Reproducible Science with R (Day 1)

M.K. Lau

Class Time: 4.5 hours

Goals: Use R and other tools to conduct reproducible research.

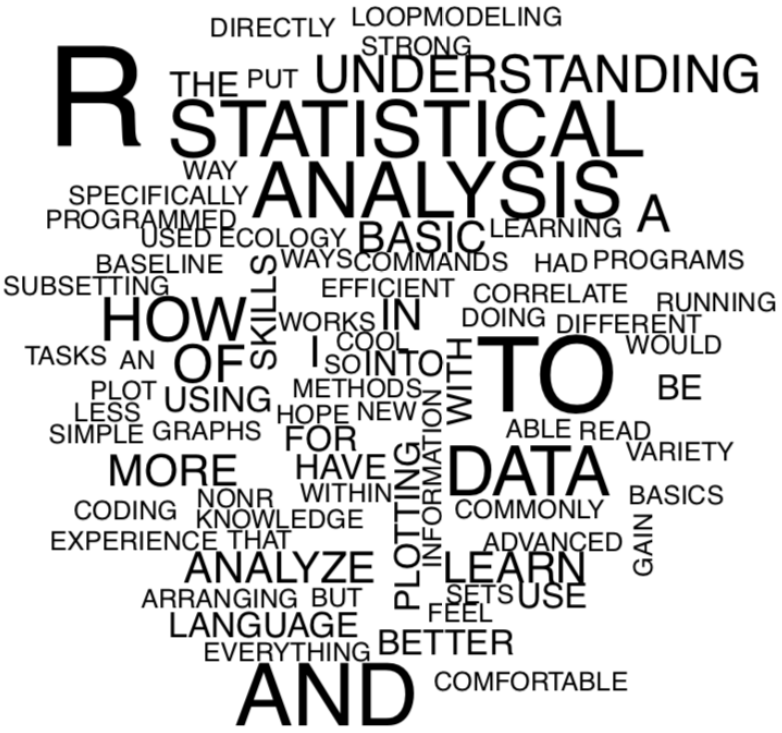

Figure 1: Wordcloud pre-class comments

This is a diagram exploring the comments made by workshop participants regarding what they hope to gain from the course. A few things that stood out in looking this over:

- People wanted to learn **R** (Kaboom, my mind explodes!)
- A lot of interest in statistical analysis. Although we won't have time to actually cover staistical analyses, this class will enable folks to learn and implement new stats methods on their own.
- Everyone had something that they wanted to learn how to do. Although there will be a wide range of abilities in this class, everyone can look at this as an opportunity to learn something.

Overview

- Day 1: Motiviation and framework for scientific reproducibility and the basics of using **R** in *RStudio*
- Day 2: Scientific scripts and best practics for code and projects
- Day 3: Tools for keeping track of versions and code dependencies

Being a good code community member (5 min)

We're about to start an activity, and it should be collaborative. If you don't understand something, try to figure it out and then ask the internet or someone else participating in the workshop (like me!). To make sure that we foster an open, productive and safe community everyone participating in this course will follow the Code of Conduct outline by the folks at ROpenSci (`https://ropensci.org/coc/`).

Similar to science in general software development is empowered by community. Please contribute to this by, in particular:

- *We all get out of this class what we put into it.*
- *Be considerate and respectiful of each other in speech and actions.*
- *Contribute a safe and effective learning experience for everyone.*

Also, please take a look at the Ada Initiative webpage for more information about creating an inclusive coding community: `https://adainitiative.org/`

Day 1

Overview: Why make your work reproducible and why R? (15 min)

Science is driven by the exchange of information and knowledge. Currently, there is a lot that we can do to make research more transparent and useful. A recent study (Stodden *et al.* 2018) demonstrated that only 26% of studies published in *Science* could be reproduced. This was even more striking given that the study was conducted after the *Science* had instituted it's open data policy.

What this and other studies point to is the need to provide well documented data as well as the software that were needed to conduct the study. Luckily, advances in open-source computer languages, such as **R** and **python**, provide a way to produce computations that can more easily document scientific research in a transparent, easily shared way.

In this course, we will cover how to conduct **reproducible** scientific research using the **R** programming language and supporting software that will enable researchers to more clearly and easily docu-

ment projects. Participants will gain experience coding in **R** using the *RStudio* IDE and the *git* version control system.

Possible additional topics, if time and interest allows:

- *RMarkdown*
- *dplyr*
- *ggplot*
- *github*
- *Shiny Apps*
- *R packages*
- *Continuous Integration (e.g. Travis)*
- *Posting data to web archives (e.g. Figshare)*
- *Open Licensing*
- *Code performance with profVis*
- *Data Provenance in R*
- *Code cleaning with Rclean*

Reproducibility Framework (10 min)

Max(reproducibility) = Data * Software * Documentation

1. Setup your project so that there is a clear architecture (*RStudio*)
2. Work so that your computation from initial data to finished results will be coded (wherever possible), including data cleaning and processing steps (**R**)
3. Keep track of versions of your code (*git*)
4. Make initial data available (whenever possible, *git*)
5. Keep track of software dependencies (*packrat*, *git*)
6. Be organized, succinct in style, coding and documentation (**R**, *Markdown*, *formatR*)

If you're interested in more details on how to conduct reproducible research, see the **TEE** (Transparency in Ecology and Evolution) website `http://www.ecoevotransparency.org/` and the **FAIR** reproducible research guide at `https://www.go-fair.org/fair-principles/`.

RStudio Tour (10 min)

To use RStudio, you will need to download and install both of the following:

1. **R** `https://cran.r-project.org/`
2. *RStudio* `https://www.rstudio.com/products/rstudio/download/#download/`

I highly recommend referring to RStudio's *Cheat Sheets*, especially for using RStudio and basics of R.

- `https://www.rstudio.com/resources/cheatsheets/`

R and RStudio

R is a statistical programming language. It is just text that follows specific rules that a computer can interpret as "things" that it is supposed to do.

RStudio provides an interface to R and other tools.

Here are some of the parts of *RStudio*:

Console

The Console is a window that can be used to "speak" R directly to the computer. It has a little ">" symbol that indicates where you can enter code (where the cursor is) and the code that you have already run (above where the cursor is). This is a handy place to conduct tasks that you don't want to keep for later usage.

Script(s)

Scripts are text files whose syntax follows the specifications of a programming language. Usually scripts are code that is aimed at doing a bunch of tasks, like analyze some data and produce output.

The "Workbench"

A set of windows (usually to the right of the Console and Scripts) with useful tools from *RStudio*. There are many possible tabs that could occur here. These are some of the possbiliites:

- **Environment** shows you what "data" you currently have loaded into **R**'s memory
- **History** shows you what you've done with **R**
- **Files** is a tab that will show you the files in your system
- **Plot** shows plots that you have generated
- **Help** will allow you to search for topics specific to the **R** language
- **Packages** we'll get to this later, but this is a way to add-on to the functionality of **R**
- **Viewer** similar to the *Plot* tab, this will show you a data table

Activity: Example Project (15 min)

Download, unzip and open up the example project in RStudio, check your working directory and run the *basics.R* script.

`https://github.com/HarvardForest/repro_example`

1. Click the "Clone or download" button and choose "Download ZIP"
2. Un-zip the repository using your system's unzip tool (often this can be accessed by right-clicking on the Zip)
3. Open R-Studio
4. From the "File" menu, choose "Open Project" and navigate to the unzipped project and click "Open"
5. Run the script
6. Make sure that the working directory is set to the project directory (search the *RStudio* help system for "Set Working Directory")

What is the Working Directory?

The *working directory* is a specific directory location in your file system that you can set in **R**'s memory. Having this makes life a lot easier in many ways, but the main one is that it allows **R** to make assumptions about the files that you're referring to in your code.

It's time to take a break! (10 min)

Anatomy of an R Script (10 min)

Let's look a little more closely at basics.R

```
# Basic R script

# MK Lau

# 15 April 2018

dat <- read.csv(file = "./data/data.csv")

cor.test(dat[, "x"], dat[, "y"])

plot(y ~ x, data = dat)
```

What do you notice about it?

Here are some general things that this script represents for scripts in **R** generally:

1. It is ordered such that the code at the top should be run before the code lower in the script
2. Code that is to the right of hashes ("#") is not executed

3. There are commands (aka. functions) that are followed by an open parenthesis, then some more words and then a close parenthesis
4. Typically data analysis workflow consists of read-in, wrangle and then analyze data
5. **R** is similar to English, so if you want to do something (like plot some data) it is often similar to what you think it might be

Function Anatomy

1. Function have names, like "plot", "mean", "help", etc.
2. Information (aka. "arguments") are given to objects inside of parentheses
3. Arguments are defined using equal signs ("=") and separated by commas (",")

Note that arguments **usually** exist only inside of functions, so you can't access the information inside of functions unless it's specifically passed out of that function.

Also, functions can have many arguments. Functions have to define default settings and orderings for arguments. Because of this, you don't always have to tell a function the value of every argument and you don't always have to define an argument by name with an equal sign.

Objects and Assignments

Objects are a used to tell **R** that some data should be kept in its working memory that can be called by that objects name, as in "dat" in the above example.

Assignments are used to create objects with data. This is done in the above example with the "<-" (aka. left arrow). There are lots of other ways to conduct assignments, including "=", but the left arrow is generally the best to use for code clarity.

Getting help with R

Now that you have a basic understanding of functions and arguments, you can expand on your **R** vocab by trial and error and getting help. Here are some useful ways to get help:

- *?*, *??*, *help*
- Google ("How do I _____ in R?")
- Community (This is not a a test, this is life)

Review and Q/A (10 min)

1. Importance of reproducibility

2. Max(reproduciblity) = data * software * docs
3. RStudio
4. R language (Arguments, Functions, Scripts)
5. Getting help