Reproducible Research - Notes

Tanner Prestegard

Course taken from 5/4/2015 - 5/31/2015

Reproducible research: concepts and ideas

- Replication - the ultimate standard for strengthening scientific evidence is replication of findings and conducting studies with independent investigators, data, analytical methods, laboratories, and instruments.
- Replication is particularly important in studies that can impact broad policy or regulatory decisions.
- What's wrong with replication?
  - Some studies can't be replicated due to time, money, or opportunity constraints.
  - Some studies are just unique and the conditions can't be replicated.
- Reproducible research: make analytic data and code available so that others may reproduce your findings.
- Why do we need reproducible research?
  - New technologies increasing data collection throughput; data are more complex and high-dimensional.
  - Existing databases can be merged into new "megadatabases."
  - Computing power is greatly increase, allowing more sophisticated analyses.
  - For every field "X" there is now a field "Computational X".
- What do we need for reproducible research?
  - Analytic data are available.
  - Analytic code is available.
  - Documentation of code and data.
  - Standard means of distribution.
- Who are the players?
  - Authors
    * Want to make their research reproducible.
    * Want tools for reproducible research to make their lives easier.
  - Readers
    * Want to reproduce and perhaps expand upon interesting findings.
    * Want tools for reproducible research to make their lives easier.
- Challenges

- Authors must undergo considerable effort to make their data/results available on the web.
  - Readers must download data/results individually and piece together which data go with which code sections, etc.
  - Readers may not have the same resources as authors.
  - Few tools to help authors/readers, although the toolbox is growing.

- In reality, what happens is:
  - Authors
    - Just put stuff up on the web.
    - Journal supplementary materials.
    - There are some central databases for various fields.
  - Readers
    - Just download the data and try to figure it out.
    - Piece together the software and try to run it.

- Idea: literate statistic programming
  - An article is a stream of text and code.
  - Analysis code is divided into text and code "chunks."
  - Each code chunk loads data and computes results.
  - Presentation code formats results into tables, figures, etc.
  - Article text explains what is going on.
  - Literate programs can be **weaved** to produce human-readable documents and **tangled** to produce machine-readable documents.

- Literate programming is a concept that requires:
  - A documentation language (human readable).
  - A programming language (machine readable).

- Sweave uses LaTeX and R as the documentation and programming languages, respectively.
  - Main website: http://www.statistik.lmu.de/~leisch/Sweave
  - Sweave limitations:
    - Focused on LaTeX, which is difficult to learn.
    - Lacks features like caching, multiple plots per chunk, mixing programming languages, etc.
    - Not frequently updated or very actively developed.

- The `knitr` package for R is an alternative for literate programming.
  - Uses R as the programming language (although others are allowed) and a variety of documentation languages, including LaTeX, Markdown, and HTML.

- Reproducible research is important as a minimum standard, particularly for studies that are difficult to replicate.

- Infrastructure is needed for creating and distributed reproducible documents, beyond what is currently available.

- There are a growing number of tools for creating reproducible documents.

Structure of a data analysis

- Steps in a data analysis
  - Define the question.
  - Define the ideal data set.
  - Determine what data you can access.
  - Obtain the data.
  - Clean the data.
  - Exploratory data analysis.
  - Statistical prediction/modeling.
  - Interpret results.
  - Challenge results.
  - Synthesize/write up results.
  - Create reproducible code.
- Defining a question
  - The way you define your questions is extremely important!
  - It's the most useful "dimension reduction" tool that you can employ.
  - Example:
    - General question: can I automatically detect emails that are spam and those that are not?
    - Make it concrete: can I use quantitative characteristics of the emails to classify them as spam?
- Define the ideal data set
  - The data set may depend on your goal.
    - Descriptive: a whole population.
    - Exploratory: a random sample with many variables measured.
    - Inferential: the right population, randomly sampled.
    - Predictive: a training and test data set from the same population.
    - Causal: data from a randomized study.
    - Mechanistic: data about all components of the system.
- Determine what data you can access
  - Sometimes you can find data for free on the web.
  - Other times you may need to buy the dta.
  - Be sure to respect the terms of use.
  - If the data don't exist, you may need to generate them yourself.
- Obtain the data
  - Try to obtain the raw data.
  - Be sure to reference the source.
  - Polite emails go a long way.
  - If you will load the data from an internet source, record the URL and the time you accessed the data.

- Clean the data
  - Raw data often needs to be processed.
  - If it is pre-processed, make sure that you understand how it was pre-processed.
  - Understand the source of the data (census, sample, convenience sample, etc.).
  - May need reformatting, subsampling, etc. Record these steps!
  - Determine if the data are good enough - if not, get new data or quit.
- Exploratory data analysis
  - Look at summaries of the data.
  - Check for missing data.
  - Create exploratory plots.
  - Perform exploratory analyses like clustering.
- Statistical prediction/modeling
  - Should be informed by the results of your exploratory analysis.
  - Exact methods depend on the question of interest.
  - Transformations/processing should be accounted for when necessary.
  - Measures of uncertainty should be reported.
- Interpret results
  - Use the appropriate language - words like "describe," "correlates with/associated with," "leads to/causes," "predicts."
  - Give an explanation.
  - Interpret coefficients.
  - Interpret measures of uncertainty.
- Challenge results
  - Challenge all steps: question, data source, processing, analysis, conclusions.
  - Challenge measures of uncertainty.
  - Challenge choices of terms to include in models.
  - Think of potential alternative analyses.
- Synthesize/write up results
  - Lead with the question.
  - Summarize the analyses into the story.
  - Don't include every analysis, only include it if:
    * It is needed for the story.
    * If it is needed to address a challenge.
  - Order analyses according to the story, rather than chronologically.
  - Include "pretty" figures that contribute to the story.
- Create reproducible code.
  - Use Markdown, knitr, etc. to document your code and your analysis.

Organizing your analysis

- Data analysis files
  - Data: raw data, processed data.
  - Figures: exploratory figures, final figures.
  - R code: raw/unused scripts, final scripts, R Markdown files.
  - Text: README files, text of analysis/report.
- Raw data
  - Should be stored in your analysis folder.
  - If accessed from the web, include URL, description, and date accessed in a README file.
- Processed data
  - Should be named so it's easy to see which script generated the data.
  - The processing script -> processed data mapping should be described in the README file.
  - Processed data should be **tidy**.
- Exploratory figures
  - Made during the course of your analysis, not necessarily part of your final report.
  - They do not need to be "pretty."
- Final figures
  - Usually a small subset of the original figures.
  - Axes/colors set to make the figure clear.
  - Possibly multiple panels.
- Raw scripts
  - May be less commented.
  - May be multiple versions.
  - May include analyses that are later discarded.
- Final scripts
  - Clearly commented.
    - Small comments used liberally - what, when, why, how.
    - Bigger commented blocks for whole sections.
  - Include processing details.
  - Only analyses that appear in the final write-up.
- R markdown files
  - Can be used to generate reproducible reports.
  - Text and R code are integrated.
  - Very easy to create in RStudio
- README files

- Not necessary if you use R markdown.
- Should contain step-by-step instructions for analysis.
- Example: https://github.com/jtleek/swfdr/blob/master/README.md

- Text of the document
  - Should include a title, introduction (motivation), methods (statistics you used), results (including measures of uncertainty), and conclusions (including potential problems).
  - It should tell a coherent story.
  - It should not include every analysis that you performed.
  - References should be included for statistical methods.

Coding standards in R

- Always use text files and a text editor.
- Indent your code.
  - Indenting improves readability.
  - Suggested to use 4-8 spaces for indents.
- Limit the width of your code (80 columns or so).
- Limit the length of individual functions.

Markdown

- Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid XHTML or HTML.
- Syntax:
  - Italics: *text*
  - Bold: **text**
  - Headings:
    - # Primary heading
    - ## Secondary heading
    - ### Tertiary heading
  - Unordered lists (can use characters other than "-"):
    - - first item
    - - second item
    - - third item
  - Ordered lists (don't have to be in order, Markdown will automatically put them in order):
    - 1. first item
    - 2. second item
    - 3. third item
  - Links (two methods):
    - [Text](URL)
    - [Text][1], then at the bottom put [1]: URL "Text"
  - Newlines: double space after the end of a line.
- Resources: The Official Markdown Guide (daringfireball.net/projects/markdown/basics)

R Markdown

- Markdown is a simplified version of "markup" languages.
- Allows you to focus on writing as opposed to formatting.
- Simple/minimal intuitive formatting elements.
- Easily converted to valid HTML (and other formats) using existing tools.
- What is R Markdown?
  - The integration of R code with markdown.
  - Allows you to create documents containing "live" R code.
  - R code is evaluated as part of the processing of the markdown.
  - Results from R code are inserted into markdown document.
    * You know that the code in the document will work, because it HAD to work in order to produce the document!
  - A core tool in literate statistical programming.
  - R Markdown can be converted to standard markdown using the `knitr` package in R.
  - Markdown can be converted to HTML using the `markdown` package in R.
  - Any basic text editor can be used to create a markdown document; no special editing tools needed.
  - The R Markdown -> Markdown -> HTML work flow can be easily managed using RStudio.
  - Can convert R Markdown to slides using the `slidify` package.

The `knitr` package

- How do I make my work reproducible?
  - Decide to do it! (ideally from the start)
  - Keep track of things, perhaps with a version control system to track snapshots/changes.
  - Use software whose operation can be coded (i.e, not GUI-based programming).
  - Don't save output like temporary data transformations, pre-processing, etc.
    * Can provide raw data and pre-processing code.
  - Save data in non-proprietary formats.
- Literate programming
  - Pros:
    * Text and code all in one place, in logical order.
    * Data and results are automatically updated to reflect external changes.
    * Code is live - automatic "regression test" when building a document.
  - Cons:
    * Text and code all in one place; can make documents difficult to read, especially if there is a lot of code.
    * Can substantially slow down processing of documents (although there are tools to help).
- What is `knitr`?
  - An R package (available on CRAN) that supports RMarkdown, LaTeX, and HTML as documentation languages.

- Can export to PDF and HTML.
- Built right into RStudio for your convenience.
- Requirements:
  * A recent version of R.
  * A text editor.
  * Some support packages that are available on CRAN.
  * Some knowledge of Markdown, LaTeX, or HTML (we will use Markdown).

- What is `knitr` good for?
  - Manuals.
  - Short/medium-length technical documents.
  - Tutorials.
  - Reports (especially if generated periodically).
  - Data pre-processing documents/summaries.

- What is `knitr` not good for?
  - Very long research articles.
  - Complex, time-consuming computations.
  - Documents that require precise formatting.

- To run `knitr` in R (not RStudio):
  - ```
    library(knitr)
    setwd("dir_name")
    knit2html("document.Rmd")
    browseURL("document.html")
    ```

- A few notes:
  - `knitr` will fill a new document with filler text - delete it!
  - Code chunks begin with ```` ```{r} ```` and end with ```` ``` ````.
  - All R code goes in between these markers.
  - Code chunks can have names, which is useful when we start making graphics.
    * ````
      ```{r firstchunk, echo=FALSE}
      ## R code goes here
      ```
      ````
    * `echo=FALSE` means that the code won't be echoed in the output document, only the result will be.
    * Set `results="hide"` to hide the results.
  - By default, code in a code chunk is echoed, as are any results of a computation.
  - Don't edit or save the .md or .html documents produced by `knitr` until you are finished!
  - Can add code directly into a sentence.
    * Example: `The current time is `r time`. My favorite random number is `r rand`.`

- Adjust figure height: ```` ```{r scatterplot, fig.height=4} ````

- `knitr` embeds the figures in the HTML.
- Making tables with `xtable`:

  - ```
    ```{r fitmodel}
    library(datasets)
    data(airquality)
    fit <- lm(Ozone ~ Wind + Temp + Solar.R, data = airquality)
    ```
    Here is a table of regression coefficients.
    ```{r showtable, results="asis"}
    library(xtable)
    xt <- xtable(summary(fit))
    print(xt, type="html")
    ```
    ```

- Setting global options (for the entire document).

  - For example, we may want to suppress all code echoing and results output.
  - To do this, create a separate code chunk at the beginning of the document and use the `opts_chunk` function.
  - ```
    ```{r setoptions, echo=FALSE}
    opts_chunk$set(echo = FALSE, results = "hide")
    ```
    ```
  - You can override the global options on a chunk-by-chunk basis.
  - Some common options:
    - `results: "asis", "hide"`
    - `echo: TRUE, FALSE`
    - `fig.height: numeric`
    - `fig.width: numeric`

- Caching computations

  - What if one chunk takes a long time to run?
  - All chunks have to be re-computed every time you re-knit the file.
  - The `cache=TRUE` option can be set on a chunk-by-chunk basis to store results of a computation.
  - After the first run, the results for that chunk will be loaded from a cache (as long as nothing has changed).

- Caching caveats

  - If the data, code, or anything external changes, you have to re-run the cached code chunks.
  - Dependencies are not checked explicitly.
  - Chunks with significant side effects may not be cacheable (for example, if the code has some effect outside of the document).

## Communicating results

- Hierarchy of information: research paper
  - Title/author list
  - Abstract
  - Body/results
  - Supplementary materials/gory details
  - Code/data/really gory details
- Hierarchy of information: e-mail presentation
  - Subject line/sender information
    * At a minimum, at least include one.
    * Can you summarize your findings in one sentence?
  - E-mail body
    * A description of the problem in 1-2 paragraphs.
    * If action needs to be taken as a result of this presentation, suggest some options and make them as concrete as possible.
    * If questions need to be addressed, try to make them yes/no questions.
  - Attachments
    * R Markdown file, knitr report, etc.
    * Stay concise, don't spit out pages of code.
  - Links to supplementary materials
    * Code/software/data
    * GitHub repository/project website

## RPubs

- RPubs.com, brought to you by RStudio.
- Easy web publishing from R, useful to share with other people or with the general public.
- Need to create an account.
- Can publish your documents directly from RStudio.

## Reproducible research checklist

- DO: start with good science.
  - Garbage in = garbage out.
  - A coherent, focused question simplifies many problems.
  - Working with good collaborators reinforces good practices.
  - Something that's interesting to you will motivate good habits.
- DON'T: do things by hand.
  - Editing spreadsheets of data to clean it up.
  - Editing tables or figures (rounding, formatting, etc.).
  - Downloading data from a website by clicking links in a web browser.

  - Moving data around on your computer, splitting/reformatting data files.
  - "We're just going to do this once..."
  - Things done by hand need to be precisely documented.

- DON'T: point and click.
  - Many data processing/statistical analysis packages have GUIs.
  - GUIs are convenient/intuitive, but the actions you take in them can be difficult to reproduce.
  - Some GUIs produce a log file or script which includes equivalent commands; these can be saved for later examination.
  - In general, be careful with data analysis software that is highly interactive; ease of use can sometimes lead to non-reproducible results.
  - Other interactive software, like text editors, are usually fine.

- DO: teach a computer.
  - If something needs to be done as a part of your analysis/investigation, try to automate it.
  - In order to give your computer instructions, you need to write down exactly what you want to do and how it should be done.
  - Teaching a computer almost guarantees reproducibility.

- DO: use version control.
  - Slow things down.
  - Add changes in small chunks (don't just do one massive commit).
  - Track/tag snapshots; revert to old versions.
  - Software like GitHub, BitBucket, or SourceForge make it easy to publish results.

- DO: keep track of your software environment.
  - If you work on a complex project involving many tools/datasets, the software and computing environment can be critical for reproducing your analysis.
  - Computer architecture: CPU, GPUs.
  - Operating system: Windows, Mac, Linux/Unix.
  - Software toolchain: compilers, interpreters, command shell, programming languages, database backends, data analysis software.
  - Supporting software/infrastructure: libraries, packages, dependencies.
  - External dependencies: web sites, data repositories, remote databases, software repositories.
  - Version numbers: ideally, for everything (if available).
  - Can use R function `sessionInfo()` to get a lot of this information.

- DON'T: save output.
  - Avoid saving data analysis output (tables, figures, processed data, etc.) except temporarily for efficiency purposes.
  - If a stray output file cannot easily be connected with the means by which it was created, then it is not reproducible.
  - Save the data and code that generated the output rather than the output itself.
  - Intermediate files are OK as long as there is clear documentation of how they were created.

- DO: set your seed.
  - Random number generators generate pseudo-random numbers based on an initial seed.
    * In R, you can use the `set.seed()` function to set the seed and specify the random number generator to use.
  - Setting the seed allows for the stream of random numbers to be exactly reproducible.
  - Whenever you generate random numbers for a non-trivial purpose, always set the seed!
- DO: think about the entire pipeline.
  - Data analysis is a lengthy process; it is not just tables/figures/reports.
  - Raw data -> processed data -> analysis -> report.
  - How you got the data is just as important as the end result.
  - The more of the data analysis pipeline you can make reproducible, the better for everyone.

## Evidence-based data analysis

- Replication
  - Focuses on the validity of a scientific claim - is this claim true?
  - The ultimate standard for strengthening scientific evidence.
  - New investigators, data analytical methods, laboratories, instruments, etc.
  - Particularly important in studies that can impact broad policy or regulatory decisions.
- Reproducibility
  - Focuses on the validity of the data analysis - can we trust this analysis?
  - Arguably a minimum standard for any scientific study.
  - New investigators, same data, same methods.
  - Important when replication is impossible.
- Background and underlying trends
  - Some studies cannot be replicated due to lack of money, time, or opportunity.
  - Technology is increasing data collection throughput; data are more complex and high-dimensional.
  - Existing databases can be merged to become bigger databases.
  - Computing power allows more sophisticated analyses, even on "small" data.
  - For every field "X", there is a "Computational X".
- The result?
  - Even basic analyses are difficult to describe.
  - Heavy computational requirements are thrust upon people without adequate training in statistics and computing.
  - Errors are more easily introduced into long analysis pipelines.
  - Knowledge transfer is inhibited.
  - Results are difficult to replicate or reproduce.
  - Complicated analyses can be hard to trust.
- What problem does reproducibility solve?

- What we get:
    * Transparency
    * Data availability
    * Software/methods availability
    * Improved transfer of knowledge
- What we do not get:
    * Validity/correctness of the analysis.
- An analysis can be reproducible but still WRONG.
- We really want to know if we can trust an analysis.
- Does requiring reproducibility deter bad analyses?

- Problems with reproducibility
  - The premise of reproducible research is that with data/code available, people can check each other and the whole system is self-correcting.
    * This addresses the most "downstream" aspect of the research process - post-publication.
    * Assumes that everyone plays by the same rules and wants to achieve the same goals (scientific discovery).

- Who reproduces research?
  - For reproducibility to be effective as a means to check validity, someone needs to do something!
    * Re-run the analysis, check that results match.
    * Check the code for bugs/errors.
    * Try alternative approaches, check sensitivity.
  - The need for someone to do something is inherited from the traditional notion of replication.
  - Who is "someone" and what are their goals?

- The story so far:
  - Reproducibility brings transparency and increased transfer of knowledge.
  - A lot of discussion about how to get people to share data.
  - Key question of "can we trust this analysis?" is not addressed by reproducibility.
  - Reproducibility addresses potential problems long after they've occurred ("downstream").
  - Secondary analyses are inevitably colored by the interests/motivations of others.

- Evidence-based data analysis
  - Most data analyses involve stringing together many different tools and methods.
  - Some methods may be standard for a given field, but often others are applied ad hoc.
  - We should apply thoroughly studied methods that are mutually agreed upon to analyze data whenever possible.
  - There should be evidence to justify the application of a given method.
  - Create analytic pipelines from evidence-based components - standardize it.
  - Once an evidence-based analytic pipeline is established, we shouldn't mess with it.
  - Analysis with a "transparent box."
  - Reduce the "researcher degrees of freedom."
  - Analogous to a pre-specified clinical trial protocol.

## Caching computations

- The `cacher` package - add-on package for R
- Evaluates code written in files and stores intermediate results in a key-value database.
- R expressions are given SHA-1 hash values so that changes can be tracked and code reevaluated if necessary.
- "Cacher packages" can be built for distribution.
- Others can "clone" an analysis and evaluate subsets of code or inspect data objects.
- Using `cacher` as an author
  - The `cachepackage` function creates a `cacher` package storing the source file, cached data objects, and metadata.
  - The file is zipped and can be distributed.
  - Readers can unzip the file and immediately investigate its contents using the `cacher` package.
- Using `cacher` as a reader
  - `library(cacher)`
  - `clonecache(id=sha1_string)`
  - `showfiles()`
- Cloning an analysis
  - Local directories are created.
  - Source code files and metadata are downloaded.
  - Data objects are not downloaded by default.
  - References to data objects are loaded and corresponding data can be "lazy-loaded" on demand.
- Tracing code backwards: `objectcode("objectname")`. (must be in quotes)
- Running code
  - The `runcode` function executes code in the source file.
  - By defaults, expressions that result in an object being created are not run, and the resulting objects are lazy-loaded into the workspace.
  - Expressions not resulting in objects are evaluated.
- Checking code and objects
  - The `checkcode` function evaluates all expressions from scratch (no lazy-loading).
  - Results of the evaluation are checked against stored results to see if the results are the same as what the author calculated.
    - Setting RNG seeds is critical for this to work.
  - The integrity of the data objects can be verified with the `checkobjects` function to check for possible data corruption.