STAT 213: R/RStudio Intro

Colin Reimer Dawson

Last Revised February 10, 2016

1 Starting R/RStudio

Skip to the section below that is relevant to your choice of implementation.

Installing R and RStudio Locally

This section is only needed if you are working on your own laptop and you want to have a local install of the software. If you just want to work on the server, you can skip this part.

Download and Install R

1. Visit `cran.r-project.org` and click the link at the top of the page for your operating system. Most likely you want the first available link on the resulting page.
2. *Mac*: Assuming you have a recent version of OS X, click on the first available link, download the `.pkg` file, and install it as you would normally install an application on OS X. *Windows:* Select "base", and then "Download R 3.2.2 for Windows". Open the executable and follow the instructions of the installer.

Download and Install RStudio

1. Visit `www.rstudio.com`, and click the button labeled "Download RStudio" in the top panel (the panel will cycle through a few screens — the one we want is on the first one).
2. Click the link for "RStudio Desktop", and then "Download RStudio Desktop" under "Open Source Edition".
3. Under "Installers", select the link for your operating system to download the installer.
4. Open the installer, and follow the instructions to install RStudio.

Log in to the RStudio Server

This section is for those who want to access the software from a web browser. The advantage of this approach is that you can get to your account, and your files, from anywhere, and you don't have to install anything. A disadvantage is that you have to upload and download files to the server if you want them to start/end from your computer.

Note: If you are not officially enrolled, you will need to use a temporary account, which means that any files you create will not be accessible later unless you save them to your computer (or save them to the lab computer and send them to yourself / upload them to the cloud at the end of class). Students who are enrolled should have accounts set up. I will tell you how to log in in class.

Instructions

1. In your web browser, visit `rstudio.oberlin.edu`.
2. If this is your first time using the server, your username and password are both your Obie ID.
3. You should change your password so that your account will be secure. To do this, go to the Tools dropdown menu (in RStudio), and select Shell. Type `passwd`. You will first be prompted for your old password. As you type you will not see the cursor move, but the server is (most likely) registering your

keystrokes. Press Enter when done. You will then be prompted to enter a new password twice. Again the cursor will not move as you type.

2 Navigating the RStudio Environment

The RStudio environment consists of **panes**, each of which may have several tabs (as in a web browser).

When you first open RStudio, the left-hand pane (by default) displays the **Console**. Here, you can enter commands one at a time to see what they do.

Enter some simple arithmetic expressions at the console and verify that the output is as expected.

```
2 + 5
13 * 17
17.04 / 2.8
sqrt(49)
```

In the lower right is a pane with various tabs to do things like keep track of your files, view graphs and plots that you create, get help messages, and see what **packages** are installed. Since R is open and free, anyone who wants to can create a module of code to extend or streamline functionality, and that others can download and use.

Installing and Using R Packages In each session, any time you want to use commands that are part of a package that is not part of core R, you need to tell R to look in that package for names of functions and datasets, using the `library()` function. One such package we will use a lot is called `mosaic`. Try typing the following:

```
library("mosaic")
```

Note the double quotes. (In this particular case the command would work without the quotes, but it is good practice to include them, as often times leaving out quotes will produce an error.)

If you just installed R, you probably got an error message, to the effect that the `mosaic` package does not exist! It is already installed on the server, but you will

need to install it on your own machine (you should only need to do this once). Type

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

You may get an error message that some other package (which `mosaic` depends on) is not found. In this case modify the above line as follows:

```
install.packages("mosaic", dependencies = TRUE)
```

This tells R to automatically install any other packages that are required.

While you're at it, also install some other packages that contain datasets we will use (again, you can skip this step if you are on the server).

```
install.packages("Stat2Data", dependencies = TRUE)
install.packages("Lock5Data", dependencies = TRUE)
```

3 Data Frames

Datasets are represented in R as tables called **data frame**s, where each row represents a case/observational unit, and each column represents a variable. We can see a list of datasets available in a particular package (say, `Stat2Data`) by typing the following:

```
data(package = "Stat2Data")
```

Note the quotes around the package name, but *not* around the word package. We will see why this is the case soon.

If we want to work with a particular dataset, we can make it visible by typing its name inside the `data()` function, for example:

```
data(Pollster08, package = "Stat2Data")
```

We can look at the first few rows of the dataset with the `head()` function:

```
head(Pollster08, n = 4)
```

The `n = 4` indicates that we want the first 4 rows.

We can ask R to report the number of rows (cases) and the number of columns (variables) in the data using `nrow()` and `ncol()`:

```
nrow(Pollster08)
ncol(Pollster08)
```

We can also look at the code book (i.e., the description of what the data is, how it was collected, and what each variable means) by typing

```
?Pollster08
```

This will only work if we have first made the data visible using the `data()` function as above.

Exercise 1 (updated 2/10) Answer the following questions about some datasets in the `DCF` package. (The package is already installed on the server. UPDATE: `DCF` is not available from the standard package repository, so you'll need to get it from the author's own repository. First install the `devtools` package, then type:

```
devtools::install_github("dtkaplan/DCF")
```

This will download `DCF` from the author's page, and install it)

(a) How many variables are there in `CountryData` dataset?

(b) How many cases are there in `WorldCities`?

(c) What's the third variable in `BabyNames`?

(d) What are the codes for the categorical variable `party` in the `RegisteredVoters` dataset, and what does each stand for?

4 Functions, Arguments, and Commands

Most of what we do in R consists of applying **functions** to data objects, specifying some options for the function, which are called **arguments**. Together, the application of a function, together with its arguments, is called a **command**.

A useful analogy is that commands are like sentences, where the function is the verb,

and the arguments (one of which usually specifies the data object) are the nouns. There is often a special argument that comes first. This is like the direct object of the command.

For example, in the English command, "Draw a picture for me with some paint", the verb "draw" acts like the function (what is the listener supposed to do?); the noun "picture" is the direct object (draw what?), and "me" and "paint" are extra (in this case, optional) details, that we might call the "recipient" and the "instrument".

By comparison, when above we said

```
head(Pollster08, n = 4)
```

we are applying the `head()` function to the dataset `Pollster08`, and giving an extra detail, `n = 4` that modifies what happens. Arguments go inside the parentheses, and are always separated by commas when there is more than one. Note that the first argument wasn't given a name, but the modifier has the name `n` and the value `4`. This is typical. Named arguments appear as above, with the name first, then an equals sign, then the value. They can occur in any order.

Sometimes the value will have quotes around it. The name will not. More on this distinction shortly.

Exercise 2 Inspect the following command that was used above:

```
install.packages("DCF", dependencies = TRUE)
```

Identify all of the function names, argument names, and argument values.

5 Objects and Variables

We already saw one kind of object: a data frame. Other kinds of objects include **text strings**, which are written with quotes around them (these are often used as argument values), **literals**, such as numbers (like the 4 above — no quotes) and the special values `TRUE` and `FALSE` (note the all caps). There are also objects called **vectors**, which are like lists that can either contain text strings or numbers. We haven't seen any of these yet, but each variable (column) in a data frame is represented as one of these.

When we want to use an object a lot (such as a numeric value, like a mean, from some statistical computation), it is helpful to give it a name so we can refer to it by what it represents, instead of by its values. Data frame objects almost always have names, so that we can refer to them.

We can give a name to an object using the `name <- value` form. This process is called **assignment**, and the named thing is called a **variable** (which is a different sense than a variable in statistics). For example:

```
my.name <- "Colin Dawson"
my.age <- 34
```

You can read the `<-` symbol as "gets", as in "(The name) `my.name` gets (the value) `''Colin Dawson''`". Notice that there is just one hyphen. A common error is to add an extra hyphen to the arrow, which R will misinterpret as a minus sign. The periods in the variable names are just part of the name; they don't have any special meaning other than as separators between words. It is also legal to use underscores and digits in variable names, but none of these can be used at the very beginning.

We can also store the result of a command in a named variable. A simple example is the following:

```
hypotenuse <- sqrt(25)
```

We can also use variables as the values of arguments, such as in:

```
a.squared <- 3^2
b.squared <- 4^2
a.squared.plus.b.squared <- a.squared + b.squared
```

Notice that every time we define a variable, it appears in the upper right pane, in the Environment tab. This shows us everything we've defined.

If you want to read in a dataset from a file, you need to give it a name within R. There is a dataset on my website involving measurements of depression in 18 people. Type the following to get it and save it to a data frame called `Depression`.

```
Depression <- read.file("http://colinreimerdawson.com/data/depression.csv")
```

Note that file names are always in quotes because they refer to a location outside the R environment itself, not to a variable.

If you are on a local installation, you may get an error message at this point to the

effect that R can't find the function `read.file()`. This is because it lives in the `mosaic` package, and we haven't successfully activated it. Do

```
library("mosaic")
```

and then run the `read.file()` line again. Notice that, when your cursor is at the console, you can use the up and down arrows to cycle through recently typed commands.

Now that the `Depression` data frame is defined, take a look at the first five rows with the `head()` function:

```
head(Depression, n = 5)
```

Exercise 3 Each of the following commands has an error in it. What is it?

(a) `ralph <- sqrt 10`

(b) `ralph2 <- "Hello to you!"`

(c) `3ralph <- "Hello to you!"`

(d) `ralph4 <- "Hello to you`

6 R Scripts for Reproducible Research

Typing directly into the console is useful for experimenting. But when it comes time to do finished work, you want to be able to reproduce what you did. Instead of typing directly into the console, you can save lines of code into a file (called a "script"), and have R execute them back to back.

Creating a Script in RStudio Under the File menu, select New File, and choose R Script. used to create new files. Click here and select "R Script".

The left-hand pane will divide in two, with the console at the bottom, and the top a blank page. You can enter code on this page, line by line, and click "Run" to execute the current line, or the highlighted region. Or you can click "Source" to run the whole file.

Any code preceded on the same line by a pound sign (#) is interpreted as a comment, and will be ignored when running that line or the whole script. For example:

```
### This part of the code is an illustration of comments

## This line is a comment, so it doesn't have to be valid R code
library(Stat2Data) # Making Stat2Data visible to commands
data("BritishUnions") #Loading a dataset
```

Note that comments can either be on a line by themselves, or at the end of a line. A single pound sign is enough to identify something as a comment, but by convention, in-line comments have a single pound, whole line comments have two pound signs, and comments that pertain to a whole section of code have three or more.

Exercise 4 Type your answers to the previous three exercises in a script, with clearly labeled headings for each exercise. Include any code you needed to run as code, and any verbal responses (and headings) as comments. In addition, store the number of cases in the `Depression` dataset (from the file on the web) in a named variable.

Include any previous code needed to make your code run. To verify that your script is self-contained, select "Restart R", followed by "Clear Workspace" from the Session menu. This will wipe away any packages you have made visible, and any variables you have defined. Then, run the script. If you get errors that you did not get before clearing everything, you are missing something.

Save your script as a file and turn it in on Blackboard by Thursday at 6pm.