Data Management and Collaborative Coding

How to stay organized while doing team science

Why?

All science is collaborative

You are your own best (or worst) collaborator

These practices improve your efficiency in the long run

The big goal: project organization and code must be *intuitive* and *understandable*

Remember: This is an iterative process

Data Management

Data Management

In any project you will probably need some version of the following folders:

- **Raw inputs** - raw data you have collected
- **Cleaned inputs** - the result once the raw data have been cleaned
- **Scripts** - for your code
- **Output** - non-figure outputs
- **Figures** - figures generated from the analysis
- **Writing** - for resources for writing the paper; including methods!

Data Management Tip 1: Make file names unique and meaningful

Consider including

- Project name or acronym
- Study title
- Location
- Data type
- Researcher initials
- Date
- Data stage (raw, filtered, etc.)
- Version number
- File type
- If script, name should describe what it does

Data Management Tip 2: Always include metadata

- “Metadata” = data about the data

Always include

- Units
- Resolution
- Meaning of column names
- Description of caveats, issues, or missing values
- How data was collected
- Filtering or processing steps the data has been through (if applicable)

Data Management Tip 3: Remember your audience

What would someone unfamiliar with your data need to evaluate, understand, and reuse your data?

Remember your “audience”

Are they...

- ...a lab mate?
- ...colleague in your field?
- ...colleague in an related interdisciplinary field?
- ...colleague in unrelated field?
- ...newspaper journalist?
- ...congress?

Data Management Tip 4: Use README files

- Include a README file - README files are used to explain each file and its contents. Or to provide installation instructions for new software. Generally they're used to provide extra information your audience (including Future You). Get into the habit of updating the README whenever you change the contents of the folder

Example README for our NTU tables:
https://drive.google.com/drive/folders/1QS_ciQoWz1CyvG9cV4b09SuvAcoP8R5U

Data Management Tip 5: Change code not data

- Make as many changes as possible in the code; where you can't code make sure to include documentation of the changes you've made.
- This applies to data, but also to figures.

The more changes made in code, the better documented your work is, and the less effort you will need to put in later.

Activity 1: Data management

1: Think about a project you are currently working on.

2: Brainstorm 3 things that you can do to improve how you currently manage data for the project. (Be specific)

Collaborative Coding

Collaborative Coding Tip 1: Use a README.md file

Create a README file

Coding READMEs should include:

1. An abstract - why are you creating these scripts, what is the big-picture question
2. How to use (what packages need to be installed, how to run, what is the workflow, etc.)
3. Descriptions of all scripts

Bonus tip: If your readme is written in markdown, you can use pretty formatting!

Headers you should consider for readme files:

Introduction/Background/Motivation (why are you doing this, what are these files for?)

Contents (what’s in the folder)

Contact (your contact info in case someone has questions)

Installation (might not apply)

Usage (what do you need to know/have to run the code?)

Dependencies (what does your work depend on)

Get into the habit of using markdown for readme files

Markdown is a way of writing that can be translated by a computer into formatted text but can still be easily read by a human in plain-text format

Some examples:

- **bold text inside two asterisks** -> **bold text inside two asterisks**
- *Italics inside one asterisk* -> *Italics inside one asterisk*
- # Large Heading after pound sign -> **Large Heading after pound sign**
- “- “ for bullets

Checkout this cheatsheet:
https://docs.github.com/en/github/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax

Collaborative Coding Tip 2: Organize your scripts as a workflow

- Rather than putting all of your analysis in a single script of thousands of lines, divide your analysis into many scripts, each of which do one thing.
- Don’t be afraid to write-out outputs and pass them between scripts

Name	Size
00_directory_setup.R	2.3 kB
01_loading_data_and_cleaning_mapping_file.R	4.5 kB
02_running_pre-filtering_calculations.R	6.0 kB
03_filtering_taxa_and_rarefying.R	4.6 kB
04_comparing_blanks_and_samples.R	23.8 kB
05_preparing_cleaned_otu_table.R	18.8 kB
06_preparing_tree.R	3.0 kB
07_basic_community_stats.R	10.6 kB
07_Plotting_Sites_and_Climate.R	8.1 kB
08_Calculate_Richness.R	8.7 kB
09_Calculate_Biomass.R	7.7 kB
10_Calculate_Community_Composition.R	15.0 kB
11_Combine_Q1_tables_and_figures.R	6.7 kB
12_Richness_and_Climate_Variables.R	18.4 kB

Collaborative Coding Tip 3: Scripts should do one thing

- Rather than putting all of your analysis in a single script of thousands of lines, divide your analysis into many scripts, each of which do one thing.
- Don’t be afraid to write-out outputs and pass them between scripts

Name	Size
00_directory_setup.R	2.3 kB
01_loading_data_and_cleaning_mapping_file.R	4.5 kB
02_running_pre-filtering_calculations.R	6.0 kB
03_filtering_taxa_and_rarefying.R	4.6 kB
04_comparing_blanks_and_samples.R	23.8 kB
05_preparing_cleaned_otu_table.R	18.8 kB
06_preparing_tree.R	3.0 kB
07_basic_community_stats.R	10.6 kB
07_Plotting_Sites_and_Climate.R	8.1 kB
08_Calculate_Richness.R	8.7 kB
09_Calculate_Biomass.R	7.7 kB
10_Calculate_Community_Composition.R	15.0 kB
11_Combine_Q1_tables_and_figures.R	6.7 kB
12_Richness_and_Climate_Variables.R	18.4 kB

Collaborative Coding Tip 4: Treat coding like writing

Writing and coding are not that different. Both are iterative processes - start with your main ideas (tasks) and work your way through them.

1. Outline first - Write this down as comments (this is sometimes called writing “pseudocode”)
2. Write code
3. Test code
4. Edit, edit, edit

Pseudocode example:

1

```
# plot_ordinations.R

# Step 1: Read in data
# Step 2: re-format data
# Step 3: Calculate ordiantion
# Step 4: Plot ordination
# Step 5: save graphs, and distance matrix
```

Pseudocode example:

1

```
# plot_ordinations.R

# Step 1: Read in data
# Step 2: re-format data
# Step 3: Calculate ordiantion
# Step 4: Plot ordination
# Step 5: save graphs, and distance matrix
```

2

```
# plot_ordinations.R

# Step 1: Read in data
# Step 2: re-format data
# Step 3: Calculate ordination
### A) calculate distance matrix
### B) run ordination
# Step 4: Plot ordination
# Step 5: save graphs, and distance matrix
```

Pseudocode example:

1

```
# plot_ordinations.R

# Step 1: Read in data
# Step 2: re-format data
# Step 3: Calculate ordiantion
# Step 4: Plot ordination
# Step 5: save graphs, and distance matr
```

2

```
# plot_ordinations.R

# Step 1: Read in data
# Step 2: re-format data
# Step 3: Calculate ordination
### A) calculate distance matrix
### B) run ordination
# Step 4: Plot ordination
# Step 5: save graphs, and distance matrix
```

3

```
# plot_ordinations.R

# Step 1: Read in data
# Step 2: re-format data
# Step 3: Calculate ordination
# Calculating Bray-curtis dissimilarities
sb_transformed <- t(sqrt(input_rar_filt_reps$data_loaded))
dm_bc <- vegdist(sb_transformed, method = "bray")

# NMDS oridination (only for bray-curtis dissimilarity)
sb.nmds <-  metaMDS(dm_bc, k = 2, trymax = 100)

# Step 4: Plot ordination
# Step 5: save graphs, and distance matrix
```

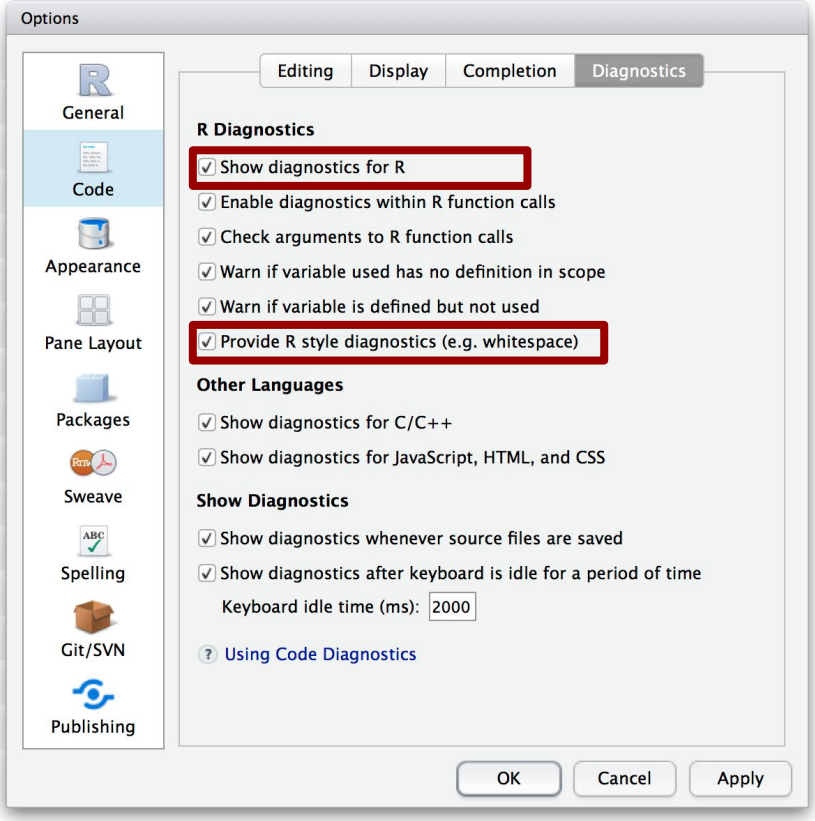
Collaborative Coding Tip 5: Comment, comment, comment

Use copious comments to explain what you are doing. Example workflow:

1. Write comment explaining what you are trying to do and why
2. Write first draft of code; troubleshoot; (add comments as needed)
3. Edit comments to reflect the code you've written

Collaborative Coding Tip 6: Be consistent with your style

- Make sure your variable names, function names, and spacing have consistent style
- No style is “correct” but you shouldn’t mix and match.
- You can get RStudio to correct your style “grammar” to force yourself to learn good habits.



Collaborative Coding Tip 7: Choose names that improve readability

- Name should describe the variable purpose
- If-then values should answer a question; variables should be nouns; functions should be verbs.
- Make names searchable, easy to distinguish, and pronounceable
- Avoid confusing characters or words with double meaning (“list”, “function”, “variable”, “return”, etc.)

```
if (is_covid_pos) {
    plot(x,y)
}
```

```
covid_positive_samples <- c("BAK_10",
"REG_8", "REG_10")
```

```
# names hard to distinguish
weekend_plot
weekend.plot
Weekend_plot1

# Confusing characters
1 l; 0 O o
```

Collaborative Coding Tip 7: Choose names that improve readability

- Use consistent “name molds” - templates that words fit into: ex: maxX, maxXperY
- Use dictionary words vs letters/abbreviations; and be consistent in usage
- Remember: Names are being chosen while your brain is occupied with problem solving/coding; So budget time to review the names after writing the code. Like a spellcheck.

```
if (is_covid_pos) {
    plot(x,y)
}
```

```
covid_positive_samples <- c("BAK_10",
"REG_8", "REG_10")
```

```
# names hard to distinguish
weekend_plot
weekend.plot
Weekend_plot1

# Confusing characters
1 l; 0 O o
```

Collaborative Coding Tip 8: Use spacing to make your code more readable

- Add spaces between operators
- Use tabs to align “=” or “<-” in lists
- Indent “unfinished” lines - for example: when using a “piped” dplyr string or ggplot figure creation

```r
# Good
average <- mean(feet / 12 + inches, na.rm = TRUE)

# Bad
average<-mean(feet/12+inches,na.rm=TRUE)
```

```r
list(
  total = a + b + c,
  mean  = (a + b + c) / n
)
```

```r
ggplot(data = yearly_counts, aes(x = year, y = n)) +
    geom_line() +
    facet_wrap(facets = vars(genus))
```

Collaborative Coding Tip 9: Give your code to someone else

- Code reviews are common in industry but less common in academic bioinformatics settings.
- Get a friend or mentor to read through your code and give you feedback. Ideally they should be able to both run AND understand your code

Activity 2: Code Review

Link to code:

https://raw.githubusercontent.com/hhollandmoritz/Data_Management_and_Collaborative_Coding/main/examples/10_Calculate_Community_Composition_original.R

1: open the link above, copy everything (ctrl + A) and paste it into a text editor on your computer (don’t use Word, instead use a text editor like text wrangler, MS text editor, or Rstudio to open it).

2: think about where the code needs edits, make comments in the code (start with #) about what should be changed.

Collaborative Coding Bonus Tip: use version control

Invest time to learn to use a version-control software like git

There are many resources out there to learn, but here's one that I made:

https://github.com/hhollandmoritz/git_guide/blob/master/knit/git_lesson_pres.pdf
https://github.com/hhollandmoritz/git_guide

Universities often host workshops

Take homes:

Document, document, document

Prioritize clarity and editability

Don’t be afraid to edit

Resources:

Data Management:

Code and data management from NSIDC: https://depts.washington.edu/mtnhydr/snowschool/Rosati_data.pdf

Ten simple rules for data management:
https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004525

Data Carpentry lessons: https://datacarpentry.org/lessons/#ecology-workshop

Data organization: https://kbroman.org/dataorg/

Coding

https://drive.google.com/file/d/1TraVwRkbkCbHq-s_-NS69ZEbRNwH8XNh/view (Dan Larremore presentation)

R style guide: http://adv-r.had.co.nz/Style.html

Initial steps towards reproducible research: https://kbroman.org/steps2rr/

The Programmer's Brain - Felienne Hermans (excellent book)