Data Management in the Dan Pad Lab

Daniel Padfield

July 21, 2025

Table of contents

Outline

This documents covers the broad principles that we use to manage and organise our data in the Dan Pad Lab. It is not designed to be exhaustive or overly prescriptive, but instead to give a minimum set of guidelines to follow that will save us time and ensure our data is as useful as possible. **When you have completed a project or when you leave the Dan Pad Lab, you should make sure all your data is uploaded to the PadLabNas.**

It is a living document that will evolve as we remember important things we do that need to be in here, and as we learn new important things that we should be doing.

Any questions or comments email me at d.padfield@exeter.ac.uk, message me on Teams, or find me in person!

Why?

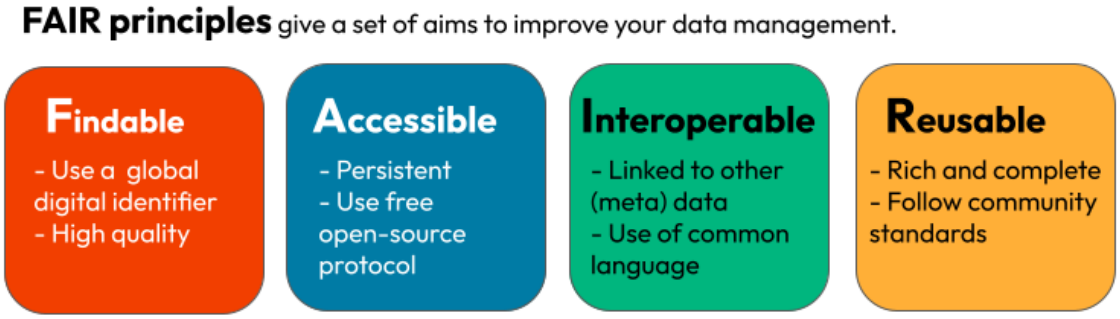
There are many reasons for good, consistent data management practices, and I will list the ones in order I think are most persuasive. I am aware it starts with the ones that are most selfish.

1. It will save you time. You likely work across projects and sometimes will need to revist a project that you have not looked at for weeks, months, and if like me, possibly years. If you have good data management and documentation, you will be able to pick up where you left off much quicker. Saving you **SO MUCH TIME.** Compared to coming back to a mess.
2. If you do it well enough for future you to understand, it will be understandable for other people in the group and beyond. This is important for collaboration and sharing your work with others, and makes your work more useful! This is good for your career, and for science as a while.

3. It makes your work reproducible - code wise. This is important for science. If we all did this, we would not reinvent the wheel as much in terms of code. Think how it would go if **lme4** did not exist. All of our projects are reproducible and the code and data are available on GitHub and Zenodo for anyone to use.
4. It is now mandated by most funders and journals that code and data is published for each study. As this is going to be needed anyway, we might as well adopt good practice early to make it easier for ourselves when we come to publishing.

What?

Data management involves the organisation, storage, preservation, and dissemination of the data and code that we generate in our projects. It is a broad definition and there are many frameworks that already exist, one of the most popular being the FAIR principles, which give a set of guidelines to improve data management.



General rules

TLDR

There are a few general rules that will get you most of the way to good data management in the Dan Pad Lab. Feel free to add others to your toolkit, but these are my bare bone recommendations.

1. Have an exhaustive Lab Notebook.
2. Never manually edit raw data files.
3. Always have a README for each project.
4. Have a consistent folder structure for each project.
5. Have a consistent naming scheme.
6. Split your scripts up into small, units doing one thing (ish).
7. Comment your code.

Lab Notebook

People working in the Lab should treat their lab notebooks like holy scripture. If you work inh the lab, you are required to have a lab notebook. **The purpose of the lab notebook is not only to help you remember details of what you did and why months or potentially years later, but also as solid proof that we did the things the way we said we did**. Anything that has to do with an experiment you run should be documented here. Pages should be dated and written in ink. Write down details of the experimental planning, scheduling, recipes for media, observations from your plates, outline of the projected steps of the analysis. Trust me, you will forget details and you may need to reference them after months or (several) years. Never throw your lab notebooks out. When you leave, the lab notebook, or photos of every page, will stay in the lab.

Raw data files

Never manually edit raw data files. Period. Microsoft Excel or other spreadsheet software is super useful for data entry (e.g. recording plate counts and dilutions), but it should not be used beyond this. After data entry has been done, the data should be saved and not touched again. If the raw files are outputs from an machine, such as a plate reader, these should never be edited. This prevents accidental and irreversible changes to the data. Any filtering or processing of the data can then be done in a programming language like R, and processed datasets can be saved as separate files. This way, we have an electronic "paper trail" of our steps from raw to processed data.

README

README's allow us to easily get a summary of a project and allow us to keep a record of what you did in your project, why it is useful, and what data and code you produced during your work. It allows you to communicate important information about your work, such as what column names in your datasets mean, what units they are in, and any information about how best to run and use the code you used to do your analysis and create your graphs. This is good practice for making your work reproducible so anyone else in the future can understand it and is useful for everyone (including future you)!

A template README for your project is available here. Follow the **How to use this document** section to fill it out for every project. An example README of mine can be found here.

Naming scheme

A naming scheme for your files, folders, and variables within scripts will save you so much time in the long run. It will save you time, creates

interoperability between your projects, and makes it easier to share you work with others in the groups. This will make it easier to interpret your work, and improve its reproducibility. A consistent naming scheme is one part of a style guide that people have to that standardises the formatting of their code and data management. An in-depth discussion of style guides is found here.

General rules I try to follow are:

- when naming variables in code, I use **snake_case**, which replaces spaces with underscores and uses only lowercase letters. Being consistent with this makes it easier for me to read and understand my code.
- when naming files, I again try to use **snake_case**. I often store metadata or treatment data in the name of the data file, so this can be split into later on in R. For example, if we measured OD of *E. coli* in a plate reader on the 24th July 2025, and it was at 30ºC, I might use the name **20250725_30_ecoli.xlsx**. This can then be split into the data, temperature, and organism later on. The components of the name separated by underscores may change from project to project, or even for different types of data within the same project, but the general principle is to use a consistent naming scheme that is easy to read and understand.
- when naming scripts, I try to describe what the script does, use **snake_case** and I number them sequentially. For example, if the first script reads data in, and the second cleans the data, I might use the names **01_read_data.R** and **02_clean_data.R**. This makes it easier to follow the flow of the project, and to find the script that does what you want. If I have a script that contains helper functions that are used in the other scripts, I usually name it **00_functions.R**. It can then be called at the start of the relevant scripts using `source("00_functions.R")`.

Folder structure

Having a consistent folder structure for each project makes your life so much easier. In depth recommendations of folder structure can be found here. I use Google Drive to store my projects, and each project generally gets its own folder. At the base of this folder is where I put the README file for that project, and the base folder is where I open an **RStudio Project** or a **Positron Folder** which allows for consistent working directory settings across computers and workspaces (e.g. means if you send me your code I can run it regardless of where I put it on my computer).

Inside each project folder, I always have the following 3 folders:

1. **data**: This is where the data goes. Within this folder there are likely subfolders such as **raw** and **processed**.
2. **scripts**: This is where the scripts go. They are normally named sequentially, such that the first script starts with **01_** etc.
3. **plots**: This is where the plots are saved to.

An example folder structure of a project might look like.

```
my_fun_project
|-- README.docx
|-- data
|   |-- raw
|   `-- processed
|-- plots
|   `-- my_first_ggplot.png
`-- scripts
    |-- 00_functions.R
    |-- 01_read_data.R
    |-- 02_processing.R
    `-- 03_analysis.R
```

Split your scripts

Writing R code can get messy. We can go down rabbit holes and write hundreds of lines of code that turn out to be pointless. Many of us keep all of this in a single, increasingly long script that has all of the different steps, from reading in raw data to analysis and plotting in it. This can make it very difficult to read. Instead, keep things that are logically related closer together in individual scripts, and name them sequentially. For example, for loading in and binding all the raw files, create a script **01_read_data.R**. For cleaning the data, have a script named **02_clean_data.R**. For analysis, have a script **03_analysis.R**. This way, you can easily find the script that does what you want, and it is easier to read and understand your code.

Comment your code

Code comments are invaluable for you to be able to understand exactly what you did and why you are doing it! A good primer on code comments can be found on StackOverflow. But essentially # is the universal comment character, meaning if a line starts with a #, it is ignored by R and not run as code. In general, write clear comments that explain what you have done and why. If you have found the method from somewhere, or you justified your approach using something you found in a paper or online, link to it in the comments. Again, this will save you time in the long run, and make your code easier to read and understand.

Also, it can allow you to let off some steam and comment on how exasperated you are with how everything is going, as I did here. Just make sure you clean them up before you publish it!

```
# what the fuck is going on
filter(extra_info, sample == 'sample_s46') %>%
  View()

filter(extra_info, sample == 'sample_s47') %>%
  View()
```

Figure 1: A difficult day of coding was had.

Inspiration

We took a lot of inspiration from Crystal Lewis, a research data management consultant, and her online book Data Management in Large Scale Education Research. We also found Kate Laskowski's Lab Values and Expectations page very useful.