Steps toward reproducible research

Karl Broman

Biostatistics & Medical Informatics
Univ. Wisconsin–Madison

kbroman.org
github.com/kbroman
@kwbroman
Slides: bit.ly/jax2017-05

These are slides for a talk I've given a whole bunch of times, most recently for a course on Big Genomic Data at the Jackson Lab, on 17 may 2017.

Source: https://github.com/kbroman/Talk_ReproRes
Slides: http://bit.ly/jax2017-05_nonotes
With notes: http://bit.ly/jax2017-05

```
Karl -- this is very interesting,
however you used an old version of
the data (n=143 rather than n=226).

I'm really sorry you did all that
work on the incomplete dataset.

Bruce
```


This is an edited version of an email I got from a collaborator, in response to an analysis report that I had sent him.

I try to always include some brief data summaries at the start of such reports. By doing so, he immediately saw that I had an old version of the data.

Because I'd set things up carefully, I could just substitute in the newer dataset, type "`make`", and get the revised report.

This is a reproducibility success story. But it took me a long time to get to this point.

The results in Table 1 don't seem to correspond to those in Figure 2.

My computational life is not entirely rosy. This is the sort of email that will freak me out.

In what order do I run these scripts?

Sometimes the process of data file manipulation and data cleaning gets spread across a bunch of scripts that need to be executed in a particular order. Will I record this information? Is it obvious what script does what?

Where did we get this data file?

Record the provenance of all data or metadata files.

Why did I omit those samples?

I may decide to omit a few samples. Will I record why I omitted those particular samples?

How did I make that figure?

Sometimes, in the midst of a bout of exploratory data analysis, I'll create some exciting graph and have a heck of a time reproducing it afterwards.

“Your script is now giving an error.”

It was working last week. Well, last month, at least.

How easy is it to go back through that script’s history to see where and why it stopped working?

“The attached is similar to the code we used.”

From an email in response to my request for code used for a paper.

Reproducible

vs.

Replicable

Computational work is reproducible if one can take the data and code and produce the same set of results. Replicable is more stringent: can someone repeat the experiment and get the same results?

Reproducibility is a minimal standard. That something is reproducible doesn't imply that it is correct. The code may have bugs. The methods may be poorly behaved. There could be experimental artifacts.

(But reproducibility is probably associated with correctness.)

Note that some scientists say replicable for what I call reproducible, and vice versa.

Steps toward reproducible research

`kbroman.org/steps2rr`

The above website contains my thoughts on how to move towards full reproducibility.

Don't try to change every aspect of your workflow all at once.

1. Organize your data & code

Your closest collaborator is you six months ago, but you don't reply to emails.

(paraphrasing Mark Holder)

The first thing to do is to make your project understandable to others (or yourself, later, when you try to figure out what it was that you did.

Segregate all the materials for a project in one directory/folder on your harddrive.

I prefer to separate raw data from processed data, and I put code in a separate directory.

Write `ReadMe` files to explain what's what.

2. Everything with a script

If you do something once,
you'll do it 1000 times.

The most basic principle for reproducible research is: do everything via code.

Downloading data from the web, converting an Excel file to CSV, renaming columns/variables, omitting bad samples or data points...do all of this with scripts.

You may be tempted to open up a data file and hand-edit. But if you get a revised version of that file, you'll need to do it again. And it'll be harder to figure out what it was that you did.

Some things are more cumbersome via code, but in the long run you'll save time.

3. Automate the process (GNU Make)

```
R/analysis.html: R/analysis.Rmd Data/cleandata.csv
    cd R;R -e "rmarkdown::render('analysis.Rmd')"

Data/cleandata.csv: R/prepData.R RawData/rawdata.csv
    cd R;R CMD BATCH prepData.R

RawData/rawdata.csv: Python/xls2csv.py RawData/rawdata.xls
    Python/xls2csv.py RawData/rawdata.xls > RawData/rawdata.csv
```


GNU Make is an old (and rather quirky) tool for automating the process of building computer programs. But it's useful much more broadly, and I find it valuable for automating the full process of data file manipulation, data cleaning, and analysis.

In addition to automating a complex process, it also documents the process, including the dependencies among data files and scripts.

4. Turn scripts into reproducible reports

Gough project diagnostics

Karl Broman, 3 March 2014

Comb

I've comb
the well-b
informati
informati
give one

There are
data have
mice and

```
I've combined the initial genotypes (using the re-clustered genotypes
for plates 14-16) with the well-behaved portion of the re-run
genotypes. I'm focusing on `r totmar(g)` markers that are informative
(though, as we'll see, there are still a lot of badly behaved and
basically non-informative markers that need to be removed).
I've combined data on replicate samples, to give one set of genotype
calls for each sample.

There are `r nind(g)` genotyped mice and `r nrow(phe)` phenotyped
mice. All of the mice in the phenotype data have genotypes, but there
are `r sum(is.na(match(gid, pid)))` genotyped mice with no phenotypes,
including `r sum(g$pheno$gen[which(is.na(match(gid, pid)))]==0)`
Gough mice and `r sum(g$pheno$gen[which(is.na(match(gid, pid)))]==2)`
F2 progeny.
```


I love R Markdown for making reproducible reports that document the full details of my analysis. R Markdown mixes Markdown (for light-weight markup of text) and R code chunks; when processed with knitr, the R code is executed and results inserted into the final document.

With these informal reports, I seek to fully capture the entirety of my data explorations and decisions.

Python people should look at iPython notebooks.

5. Turn repeated code into functions

```
# Python
def read_genotypes (filename):
    "Read matrix of genotype data"
```

```
# R
plot_genotypes <-
function(genotypes, ...)
{
}
```


Pull out complex or repeated code as a separate function. This makes your code easier to read and maintain.

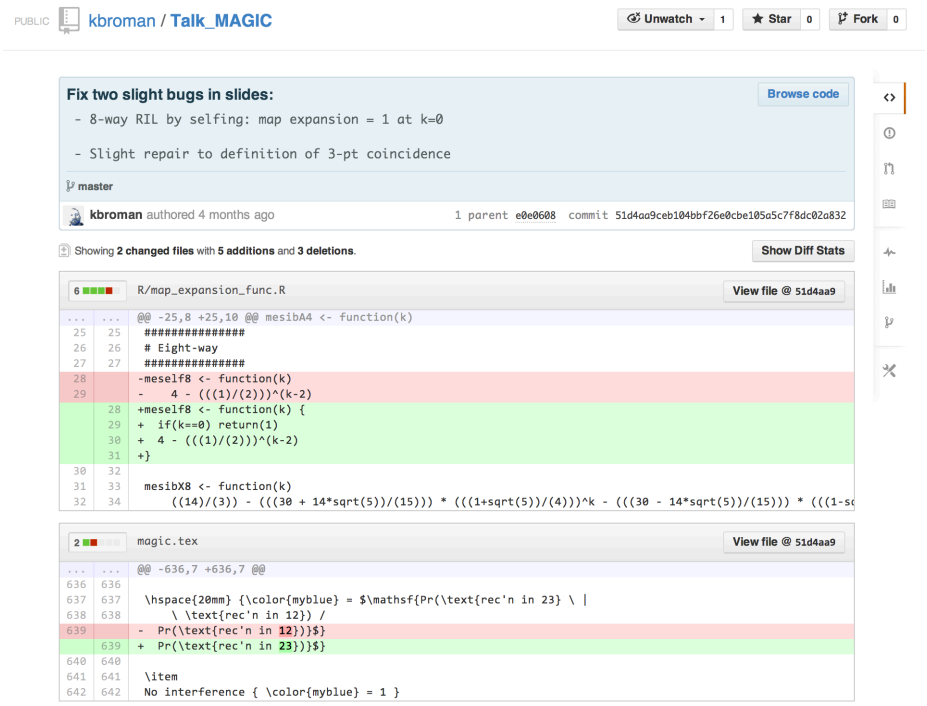
6. Create a package/module

Don't repeat yourself

It's surprisingly easy to create an R package (see `http://kbroman.org/pkg_primer`) and it's even easier to make a Python module.

When writing functions, try to write them in a somewhat-general way and then pull them out of the project as separate package or module, so that you (and/or others) may reuse them for other purposes.

7. Use version control (git/GitHub)



git has a steep learning curve, but ultimately I think you'll find it really helpful.

The big selling point is in collaboration: merging changes from collaborators, and keep your work synchronized.

Longer term, there's great value in having the entire history of changes to your project. If something stops working, you can go back to any point in that history to see when it stopped working and why.

With git, you can also work on new features or analyses without fear of breaking the parts that are currently working well.

8. License your software

Pick a license, any license

– Jeff Atwood

If you don't pick a license for your software, no one else can use it.

So if you want to distribute your code so that others can reproduce your analyses, you need to pick a license, any license.

I choose between the MIT license and the GPL.

Don't use the Creative Commons licenses for code. But feel free to use them for other things.

Other considerations

- Testing
  - are you getting the right answers?
- Software versions
  - will your stuff work when dependencies change?
- Large-scale computations
  - computation time + dependence on cluster environment
- Collaborations
  - coordinating who does what and where things live
- Distribution
  - where and how to distribute data and code?

I've focused on issues for small-scale, single-investigator projects, and even with that limited scope, I've not covered everything.

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.

Summary

1. Organize your data & code
2. Everything with a script
3. Automate the process (GNU Make)
4. Turn scripts into reproducible reports
5. Turn repeated code into functions
6. Create a package/module
7. Use version control (git/GitHub)
8. Pick a license, any license

It's always good to include a summary.

Slides: bit.ly/jax2017-05

kbroman.org

github.com/kbroman

@kwbroman

Here's where you can find me, as well as the slides for this talk.