Week2: Getting used to Rstudio
Demography 213

Carl Mason

August 29, 2016

Abstract

Our two goals for this week are first to setup our personal computers to run the noMachine client – and thereby gain the ability to do science from home. And second to master Rstudio by working through Chapters 3 and 2 of Getting Started with Rstudio.

Before we meet on Wednesday:

- Please read Chapter 3 of Getting Started with Rstudio. You can access this book for free either from a campus ipaddress or via the library proxy server. Find it at: `http://proquest.safaribooksonline.com/9781449314798`.
  Chapter 3 is about R console window and the various tricks you can do with it. Chapter 3 is descriptive rather than "hands on" so you need only read it **carefully**. The most useful part of the chapter is the section called "Command-line Conveniences". We'll work through Chapter 2 in class.
- Try to get noMachine working on your portable computer and bring that computer to class on Wednesday. (See Section 1 for explanations and instructions). Don't worry (much) if you are unable to get noMachine to work on your own. We will get it working both on your laptop and on the department workstations on Wednesday. **We won't need to use noMachine until later. We're just planning ahead. We will generally be using Rstudio in the browser**

1 Installing and configuring noMachine

noMachine is a remote desktop system that lets you use the Demography Lab from home or from anywhere on the Internet. noMachine consists of a "server" which runs on the machine we call `quigley` and a **client** which runs on your portable computer.

The client is a small program that encrypts and decrypts and greatly compresses the information that flows between your portable and `quigley` allowing you to efficiently work on your Demography Lab desktop without being here.

Instructions for installing and configuring noMachine can be found at `lab.demog.berkeley.edu/LabWiki`. Click the "How to Do Stuff" link on the left navigation bar then click on "Help:Getting Started".

Please have a go at installing and configuring the noMachine client on your laptop but do not worry if it doesn't work. We'll get it going for you by the time you need it.

If you are in a hurry and are good at this sort of thing –or in other words are possessed of sufficient hubris, impatience and laziness[1] – just google noMachine; download and install the appropriate client for your computer and choose all the defaults. When you set up your connection specify hostname: `nmx.demog.berkeley.edu`. Of course you'll login using your Demography Lab userid and password.

But don't worry about noMachine this week.

[1] The essential qualities of good programmers according to Larry Wall

2 Mastering Rstudio

These instructions have to do with working through Chapter 2 of Getting Started with Rstudio. Please be sure to read Chapter 3 first.

You have already experienced the wonders of R, but as you have no doubt noticed, plain old R is a bit barbaric. Modern scholars expect fancy graphic front ends with intelligent features that help them work better and keep track of their stuff. That is where Rstudio comes in.

It is important to understand that Rstudio and R are separate programs. Rstudio is what might be called an "integrated development environment" or IDE. Not to be confused with IEDs, IDEs are sophisticated programs that understand the structure of your programming language and the broader environment in which you are working. As such, Rstudio can help you keep track of your data files and help you work with other tools to for example clean up your data before reading it into R.

Rstudio also runs in your browser and thus makes your research project available from anywhere.

This week, we will learn about Rstudio's features for dealing with files that are Not R code and several efficiency enhancing tricks that will hopefully make learning and using R easier.

The main difficulty with this week's Rstudio exploration is that, in places, it assumes a deeper knowledge of R than we currently have. Below, I try to smooth over those bits with kinder gentler examples and explanations but if you find it impenetrable, don't worry (much) we will meet `lapply` and anonymous functions later. Getting a rough idea of how these things work now will ease the pain of learning them for real later. The goal for this week is get comfortable with Rstudio – which will help us get better at R faster... but later.

2.1 Step by step

IN A TERMINAL WINDOW ...

1. Unless something went terribly wrong last week, you have already created a directory called `~/213/Week2` in which you should do this week's assignment. If for some reason you have not already created this directory, please refer to last week's handout to figure out how. Once the Week2 directory exists, **cd into it**
2. **copy `~carlm/213/Rstudio/Degas8_13_2010_12_1AM.rtf` into your new subdirectory of the week**. Again, you know where to look if you can't remember how to copy files.
3. **give Degas8_12_2010_12_1AM.rtf a shorter name**. You could of course do this with the GUI file manager application – but how about using one of the 12 most important Unix commands: **mv**.

   @:> `mv Deg` TAB `shortname.rtf`

   Whatever new convenient name you give the file, make sure that it still has the `.rtf` suffix to avoid problems later.

IN A BROWSER

1. **Browse to `rstudio.demog.berkeley.edu` to launch rstudio and create a new project in an EXISTING directory.** Project→New Project... Then be sure to select **Existing Directory**. Your goal is to create a new project in the directory that you just created for this week's assignment, the working directory of the new Rstudio **Project**. In Rstudio, a "project" is the container that keeps all of the various files for a particular purpose together. You don't need to use projects in order to use Rstudio or R. But for larger and longer lasting endeavors than the weekly assignments in 213, Rstudio **projects** will add a pleasing order to your life, for small stuff it doesn't matter so much.
2. **Gain access to Getting Started with Rstudio**. The library has an electronic copy at: `http://proquest.safaribooksonline.com/9781449314798`. If you are reading this on paper, you can go to the course website and look the link under this assignment.
3. Follow along with Chapters 2 : Case Study: Data Cleaning as it describes how to change the file suffix of the file that you copied into your Week2 directory earlier and how to read that into Rstudio's editor where it can be modified in order to later be read into R as data.
4. When you get to the subsection called "Data Cleaning"... Read the rest of this handout which attempts to disambiguate some of the R voodoo. Note that the file that you copied (and renamed) is the one that you will read in as part of this exercise in Chapter 2.

In the subsection of Chapter 2 titled "Data Cleaning" a considerable knowledge of R is assumed. You need not fully comprehend the example–since our purpose here is to learn to use Rstudio, nonetheless, here is the back story:

- **list objects**. You've seen lists in datacamp, but for more information see Chap 2 of Venebles and Smith and/or R In Action Section 2.2.6, a **list** is an object that holds a collection of other objects – which need not be of the same class or size. Recall that **Vectors** and **arrays**, can only hold items of the same mode (e.g. logical, numerical or character), With dataframes, each column can hold objects of a different mode, but the columns all must be of the same length. Lists, however, are more general. With lists, element 1 could be a vector of numbers, while list element 2 could be a dataframe and list element 3 could be character string.
  In the "Data Cleaning" example, a list (called "l") is needed because each naked mole rat (RFID) may have a different number of observed gate crossings. The result of the `split()` function is a **list** of **dataframes** with differing numbers of rows in each.
  That is each **element** of list **l** is a dataframe. The `lapply()` function is a very convenient tool for dealing with lists such as **l**.
  We will learn how to use `lapply()` later, for now you only need to know that `lapply(listX, function())` "applies" or executes the given function sequentially on each element of `listX` and returns the result as a list object with one element for each element in original `listX`. In the "Data Cleaning" example, `lapply()` does a few things to each element of list "l" (that is to each dataframe in list "l").
  The `lapply()` example in the "Data Cleaning" section is particularly difficult to follow, because it uses what is known as an **anonymous function**. Rather than using, for the second argument of `lapply()`, a function that has already been stored in an object, the example includes the code that would be used to create a function object instead.
  for example, rather than writing:

  - `> lapply(L,mean)`

    which would take the mean of each element of list L, (Recall that `mean()` is a function; you can type `mean(x)` to get the mean of a numeric vector called `x`). The anonymous function version would look like this:

  - `> lapply(L,function(x){return(sum(x)/length(x))})`

    The expression: " `function(x){return(sum(x)/length(x))}`" is an expression for creating a function that takes the mean of an object of mode `numeric`. Thus by executing:

  - `> Mean<-function(x)return(sum(x)/length(x))`

    (Note the capitalization Mean is not the same as mean.)
    AND

  - `> lapply(L,Mean)`

    one could achieve the same result as executing this:

  - `> lapply(L,function(x){return(sum(x)/length(x))})`

5. **Naming a variable lower case L, as is done in the "Data Cleaning" section is an act of unconscionable hostility toward your biographer.** Letters are very cheap, and lower case L looks way too much like the number one.

When you are finished working through the "case study" in Chapter 2 (having already read Chapter 3), you are finished for this week. An email indicating what you found particularly awful about this exercise would be appreciated.