13: Summary + Extras

bit.ly/SISBID3

We'll briefly summarize all that we discussed, and further touch on how to share your research data and code. Then we'll give pointers to the many things that we didn't have time to talk about.

The most important tool is the mindset, when starting, that the end product will be reproducible.

– Keith Baggerly

So true. Desire for reproducibility is step one.

Steps toward reproducible research

- Slow down
- Organize; document
- Everything with code
- Scripts → RMarkdown
- Code → functions → packages
- Version control with Git
- Automation with Make
- Choose a license
- Share your work with others

Moving from "standard practice" to "fully reproducible" is hard. There are a lot of tools to learn and a lot of workflow changes to make. Don't try to change everything all at once. Focus on improving one aspect at a time, ideally jointly with your friends and colleagues. Your goal should be to have each project be a bit better organized than the previous.

Challenges

- Daily maintenance
  - READMEs up to date?
  - Documentation match code?
- Cleaning up the junk
  - Move defunct stuff into an `Old/` subdirectory?
- Start over from the beginning, nicely?

The organization of a project is dependent on your worst day with it. You keep everything carefully arranged for months and then one day you need to rush, rush, rush to get a manuscript out the door, and you leave a big mess.

And a common problem is that you don't really know what you're doing until it's all done. So if you can, spend a week at the end making a separate, clean version of the whole thing.

Sharing your work

- Why share?
  - Funding agency or journal requirement
  - Increased visibility
  - So that others can build on your work
- When?
  - Continuously and instantaneously
  - When you submit a paper
  - When your paper appears
- Risks?

There a lot of advantages to making your work public, and you may be required to make it public.

Some scientists put all of the work in the open as they're doing it. Others wait until they submit a manuscript; others until the manuscript actually appears. Still others want to delay releasing data yet further. The earlier the better, I think.

Are there risks? Many worry about being scooped: having someone find something cool in your data before you got a chance to find it yourself. I think the risk of this is far outweighed by the advantages of having data and code in the open. You may also worry about people finding problems in your analyses. But if there are problems, wouldn't you rather know about them? You can't build upon it if it's not right.

The biggest risk is that you'll be ignored. Sharing more, sooner, and in a more convenient form will encourage broader visibility.

What to share?

- For sure
  - Primary dataset
  - Metadata
  - Data cleaning scripts
  - Analysis scripts
- It could help
  - Very-raw data
  - Processed/clean data
  - Intermediate results
- No
  - Confidential data (e.g. HIPAA data)
  - Passwords, private keys

It can be tricky to define what is the "primary dataset". How raw of data should you share? Where would someone want to start, and how can you make it easiest for them to dig into your analyses or make use of the data for other purposes, such as for a meta-analysis study? In some cases, it can be valuable to share many of the intermediate results, so that folks can jump into the bit that they care about without having to first run a bunch of complex and time-consuming analyses to get there.

You definitely want to make sure that you're not including any confidential information, particularly regarding patient-level data, but also private keys for data APIs which you might be including in your scripts.

Where to share?

- Domain-specific repository
  - Genbank, dbGaP, etc.
  - See `re3data.org`
- Figshare, Dryad, Zenodo
- Institutional repository
- GitHub, BitBucket

Share code at GitHub or BitBucket. But how about data? Consider domain-specific repositories like dbGaP, then general data repositories like Zenodo, and then look to see whether your institution has a data repository.

Another option is as supplemental material for a publication, at the Journal's website, but this is often the most cumbersome for users.

Resources

- R Markdown
  - rmarkdown.rstudio.com
- R Packages
  - Releasing to CRAN: r-pkgs.had.co.nz/release.html
  - Leek group: github.com/jtleek/rpackages
  - When to trust an R package: bit.ly/trust_r_pkg
- Make
  - kbroman.org/minimal_make
- Git
  - Git branches: nicercode.github.io/git/branches.html
  - Hadley on Git/GitHub: r-pkgs.had.co.nz/git.html
  - Git subtrees: bit.ly/git_subtree

Links to some resources related to the course material.

Some of the things we didn't cover

- Command-line, including 'R CMD BATCH'
- Software testing (and debugging)
- Code review / paired programming
- Capturing versions of dependent software (e.g. packrat)
- Containers (e.g. `docker.com`)
- Coding conventions, e.g. Hadley's `adv-r.had.co.nz/Style.html` and Google's `google.github.io/styleguide/Rguide.xml`
- Report templates, `rmarkdown.rstudio.com/developer_document_templates.html`

There's a lot that we didn't cover.