Reproducible Science with R

M.K. Lau

Class Time: 4.5 hours

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

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 document 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*)

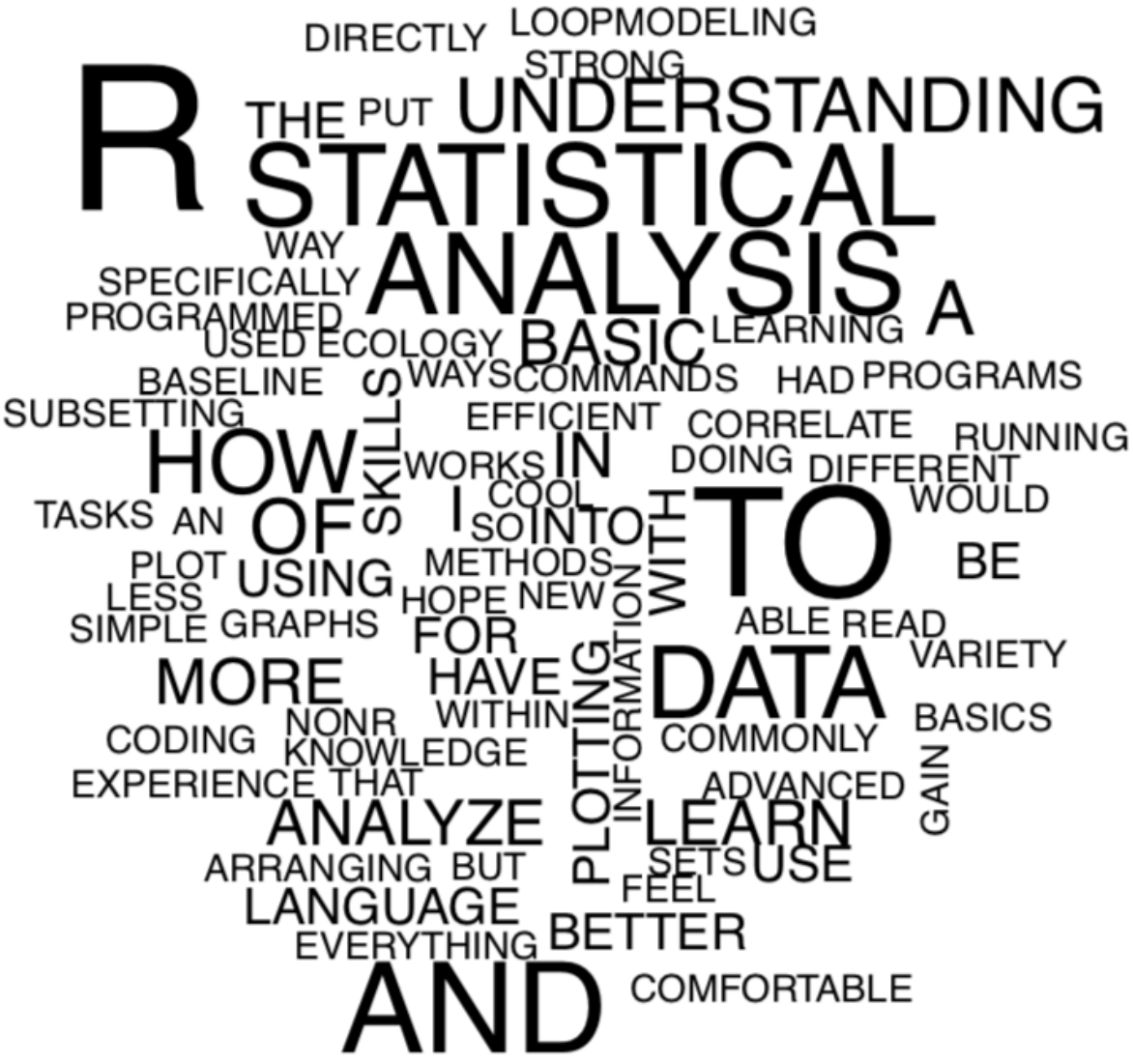

Figure 1: Wordcloud pre-class comments

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)

Download and install:

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

Cheat Sheets

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

https://www.rstudio.com/wp-content/uploads/2016/10/r-cheat-sheet-3.pdf

R is a language, *RStudio* is an interface to R and other tools.

Parts of RStudio

- Console: where R code can be entered directly
- Script(s): R code is text that is understood by an interpreter
- The "Workbench": a collection of tools
- File system
- Help (with RStudio and R)
- Working directory

Being a good code community member (5 min)

Like science, open-source software development is empowered by community. Everyone participating in this course will follow the Code of Conduct outline by the folks at ROpenSci (https://ropensci.org/coc/).

- *We all get out of this class what we put into it.*
- *Be considerate and respectful 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/

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/archive/master.zip

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

Anatomy of an R Script (10 min)

- Typical flow: read-in, wrangle and then analyze
- comments
- function anatomy
- objects

- assignments
- basic plotting

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)
```

Getting help with R

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

Activity: Write your own plot code (10 min)

1. Open up a new script window
2. Write a header
3. Comment out your header
4. Save your initial script
5. Copy the necessary code from the example script
6. Make a different plot

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. Your own script
6. Plotting

Day 2

Project Architecture (10 min)

Max(reproducibility) = Data * Software * Documentation

In this section, we'll go over some project best practices that generally work for computational projects. We'll initiate a new project from *RStudio* and set up a file system with:

- **README**: describes the project and associated files

- **data**: folder to collect relevant data files or links
- **src**: where all of the **R** scripts should be kept
- **results**: this is where output from scripts can go
- **docs**: further documentation and relevant files (e.g. notes, papers)
- **bin**: ADVANCED - if you need to include other associated software

ACTIVITY: Ecological data project (10 min)

Setup a new project and outline a script that will:

1. Import data
2. Do some data management
3. Do some analyses
4. Make a plot

Data wrangling (15 min)

File I/O = read.* and write.*

Working directories and relative paths = getwd, setwd, "./../../"

Objects and assignments = x <- 10 (=, ->, assign)

Mode = Scalars, vectors, matrices, arrays, lists, data.frames

Class = numeric, character, factor, logical, etc.

Indexing = x[1], x[1,1], x[[1]],

Sorting = sort(x), order(x)

Activity: Wrangle those data! (10 min)

Yo, Matt, it's time to take a break! (10 min)

Style: Best Practices and Intro to Packages (5 min)

Hadley Whickam's guide to style http://adv-r.had.co.nz/Style.html

Re-format code in *RStudio* with:

1. Code -> Reformat Code

Data testing (10 min)

Conditional operations Conditional statements Logical type

```
if (!(is.numeric(x[,1]))){warning("Danger Will Robinson!")}else{}
```

Activity: Write your own test! (10 min)

Review and Q/A (10 min)

1. Project organization

2. Ecological data
3. Data testing

Further Study:

- *dplyr* = reorganize data
- *stringr* manipulate text (aka. "strings")
- *lubridate* for handling dates and times
- *sp* and *raster* for spatial datasets
- *purrr* "apply" functions = functions for repeating functions

Day 3

"Backing Up" Version Control (10 min)

Max(reproducibility) = Data * Software * Documentation

In this module, you'll learn how to use the version control system, known as git, from RStudio. We will cover the basic topics of how to:

1. Initiate a project "repository"
2. Create, "add" and "commit" changes
3. View "diffs" and share a compressed project

We will not cover how to use the online git repository known as *github*. Using *github* provides a central server through which projects can be shared among collaborators in real-time via a system that keeps users from stepping on each other's "digital toes". To setup a free, private repository through *github.com*, go to their educational program webpage: https://education.github.com

Activity: Setup your ecological data project as a repository (10 min)

1. Using RStudio's menu system, turn on git support
2. Make some changes to your script
3. Add them to the "stage"
4. Review the changes
5. Commit them

Commit best practices (5 min)

1. Commits should represent a task that has progressed or completed
2. Files with changes relevant to those tasks should be staged
3. In general commit often and describe commits succinctly but informatively

Examples:

- "Some changes"
- "Added data import of plant data"

Branching (10 min)

- Separates commits

Activity: Commit! (5 min)

Activity: Wanna go back in time! (10 min)

Rolling back versions

1. View history
2. Choose the time point (here's where commit messages are key!)
3. Revert

Yo, Matt, it's time to take a break!

Software Dependencies: R Packages and CRAN *et al.* (5 min)

Functions and other information are grouped into packages and hosted on CRAN (Comprehensive **R** Archive Network). This modular structure allows **R** to grow but still be usable. As of 2016 there were over 8,000 packages on CRAN, an increase of 2,000 from 2015.

https://www.r-bloggers.com/on-the-growth-of-cran-packages/

With regard to reproducibility, this creates an issue as CRAN and **R** are constantly changing. That is why it is imperative to keep track of the "dependencies" of your code. Ways to deal with dependencies:

1. Minimize them
2. Track them (*sessionInfo*, *packrat*, containers/*rocker*, VM/*encapsulator*)

https://rviews.rstudio.com/2018/01/18/package-management-for-reproducible-r-code/

One easy way would be to just record the system and library information using *sessionInfo*:

```
print(sessionInfo(), locale = FALSE)
```

```
## R version 3.4.4 (2018-03-15)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.5
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_3.4.4  backports_1.1.0 magrittr_1.5    tufte_0.3
##  [5] rprojroot_1.2   formatR_1.5     tools_3.4.4     htmltools_0.3.6
##  [9] yaml_2.1.17     tinytex_0.5     Rcpp_0.12.12    stringi_1.1.6
## [13] rmarkdown_1.9   knitr_1.20      stringr_1.3.0   digest_0.6.15
## [17] evaluate_0.10.1
```

A good place for this information is in your README.

packrat (10 min)

Although there are other more sophisticated ways to implement dependency reproducibility, we'll go over *packrat*, which is implemented in *RStudio*:

1. Tools -> Project Options
2. Packrat -> Use packrat for this project
3. Use default settings and continue

More info on *packrat* can be found here: https://rstudio.github.io/packrat/rstudio.html

One handy thing that *packrat* enables is that it can detect what packages your project isn't using and remove them. Use the following bit of code to do this:

```
packrat::clean()
```

ProTip: this syntax uses "::" (i.e. a double colon) allowing you to specify the package ("packrat") that contains the function ("clean") that you want to use. This is extremely useful as it means that you don't always have to load (i.e. *library*) the entire package when you just want to use one or a few functions.

Wrap-up on Reproducibility with R (5 min)

1. Max(reproducible) = data * software * docs
2. Project architecture for better access
3. Document (comment, follow style, "Brevity is the soul of wit.")
4. Share your results
5. Planned obsolescence: Backup and version

Possible: Advanced Topics Courses (5 min)

1-hour sessions

Zoom + Slack

Museum or remote locations

- Scientific notebooks with *rmarkdown* or *jupyter*
- Advanced command line BASH
- Advanced git and github
- Web apps with *shiny*
- Advanced plotting with *ggplot*
- Community analysis with *vegan* and *ecodist*
- Matrix operations and linear algebra **%*%**
- Efficiency with *profVis*
- Network analysis with *enaR*
- Mapping with *googlevis*
- Spatial data analysis with *sp* and *raster*
- Automated workflow tracking with data provenance *RDatatracker*
- Code cleaning with *Rclean*

R Markdown Notebooks

Markdown is a language use for "typesetting", like the HTML that makes up webpages. There is also a specific syntax that can be used to combin Markdown with the R language. This allows the creation of documents that combine text, code and output. This is an extremely useful tool for the scientific reproduciblity kit. One particular aspect that is extremely powerful is the ability to automatically create many different "rendered" documents from a single document, such as webpages, PDFs, Shiny Apps and many more formats.

For more information, the following is a useful introduction to R Markdown notebooks.

https://rmarkdown.rstudio.com/r_notebooks.html

Survey (5-10 min)