Class 2: Research Reproducibility I

Kim Johnson

January 25, 2018

Lecture Outline

- Housekeeping (Concept proposal and peer review for HW3)
- Research Reproducibility
- Promising coding practices
- Literate programming
- Using R markdown for writing reproducible code

Learning Objectives

1. Be able to distinguish between replication, reproducibility, and repeatability
2. Know why research reproducibility is critically important
3. Be able to understand and implement promising practices for reproducible code
4. Be able to use R markdown with some basic options and generate a project analysis html file

The reproducibility crisis in Science

The reproducibility crisis in science

A statistical counterattack

More people have more access to data than ever before. But a comparative lack of analytical skills has resulted in scientific findings that are neither replicable nor reproducible. It is time to invest in statistics education, says **Roger Peng**

Reproducibility crisis

The reproducibility spectrum (From Peng reading)

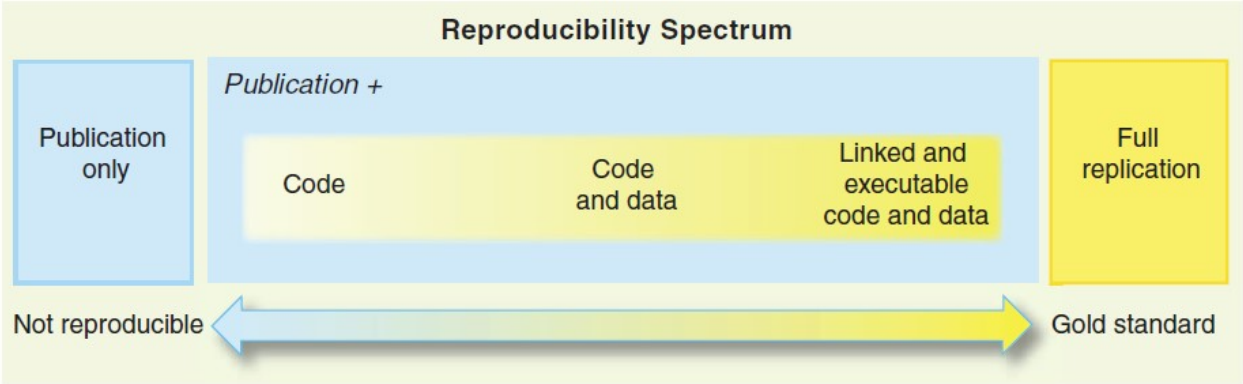


Fig. 1. The spectrum of reproducibility.

Reproducibility spectrum

What is reproducibility? (with credit to Jenine Harris)

same data + same code = same results

- This study is **repeatable**

same data + same or new code = same results

- This study is **reproducible**

new data + new code = same results

- This study is **replicable**

Some evidence for a replication/reproducibility problem (with credit to Jenine Harris)

- 21% of 67 drug studies were reproducible (Prinz et al., 2011)
- 40-60% of psychology studies were reproducible (Open Science Collaboration, 2015)
- 61% of economics studies were reproducible (Camerer et al., 2016)
- 6% of p-values incorrectly reported in psychology papers (Nuitjen et al., 2004)
- 11% of p-values incorrectly reported in medical papers (Garcia-Berthou et al., 2004)

Making research reproducible components

1. ***All*** methods are fully and clearly reported
2. All data and code files used for the analysis are available.
3. The process of analyzing raw data is well-reported and preserved.

From: https://www.r-bloggers.com/what-is-reproducible-research/

There are barriers…

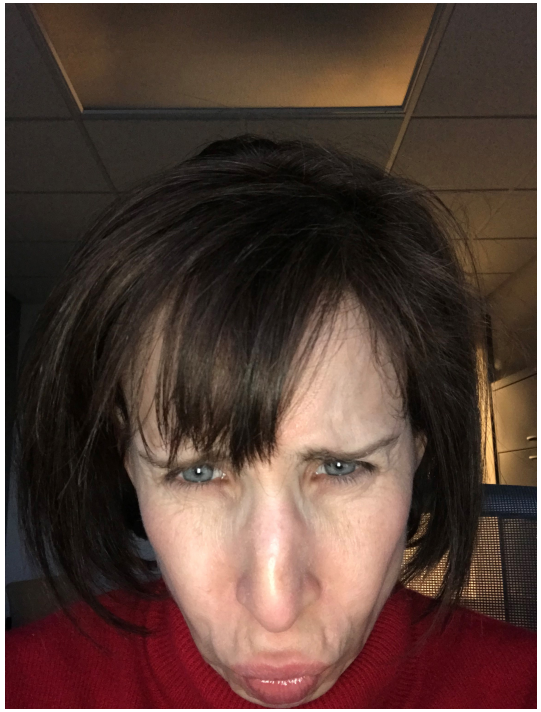
- Training
- Cost and privacy
- Time

Making research reproducible (statistical code and documentation)

- Files can proliferate like cancer and before you know it is out of control and you have no idea who did what when…even if the *who* is *you* sometime in the future
- **FINAL** is never **FINAL**
- Statistical code can be messy and hard to interpret with random curiousity code contaminating code files
- How do we make code more interpretable?

This is a personal problem!!

- I couldn't find the right files quickly for a student to get started on a project just last week!



Do not pass around

Number of code files per analysis project

- I am going to say 2 max (one for data managment and one for analysis)
- Think about this carefully at the start of a project

Formatting and organizing code

The next section covers 9 practices that apply regardless of the statistical language you are using. These practices fall into several categories:

- Introduce and explain the code
    1. *Write a prolog to introduce the code*
    2. *Annotate to clarify code purpose*
- Choose meaningful names with consistent formatting
    3. *Use meaningful names for objects*
    4. *Use verbs for functions*
    5. *Use camelCase or under_line for multi-part names*
    6. *Add meta-data to file names*
- Use space wisely
    7. *Use white space to separate processes*
    8. *Limit line length to 80 characters*
    9. *Indent to group lines of code that belong together*

Introduce and explain the code:

Promising Practice #1: Write a prolog to introduce the code

- **Prolog**- a block of comments at the beginning of a code file offset with special characters as appropriate in the software being used.

A *prolog* can be used to *provide sufficient information about code* for a collaborator (maybe future you!) or someone outside your project to understand the *who*, *what*, and *why* of your code.

For example, here is a prolog formatted for a statistical code file in SAS:

```
/* PROLOG   ********************************************************************

   PROJECT: NAME OF PROJECT HERE
   PURPOSE: MAJOR POINT(S) OF WHAT I AM DOING WITH THE DATA HERE
   DIR:     list directory(-ies) for files here
   DATA:    list dataset file names/availability here, e.g.,
            filename.correctextention
            somewebaddress.com
   AUTHOR:  AUTHOR NAME(S)
   CREATED: MONTH dd, YEAR
   LATEST:  MONTH dd, YEAR
   NOTES:   indent all additional lines under each heading,
            & begin the prolog with a forward slash followed by an asterisk,
            end with an asterisk followed by a forward slash.
            KEEP PURPOSE, AUTHOR, CREATED & LATEST ENTRIES IN UPPER CASE,
            with appropriate case for DIR & DATA, lower case for notes
            If multiple lines become too much,
            simplify and write code book and readme.
            HINT #1: Decide what a long prolog is.
            HINT #2: copy & paste this into new script & replace text.

   PROLOG   ******************************************************************** */
```

Here is a prolog formatted for a code file in R:

```
# PROLOG   ********************************************************************'

# PROJECT: NAME OF PROJECT HERE
# PURPOSE: MAJOR POINT(S) OF WHAT I AM DOING WITH THE DATA HERE
# DIR:     list directory(-ies) for files here
# DATA:    list dataset file names/availability here, e.g.,
```

```
#          filename.correctextention
#          somewebaddress.com
# AUTHOR:  AUTHOR NAME(S)
# CREATED: MONTH dd, YEAR
# LATEST:  MONTH dd, YEAR
# NOTES:   indent all additional lines under each heading,
#          & use the apostrophe hashmark bookends that appear
#          KEEP PURPOSE, AUTHOR, CREATED & LATEST ENTRIES IN UPPER CASE,
#          with appropriate case for DIR & DATA, lower case for notes
#          If multiple lines become too much,
#          simplify and write code book and readme.
#          HINT #1: Decide what a long prolog is.
#          HINT #2: copy & paste this into new script & replace text.

# PROLOG   ########################################################
```

Prolog templates for SAS, SPSS, Stata, and R are available on GitHub at https://github.com/coding2share/Prolog-templates

Introduce and explain the code:

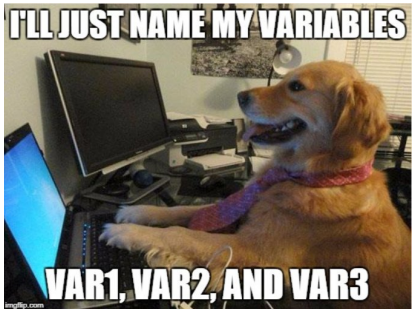
Promising Practice #2: Annotate to clarify code purpose

- A prolog contains general information about the code
- Annotation in contrast is used throughout the code to describe what is going on.
- There are different schools of thought on how much or how little annotation is needed.
- The general goal would be to write *clear* enough code that a small amount of annotation would be enough and then add annotation as needed to clarify steps in the analyses.

Choose meaningful names with consistent formatting:

Promising Practice #3: Use meaningful names for objects

- It is common in research for variables to be named something like *var1* or *vm1q26b* during data collection.
- Names may be generated automatically from survey software or from using a numbered list in a word processing program.
- Following data collection, *consistent and meaningful* names for variables, functions, and files should be assigned.
- Important for code that is easy to read and use by you and others



Choose meaningful names with consistent formatting:

Promising Practice #4: Use verbs for functions

- A verb is the part of speech that describes an action. Using parts of speech in code can make your code more intuitive.
- Use *verbs* for *function names* works well because functions typically do some action.
- Use *nouns* and adjectives for variables (e.g. race, marital status, age, or sex)
- When future you or another researcher uses your code it will be easier to discern variables from functions.

What are *Meaningful names* for variables and functions?

- They will describe what information is stored in the variable and what the functions does
- *BMI* vs. *var1* is much more descriptive of the nature of the variable
- A function that multiplies *age* by *packs smoked per day* to find pack years for smokers might be called *find_pack_years* rather than *funcpy* or even just *f*.

Choose meaningful names with consistent formatting:

Promising Practice #5: Use camelCase or under_line for multi-part names

- Multiword variable, function, or file names are more easily read by humans when they are formatted using *upper CamelCase*, *lower camelCase*, or *under_lines* to separate words.
- The most useful may be *under_lines* since they are more *machine readable* than camel case. Example: *Last_men_period* for "last menstrual period"
- However, empirical work does suggest that camel case leads to great accuracy and speed… http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.158.9499
- It is really a matter of personal preference

Choose meaningful names with consistent formatting:

Practice #6: Add meta-data to file names

Including **consistent** meta-data like the *date* and *project name* in file names makes it easier to search for files and process files automatically. There are three key principles for filenames:

1. Machine readable
2. Human readable
3. Works with default ordering

- *Machine readable* file names are formatted in a way that can be read into a program and interpreted by the program.
- This is especially useful if a project includes many data or code files that are used together.
- Machine readable file names typically include separators, usually an underline, to separate parts of the file name. For example, a CSV data file collected on January 23 of 2017 might be saved as *01232017_projectName.csv*. Note that this file name includes a 0 at the beginning rather than a 1 for January. This is important due to *default ordering* of numbers and letters. If January were coded as simply **1**, the file name would be 12317 and default ordering would interpret it as coming after data files from October or November, months 10 and 11. *Human readable* file names include dates and words formatted in ways that are familiar to people.

Use space wisely:

Promising Practice #7: Use white space to separate processes

- Code that is written without blank lines between procedures can run together and be difficult to interpret.
- Add an extra line of space before new procedures to make the code easier to follow.

Use space wisely:

Promising Practice #8: Limit line length to 80 characters

- Long lines of code are difficult or impossible to see on some computer screens, especially for people using laptop computers.
- Instead of continuing code on a single line, add hard returns so that code does not go beyond about 80 characters.

Use space wisely

Promising Practice #9: Indent to group lines of code that belong together

- When a function or procedure is very long, it can take multiple lines of code.
- To signal that several lines of code go together, indent each line after the first one.

An example

This R code does not follow the promising practices above and is arguably difficult to read:

```
library(RNHANES); library(ggplot2)
dat <- nhanes_load_data("AUQ_G", "2011-2012", demographics = TRUE)
summary(dat$AUQ300)
dat$AUQ300[dat$AUQ300 > 2] <- NA
dat$AUQ300 <- factor(dat$AUQ300,levels=c(1,2),labels=c("Yes","No"))
summary(dat$AUQ300)
ggplot(subset(dat, !is.na(AUQ300)),aes(x=AUQ300, y=(..count..)/sum(..count..),fill=AUQ300)) + geom_bar() + theme_minimal() + scale_y_continuous(labels=scales::percent) +  scale_fill_manual(values=c("
```

First,

Use space wisely. Adding *white space to separate processes*, *limiting line length to 80 characters*, and *indenting to group lines of code that belong together* can help make the code easier to follow:

```
library(RNHANES)
library(ggplot2)

dat <- nhanes_load_data("AUQ_G", "2011-2012",
                        demographics = TRUE)
summary(dat$AUQ300)

dat$AUQ300[dat$AUQ300 > 2] <- NA
dat$AUQ300 <- factor(dat$AUQ300,
                     levels=c(1,2),
                     labels=c("Yes","No"))
summary(dat$AUQ300)

ggplot(subset(dat, !is.na(AUQ300)),
       aes(x=AUQ300,
           y=(..count..)/sum(..count..),
           fill=AUQ300)) +
  geom_bar() +
  theme_minimal() +
  scale_y_continuous(labels=scales::percent) +
  scale_fill_manual(values=c("orange","gray"),
                    guide=FALSE)
```

Second,

Choose meaningful names for objects. Since the code examines the National Health and Nutrition (NHANES) survey data, naming the data object *nhanes* makes sense. Likewise, the variable AUQ300 measures whether the participant has ever used a gun, so try naming it *gunUser*:

```
#install.packages("RNHANES")
library(RNHANES)
library(ggplot2)

nhanes <- nhanes_load_data("AUQ_G", "2011-2012",
                        demographics = TRUE)
summary(nhanes$AUQ300)

nhanes$AUQ300[nhanes$AUQ300 > 2] <- NA
nhanes$gunUser <- factor(nhanes$AUQ300,
                      levels=c(1,2),
                      labels=c("Yes","No"))
summary(nhanes$gunUser)

ggplot(subset(nhanes, !is.na(gunUser)),
       aes(x=gunUser,
           y=(..count..)/sum(..count..),
           fill=gunUse)) +
  geom_bar() +
  theme_minimal() +
  scale_y_continuous(labels=scales::percent) +
  scale_fill_manual(values=c("orange","gray"),
                    guide=FALSE)
```

Finally,

Introduce and explain the code by adding a prolog and annotation. Compare this final version of the code with the first version above:

```
#################################################################
# PROJECT: Gun use policy brief
# PURPOSE: Bar graph of percentage of gun use for policy brief
# DIR:     C:/Users/jenine/Desktop/datacamp
# DATA:    NHANES 2011-2012 data available via RNHANES package
# AUTHOR:  Jenine Harris
# CREATED: 11/28/17
# LATEST:  11/28/17
# NOTES:   For coding2share formatting code module
#################################################################

#open NHANES package to bring in data
#open ggplot2 for graphing
library(RNHANES)
library(ggplot2)

#bring in NHANES 2011-12 audiology data
#AUQ300 question asks ever used gun
#where 1 = Yes, 2 = No, 7 = Refused, 9 = Don't know
nhanes <- nhanes_load_data("AUQ_G", "2011-2012",
                           demographics = TRUE)
summary(nhanes$AUQ300)

#delete Refused and Don't know responses
#rename variable to gunUser, add labels to levels
nhanes$AUQ300[nhanes$AUQ300 > 2] <- NA
nhanes$gunUser <- factor(nhanes$AUQ300,
                         levels=c(1,2),
                         labels=c("Yes","No"))
summary(nhanes$gunUser)

#plot bar graph of percent of 2011-2012 NHANES
#participants who ever used gun
ggplot(subset(nhanes,
              !is.na(gunUser)),
       aes(x=gunUser,
           y=(..count..)/sum(..count..),
           fill=gunUser)) +
  geom_bar() +
  theme_minimal() +
  scale_y_continuous(labels=scales::percent) +
  scale_fill_manual(values=c("orange","gray"),
                    guide=FALSE)
```

Literate programming

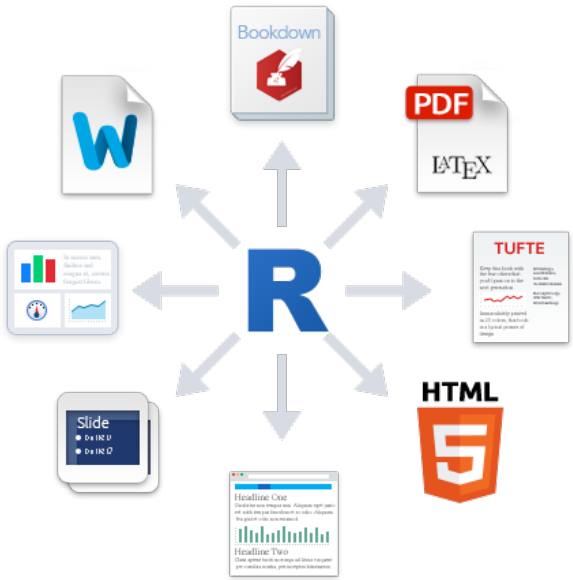
- First introduced by Donald Knuth of Stanford University
- It is a program where the English language is interspersed with code to describe what each piece of the program does
- In his seminal paper in 1984 Dr. Knuth says “I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be **works of literature**. Hence, my title”Literate Programming."
- It is code embedded within documentation rather than documentation embedded within code

R-markdown with literate programming

[Scary Video][https://youtu.be/s3JldKoA0zw]

R Markdown (RMD)

- R Markdown is a file format in R (ending in .RMD) for making literate programming documents in R written in markdown (human readable plain text)
- Code chunks are embedded within plain text that describes what the code does
- Can quickly generate (aka "knit") data analysis reports in several file formats including PDF, Word, HTML



YAML Header

- What the heck is a YAML?
- According to Wikipedia (my personal favorite source), it stands for *YAML Aint Markup Language*
- From Wikipedia (again): *"It is a human-readable data serialization language."*
- It is used to specify output document types as well as parameters from RMD
- It is at the top of your markdown file and looks like this:

```yaml
---
title: "Untitled"
author: "Kim Johnson"
date: "January 14, 2018"
output: html_document
---
```

YAML

Activity and HW3

1. Open R studio and let's explore R Markdown
2. HW3