Reproducible guide

Draft

Anthony Davidson

2022-02-14

Contents

Chapter 1

Overview

This guide is really a combination of reproducible resources for graduate research and beyond using RMarkdown through RStudio.

Welcome!

This is a minimal example of a book based on R Markdown and **bookdown** (https://github.com/rstudio/bookdown).

This template provides a skeleton file structure that you can edit to create your book.

The contents inside the .Rmd files provide some pointers to help you get started, but feel free to also delete the content in each file and start fresh.

Additional resources:

The **bookdown** book: https://bookdown.org/yihui/bookdown/

The **bookdown** package reference site: https://pkgs.rstudio.com/bookdown

If you prefer text as the link instead of a numbered reference use: any text you want can go here.

1.1 Working with this resource

1.2 Description

Using tools like git for version control and knitr for dynamic figure generation are great steps forward towards better research transparency and reproducibility.

But there are also steps to be made in improving code and preparing data for re-use. These steps don't necessarily involve the use of new tools, but instead would be the result of applying "best practice" guidelines.

We aim to begin compiling resources here. This is only a start, and we'd encourage anyone to suggest further guidelines, particularly those that are specific to various statistical software packages.

1.3 Guidelines for writing code

Very useful manual:

Gentzkow and Shapiro, "Code and Data for the Social Sciences: A Practitioner's Guide" (including section "RA Manual: Notes on Writing Code.") 2014

We'd recommend reading the full guide, but briefly outline the sections of "RA Manual: Notes on Writing Code" here so as to give a preview. We also include one example from each section, but there are many other examples in the original paper.

1.3.0.0.0.1 Code should be logical

- Despise redundancy
- Separate functional code and metadata
- Use the right data structures
- Make your functions shy
- Use overriding methods instead of switches where appropriate

Example: use the right data structures

Instead of writing a function like:

```
myfunction <- function(name, surname, age, height) {
  if (age < 18) {}
}
```

We can use a structure like an object or array for that.

```
myfunction <- function(person) {
  if (person.age < 18) { }
}
```

This would make the code easier to read and maintain.

1.3.0.0.0.2 Code should be readable

- Keep it short
- Order your functions for linear reading
- Choose descriptive names

- Use white space and indents to make code scannable
- Pay special attention to coding algebra
- Make logical switches intuitive
- Use enough comments and no more
- Be consistent
- Avoid commands that make code hard to read

Example: choose descriptive names

It might be easier for other users to read and adapt existing code if the variables, folders, classes and other elements have simple and intuitive names. So prefer writing code like:

```
calculate_status <- function(person) {}
```

Rather than:

```
cs <- function(p) {}
```

1.3.0.0.0.3 Code should be robust

- Check for errors
- Write tests

Example: check for errors

Users can use your code with different parameters and in different environments. It is a good practice to include code that check for erroneous values and provide clear feedback.

```
myfunction <- function(x, y) {
  return x / y;
}

myfunction <- function(x, y) {
  if (y == 0)
    return 'y must be different than zero!';
  return x / y;
}
```

Of course you can also write tests to make sure your verifications are working correctly ;-)

1.3.0.0.0.4 Code should be efficient

- Profile slow code relentlessly
- Store "too much" output from slow code
- Separate slow code from fast code

1.4 Packages and libraries needed:

Chapter 2

Introduction

Modern challenges of reproducibility in research, particularly computational reproducibility [cite], have produced a lot of discussion in papers, blogs and videos, some of which are listed here.

2.1 BES guidelines

In 2017 The britich ECological Society produced a set of reproduciblity guidelines here.

[pdf and notes coming....]

2.2 What's a reproducible report?

For the purposes of this guide, a report is a scientific document that contains not only the text that makes up the manuscript, but also the code that generates the figures and the statistics that are reported in your manuscript. Ideally, the report is part of a self-contained project that may contain your data, your initial exploratory analyses, the final product, and the code needed to generate them.

This manuscript can be a scientific article, a conference presentation, a technical report, or a document to share your progress with your collaborators. The end product may not show any code and therefore it may not look like it was generated differently from other documents.

Typically a report contains code for data manipulation, data analysis, and figure generation alongside the text that constitutes the heart of the report. Because of this hybrid nature, if left unchecked, this mix can lead to a big mess that can be difficult to maintain and debug. In this guide, we will provide you with some advice on how to keep your report manageable.

Box xx: What is the difference between repeatability and reproducibility?

Repeatability describes how close are the results of an experiment conducted under the same conditions (same instruments, same operators, etc.). **Reproducibility** describes how close are the results of an experiment conducted under similar but different conditions. Repeatability ensures that you would obtain similar results when running your code on your own laptop at different times; while reproducibility ensures that giving your code to someone else would allow them to obtain the same results as yours.

2.3 Why a reproducible report?

Did you ever have to redo an analysis 6 months later, and it was difficult. You forgot which one of the 15 files with "final" in their names was really the one you should have used? Have you ever spent several hours assembling an intricate figure with your favorite drawing program, just to realize that your collaborators had forgotten to send you the latest batch of data? Writing a reproducible report alleviates some of these hurdles. By automating how the figures and the statistics in your report are generated, you are leaving a code trail that you, your collaborators, or your readers can take, and that leads to your original data. This path to the raw data increases the transparency of your science. However, in order for the six-month-in-the-future you, your collaborators, and your readers, to be able to take this path, it is important that you organize your code and your data files consistently.

Not only does writing a reproducible report increase the transparency of your science, it reduces the mistakes that result from copying and pasting across software. Keeping the content of your manuscript in sync with the output of your statistical program is challenging. By specifying directly the output of your model in your text, it is easier to make sure you are referring to the correct model with the correct parameters. To be the devil's advocate, one could argue that the additional code that will need to be written to integrate the results within the text could lead to additional errors. However, these bugs are possible to detect (contrary to mistakes done by copying and pasting the correct numbers), and their consequences can be assessed by re-running the code generating the manuscript after fixing them.

Writing a reproducible report allows you to tell a much richer story than the narrative in the report by itself does. The text in your report does not usually show the different approaches and analyses you have tried before coming up with the final results. With a reproducible report, you can provide readers who want to know more about how you obtained the results in your paper, the approaches you tried and the their results. These can be included as supplementary material or tagged in the history of your version control system.

To make your report reproducible, your code will need to be self-contained. As a consequence, you will be able to re-use the code you wrote for one project in

another one. Therefore, if initially it might slow you down to make your code reproducible, it is an investment in the future as you will be able to re-use and build upon it in the future. Additionally, others might be able to also re-use your code, and apply it for their own data. Your efforts may speed up the overall scientific process (you or your colleagues won't need to re-invent the wheel for each project), and you could get more citations on your papers.

It can feel daunting to get started with writing a reproducible report because of the technical skills and knowledge required. However, a partially reproducible report is better than a non-reproducible one. So each step you take towards reproducibility is worth taking, and sets you up to take the next one for the next project.

2.3.1 How to deal with citations?

You need two things: a BibTeX file that contains all the citations you use in your manuscript and a CSL (Citation Style Language) file that specifies the format of your citation. Software citation managers such as Zotero or Mendeley provide options to generate BibTeX files for your citations. CSL files exist for most journals, and can be downloaded from: https://www.zotero.org/styles. This is a convenient search interface provided by Zotero but you do not need to use Zotero to download or use these files.

2.4 Where can I find more information?

- The RStudio Markdown website: http://rmarkdown.rstudio.com/
- The bookdown website: https://bookdown.org/yihui/bookdown/

Chapter 3

Documenting and managing dependencies

Reproducibility is also about making sure someone else can re-use your code to obtain the same results as yours. Understanding why your analysis may not lead to the same results on a different computer can be useful to determine how careful you need to be in documenting your setup.

For someone else to be able to reproduce the results included in your report, you need to provide more than the code and the data. You also need to document the exact versions of all the packages, libraries, and software you used, and potentially your operating system as well as your hardware.

R itself is very stable, and the core team of developer takes backward compatibility (that old code works with recent version of R) very seriously. However, default values in some functions have changed, and new functions get introduced regularly. If you wrote your code on a recent version of R and give it to someone who hasn't upgraded recently, they may not be able to run your code. If R itself is stable, the packages are generally much less stable. New functionalities get introduced with each versions, some functions get deprecated, and defaults options change. Code written for one version of a package may produce very different results with a more recent version.

Documenting and managing the dependencies of your project correctly can be a complicated. However, even simple documentation that helps others understand the setup you used can have a big impact. Here we present three levels of complexity to document the dependencies for your projects.

3.1 Show the packages you used

With R, the simplest (but useful and important) approach to document your dependencies is to report the output of `sessionInfo()` (or `devtools::session_info()`). Among other information, this will show all the packages (and their versions) that are loaded in the session you used to run your analysis. If someone wants to recreate your analysis, they will know which packages they will need to install.

3.2 Use packages that help recreate your setup

The `checkpoint` package provides a way to download all the packages at a given date from CRAN. Thus, from the output provided by `sessionInfo()`, they could recreate your setup. It however makes two important assumptions: all your packages were up-to-date with CRAN at the time of your analysis; you were not using packages that are not available from CRAN (e.g. the development version of a package directly from a git repository).

Another approach is to use the `packrat` package. This package creates a library (a collection of packages) directly within your analysis directory. It increases the size of your project as all the source code for the packages is included, but it ensures that someone can recreate more reliably the same environment as the one you used for your analysis. It also makes it easier because the installation of these packages is fully automated for the person wanting to have the same setup.

3.3 Use containers to share your setup

A step further in complexity is to use Docker. With Docker you recreate an entire operating system with all the software, data, and packages needed for your analysis. It is more technical to set up but it allows you to distribute the exact same environment as the one you used. If you want others to be able reproduce your results, and your analysis depends on software that can be difficult to install, it is an option that might be worth exploring.

Original content from “Althea ArchMiller”

Chapter 4

Methods

You can add parts to organize one or more book chapters together. Parts can be inserted at the top of an .Rmd file, before the first-level chapter heading in that same file.

Add a numbered part: `# (PART) Act one {-}` (followed by `# A chapter`)

Add an unnumbered part: `# (PART\*) Act one {-}` (followed by `# A chapter`)

Add an appendix as a special kind of un-numbered part: `# (APPENDIX) Other stuff {-}` (followed by `# A chapter`). Chapters in an appendix are prepended with letters instead of numbers.

Chapter 5

Footnotes and citations

5.1 Footnotes

Footnotes are put inside the square brackets after a caret `^[]`. Like this one [1].

5.2 Citations

Reference items in your bibliography file(s) using `@key`.

For example, we are using the **bookdown** package [Xie, 2021] (check out the last code chunk in index.Rmd to see how this citation key was added) in this sample book, which was built on top of R Markdown and **knitr** [Xie, 2015] (this citation was added manually in an external file book.bib). Note that the `.bib` files need to be listed in the index.Rmd with the YAML `bibliography` key.

The RStudio Visual Markdown Editor can also make it easier to insert citations: https://rstudio.github.io/visual-markdown-editing/#/citations

[1] This is a footnote.

Chapter 6

Blocks

6.1 Equations

Here is an equation.

$$f(k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{6.1}$$

You may refer to using `\@ref(eq:binom)`, like see Equation (6.1).

6.2 Theorems and proofs

Labeled theorems can be referenced in text using `\@ref(thm:tri)`, for example, check out this smart theorem 6.1.

Theorem 6.1. *For a right triangle, if c denotes the* length *of the hypotenuse and a and b denote the lengths of the* ***other*** *two sides, we have*

$$a^2 + b^2 = c^2$$

Read more here https://bookdown.org/yihui/bookdown/markdown-extensions-by-bookdown.html.

6.3 Callout blocks

The R Markdown Cookbook provides more help on how to use custom blocks to design your own callouts: https://bookdown.org/yihui/rmarkdown-cookbook/custom-blocks.html

6.4 github and git

Reproducibility practices for scientific code are sorely needed. Now is the time to come together as a scientific community to decide what works and what doesn't.

One of the best ways to collaborate on a digital project is to use the version control system, git, and have it freely available through an online hosting system, in this case github. Every file that is used to build this guide is on a repository called reproducibility-guide on github.

There are several ways to contribute to this website. You don't have to contribute code. No matter if you are a programmer new to science, or a scientist new to programming, your perspective is needed for the advancement of scientific reproducibility.

6.5 Issues

One way to begin contributing is to start an issue. Issues are how we discuss the project. It can be as large as changing the entire structure of the site, or small, like adding a new tool to the Tools section.

Everyone who watches the project is alerted when an issue has been established, and everyone can comment on it. If you have an idea for making the site better, or just want to start a discussion on reproducibility in sciences, please start a new issue.

If you're not sure where to start, take a look at already started issues. Feel free to voice your opinion on any issue already mentioned, come join the discussion!

If you understand the basics of git, maybe you would like to take action and fix the issue by adding code to the repository.

6.6 Contributing to Codebase

Each collaborative project on github has an ecosystem, so the rules for collaborating are different depending on the project. Often housed in a CONTRIBUTING.md file, there are rules that have been adapted to suit the functionality of the project. For our project, we are flexible on how you contribute and encourage newbie git collaborators: everyone has to start somewhere. Here is a general guide for how you can contribute to the basecode for this site. If you are a beginning git user, this may seem a bit confusing, but we promise it becomes clear after trying it.

6.6.1 Tiny Tutorial on Contributing to Repositories

(requires base knowledge of git. Remixed from Karl Broman's github Guide - Contribute to Someone's repository

- Say you want to contribute changes to our code repository.
- Go to the repository on github. (This site is from `ropensci`, and is called `reproducibility-guide`, you'll find it at `https://github.com/ropensci/reproducibility-guide`.
- Click the fork button at the top right.
- You'll now have your own copy of that repository in your github account.
- Open a terminal/shell.
- Type

```
$ git clone https://github.com/username/reproducibility-guide.git
```

where `username` is *your* username.

- You'll now have a local copy of *your version* of that repository called "origin"
- Add a connection to the original owner's repository and calling it "master".

```
$ git remote add master https://github.com/ropensci/reproducibility-guide.git
```

- You can check if this worked with the command. You should see the connection to *your version* of the repository and the ropensci "master" version.

```
$ git remote -v
```

- Now you can make changes to files. The main branch for the Reproducibility Guide site is `gh-pages`. You may also start another branch for to work on. To check which branch you are on type `git branch`.
- `git add` and `git commit` those changes
- `git push` them back to github. These will go to *your version* of the repository.
- Go to *your version* of the repository on github.
- Click the green "Pull Request" button at the top of the page.
- Note that the ropensci repository will be on the left and *your repository* will be on the right.
- Give a short explanation of the changes and click the "Send pull request" button.

6.7 Pulling others' changes

- Before you make further changes to the repository, you should check that your version is up to date relative to your friend's version.
- Go into the directory for the project and type:

```
$ git pull ropensci master
```

This will pull down and merge all of the changes that your friend has made.

- Now push them back to your github repository.

```
$ git push
```

6.8 Add yourself to the contributors section

Now that you have contributed, you can add yourself to the list of contributors. The file to add your name to is located in the contributors folder in the main directory, just add your name to the index.md file housed here.

6.9 Resources

Full git/github - Karl Broman

How to Collaborate on github - tuts+

6.10 Captioned figures and tables

Figures and tables *with captions* can also be cross-referenced from elsewhere in your book using `\@ref(fig:chunk-label)` and `\@ref(tab:chunk-label)`, respectively.

See Figure 6.1.

Don't miss Table 6.1.

6.11 How to do a report using RMarkdown?

Programming languages typically used by scientistics for data analysis have libraries or packages that can be used to generate reproducible reports. The most popular ones are Jupyter Notebooks for scientists who primarily use python and RMarkdown for those who use R. While they both share many commonalities, their implementation and everyday applications differ. Here, we focus on RMarkdown.

RMarkdown is a file format (typically saved with the `.Rmd` extension) that can contain: a YAML header (see next section), text, code chunks, and inline code.

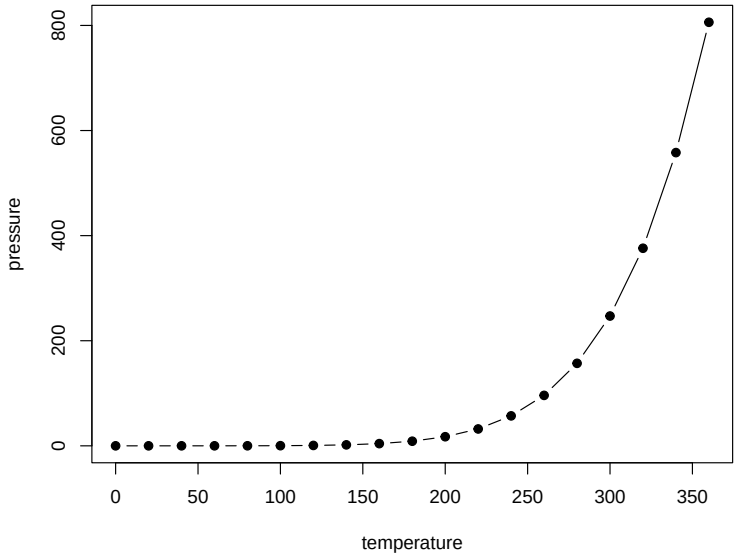


Figure 6.1: Here is a nice figure!

Table 6.1: Here is a nice table!

temperature	pressure
0	0.0002
20	0.0012
40	0.0060
60	0.0300
80	0.0900
100	0.2700
120	0.7500
140	1.8500
160	4.2000
180	8.8000

The `.rmarkdown` package converts this file into a report most commonly into HTML or PDF.

The `.rmarkdown` package automates a multi-step process (Fig. xx). Under the hood, it calls the `.knitr` package that converts the Rmd file into a markdown file. In the process, `.knitr` takes all the code chunks and the inline code, run them through R (or other programs), capture their output, and incorporates them in the report. Afterwards, `.rmarkdown` calls the pandoc program (it is an external program that is not related to R) that can take the markdown file and converts to a variety of formats. For pandoc to generate PDF files, you will need a functional installation of LaTeX that you will need to install separately.

The `bookdown` package comes in to take care of numbering the figures and tables, as well as dealing with citations. As its name suggests, this package can be used to author books, but it is also well-suited to help generating reports.

6.11.1 the YAML header

The YAML header is at the top of your file, it is delineated by three dashes (`---`) at the top and at the bottom of it. It is optional, but can be used to specify:

- the characteristics of your document: the title, authors, date of creation.
- the arguments to pass to pandoc to control the format of the output as well as additional information such as the bibliography file and the formatting of the list of references.
- parameters for your report: for instance, you can specify a parameter such that your report will only use a subset of your data so the final product will be generated quickly when you are developing the code for your project. Once your code is working, you can switch to the full dataset.

6.11.2 Code chunks

Code chunks are interspersed within the text of the report. They are delineated by three backticks (`$```$`) at the top and at the bottom of it. The top backticks are followed by a curly bracket that specify: (1) the language in which the code chunk is written, (2) the name of the chunk (optional but good practice), (3) `knitr` options that control whether and how the code, the output, or the figure are interpreted and displayed. Everything that comes after the name of the chunk has to be a valid R expression: the strings need be quoted, the arguments are separated by commas, and logical values (`TRUE`/`FALSE`) need to be capitalized.

6.11.3 How to deal with figures?

The `knitr` package provides many options to finely control how your figures are going to be generated. Some of `knitr`'s options can be set individually for

each chunk or be set globally. For a reproducible report, it is common practice to have chunk at the beginning of the report that sets default options for the figures. It is also usually a good place to load all the packages you will need for your analysis. For instance the following chunk will do the following:

- all the figures generated by the report will be placed in the `figures/` sub-directory
- all the figures will be 6.5 x 4 inches and centered in the text.

Additionally, this chunk will be named `figuresetup`, and we use the `echo=FALSE` option so the code for the chunk will not be displayed in the report, and use the `include=FALSE` option so no output produced by this chunk will be included in the report.

For our figures, we can now do

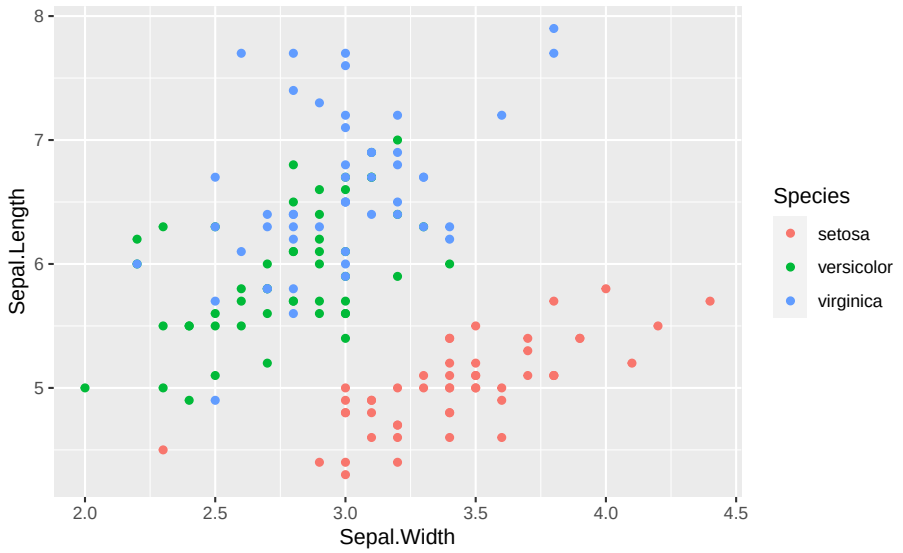
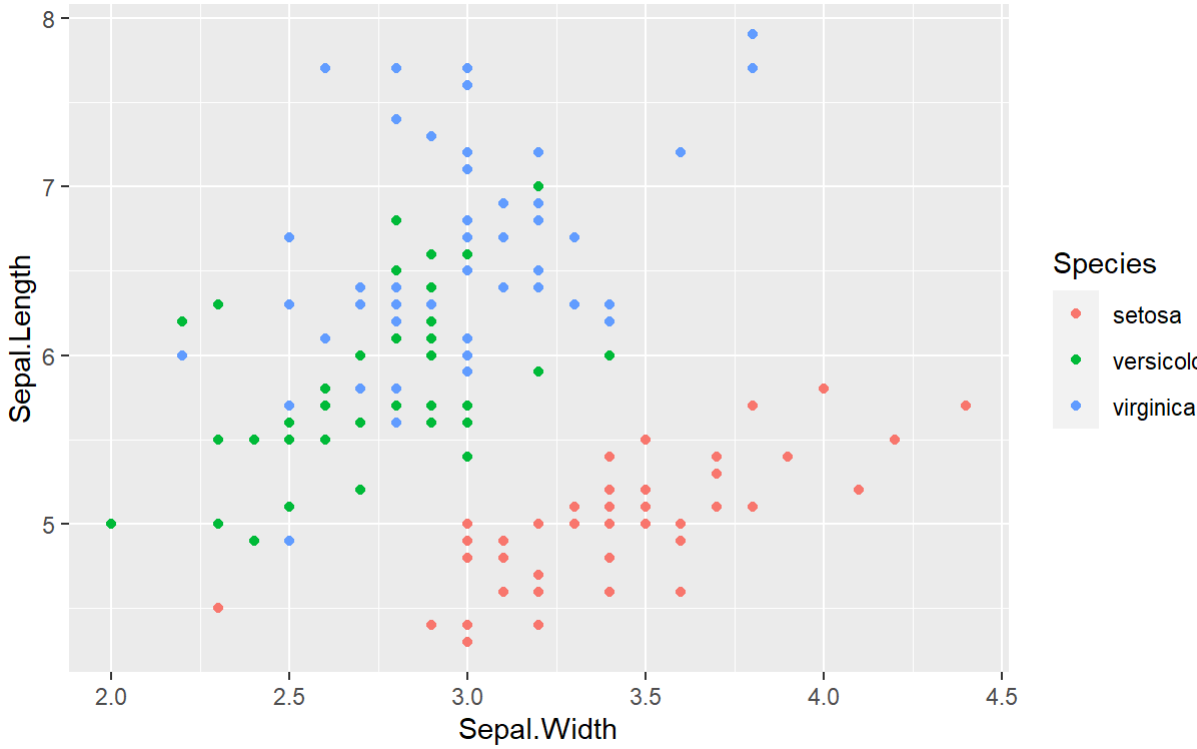


Figure 6.2: Relation between sepal width and length in three species of Iris.

When this file will be processed, it will create an image file (`figures/sepalwidthlength.png`) with the default dimension and the caption specified by the value of the `fig.cap` argument. You can use markdown formatting within the captions of your figures. This figure will have the label `fig:sepalwidthlength` that we will be able to use for cross referencing (see below).

If you wish to incorporate a figure that is not generated by code (a photo of your field site or study organism), using the function `#knitr::include_graphics()` takes care of many details for you, and generates labels and captions as if it was generated by code.



6.11.4 How to deal with tables?

To generate tables, `knitr` comes with the function `kable` that might be sufficient to make simple tables to represent data frames within your report. However, there are many packages that provide more sophisticated approaches to display and format tabular data within your reports. This page provides an overview of the capabilities of the different packages.

Chapter 7

Sharing your book

Additional resources for reference

7.1 Publishing

HTML books can be published online, see: https://bookdown.org/yihui/bookdown/publishing.html

7.2 404 pages

By default, users will be directed to a 404 page if they try to access a webpage that cannot be found. If you'd like to customize your 404 page instead of using the default, you may add either a `_404.Rmd` or `_404.md` file to your project root and use code and/or Markdown syntax.

7.3 Metadata for sharing

Bookdown HTML books will provide HTML metadata for social sharing on platforms like Twitter, Facebook, and LinkedIn, using information you provide in the `index.Rmd` YAML. To setup, set the `url` for your book and the path to your `cover-image` file. Your book's `title` and `description` are also used.

This `gitbook` uses the same social sharing data across all chapters in your book- all links shared will look the same.

Specify your book's source repository on GitHub using the `edit` key under the configuration options in the `_output.yml` file, which allows users to suggest an edit by linking to a chapter's source file.

Read more about the features of this output format here:

https://pkgs.rstudio.com/bookdown/reference/gitbook.html

Or use:

Welcome!

This is a minimal example of a book based on R Markdown and **bookdown** (https://github.com/rstudio/bookdown).

This template provides a skeleton file structure that you can edit to create your book.

The contents inside the .Rmd files provide some pointers to help you get started, but feel free to also delete the content in each file and start fresh.

Additional resources:

The **bookdown** book: https://bookdown.org/yihui/bookdown/

The **bookdown** package reference site: https://pkgs.rstudio.com/bookdown

There are many many different resources on reproducible workflows. This document collects the current resources avaliable with a strong focus on R and (RMarkdown; blog here).

I have started this repository from a combination of different git repositories:

- BES guidelines

My workflow has a distinctly ecological feel along with the tidyverse approach of tooling. I apologise for this in advance. If you are not a R user I would recommend modifying this workflow or finding another workflow with the same components but in your field.

Chapter 8

R resources

Since the development of R and RStudio (and a magnitude of other IT changes happening at the same time) there are now tools for working with issues to do with reproducibility. There are many blogs from a simple web search. Here is a collection of the posts I have drawn inspiration from:

- R bloggers posts: This is a collection of the current blogs on R bloggers.
- R workflow: Mara Averick has a post that looks better than this and has many of the same resources just published a few years ago.
- Methods in Ecology: A good blog that goes with the reproducible guidebook (2018).
- workflow general tips
  - **Key points**

  1. Start without writing code but with a clear mind and perhaps a pen and paper. This will ensure you keep your objectives at the forefront of your mind, without getting lost in the technology.
  2. Make a plan. The size and nature will depend on the project but time-lines, resources and 'chunking' the work will make you more effective when you start.
  3. Select the packages you will use for implementing the plan early. Minutes spent researching and selecting from the available options could save hours in the future.
  4. Document your work at every stage: work can only be effective if it's communicated clearly and code can only be efficiently understood if it's commented.
  5. Make your entire workflow as reproducible as possible. knitr can help with this in the phase of documentation. Reference here

- Data Science and R/Python

- Truely reproducible
- EEB313H1
- Good enough practices for Scientific Computing
- Pitfalls to non-reproducible research: This is a nice simple post on the three danger zones: R session context; OS context; Data versioning.
- What is it?: From Rbloggers.
- Truely reproducible web
- PeerJ stats help
- Implmentation guide: Great post. I have used much of this for my workflow.
- A list of sites I haven't sorted yet

8.0.0.1 Databases for reproducible packages

- rOpenSci is a non-profit initiative founded in 2011 by Karthik Ram, Scott Chamberlain, and Carl Boettiger to make scientific data retrieval reproducible. Over the past seven years we have developed an ecosystem of open source tools, we run annual conferences, and review community developed software.
- The Reproducible Research CRAN Task View

Both of these pages are doing a great job at producing searchable interface for reproducible packages in R with documentation.

Chapter 9

Journals

9.0.0.0.1 PLOS

- Best practices for Scientific Computing
- 10 simple rules for reproducible computational research
- A quick guide to organizing computational biology projects
- Ten Simple Rules for Digital Data Storage

9.1 Coding groups

- Uni of Toronto Coders -R course
- UBC statistics course

With all the resources above I have created a evolving "workflow" for my research.

- Distributed Workflows: This is a class on reproducible workflows in RStudio. Ian Carroll says:

A single collaboration model – the centralized workflow – dominates collaborative research. There is a central hub, and everyone synchronizes their work to it. A number of researchers are nodes – consumers of that hub – and synchronize to that one place.

[1] **Scholarly Context Not Found: One in Five Articles Suffers from Reference Rot**, Klein et al, https://doi.org/10.1371/journal.pone.0115253

[2] **Software citation principles**, Smith et al, https://doi.org/10.7717/peerj-cs.86

[3] **Code publication and citation**, Croucher et al, https://doi.org/10.5281/zenodo.801586

Chapter 10

Resources

Over the past few years there has been a hige development of reproducible research tools using R.

10.1 RStudio tools and packages

Since the development of R and RStudio (and a magnitude of other IT changes happening at the same time) there are now tools for working with issues to do with reproducibility. There are many blogs from a simple web search.

Here is a collection of the posts I have drawn inspiration from:

- R bloggers post: Jeromy Anglim
- Reproducible workflows in RStudio
- R workflow: Mara Averick
- Methods in Ecology
- workflow general tips
- Data Science and R/Python
- Truely reproducible
- EEB313H1

10.2 Scientific publications

There are many many different posts on reproducible workflows. This document collects the current resources available in R and RMarkdown. I have developed this document from a combination of different git repositories:

- BES guidelines as a start
- Added Wickhams etc paper

- Best practices for Scientific Computing (http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745)
- Good enough practices for Scientific Computing (https://swcarpentry.github.io/good-enough-practices-in-scientific-computing/)
- 10 simple rules for reproducible computational research: http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003285
- A quick guide to organizing computational biology projects: http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1000424
- Ten Simple Rules for Digital Data Storage (http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005097)
- The Reproducible Research CRAN Task View: (https://cran.r-project.org/web/views/ReproducibleResearch.html)

10.3 Workflow options

However because of this my workflow has a distinctly ecological feel along with the tidyverse approach of tooling. I apologise for this in advance. If you are not a R user I would recommend finding another workflow with the same components.

- Packaging data publication
- Tidytools package

Chapter 11

Files within this repository

Resources include the templates and resources within this repository.

```
 [1] "checklist"          "collaborationGuide" "copyrightLicensing"
 [4] "dataSharing"        "dataStorage"        "introduction"
 [7] "metaData"           "references"         "tools"
[10] "versionControl"     "workflows"          "writingCode"
```

11.1 Checklist

Reproduciblity can occur at every step in the history of your project. How easy will it be for others or your future self to answer these questions?

11.1.1 Documentation

Is it clear where to begin? (e.g., can someone picking a project up see where to start running it)
can you determine which file(s) was/were used as input in a process that produced a derived file?
Who do I cite? (code, data, etc.)

Is there documentation about every result?

Have you noted the exact version of every external application used in the process?

For analyses that include randomness, have you noted the underlying random seed(s)?

Have you specified the license under which you're distributing your content, data, and code?

Have you noted the license(s) for others peoples' content, data, and code used in your analysis?

11.1.2 Organization

Which is the most recent data file/code?

Which folders can I safely delete?

Do you keep older files/code or delete them?

Can you find a file for a particular replicate of your research project?

Have you stored the raw data behind each plot?

Is your analysis output done hierarchically? (allowing others to find more detailed output underneath a summary)

Do you run backups on all files associated with your analysis?
How many times has a particular file been generated in the past?
Why was the same file generated multiple times?
Where did a file that I didn't generate come from?

11.1.3 Automation

Are there lots of manual data manipulation steps are there?

Are all custom scripts under version control?

Is your writing (content) under version control?

11.1.4 Publication

Have you archived the exact version of every external application used in your process(es)?

Did you include a reproducibility statement or declaration at the end of your paper(s)?

Are textual statements connected/linked to the supporting results or data?

Did you archived preprints of resulting papers in a public repository?

Did you release the underlying code at the time of publishing a paper?

Are you providing public access to your scripts, runs, and results?

11.1.4.1 References

Originally created at the Reproducibility Hackathon 2014

This takes a combination of reproducible guides, inputs them into a bookdown project and begins to write R scripts to access the dynamic database of reproducibility literature underneath it.

11.2 Coding groups

- Uni of Toronto Coders -R course

Chapter 12

Software options

rOpenSci is a non-profit initiative founded in 2011 by Karthik Ram, Scott Chamberlain, and Carl Boettiger to make scientific data retrieval reproducible. Over the past seven years we have developed an ecosystem of open source tools, we run annual unconferences, and review community developed software.

They have produced a great interface for reproducible packages in R with documentation. The key software and packages I use for my workflow are:

12.1 File building

Im not sure what this is actually meant to represent but I see these options as packages making packages

12.1.1 packrat

In mid-August of 2016, Eric Nantz of the R-Podcast converted me to packrat (by Kevin Ushey and others at RStudio), a package that lengthens the shelf life of R projects. Packrat maintains local snapshots of dependencies so that your project won't break when external packages are updated. Just be sure your current working directory is the root directory of your project when you run remake::make() or the Makefile. Also, if you use a shell.sh with your Makefile, be sure to modify module load R so that it points to the version of R corresponding to your packrat library. You can learn more about packrat with the hands-on walkthrough.

12.1.2 ProjectTemplate

- ProjectTemplate and the webpage is here.
- A workshop using this package

12.2 Packages

Hadley wickham book on R Packages.

12.2.1 R

- rrtools

12.3 Version control

12.4 Markdown

Bibliography

Yihui Xie. *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition, 2015. URL http://yihui.org/knitr/. ISBN 978-1498716963.

Yihui Xie. *bookdown: Authoring Books and Technical Documents with R Markdown*, 2021. URL https://CRAN.R-project.org/package=bookdown. R package version 0.24.