Best Coding Practices

Lecture 01.2: Coding Tips

Lauren Sullivan

Module: Data Management, Visualization & Reproducibility

Readings

Required for class:

- Carey & Papin (2018) Ten Simple Rules for Biologists Learning to Program. *PLOS Computational Biology*.
- Schnell (2015) Ten Simple Rules for a Computational Biologist's Laboratory Notebook. *PLOS Computational Biology*.

Optional:

- R for Data Scientists - Online or Hard Copy
- R Markdown
- R Markdown Cheat Sheet

Thanks to Dr. Bobbi Carothers (Wash U) for ideas and code from an R-Ladies STL workshop on Reproducible Research

Why Care About Clean Code?

Ever inherit a project and have *no* idea how the data were managed and/or analyses performed?

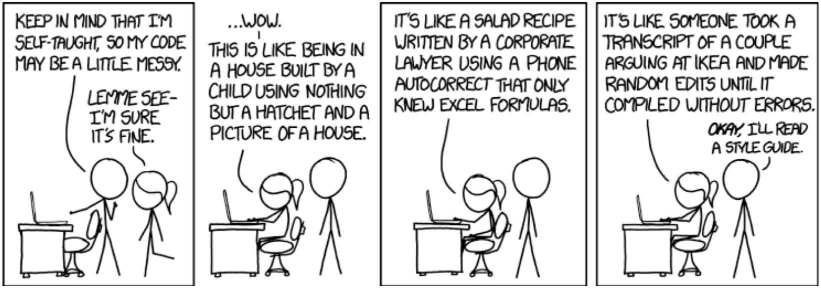
Ever come back to your own project after a few months and have no recollection of what you did or why you did it, even after looking at your own code?

In this lecture we'll be talking about ways to help others and future-you through:

- Literate Programming
- Clear Code Formatting

Clean Coding

There are lots of ways to code the same thing. Using clean code conventions means we can avoid this…


KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE– I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

In order to make sure your code is FAIR, you want to make sure it is clean, readable, and well annotated. So how do we do that?

Tip #1: Use White Space to Separate Processes

Bad:

```
tvtest <- lmer(sqrt(tv) ~ Region * log(DD5_TOT) + (1|Popula
r.squaredGLMM(tvtest) ggplot(tvdat, aes(x=log(DD5_TOT),y=sc
size=3)+geom_point(shape=1,size=3,colour='black')+geom_smoo
(aes(col=Region),method='lm')+xlab(bquote('log(Degree Days
```

Better:

```
tvtest <- lmer(sqrt(tv) ~ Region * log(DD5_TOT) + (1|Popula
summary(tvtest) r.squaredGLMM(tvtest)

ggplot(tvdat, aes(x=log(DD5_TOT), y=sqrt(tv),col=Region))+g
size=3)+geom_point(shape=1,size=3,colour='black')+geom_smoo
+xlab(bquote('log(Degree Days above' ~5^o~C*')'))+ylab('sq
```

Tip #2: Limit Line Length to Reduce Scrolling

```
tvtest <- lmer(sqrt(tv) ~ Region * log(DD5_TOT) +
(1|Population/Sample_ID) + (1|Year), data=tvdat,
REML=FALSE) summary(tvtest) r.squaredGLMM(tvtest)

ggplot(tvdat, aes(x=log(DD5_TOT), y=sqrt(tv),
col=Region))+geom_point(aes(col=Region),size=3)+
geom_point(shape=1,size=3,colour='black')+
geom_smooth(aes(col=Region),method='lm')
+xlab(bquote('log(Degree Days above' ~5^o~C*')'))+
ylab('sqrt(Terminal Velocity (m/s))')
```

Tip #3: Use Indentation to Group Code Together

```r
tvtest <- lmer(sqrt(tv) ~ Region * log(DD5_TOT) +
    (1|Population/Sample_ID) + (1|Year), data=tvdat,
    REML=FALSE)
summary(tvtest)
r.squaredGLMM(tvtest)

ggplot(tvdat, aes(x=log(DD5_TOT), y=sqrt(tv),
    col=Region))+
  geom_point(aes(col=Region),size=3)+
  geom_point(shape=1,size=3,colour='black')+
  geom_smooth(aes(col=Region),method='lm')+
  xlab(bquote('log(Degree Days above' ~5^o~C*')'))+
  ylab('sqrt(Terminal Velocity (m/s))')
```

Tip #4: Use Meaningful Names for Objects

Bad:

```
g$SM10_21[g$SM10_21==-99] &lt;- NA
prop.table(table(g$SM10_21))
```

Better: replace `g` with `GEUM_DATA` and `SM10_21` with `seed_mass`.

```
GEUM_DATA$seed_mass[GEUM_DATA$seed_mass==-99] &lt;- NA
prop.table(table(GEUM_DATA$seed_mass))
```

Tip #5: Consistent Formatting for Multi-Part Names

- The classics include: dot.case, camelCase, or snake_case
- Variations include lower camelCase, upper CamelCase, UPPER_SNAKE_CASE and so on
- Consider using one format for functions, another for dataframes, and a third for variable names

```
find.mode(GEUM_DATA$seed_mass)
```

Here, the function is dot case, the data frame is upper snake case, and the variable name is lower snake_case

{*Note that some formats might not work in certain software packages.*}

Tip #6: Add Metadata to File Names

- Include metadata (e.g. date and project name) in file names.
  1. **Machine readable** (no spaces in the file name)
  2. **Works with default ordering** (numbers refer to dates - yymmdd, use leading zeros where necessary. This is extremely helpful in code file names for versioning)
  3. **Human readable** (makes sense)

- Examples:
  - 17_02_03_raw_GeumTraitData.csv
  - 17_02_05_clean_GeumTraitData.csv

Tip #7: Write a Prolog to Introduce Code

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

# PROJECT: NAME OF PROJECT
# PURPOSE: MAJOR POINT(S) OF WHAT I DO WITH DATA HERE
# DIR:     list directory(-ies) for files here
# DATA:    list dataset file names here, e.g.,
#            filename.correctextention
#            somewebaddress.com
# AUTHOR:  AUTHOR NAME(S)
# CREATED: MONTH dd, YEAR
# LATEST:  MONTH dd, YEAR
# NOTES:   indent additional lines under each heading,
#            Use case system to keep things clear -
#            UPPER CASE FOR INFO, lower case for notes
#
#          Include changes made for this version

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

Tip #8: Annotate to Clarify Code Purpose

Use comments to:

- Explain the reason for the code (if needed)
- Explain functionality or choices that are not obvious
- Identify places to return to code to fix

Avoid using comments to:

- Explain poorly named objects

```
# Calculate avg wind speed and convert km/h into m/s
wind_avg <- ddply(wind, .(measurement), summarize,
     wind_average = (mean(value)*1000)/3600)
```

Tip #9: Clear Your Code History at the Start of Each File

Always be sure to include `rm(list=ls())` at the start of each code file, before you enter in libraries or datasets. This code clears the working memory so you don't accidentally have something named, for example, `test` in the system that you think is your current `test` but is really a file called `test` from a different analysis.

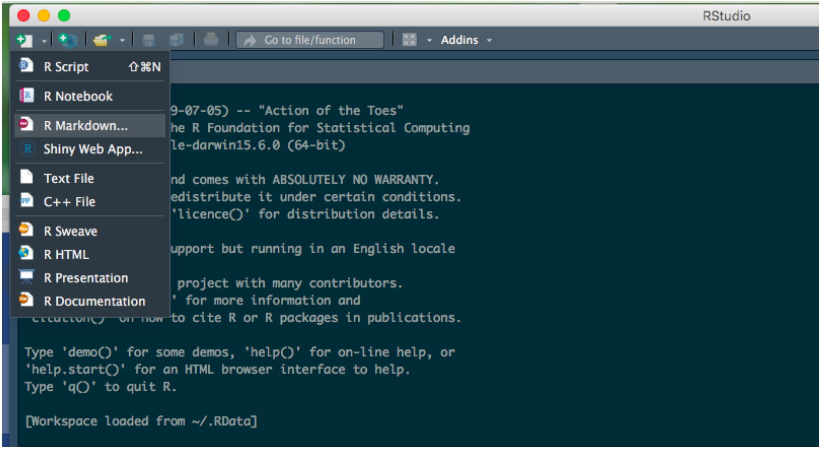
I can not emphasize this practice enough

R Markdown

- Create polished documents detailing data, analyses and figures, etc.
- Merge prose, code, analyses all in one document.
  - great for use as a metadata file. Cleanly describes your work for publications or data storage.
  - great when trying to be FAIR
- Final document is in a non-proprietary format (.pdf or .html) that can be shared publically (as opposed to simply providing people with R code)
- More attractive and more easily shared than syntax/code files

R Markdown

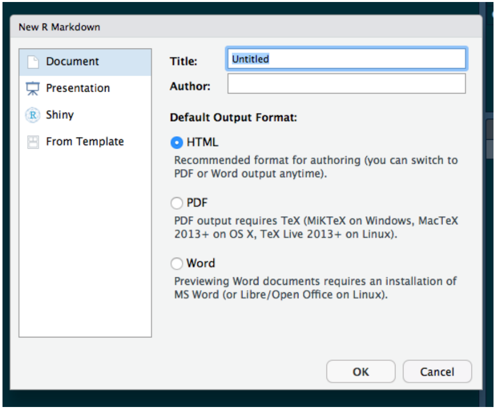
Open up RStudio, then open up a new R Markdown document.



```
9-07-05) -- "Action of the Toes"
he R Foundation for Statistical Computing
le-darwin15.6.0 (64-bit)

nd comes with ABSOLUTELY NO WARRANTY.
edistribute it under certain conditions.
'licence()' for distribution details.

upport but running in an English locale

 project with many contributors.
' for more information and
citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]
```

R Markdown

Then choose what type of document you want to create. You can see there are a lot of options. I created these slides in RMarkdown, and there is a new option to create powerpoint slides in Markdown that you can then edit in Powerpoint. You can even write your entire manuscript in Markdown if you want!



Title: Untitled

Author:

Default Output Format:

[✓ HTML]
Recommended format for authoring (you can switch to PDF or Word output anytime).

PDF
PDF output requires TeX (MiKTeX on Windows, MacTeX 2013+ on OS X, TeX Live 2013+ on Linux).

Word
Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux).

R Markdown Syntax

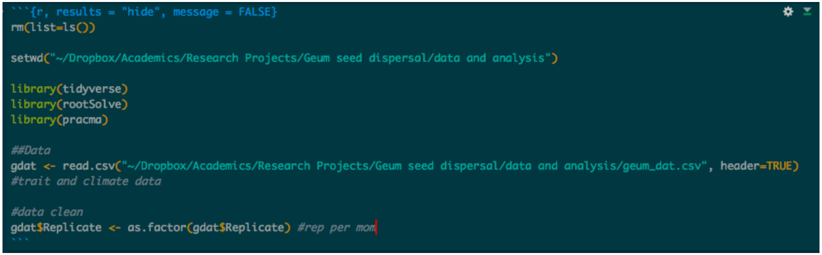
You need to create a title for each document using YAML (Yet Another Markdown Language). These differ depending on the type of document you want to create. You can use the internet to find options for personalizing your document. You'll notice the headers for my slides look like this:

```yaml
---
title: "Best Coding Practices"
subtitle: "Lecture 2: Coding Tips"
author:  "Lauren Sullivan"
date: "Module: Data Management, Visualization
      & Reproducibility"
output: beamer_presentation
mainfont: Calibri
---
```

Note: I put things under the incorrect type so I could get the titles and subtitles I wanted.

Code Chunks

The code that you want R to run goes in “chunks” that are wrapped in backticks with **`{r}`** at the end of the top wrapper. You can set options for how you want R to handle each chunk (e.g. do you want to display code, results, warning messages, etc). Check out the R Markdown Cheat Sheet for some of the many options for your code chunks.



~~~
```{r, results = "hide", message = FALSE}
rm(list=ls())

setwd("~/Dropbox/Academics/Research Projects/Geum seed dispersal/data and analysis")

library(tidyverse)
library(rootSolve)
library(pracma)

##Data
gdat <- read.csv("~/Dropbox/Academics/Research Projects/Geum seed dispersal/data and analysis/geum_dat.csv", header=TRUE)
#trait and climate data

#data clean
gdat$Replicate <- as.factor(gdat$Replicate) #rep per mom
```
~~~

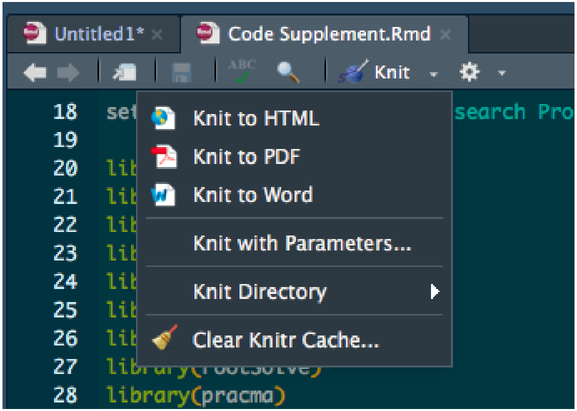
Text options

Everything written outside of the chunks will **not** be evaluated by R and will display as normal text.

There are many options for syntax here as well. Check the RMarkdown Cheat Sheet for some of the many options.

Final Output

When you’re ready to create an output file, you “knit” it:



```
library(rootSolve)
library(pracma)
```

My Thoughts

My general code usage: I use all the “clean coding” techniques above while I’m working on a non-finished project. I really like to use Markdown files once I have completed a project so I know what I did and how I did it in a super clean way. Here’s an example of a supplement I created for a paper in Markdown.