Reproducible research 1

R Style

Why is style important?

R style guidelines provide rules for how to format code in an R script.

Some people develop their own style as they learn to code. However, it is easy to get in the habit of following style guidelines, and they offer some important advantages:

- Clean code is easier to read and interpret later.
- It's easier to catch and fix mistakes when code is clear.
- Others can more easily follow and adapt your code if it's clean.
- Some style guidelines will help prevent possible problems (e.g., avoiding `.` in function names).

Style guidelines

For this course, we will use R style guidelines from two sources:

- Hadley Wickham's R style guidelines
- Google's R style guidelines

Style guideline review

Hear are a few guidelines we've already covered in class:

- Use <-, not =, for assignment.
- Guidelines for naming objects:
    - All lowercase letters or numbers
    - Use underscore (_) to separate words, not camelCase or a dot (.) (this differs for Google and Wickham style guides)
    - Have some consistent names to use for "throw-away" objects (e.g., `df`, `ex`, `a`, `b`)
- Make names meaningful
    - Descriptive names for R scripts ("random_group_assignment.R")
    - Nouns for objects (`todays_groups` for an object with group assignments)
    - Verbs for functions (`make_groups` for the function to assign groups)

Line length

Google: **Keep lines to 80 characters or less**

To set your script pane to be limited to 80 characters, go to “RStudio” -> “Preferences” -> “Code” -> “Display”, and set “Margin Column” to 80.

```
# Do
my_df <- data.frame(n = 1:3,
                    letter = c("a", "b", "c"),
                    cap_letter = c("A", "B", "C"))

# Don't
my_df <- data.frame(n = 1:3, letter = c("a", "b", "c"), cap_lett
```

This guideline helps ensure that your code is formatted in a way that you can see all of the code without scrolling horizontally (left and right).

Spacing

- Binary operators (e.g., <-, +, -) should have a space on either side
- A comma should have a space after it, but not before.
- Colons should not have a space on either side.
- Put spaces before and after = when assigning parameter arguments

```r
# Do
shots_per_min <- worldcup$Shots / worldcup$Time
#Don't
shots_per_min<-worldcup$Shots/worldcup$Time

#Do
ave_time <- mean(worldcup[1:10, "Time"])
#Don't
ave_time<-mean(worldcup[1 : 10 ,"Time"])
```

Semicolons

Although you can use a semicolon to put two lines of code on the same line, you should avoid it.

```r
# Do
a <- 1:10
b <- 3

# Don't
a <- 1:10; b <- 3
```

Commenting

- For a comment on its own line, use #. Follow with a space, then the comment.
- You can put a short comment at the end of a line of R code. In this case, put two spaces after the end of the code, one #, and one more space before the comment.
- If it helps make it easier to read your code, separate sections using a comment character followed by many hyphens (e.g., #------------). Anything after the comment character is "muted".

```
# Read in health data ----------------------------

# Clean exposure data ----------------------------
```

Indentation

Google:

- Within function calls, line up new lines with first letter after opening parenthesis for parameters to function calls:

Example:

```r
# Relabel sex variable
nepali$sex <- factor(nepali$sex,
                     levels = c(1, 2),
                     labels = c("Male", "Female"))
```

Code grouping

- Group related pieces of code together.
- Separate blocks of code by empty spaces.

```r
# Load data
library(faraway)
data(nepali)

# Relabel sex variable
nepali$sex <- factor(nepali$sex,
                     levels = c(1, 2),
                     labels = c("Male", "Female"))
```

Note that this grouping often happens naturally when using tidyverse functions, since they encourage piping (%>% and +).

Broader guidelines

- Omit needless code.
- Don't repeat yourself.

We'll learn more about satisfying these guidelines when we talk about writing your own functions in the next part of the class.