Steps toward reproducible research

Karl Broman

Biostatistics & Medical Informatics
Univ. Wisconsin–Madison

kbroman.org
github.com/kbroman
@kwbroman
Slides: bit.ly/StPaul-0

These are slides for a one-hour workshop on Reproducible Research, part of the Data Science and Plant Breeding Simulation Workshop at the University of Minnesota on 23 March 2017.

Source: `https://github.com/kbroman/RR_Workshop`
Slides: `http://bit.ly/StPaul-0-nonotes`
With notes: `http://bit.ly/StPaul-0`

Preparations

Install R, RStudio, and some R packages.

`bit.ly/StPaul-prep`

For Activity 3, we will be creating an R Markdown document, which mixes text and code to make a fully reproducible report.

You will first need to install, R, RStudio, and some R packages. It's best to have the latest versions, so if you installed them in the distant past but haven't updated them recently, please re-install them.

```
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 equivalent 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 toward full reproducibility.

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

1. Arrange data to ease analysis

Write programs for people

Organize data for computers

One may be tempted to think that, when you're writing a computer program, you're trying to communicate with the computer. But while it's important that the program executes correctly, it is just as important that it is readable. Then others (which might be you, three months from now) will be better able understand the code, for example to fix problems, or to build upon it.

On the other hand, when laying out data within a spreadsheet, you should think principally about having it be easily read by the computer; this will make analysis easier.

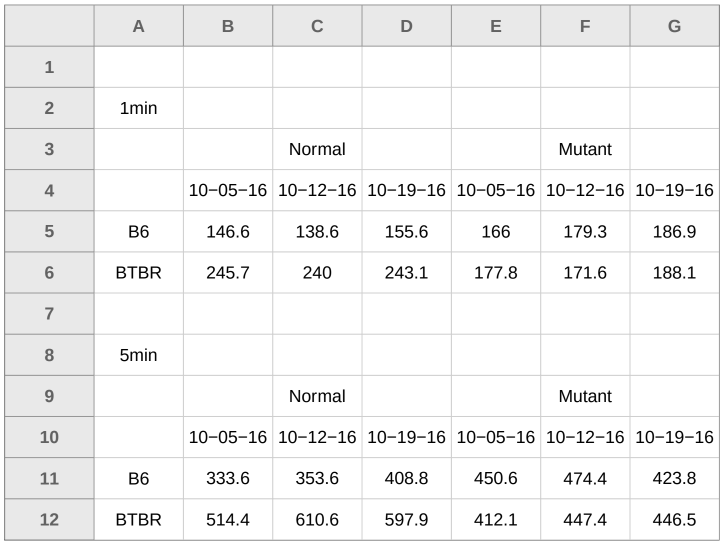
Activity 1

Arrange data to ease analysis

`bit.ly/StPaul-1`

First activity: study the data in a spreadsheet, and discuss how the arrangement could be improved.

Activity 1: original file



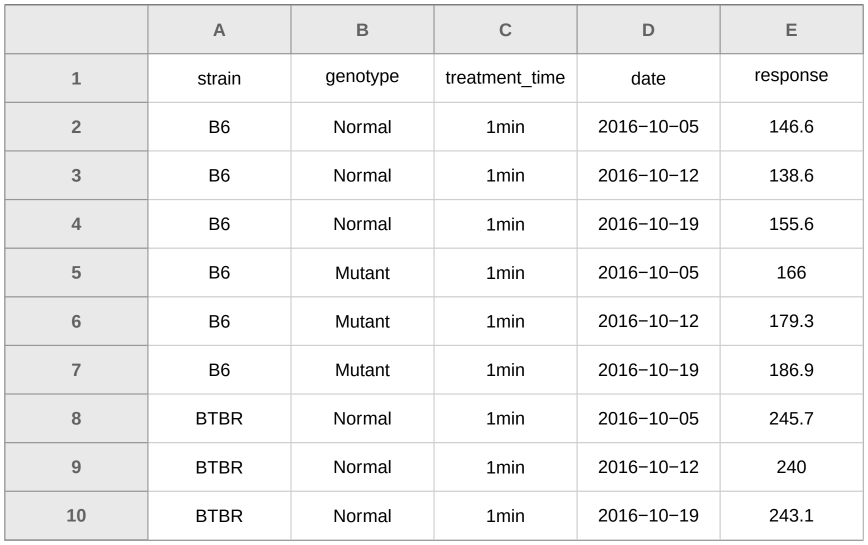
	A	B	C	D	E	F	G
1							
2	1min						
3			Normal			Mutant	
4		10−05−16	10−12−16	10−19−16	10−05−16	10−12−16	10−19−16
5	B6	146.6	138.6	155.6	166	179.3	186.9
6	BTBR	245.7	240	243.1	177.8	171.6	188.1
7							
8	5min						
9			Normal			Mutant	
10		10−05−16	10−12−16	10−19−16	10−05−16	10−12−16	10−19−16
11	B6	333.6	353.6	408.8	450.6	474.4	423.8
12	BTBR	514.4	610.6	597.9	412.1	447.4	446.5

`bit.ly/StPaul-1`

Here's an image of the original file. What could be improved?

After studying this for a bit, you can likely figure out what the data mean, but a computer is going to have a harder time. If you read this into R and want to make some plots, you're going to have to spend a good amount of time reorganizing the data.

Activity 1: tidy file



	A	B	C	D	E
1	strain	genotype	treatment_time	date	response
2	B6	Normal	1min	2016−10−05	146.6
3	B6	Normal	1min	2016−10−12	138.6
4	B6	Normal	1min	2016−10−19	155.6
5	B6	Mutant	1min	2016−10−05	166
6	B6	Mutant	1min	2016−10−12	179.3
7	B6	Mutant	1min	2016−10−19	186.9
8	BTBR	Normal	1min	2016−10−05	245.7
9	BTBR	Normal	1min	2016−10−12	240
10	BTBR	Normal	1min	2016−10−19	243.1

`bit.ly/StPaul-1-tidy`

Here's an image of a "tidy" version of the file.

The key thing: the data are now arranged as a rectangle, with the rows being observations and columns being variables. If you read this version into R, you'll be able to start making plots right away.

Organizing data in spreadsheets

- Make it a rectangle (rows = observations, cols=variables)
- Use a single header row; avoid spaces.
- Be consistent.
- Use care about dates. (3 separate columns?)
- Put just one thing in a cell.
- Fill in all cells.
- No calculations in the raw data files.
- Don't use font color or highlighting as data.
- Make backups.
- Use data validation to avoid data entry mistakes.
- Save the data in plain text files.

`kbroman.org/dataorg`

A summary of considerations in the organization of data in spreadsheets. See `kbroman.org/dataorg`

2. Organize your data & code

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

(paraphrasing Mark Holder)

Step 2 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 hard drive.

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.

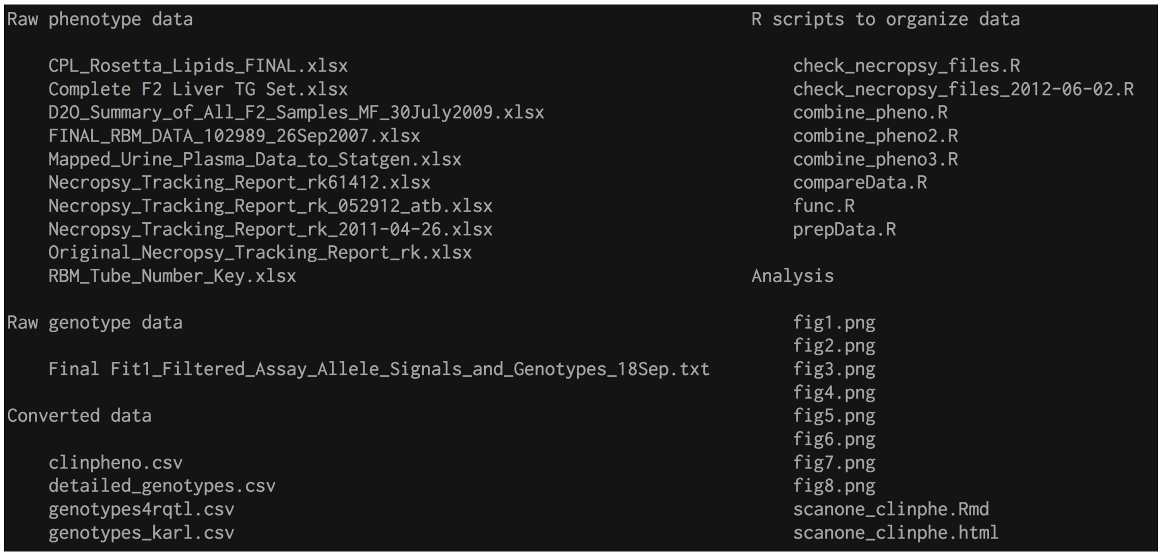
Activity 2

Organizing and naming of files for a project

`bit.ly/StPaul-2`

Second activity: given a list of files for a project, how would you organize them? Would you rename some of the files?

Activity 2



```
Raw phenotype data                                              R scripts to organize data

    CPL_Rosetta_Lipids_FINAL.xlsx                                   check_necropsy_files.R
    Complete F2 Liver TG Set.xlsx                                   check_necropsy_files_2012-06-02.R
    D2O_Summary_of_All_F2_Samples_MF_30July2009.xlsx                combine_pheno.R
    FINAL_RBM_DATA_102989_26Sep2007.xlsx                            combine_pheno2.R
    Mapped_Urine_Plasma_Data_to_Statgen.xlsx                        combine_pheno3.R
    Necropsy_Tracking_Report_rk61412.xlsx                           compareData.R
    Necropsy_Tracking_Report_rk_052912_atb.xlsx                     func.R
    Necropsy_Tracking_Report_rk_2011-04-26.xlsx                     prepData.R
    Original_Necropsy_Tracking_Report_rk.xlsx
    RBM_Tube_Number_Key.xlsx                                    Analysis

Raw genotype data                                                   fig1.png
                                                                    fig2.png
    Final Fit1_Filtered_Assay_Allele_Signals_and_Genotypes_18Sep.txt    fig3.png
                                                                    fig4.png
Converted data                                                      fig5.png
                                                                    fig6.png
    clinpheno.csv                                                   fig7.png
    detailed_genotypes.csv                                          fig8.png
    genotypes4rqtl.csv                                              scanone_clinphe.Rmd
    genotypes_karl.csv                                              scanone_clinphe.html
```

bit.ly/StPaul-2b

Here's the same list, annotated a bit.

3. 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.

4. Automate the process (GNU Make)

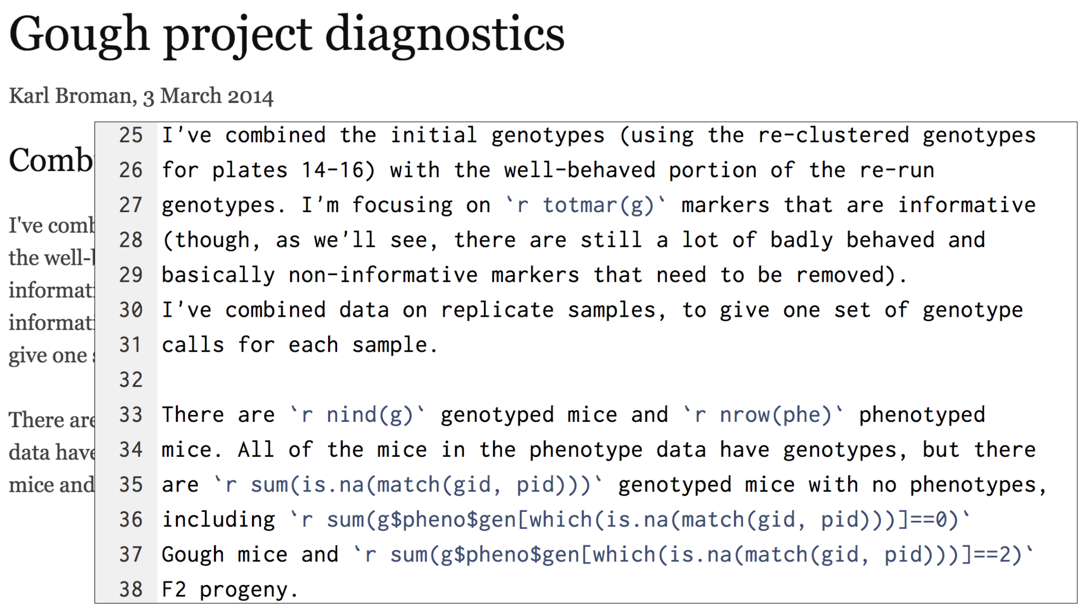
```
R/analysis.html: R/analysis.Rmd Data/cleandata.csv
    cd R;R -e "rmarkdown::render('analysis.Rmd')"

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
```

kbroman.org/minimal_make

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.

5. Turn scripts into reproducible reports


Gough project diagnostics

Karl Broman, 3 March 2014

Comb

I've comb
the well-l
informat
informat
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.

Activity 3

Create an R Markdown report within RStudio

`bit.ly/StPaul-3`

Third activity: we'll open R within RStudio and create an RMarkdown report. The link above is an example.

6. 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.

7. Create a package/module

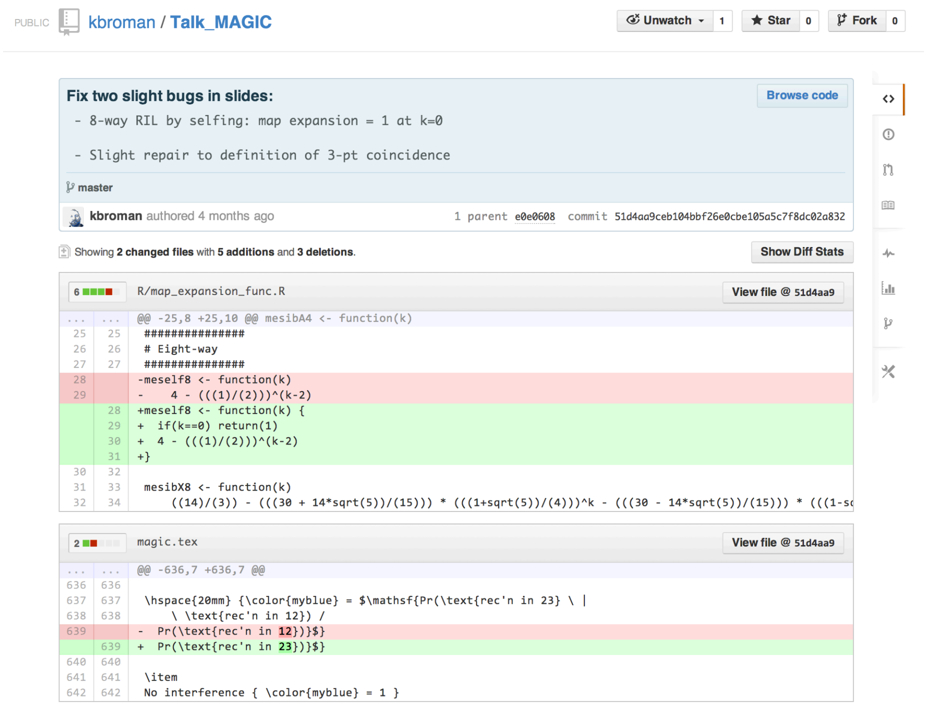
Don't repeat yourself

`kbroman.org/pkg_primer`

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.

8. Use version control (git/GitHub)



Fix two slight bugs in slides:

```
- 8-way RIL by selfing: map expansion = 1 at k=0

- Slight repair to definition of 3-pt coincidence
```

master

kbroman authored 4 months ago   1 parent e0e0608   commit 51d4aa9ceb104bbf26e0cbe105a5c7f8dc02a832

Showing **2 changed files** with **5 additions** and **3 deletions**.

6 R/map_expansion_func.R   View file @ 51d4aa9

```
@@ -25,8 +25,10 @@ mesibA4 <- function(k)
 ###############
 # Eight-way
 ###############
-meself8 <- function(k)
-    4 - (((1)/(2)))^(k-2)
+meself8 <- function(k) {
+  if(k==0) return(1)
+  4 - (((1)/(2)))^(k-2)
+}

 mesibX8 <- function(k)
     ((14)/(3)) - (((30 + 14*sqrt(5))/(15))) * (((1+sqrt(5))/(4)))^k - (((30 - 14*sqrt(5))/(15))) * (((1-s
```

2 magic.tex   View file @ 51d4aa9

```
@@ -636,7 +636,7 @@

 \hspace{20mm} {\color{myblue} = $\mathsf{Pr(\text{rec'n in 23} \ |
     \ \text{rec'n in 12}) /
-  Pr(\text{rec'n in 12})}$}
+  Pr(\text{rec'n in 23})}$}

 \item
 No interference { \color{myblue} = 1 }
```

kbroman.org/github_tutorial

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.

9. 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.

Summary

1. Arrange data to ease analysis
2. Organize your data and code
3. Everything with a script
4. Automate the process
5. Turn scripts into reproducible reports
6. Turn repeated code into functions
7. Create a package/module
8. Use version control
9. License your software

It's always good to include a summary.

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.

Slides: `bit.ly/StPaul-0`

`kbroman.org`

`github.com/kbroman`

`@kwbroman`

Here's where you can find me, as well as the materials for this workshop.