R for Reproducible Research

Philip Dixon

Department of Statistics, Iowa State University

Lunchinators, 22 May 2020

R for reproducible research

- A discussion of some R packages I've recently learnt about
- Goal: Simplify doing reproducible research
- Repeatable/Replicable research:
    - Someone else can repeat your study with new samples and new data
    - At a minimum, from same population
    - Best (if making general claims), from a new population
    - Really stringent test, often very hard
- Reproducible research:
    - Someone else can take your data and description of your methods and recreate your results

R for reproducible research

- This presentation based heavily on:
  - Jenny Bryan's blog post on workflow
    `https://www.tidyverse.org/blog/2017/12/workflow-vs-script/`
  - Anna Krystalli's presentation, 21 Apr 2020
    slides at `bit.ly/r-in-repro-research-dc-leeds`
    Recording at:
    `https://sites.google.com/site/rssleedsbradford/home/2019---2020-session/reproducibility`
- Other presentation, "Is your statistical software correct?"
  - Slides at: `https://mikecroucher.github.io/reproducible_ML/`
  - Take home message: often not.
    - Simple algorithms usually work, Usually not good enough!
    - NAG, Numerical Algorithms Group (UK), high quality algorithms
  - So, document what software you used

Why is reproducible research important?

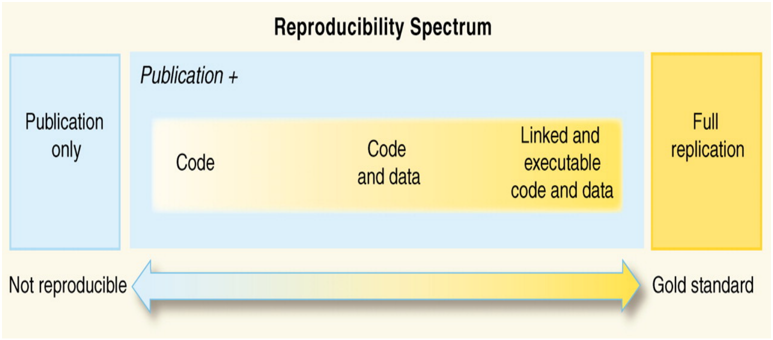
- Far too many papers describe the equipment in excruciating detail (manufacturer, model, address)
- And then the stat methods paragraph (or sentence) is something like: "We used glmer() from the R lme4 package to analyze the data."
- Could someone else recreate the results from that? **NO!**
- What's left out?
- My short list (could be more)
  - What family?
  - Overdispersion?
  - What fixed effects? Interactions?
  - What random effects?
  - What sort of estimates are being reported?
  - How were tests constructed? anova() or using emmeans?

Examples of computing errors affecting results

- From Anna's slides:
  - Cooper et al., 2016, Methods in Ecol and Evol.
    "highlight problems with users jumping straight into software implementations (e.g. in r) that may lack documentation of biases and assumptions that are mentioned in the original papers".
- From Mike's slides
  - Reinhart and Rogoff (2010): Excel error, wrong assessment of role of austerity
  - Ziemann, Eren, El-Osta (2016) Genome Biology 17:177.
    - Excel mangles gene names like Sept2,
    - accession numbers like 2310009E13 $\rightarrow$ 2.31E19
  - Ioannidis, et al, 2010 Nature Genetics. 18 microarray gene expression studies. Only 2 could be fully reproduced
  - Various other examples
  - 2017: 65% of papers indicate reliance on software - has increased over time

So what's a conscientious analyst to do?

- Croucher's law: I will make mistakes
  - corollary: What can I do to minimize this?
- Maximize transparency: document what I did

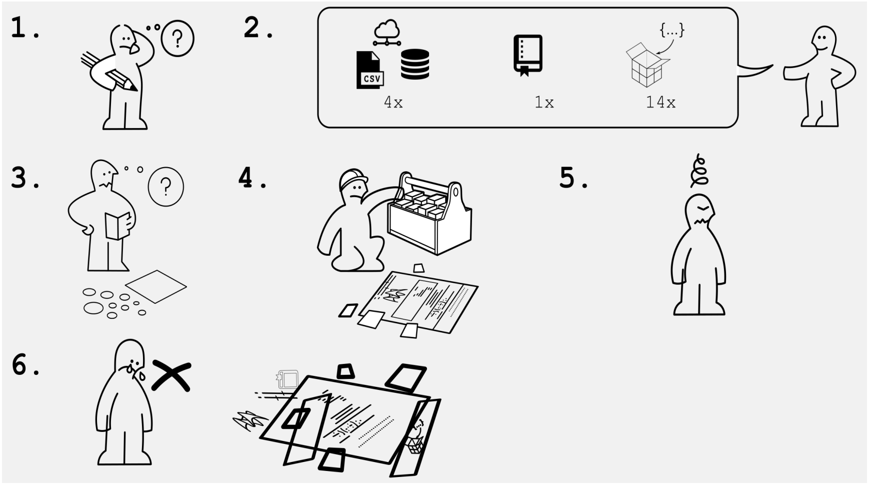


- from Peng, RD, 2011, Science 334:1226-1227

Three reasons for reproducibility

- 1) For yourself. Make it easier to revisit code in the future
  - Submit a manuscript, 3+ months later get reviews. Want revisions
  - Much easier to remember what you did and which files you really need
- 2) Required for journal.
  - provide code to do analysis / create graphs in a paper
- 3) looking for community help
  - Your Q is much clearer if you provide a compact example of the problem.
  - Equally relevant when asking me or CrossValidated, StackExchange, R-sig-ecol

The struggle

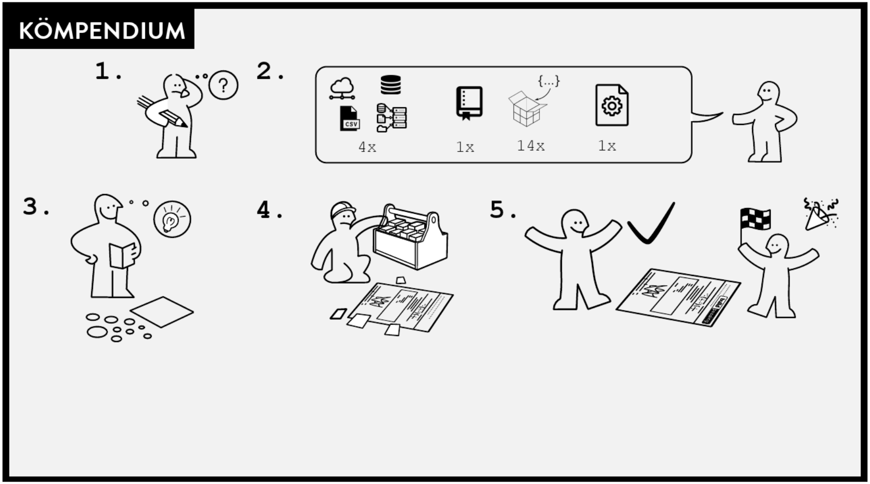


Source: Karthik Ram, 2019 RStudio Conference talk, via Anna Kristalli's slides

R tools to help make code reproducible

- RStudio projects:
  - session starts with in the project folder: setwd() done for you
  - and loads the .Rdata in that folder
  - not completely clean start: previous libraries remain
- RMarkdown document:
  - knitr-ed in a completely clean session
  - If you forget to link a library, you find out quickly
- R Notebooks are similar (Interactive Markdown files)
  - But not as reproducible:
  - Objects can be loaded from your workspace
- My current practice:
  create an RMarkdown document for each manuscript
  - It's a record of the path from data to results (tables, figures)
  - That's what I give to journals that require code and data

The reward



Source: Karthik Ram, 2019 RStudio Conference talk, via Anna Kristalli's slides

Potential issues with RMarkdown and some solutions

- Sometimes an R library links to standalone executables
  - Spatial data: rgdal is the interface to the gdal program
  - Mark recapture: RMark is the interface to Mark
  - Bayesian analysis: rjags, rstan, rstanarm, brms link to JAGS or Stan
- Docker containers bundle everything needed to run an application
  - code, runtime environment, system tools, libraries
- rrtools library
  - provides functions to simplify creating a Docker container
  - Looks like modeled on the usethis library used to create R packages
- Can link with Travis (software testing software)
  - Travis and the R testthat library provide software testing
  - does your code continue to produce the expected results?
- See Anna's slides, 71 et seq for details on Docker and Travis

Research Compendium

- "We introduce the concept of a compendium as both a container for the different elements that make up the document and its computations (i.e. text, code, data, ...), and as a means for distributing, managing and updating the collection."
  Gentleman and Temple Lang, 2007, Statistical Analyses and Reproducible Research J Comp Graph Stat 16(1):1-23

  Original is a 2004 Bioconductor working paper

Research Compendium

- What is the role of a computational paper?
  - It's the advertisement for the compendium
  - "an article about computational result is advertising, not scholarship. The actual scholarship is the full software environment, code and data, that produced the result."

    John Claerbout paraphrased in Buckheit and Donoho, 1995, Proc. Soc. of photo-optical instrumentation engineers (SPIE) 2569(1& 2):540-551
  - Example in Gentleman, 2005, Reproducible research: A bioinformatics case study, Stat Appl in Gen and Mol Bio 4(2)
- Recent example:
  - The paper: Boettiger 2018, From noise to knowledge: how randomness generates novel phenomena and reveals information,

    Ecology Letters, `doi.org/10.1111/ele.13085`
  - The compendium: cboettig/noise-phenomena: Supplement to: "From noise to knowledge: how randomness generates novel phenomena and reveals information"

    `doi.org/10.5281/zenodo.1219780`

RMarkdown isn't practical for long computations: Cache

- You have a simulation that takes 10 hours to run
  - You add code to your Markdown document and re-knitr it
  - Wait 10 hours!
  - because (usually) every code block executed every time you knitr
- Caching results
  - `‘‘‘{r long, cache=TRUE}    Code to run simulation ‘‘‘`
  - The first time you knitr, that code block will execute and the results saved
  - The second time you knitr, the saved results will be grabbed
  - remember, each knitr starts in a clean environment
- If you edit the code in the cached code block,
  the code will be re-executed
- But, cache is not aware of dependencies between code blocks

Illustration of code dependencies

- Code in Markdown cache.Rmd illustrates both cache and dependency

```
‘‘‘{r size}
n <- 10
‘‘‘
‘‘‘{r long, cache=T}
i <- rnorm(n)
‘‘‘
‘‘‘{r result}
print(length(i))
‘‘‘
```

Code dependencies

- Cache fails in this situation
  - Run code once with n=10, i saved as a length 10 vector
  - Change init block to `n <- 20` and knitr
  - i is still length 10
  - No change made to the long block, so that code not rerun
  - Could force cache to work by putting all changeable things inside the long block
  - Not easy to do with large amounts of code

Code dependencies

- Second example: From my own work many years ago
  - Field ecological study: multiple sites, multiple years
  - many data files: most are site/year, some are site, some are year
    - code did computations on each site/year,
    - merged results with site and year information,
    - did summary analyses
  - Added new data files each year; sometimes corrected a old data file.
  - Only want to redo the computations involving the data that changed
  - My 1990's SAS solution was a sheet of paper graphing the data flow
- Does this sound like a job for the UNIX make command? YES!
  - a makefile describes all the steps to produce executable code from many source files
  - make processes the makefile commands
  - only executes the steps that are needed (source newer than object)

The drake library

- drake is a make system for R data analyses
- I found you need to install downloader and visNetwork as well as drake
- Two strategies for package dependencies
  - Install everything, even if you never need it (bloatware)
  - Install packages only if you need it, e.g. visNetwork for `vis_drake_graph()`
- You write a `drake_plan`
  - specifies how to do each major step in an analysis
  - clearest when inline code packaged into functions
  - steps only executed when needed
  - can tell it about dependence on source data files
  - so steps will be re-executed when data changes
- drake stands for "Data Frames in R for Make".
- Will Landau got his PhD from ISU/Statistics (Jarad)

A drake plan

- Describes the steps in your analysis
- Each component specifies:
  - What it needs (the input)
  - What it produces (the output)
- Not a sequential list of operations
  Can be in any order; I find sequential easiest
- Need = because each step is an argument
- Bigger programs will use functions to bundle inline code

```
plan <- drake_plan(
  size =  10,
  long = rnorm(size),
  result = target(length(long))
  )
```

drake plans

- Each line defines a target (size, long, result)
  - and the computation needed to produce it
  - The target() function is implicit if omitted (e.g. for size and long)
- Targets do not appear in your workspace.
  - zipped versions saved in a .drake folder
  - When remake a target, old version still kept
- `https://books.ropensci.org/drake` is an online text

What to do with a drake plan

- Run it: `make(plan)`
  - figures out all the dependencies among components
  - runs what needs to be rerun
    - code or the function has changed
    - an input item has changed
    - and can inform drake about input files, so will rerun if input file changes
  - All hidden from the user
    - Objects saved in a drake folder, not your working directory
  - Version control automatically enabled
    - Can recover "old" versions if necessary
- Print out an item: `readd(result)`
- Load an item into the workspace: `loadd(result)`
  - Object in workspace now called by its drake name (e.g. result)

Really nifty things to do with a drake plan

- graph the dependencies between objects: `vis_drake_graph(plan)`
  - Draws the dependencies between objects
  - Colors by whether up-to-date, or needs re-executing
  - Demonstration with planB
  - Includes data files when those files are registered (see below)
- Tell drake about input files: `file_in('name')`
  - add `file_in()` to add that file to drake's list of tracked objects
  - now, any change to the input file will trigger re-execution
  - Illustrated with planC
- List outdated objects: `outdated()`
- List history of all objects: `drake_history()`
  - Searches through drakes cache and summarizes everything
  - Could use this to restore earlier objects

Practical drake plans

- Use functions to bundle the steps for a target

```
plan <- drake_plan(
  data1 = read_data(file_in("data1.csv")),
  mungedata1 = munge_data(data1),
  data2 = read_data(file_in("data2.csv")),
  mungedata2 = munge_data(data2),
  mergedata = merge_data(mungedata1, mungedata2),
  analysis = analyze_data(mergedata),
  report = rmarkdown::render(
    knitr_in("report.Rmd"),
    output_file = file_out("report.html"),
    quiet = TRUE )
  )
```

Practical drake plans

- Each step runs a function to convert input stuff to a target
  - You write that function, or
  - use `fn <- code_to_function('filename.r')` to create a function from code in a file
- Can run Rmarkdown inside a drake plan
  - The knitr button is a shortcut to the render function in the rmarkdown library
  - Can add options to transform data or create more than one target from one line
- Will Landau runs a lot of simulations, so he made it easy
- Commonly, need to evaluate all combinations of 2 or more factors
- Each combination is one target
  - Could write each target individually
  - or use target(, transform=map()) or transform=cross() to repeat for each combination
- Look up static branching, e.g. in the drake book, for more information

More you can do with drake

- drake has lots of commands and options
- I've only summarized the basics
- lots of documentation: `https://books.ropensci.org/drake/`
- Also has memory management for huge objects
- Can parallelize easily: `make(    , jobs=4)` where number is the number of cores
    - library parallel (for Windows, I believe also works on MacOS)
    - detectCores()

Most important drake functions

From the overview document

- `drake_plan()`: create a workflow data frame (like `my_plan`).
- `make()`: build your project.
- `drake_history()`: show what you built, when you built it, and the function arguments you used.
- `r_make()`: launch a fresh callr::r() process to build your project. Called from an interactive R session, `r_make()` is more reproducible than make().
- `loadd()`: load one or more built targets into your R session.
- `readd()`: read and return a built target.
- `vis_drake_graph()`: show an interactive visual network representation of your workflow.

Most important drake functions, continued

- `recoverable()`: Which targets can we salvage using make(recover = TRUE) (experimental).
- `outdated()`: see which targets will be built in the next make().
- `deps_code()`: check the dependencies of a command or function.
- `drake_failed()`: list the targets that failed to build in the last make().
- `diagnose()`: return the full context of a build, including errors, warnings, and messages.

What about goal 3: getting help, the reprex package

- Most discussion forums want a small, complete, reproducible example that demonstrates the issue
- reprex checks that a block of code is self-contained
  - uses markdown as the engine
  - but simpler to use
  - grabs code from the clipboard (by default) returns output to the clipboard
- demonstration:

```
n <- 10
i <- rnorm(n)
print(length(i))
```

- copy to clipboard
- reprex()

reprex to process console sessions

- Many folks enter code in a program window
  - execute by ctrl-enter
- Others just type “one-off” commands in the console window
- What if you (later) decide to create a program
- reprex can extract code from a console session
- To create clean R code from, e.g., a console session:
  - highlight the code in the console window
  - reprex(venue='r')
  - paste the clean R code into a program window
- can specify input= and outfile= to read/write to files

Other packages I've heard about recently

- beepr: one function, beep() plays a sound or audio file
  - Especially useful to announce the end of a long computation
  - sounds 4 and 6 are especially fun
  - beep()
  - beep(4)
  - beep(6)
- rticles: custom Rmd templates for various journals
- citr: RStudio add in for bibtex citations
- flipbookr: creates flipbooks illustrating individual items of code
  - But, looks like a pain to set up
- redoc: on Github not CRAN
  - Conversion between Markdown and Word **and back**
  - Edits to a word document copied back to the Markdown file
  - See Anna's slide 36

Other packages I've heard about recently

- renv: saves specific package versions
  - Most packages are actively maintained
  - An update may break your code
  - Replaces Packrat
  - Both archive "old" versions of packages
  - Creates a project-specific set of local libraries
  - My experience is that some functions, e.g., update(), clash with modeling functions
- here: one function, here()
  - Provides a file path to its best guess about the root directory
  - Not a project: the current working directory
  - A project: the project top directory
  - specify a lower folder or specific file as arguments
  - here('my data.csv')

And packages for statistical analysis

- CUB: ordinal data
  - response is a mixture of “feeling”: preference for a category and nearby ones
  - and “uncertainty”: discrete uniform over all categories
  - mixture proportion and “feeling” can depend on individual characteristics
  - Maximum likelihood estimates
- brms: interface to Stan for Bayesian analysis
  - can specify models using R formula notation
  - like rstanarm but larger range of models
  - compiles your model into C++ code, takes ca 60 seconds
- ctmm: continuous time movement models
  - locations of critters measured frequently
  - estimates home range and related quantities
  - accounts for autocorrelation in locations

Questions / comments?