet.

The Data: 2011 baseball statistics

```
mlb11
```

What you should see are four columns of numbers, each row representing a different team: the first entry in each row is simply the row number (an index we can use to access the data from individual years if we want), the second is the team, and the remaining columns are team-specific metrics. Use the scrollbar on the right side of the console window to examine the complete data set.

Note that the row numbers in the first column are not part of the MLB data. R adds them as part of its printout to help you make visual comparisons. You can think of them as the index that you see on the left side of a spreadsheet. In fact, the comparison to a spreadsheet will generally be helpful. R has stored MLB data in a kind of spreadsheet or table called a *data frame*.

You can see the dimensions of this data frame by typing:

```
dim(mlb11)
```

```
## [1] 30 12
```

This command should output `[1] 30 12`, indicating that there are 30 rows and 12 columns (we'll get to what the `[1]` means in a bit), just as it says next to the object in your workspace. You can see the names of these columns (or variables) by typing:

```
names(mlb11)
```

```
##  [1] "team"         "runs"         "at_bats"      "hits"
##  [5] "homeruns"     "bat_avg"      "strikeouts"   "stolen_bases"
##  [9] "wins"         "new_onbase"   "new_slug"     "new_obs"
```

You should see that the data frame contains the columns `team`, `runs`, and `at_bats`, etc. At this point, you might notice that many of the commands in R look a lot like functions from math class; that is, invoking R commands means supplying a function with some number of arguments. The `dim` and `names` commands, for example, each took a single argument, the name of a data frame.

One advantage of RStudio is that it comes with a built-in data viewer. Click on the name `mlb11` in the *Environment* pane (upper right window) that lists the objects in your workspace. This will bring up an alternative display of the data set in the *Data Viewer* (upper left window). You can close the data viewer by clicking on the x in the upper lefthand corner.

Some Exploration

Let's start to examine the data a little more closely. We can access the data in a single column of a data frame separately using a command like

```
mlb11$runs
```

This command will only show the number of runs scored each year by each team.

1. What command would you use to extract just the number of hits for each team? Try it!

Notice that the way R has printed these data is different. When we looked at the complete data frame, we saw 30 rows, one on each line of the display. These data are no longer structured in a table with other variables, so they are displayed one right after another. Objects that print out in this way are called *vectors*; they represent a set of numbers. R has added numbers in [brackets] along the left side of the printout to indicate locations within the vector. For example, `1599` follows `[1]`, indicating that `1599` is the first entry in the vector. And if `[20]` starts a line, then that would mean the first number on that line would represent the 20th entry in the vector.

One more super-useful command, `head()`, which shows the first six rows of any data frame.

```
head(mlb11)
```

```
##                 team runs at_bats hits homeruns bat_avg strikeouts
## 1       Texas Rangers  855    5659 1599      210   0.283        930
## 2      Boston Red Sox  875    5710 1600      203   0.280       1108
## 3      Detroit Tigers  787    5563 1540      169   0.277       1143
## 4  Kansas City Royals  730    5672 1560      129   0.275       1006
## 5 St. Louis Cardinals  762    5532 1513      162   0.273        978
## 6       New York Mets  718    5600 1477      108   0.264       1085
##   stolen_bases wins new_onbase new_slug new_obs
## 1          143   96      0.340    0.460   0.800
## 2          102   90      0.349    0.461   0.810
## 3           49   95      0.340    0.434   0.773
## 4          153   71      0.329    0.415   0.744
## 5           57   90      0.341    0.425   0.766
## 6          130   77      0.335    0.391   0.725
```

Related: 'tail()'

```
tail(mlb11)
```

```
##                   team runs at_bats hits homeruns bat_avg strikeouts
## 25       Tampa Bay Rays  707    5436 1324      172   0.244       1193
## 26       Atlanta Braves  641    5528 1345      173   0.243       1260
## 27 Washington Nationals  624    5441 1319      154   0.242       1323
## 28 San Francisco Giants  570    5486 1327      121   0.242       1122
## 29     San Diego Padres  593    5417 1284       91   0.237       1320
## 30     Seattle Mariners  556    5421 1263      109   0.233       1280
##    stolen_bases wins new_onbase new_slug new_obs
## 25          155   91      0.322    0.402   0.724
## 26           77   89      0.308    0.387   0.695
## 27          106   80      0.309    0.383   0.691
## 28           85   86      0.303    0.368   0.671
## 29          170   71      0.305    0.349   0.653
## 30          125   67      0.292    0.348   0.640
```

mosaic

There is an additional package for R called `mosaic` that streamlines all of the commands that you will need in this course. This package is specifically designed by a team of NSF-funded educators to make R more accessible to statistics students like you. The `mosaic` package doesn't provide new functionality so much as it makes existing functionality more logical, consistent, all the while emphasizing importants concepts in statistics.

To use the package, you will first need to install it.

```
install.packages("mosaic")
```

Note that you will only have to do this *once*. However, once the package is installed, you will have to load it into the current workspace before it can be used.

```
require(mosaic)
```

```
## Warning: replacing previous import by 'grid::arrow' when loading 'mosaic'

## Warning: replacing previous import by 'grid::unit' when loading 'mosaic'
```

Note that you will have to do this *every* time you start a new R session.

R has some powerful functions for making graphics.

The centerpiece of the `mosaic` syntax is the use of the *modeling language*. This involves the use of a tilde (~), which can be read as "is a function of". For example, we can create a simple plot of the number of runs each team scored versus its hits

```
xyplot(runs ~ hits, data=mlb11)
```

By default, R creates a scatterplot with each x,y pair indicated by an open circle. The plot itself should appear under the *Plots* tab of the lower right panel of RStudio. Notice that the command above again looks like a function, this time with two arguments separated by a comma. The first argument in the plot function specifies the variable for the x-axis and the second for the y-axis.

You might wonder how you are supposed to know that it was possible to add that third argument. Thankfully, R documents all of its functions extensively. To read what a function does and learn the arguments that are available to you, just type in a question mark followed by the name of the function that you're interested in. Try the following.

```
?xyplot
```

Notice that the help file replaces the plot in the lower right panel. You can toggle between plots and help files using the tabs at the top of that panel.

2. Is there an association between runs scored and hits? How would you describe it?

Now, suppose we want to plot only the total number of runs.

```
histogram( ~ runs, data=mlb11)
bwplot( ~ runs, data=mlb11)
```

3. Describe the distribution of runs scored among MLB teams in 2011. What is the median?

Additional R functionalities

R can do lots of things besides graphs. In fact, R is really just a big calculator. We can type in mathematical expressions like

```
210 + 930
```

to see the total number of homeruns and strikeouts by the Texas Rangers, the first row in our data. We could do this for every row, but there is a faster way. If we add the vectors together, R will compute all sums simultaneously.

```
mlb11$homeruns + mlb11$strikeouts
```

What you will see are 30 numbers (in that packed display, because we aren't looking at a data frame here), each one representing the sum we're after. Take a look at a few of them and verify that they are right.

4. Which team had the highest cumulative number of strikeouts and homeruns in 2011?

Note that with R as with your calculator, you need to be conscious of the order of operations. For example, not the following example.

```
4*6+2
```

```
## [1] 26
```

```
4*(6+2)
```

```
## [1] 32
```

Finally, in addition to simple mathematical operators like subtraction and division, you can ask R to make comparisons like greater than, `>`, less than, `<`, and equality, `==`. For example, we can ask which home run observations were greater than 200.

```
mlb11$homeruns > 200
```

This output shows a different kind of data than we have considered so far. In the `mlb11` data frame, most of our values are numerical. Here, we've asked R to create *logical* data, data where the values are either `TRUE` or `FALSE`. In general, data analysis will involve many different kinds of data types, and one reason for using R is that it is able torepresent and compute with many of them.

5. Which MLB teams had more than 200 home runs in 2011?

Summaries and tables

A good first step in any analysis is to distill all of that information into a few summary statistics and graphics. As a simple example, the function `summary` returns a numerical summary: minimum, first quartile, median, mean, second quartile, and maximum. For `runs` this is

```
summary(mlb11$runs)
```

If you wanted to compute the interquartile range for team runs, you could look at the output from the summary command above and then enter

```
734 - 629
```

R also has built-in functions to compute summary statistics one by one. For instance, to calculate the mean, median, and variance of `runs`, type

```
var(~runs, data=mlb11)
median(~runs, data=mlb11)
```

The `mosaic` command `favstats`, allows us to compute all of this information (and more) at once.

```
favstats(~runs, data=mlb11)
```

While it makes sense to describe a quantitative variable like `runs` in terms of these statistics, what about categorical data? We would instead consider the sample frequency or relative frequency distribution. The function `tally` does this for you by counting the number of times each kind of response was given. For example, to see the number of teams who hit at least 200 homeruns, type

```
mlb11$HighHR <- mlb11$homeruns >= 200
mlb11$HighHR
tally(~ HighHR, data=mlb11)
```

or instead look at the relative frequency distribution by typing

```
tally(~ HighHR, data=mlb11, format="proportion")
```

In each of the above steps, we make use of a new variable, `HighHR` (the contents of which we can see by typing `HighHR` into the console) and then used it in as the input for `tally`. The special symbol `<-` performs an *assignment*, taking the output of one line of code and saving it into an object in your workspace. This is another important idea that we'll return to later.

Notice how R automatically divides all entries in the table by 30 in the second command. This is similar to something we observed earlier; when we multiplied or divided a vector with a number, R applied that action across entries in the vectors. As we see above, this also works for tables. Next, we make a bar chart of the entries in the table by putting the table inside the`barchart` command.

```
barchart(tally(~HighHR, data=mlb11, margins=FALSE), horizontal=FALSE)
```

Notice what we've done here! We've computed the table of and then immediately applied the graphical function, `barchart`. This is an important idea: R commands can be nested. You could also break this into two steps by typing the following:

```
HR <- tally(~ HighHR, data=mlb11, margins=FALSE)
barchart(HR, horizontal=FALSE)
```

6. Create a numerical summary for `stolen_bases` and `wins`, and compute the interquartile range for each.

7. Compute the relative frequency distribution for teams with at least 90 `wins`. How many teams reached 90 wins?

The `tally` command can be used to tabulate any number of variables that you provide.

8. What is shown in the following table? How many teams won 90 or more games and hit at least 200 home runs?

```
mlb11$HighWins <- mlb11$wins >=90
tally(HighWins ~ HighHR, data=mlb11, format="count")
```

How R thinks about data

We mentioned that R stores data in data frames, which you might think of as a type of spreadsheet. If we want to access a subset of the full data frame, we can use row-and-column notation. For example, to see the sixth variable of the 7th respondent, use the format

```
mlb11[7,6]
```

which means we want the element of our data set that is in the 7th row (meaning the 7th team) and the 6th column (in this case, batting average). We know that `bat_avg` is the 6th variable because it is the 6th entry in the list of variable names

```
names(mlb11)
```

To see the `bat_avg` for the first 10 respondents we can type

```
mlb11[1:10,6]
```

In this expression, we have asked just for rows in the range 1 through 10. R uses the `:` to create a range of values, so 1:10 expands to 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. You can see this by entering

```
1:10
```

Finally, if we want all of the data for the first 10 respondents, type

```
mlb11[1:10,]
```

By leaving out an index or a range (we didn't type anything between the comma and the square bracket), we get all the columns. When starting out in R, this is a bit counterintuitive. As a rule, we omit the column number to see all columns in a data frame. Similarly, if we leave out an index or range for the rows, we would access all the observations, not just the 7th, or rows 1 through 10. Try the following to see the batting averages for all 30 teams.

```
mlb11[,6]
```

There's an easier way, which you will recall from earlier

```
mlb11$bat_avg
```

The dollar-sign tells R to look in data frame `mlb11` for the column called `bat_avg`. Since that's a single vector, we can subset it with just a single index inside square brackets. We see the batting average for the 7th respondent by typing

```
mlb11$bat_avg[7]
```

Exercises.

9. Describe the center, shape, and spread of the `stolen_bases` variable, using an appropriate plot and the appropriate metrics.

```
### Enter code here.
### Note that any comments in code after the `#` do not get read by R
```

10. A coach is interested in the link between `stolen_bases` and `runs`. Show the coach a scatter plot, and describe the association. As you make the plot, think carefully about which of these two variables is the explanatory variable (and which is the response).

```
### Enter code here.
```

11. How can you change the x and y labels on your plots? How can you add a title? Use google to guide you, and update your plot in Question 10.

```
### Enter code here.
```

12. Using visual evidence, find the variable that you think seems to boast the strongest association to `runs`. Consider any continuous variables between columns 3 (`at_bats`) and 12 (`new_obs`).

```
### Enter code here.
```

13. What is the variance in the number of strikeouts for each team during the 2011 season?

```
### Enter code here.
```

14. Make a new variable, `high_BA`, to represent teams that hit for a batting average of 0.270 or more. How many teams fit into this group?

```
### Enter code here
```

15. You can use favstats to get the summary statistics within each `high_BA` group, and `bwplot()` to make boxplots. For example, what appears to be the link between `homeruns` and `high_BA`? Do teams with higher batting averages tend to hit more home runs?

```
favstats(homeruns ~ high_BA, data = mlb11)
densityplot(~ homeruns, groups = high_BA, data = mlb11, auto.key = TRUE)
```

16. Repeat the code above, only using `stolen bases` instead of `homeruns`. Does there appear to be a link between `stolen bases` and `high_BA`?

```
### Enter code here.
```

That was a short introduction to R and RStudio, but we will provide you with more functions and a more complete sense of the language as the course progresses. Feel free to browse around the websites for R and RStudio if you're interested in learning more.