STAT 215: Lab for Week 2

Colin Reimer Dawson

Last Revised September 8, 2017

What to Turn In

For this lab you will create a file called an "RMarkdown" document, which allows you to interleave code, results, plots, and verbal commentary in a single file. For Tuesday, turn in a single `.Rmd` "source" file, as well as a "compiled" `.pdf` document, with your commands, results, plots, and commentary from Exercises 1-12.

Log in to the RStudio Server

If you are using the server, here is a reminder about how to log in.

Instructions

1. In your web browser, visit `rstudio.oberlin.edu`.
2. Your username is your Obie ID, and you should have set a password last week. If not, it is also your Obie ID (but you should refer back to Lab 1 for instructions on changing it before moving on)

1 R Markdown for Reproducible Research

Last time we learned about script files as a way to keep track of what we have done. It is good practice to record exactly what you did so that you can reproduce it later. This extends to plots and graphs as well. Last time, you created several plots. If

we wanted to present our models and descriptions of this data, we'd like to include up-to-date plots which (a) use the most current data we have, and (b) reflect exactly any manipulations we did in the code. A script only helps us so much here: at some point we would need to cut and paste graphs into a Word document or some such writeup, and we run the risk of getting things out of sync.

A powerful solution is to use a **Markdown** document, where code, plots, and descriptive text can be interleaved in a file, which gets "compiled" (or "Knit") into a well-formated document that includes all of the above (or, you can omit the code). Every time you make a change, just re-Knit, and the changes will be updated in the document.

1.1 R Markdown Workflow

The following is a sequence of steps that you should follow the first few times you create a Markdown document. Once you have done this a few times you will develop your own habits, but these are good habits to develop for starters.

1. From the RStudio File menu, select New File > R Markdown.... Now you have several options. From the list on the lefthand side, select From Template, and then on the right, pick mosaic fancy for now, to get a template that includes a bunch of example information. As you create your own documents you will want to delete much of this extra stuff, but reading this template is a good way to learn about how Markdown works.
2. The first thing you should do after creating a new Markdown document is to save the file. Do File > Save As... and give your document a name.
3. Before editing anything, press the Knit HTML button in the document toolbar. If on the server you may get a message about a blocked pop-up. Allow it to show, and you will get a new window with a mix of code, text, and plots.
4. The first thing you should edit is the Title and Author fields at the top of the document. Give the document a title such as "STAT 215: Lab 2", and put your name inside the quotes on the Author line. Re-Knit to see your change reflected in the document.

5. In general, every time you make a change, it is a good idea to Knit again. That way, if you get an error message, you know what caused it. As you get more experienced you may find yourself going longer between re-Knits, but at first you should do it very frequently.

Look over the original source document. One thing that jumps out is that all of the R code in the document is inside triple quote markers. These are actually back-quotes, which are found on the same key as the tilde (~) on most keyboards.

To create a new code chunk, place the cursor where you want the code to go, and click the Chunks button at the upper right of the toolbar, and select Insert Chunk (you will also see a keyboard shortcut that you could use instead if you prefer). This will create the "fence" markers that tell the compiler that anything inside is code (and is interpreted as in an R Script), and anything outside is ordinary text.

Protip You should *not* include code in your Markdown document that you would not want to be run over and over again, or that produces output that you would not put in a writeup. This includes `install.package()` lines, lines that access documentation (e.g., `?EmployedACS`), or generate dialogue boxes.

1.2 Markdown Exercises

1. Edit the very first code chunk to include `library()` lines for the packages we are using (i.e., `mosaic`, `Stat2Data`, `Lock5Data`).

   The `include = FALSE` option in curly braces tells the Knit process not to display the code or any output from that chunk in the final document. It should just quietly run it for its "side-effects". Re-Knit to make sure you don't get any errors.

2. Create a new code chunk and create a the histogram and box plot of the `Income` variable from the `EmployedACS` dataset (in the `Lock5Data` package). (Recall the `histogram()` and `bwplot()` functions from last week use the form `function(~Variable, data = DataSet)`.) Knit to verify that you have everything you need. You may need to add one or more lines to load the requisite data into memory, if you haven't done so already.

> Note that with a Markdown document, unlike a script, all prerequisite commands must be in the document; the Knit procedure will not be able to use any data, packages, or variables that you have loaded at the console or from a script. In other words, a Markdown document must be like a completely self-contained R session.

3. Outside the code chunk, write a sentence (it does *not* have to be a comment if it is outside a code chunk) describing what the plots show (e.g., estimate the center and spread of the data).

2 Basic Properties of Measures of Center and Spread

In the following exercise you will examine the effect of extreme values on various measures of center and spread of a quantitative variable. You will also practice using `filter()` to select cases.

Recall that we can produce a "reduced" dataset including only those cases that meet a certain condition using the `filter()` function. Last time we did something like

```
filter(EmployedACS, Sex == 0)
```

to produce a reduced Employment dataset containing only those cases where `Sex == 0` (i.e., females).

We saw three ways to "chain" or "nest" commands:

(A) Direct nesting, substituting a whole command directly into another, as in

```
histogram(~HoursWk, data = filter(EmployedACS, Sex == 0))
```

(B) "Saving" the output of one command as a named variable and using that in the second command:

```
EmployedACS.Females <- filter(EmployedACS, Sex == 0)
histogram(~HoursWk, data = EmployedACS.Females)
```

(C) "Chaining" commands using the 'pipe' operator:

```
filter(EmployedACS, Sex == 0) %>%
    histogram(~HoursWk, data = .)
```

where the . represents the output of the "pipe".

2.1 Exercise on Properties of Center and Spread

4. Filter out any cases with income above half a million dollars ($500K). What percentage of cases remain? How does this filtering affect the mean income? How does it affect the median? How does it affect the standard deviation? How does it affect the **interquartile range** (the difference between the first and third quartiles). You can use the `IQR` function to get this value, or just subtract the quartiles. Include code you used to answer these questions in a code chunk in your Markdown document, and put your answers themselves outside the chunk.

3 Confounding and Simpson's Paradox

In this section you will explore an instance of **Simpson's Paradox**, where controlling for a third confounding variable can result in a reversal of the direction of association between two other variables of interest. It is not truly a "paradox" in the technical sense of a logical contradiction, but it does produce some counterintuitive results.

In this section you will also learn some R commands to produce scatterplots, plot regression lines, compute correlations, and create new variables out of old ones.

3.1 Exercises on Simpson's Paradox

Include code and/or responses for the following in your Markdown document. Put code inside a code chunk "fence", and interpretation text outside code chunks. Remember to Knit frequently to verify that you haven't introduced an error.

5. Read the dataset `SATs.csv` from the course website (it is at `http://colindawson.net/data/SATs.csv`) into R (recall that the `read.file()` command from the `mosaic` package can read in a file from a URL supplied in quotes as an argument). The cases in this data are U.S. states. The dataset contains 2010 data on statewide average SAT scores in reading (`Read`), math

(`Math`), writing (`Write`), and in (`Total`), percent of students taking the SAT (`PercentSAT`), and average teacher salary (`Salary`).

6. Examine the data with `head()` to confirm that it looks as it should.

7. Create a scatterplot relating states' average SAT scores to their average teacher salaries, plotting a regression line on the data to predict mean SAT scores as a function of teacher salary. You can use the `xyplot()` function to do this, which has the following syntax:

```
xyplot(Y ~ X, data = DataSet, type = c("p","r"))
```

   where `X` and `Y` are the names of the variables to be plotted on the x and y axis, and the argument `type = c("p", "r")` tells the `xyplot()` function to plot both points ("p") and a regression line ("r"). The `c()` function is R's way of "concatenating" multiple values into a list.

   Describe the pattern you see. Give at least two possible explanations for the relationship, one causal and at least one involving a possible confounding variable.

8. Compute the Pearson correlation (notation, r) between the two variables. Use the `cor()` function, which has the form `cor(Y ~ X, data = DataSet)`.

   > If you are not familiar, Pearson's correlation, r, is a number between -1 and 1 that describes how strong the *linear* association is between two quantitative variables — that is, how close would the data points be to a straight line on a scatterplot — where 1 indicates a perfect positive linear association, -1 indicates a perfect negative linear association, and 0 indicates no overall linear association.

9. A possible confounding variable is the *percentage of students* who take the SAT in each state. Create scatterplots with fitted regression lines, and compute correlation coefficients, between the percentage of students taking the SAT and each of the two variables you plotted above. Do your findings here affect your interpretation of the previous correlation?

10. Let's create a categorical variable out of the `PercentSAT` variable whose values correspond to "low", "medium" and "high" percentages of SAT attempts in the state. Type the following:

```
SATS.augmented <-
    mutate(SATs,
           SAT.Attempt.Rate = cut(PercentSAT, breaks = c(0,5,50,100)))
```

replacing `SATs` with whatever name you used for the dataset.

> The `mutate()` function is taking the data and creating a new variable called `SAT.Attempt.Rate` which is defined by the formula given on the right of the `=` sign. In this case, the formula says to create a categorical variable out of the quantitative `PercentSAT` variable by "cutting" the range into bins: "between 0 and 5", "between 5 and 50", and "between 50 and 100". We are saving the resulting augmented dataset as a new object called `SATS.augmented`.

11. Let's plot each of these three groups with their own regression lines relating salary and mean SAT score. Repeat the `xyplot()` command from above, but add the following two new arguments inside the parentheses:

```
groups = SAT.attempt.rate  # color codes the data by this grouping variable
auto.key = TRUE            # adds a legend at the top
```

12. Compute the correlation between `Salary` and `Total` for each subgroup (use `filter()`). Describe how the pattern in the correlations and regression lines differ when considering each subgroup separately, and hypothesize what might account for the discrepancy.