Lecture 5

Brooke Anderson



Sharing code and data

There are a few options for how you share the code and data for the paper, including:

- Supplemental files through the journal's website
- GitHub
- Scientific repository, like [examples, NIH]
- ► Personal / academic website
- R package plus script

Sharing code and data

There are advantages to solutions that let you share everything as a directory:

- Relative pathnames work in scripts (because directory structure is preserved)
- ▶ If you set up your directory as an R Project, it's designed to be self-contained (can interact with other things in the directory but doesn't rely on the computer's file directory outside of that)

Leveraging a sharing system before you publish

Some of these methods are platforms that you can use before you publish, and they can help you work on code-heavy projects within your team. GitHub is one example, but some agency-funded platforms now combine repositories with workspaces for coding.

For GitHub (and some of the agency-funded platforms), you can toggle between having your data and code be public or private. This allows you to work privately before you publish, but then switch the repository to public when you publish the paper.

This can be much easier than trying to figure out how to post everything after publication. It also provides some helpful tools as you work on the paper.

Considerations about data

Processed data versus raw data

A lot of health-related data can require extensive pre-processing to extract the information you need to answer a scientific question.

It is helpful to include the original, raw data and the code used to get to the processed data.

Extremely large data

- Repositories built for data curation might be better suited for storing data that methods with data size limits ([example])
- Some repositories might have APIs—data can be downloaded programatically (from the code script), so the code could be connected to the data in this way
- Some repositories allow options to toggle private versus public (similar to GitHub)

Sensitive or protected data

- ► Can include mock dataset (same format, different numbers) to use to test and demonstrate the function of the code
- Can use .gitignore to include the data in a git repo on a local computer (or a secure server you're working on) but not push it to remote versions of that repo, like one on GitHub
- ▶ When working on a server for security reasons, it can be helpful to have RStudio Server installed, as this provides a friendlier interface for working with git version control on the server (avoids having to do much from the terminal)

Saving intermediate data

It can be useful to save intermediate versions of the data, as the code moves from raw data to the final figures, tables, and other results. Intermediate data can include:

- Processed data (e.g., if raw data are from flow cytometry, the results after gating and counting the data)
- Data points that are plotted in figures (i.e., data at the point immediately before it is graphed)

Note that this is in addition to the raw data.

For these intermediate data sets, it's helpful if you save them in plain text formats (e.g., ".csv" or ".txt" file).

(Sandve et al.)

Considerations about Code

Navigating the code

Can a reader or new user:

- ► Find where the code is?
- Find what order to run the code?
- Use the filepaths that are included in the code? (Or are they absolute paths that will need to change on a new user's computer?)
- ► Figure out which code was used to create specific figures and tables in the paper?
- ► Figure out how to start from a raw dataset similar to the data used for the paper and get to the final results?

Code License

Are others allowed to reuse your code? What are the rules for how they reuse it?

License choices

Versions used in original analysis

R base code changes from one version to another.

The code within packages can change even more.

Seems to be a particular source of difficulty in reproducing for work that involves Bioconductor packages.

Versions used in original analysis

You can record the information on all versions of the R software you used in your analysis (base R and packages).

 ${\tt session_info} \ from \ devtools \ (or \ {\tt sessionInfo} \ from \ base \ R)$

Include this as a line in an RMarkdown file (often at the end), and it will print out the information on versions.

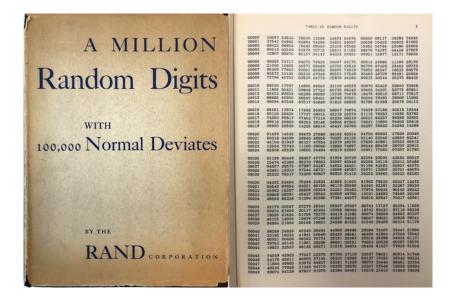
Versions used in original analysis

Help user recreate the "environment" that code ran in originally

If you have version numbers, you could do it by hand (all old versions of CRAN packages, for example, are archived and available for you to install, although it takes a bit more work that installing the current version of the package).

[Packrat alternatives]

[Dice, roulette wheel]





pseudorandom number generator

You might be using random numbers if you:

- are sampling
- are using the Monte Carlo method / simulations
- are doing Bayesian statistics

Setting seeds when code includes random number generation

Seed can be any integer

Pseudorandom number generator—depends on an initial value (the seed)

If you don't set a seed, you will get different results when you run code that involves random number generation, because of the randomness involved.

```
sample(1:5)
## [1] 5 4 2 3 1
sample(1:5)
## [1] 4 2 5 3 1
```

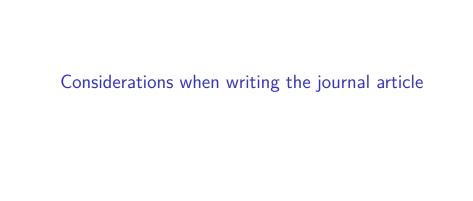
If you set the same seed each time before you run that code, you will get the same "random" results:

```
set.seed(100)
sample(1:5)

## [1] 2 3 5 4 1
set.seed(100)
sample(1:5)
```

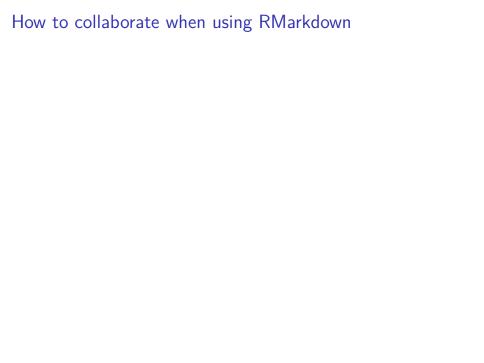
```
## [1] 2 3 5 4 1
```

```
set.seed(100)
sample(1:5)
## [1] 2 3 5 4 1
sample(1:5)
## [1] 1 2 4 3 5
```



RMarkdown for journal articles





More advanced and complex approaches

Considering containers

[Docker. Galaxy? Workspaces on agency platforms?]

Considering containers

Containers can get very large—stores not just info on the versions of each piece of software used, but full source code (?)

Works somewhat like a black box—harder for users to explore, change, adapt

Can be useful, though, if using a large collection of different open-source software (command line tools, R, Python, etc., all in the same pipeline)

Considering making a package