A Minimal Book Example

John Doe

Contents

1 **5**

1.1 Why do reproducible research? 5
1.2 What are the principles of reproducible research? 6
1.3 What are the different kinds of reproducible research? 6

2 Resources **13**

2.1 RStudio tools and packages 13
2.2 Scientific publications 13
2.3 Workflow options 14

3 Files within this repository **15**

3.1 Checklist 15
3.2 Coding groups 17

4 Software options **19**

4.1 File building 19
4.2 Packages 20
4.3 Version control 20
4.4 Markdown 20

5 Additional resources for reference **21**

5.1 What's a reproducible report? 21
5.2 Why a reproducible report? 22
5.3 How to do a report using RMarkdown? 23
5.4 Where can I find more information? 26

6 Documenting and managing dependencies **27**

6.1 Show the packages you used 28
6.2 Use packages that help recreate your setup 28
6.3 Use containers to share your setup 28

Chapter 1

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.

In this short introduction, we briefly summarise some of the principles, definitions and questions relevant to reproducible research that have emerged in the literature. We outline basic and widely applicable steps for promoting reproducibility. While our suggestions for tools and workflow are aimed primarily at the computational environment of the R programming language, the principles and definitions sketched out here are applicable to researchers working in any environment.

You are welcome to contribute to the original guide (fix typos, add to the reference list, etc.) by submitting a pull request to its github repository. Or contact me with regard to this extended guide here https://github.com/davan690/reproducible-guidebook.

If you're not familiar with R, have a look at rforcats.net for a gentle introduction, and if you're already working with R then have a look at Hadley's Advanced R programming book for a guide to best practices.

1.1 Why do reproducible research?

There are two basic reasons to be concerned about making your research reproducible. The first is **to show evidence of the correctness of your results**. Descriptions contained in scholarly publications are rarely sufficient to convince sceptical readers of the reliability of our work. In simpler times, scholarly publications showed the reader most of the work involved in getting the result. The reader could make an informed choice about the credibility of the science. Now, the reader may feel they are being asked to blindly trust in all the details that were not described in the original journal article.

Adopting a reproducible workflow means providing our audience with the code and data that demonstrates the decisions we made as we generated our results.

This makes it easier for others to satisfy themselves that our results are reliable (or not, since reproducibility is no guarantee of correctness).

The second reason to aspire to reproducibility is **to enable others to make use of our methods and results**. Equipped with only our published article, our colleagues might struggle to reconstruct our method in enough detail to apply it to their own data. Adopting a reproducible workflow means publishing our code and data in order to allow our colleagues to extend our approach to new applications with a minimum of effort. This has the potential to save a great deal of time in transmitting knowledge to future researchers.

1.2 What are the principles of reproducible research?

The motivating principle for reproducible research is that the traditional unit of scholarly communication - a published article - is only the tip of the iceberg of the research process. Jon Claerbout described the article as merely an advertisement for research (Claerbout and Karrenbach 1992):

> An article about computational results is advertising, not scholarship. The actual scholarship is the full software environment, code and data, that produced the result.

In most cases, when we have only an article, we are unable to engage with the bulk of the details and decisions that lead to the figures, tables and narrative presented in the article.

The aim of practising reproducible computational research is to expose more of the research workflow to our audience. This makes it easier for them to make a more informed assessment of our methods and results, and makes it easier for them to adapt our methods to their own research.

1.3 What are the different kinds of reproducible research?

Enabling reproducibility can be complicated, but by separating out some of the levels and degrees of reproducibility the problem can become more manageable because we can focus our efforts on what best suits our specific scientific domain. Victoria Stodden (2014), a prominent scholar on this topic, has identified some useful distinctions in reproducible research:

- *Computational reproducibility*: when detailed information is provided about code, software, hardware and implementation details.
- *Empirical reproducibility*: when detailed information is provided about non-computational empirical scientific experiments and observations. In

practise this is enabled by making data freely available, as well as details of how the data was collected.

- *Statistical reproducibility*: when detailed information is provided about the choice of statistical tests, model parameters, threshold values, etc. This mostly relates to pre-registration of study design to prevent p-value hacking and other manipulations.

Stodden et al. (2013) place **computational reproducibility** on a spectrum with five categories that account for many typical research contexts:

- *Reviewable Research.* The descriptions of the research methods can be independently assessed and the results judged credible. (This includes both traditional peer review and community review, and does not necessarily imply reproducibility.)
- *Replicable Research.* Tools are made available that would allow one to duplicate the results of the research, for example by running the authors' code to produce the plots shown in the publication. (Here tools might be limited in scope, e.g., only essential data or executables, and might only be made available to referees or only upon request.)
- *Confirmable Research.* The main conclusions of the research can be attained independently without the use of software provided by the author. (But using the complete description of algorithms and methodology provided in the publication and any supplementary materials.)
- *Auditable Research.* Sufficient records (including data and software) have been archived so that the research can be defended later if necessary or differences between independent confirmations resolved. The archive might be private, as with traditional laboratory notebooks.
- *Open or Reproducible Research.* Auditable research made openly available. This comprised well-documented and fully open code and data that are publicly available that would allow one to (a) fully audit the computational procedure, (b) replicate and also independently reproduce the results of the research, and (c) extend the results or apply the method to new problems.

This kind of spectrum is valuable because it easy for us to understand how our current practices can be defined. It helps to pinpoint limitations that we face (for example many researchers are unable to publicly release sensitive data) and identify basic changes we can make to improve reproducibility.

1.3.0.1 How can software tools can make our research more reproducible?

For many domains of science there are already several options for software tools to help overcome the technical challenges of doing reproducible research. These tools enable researchers to capture and communicate the details of their workflow with much greater efficiency that simply writing a lengthy prose narrative. There are three general types of software tools (cf. Stodden et al 2013). For more specific details on the tools named here, see our tools page:

- **Literate computing, authoring, and publishing**. These tools enable writing and publishing self-contained documents that include narrative and code used to generate both text and graphical results. In the R ecosystem, knitr and its ancestor Sweave used with RStudio are the main tools for literate computing. Markdown or LaTeX are used for writing the narrative, with chunks of R code sprinkled throughout the narrative. IPython is a popular related system for the Python language, providing an interactive notebook for browser-based literate computing.

- **Version control**. These tools enable you to keep a record of file changes over time, so specific versions can be recalled later. During exploratory data analysis you might pursue several paths that lead nowhere, with version control you can efficiently revert to an earlier point in the analysis without starting from scratch. There are several systems that are widely used, and amongst R users the git version control system is especially popular. Version control tools combined with web services such as Github and BitBucket also enable you to work with collaborators with a minimum of confusion and friction.

- **Tracking provenance**. Provenance refers to the tracking of chronology and origin of research objects, such as data, source code, figures, and results. Programs that track provenance enable you to capture a complex workflow that relies on many different tools. Tools that record provenance of computations include VisTrails, Kepler, Taverna, and several others. Many of these can include R as part of a sequence of tools. If your workflow uses many different programs, these provenance trackers will be useful for efficiently documenting the order of processes and parameters to show others your analytical process.

- **Automation** Several Unix tools are useful for streamlined automation and documentation of the research process, e.g. editing files, moving input and output between different parts of your workflow, and compiling documents for publication. Scripts for shell programs and make files enable highly efficient and easily repeatable control of your computer. These scripts can also be used to keep a record of every kind of action on your computer. There are lots of options for learning and using Unix tools, for more details check out Karl Broman's list of tools and editors for different platforms.

- **Capturing the computational environment** A substantial challenge in reproducing analyses is installing and configuring the web of dependencies of specific versions of various analytical tools. Virtual machines (a computer inside a computer) enable you to efficiently share your entire computational environment with all the dependencies intact. Popular VM applications include VirtualBox and VMWare. One challenge of working with VMs is that the files that contain the environment are not small, typically one gigbyte or more, which can be awkward to share. On the other hand, they are convenient for use with cloud-based services such as Amazon EC2.

1.3.0.2 How to give and receive credit for reproducible research?

One of the challenges going forward is how to share code and data. In many disciplines, researchers have concerns about copyright, being scooped, and getting recognition for the effort invested in writing code and collecting data. These concerns hinder people from sharing details of their research and impede reproducibility. In response to these concerns, Stodden (2009) has proposed the **Reproducible Research Standard** to encourage researchers to release their products with standardised instructions for reuse and attribution. In brief, this standard recommends that researchers:

1. Release media components (text and figures, such as markdown, LaTeX, PDF and other documents) under a Creative Commons Attribution (CC-BY) licence.
2. Release code components under the MIT license or similar.
3. Release data under the CC0 licence, that is, place data in the public domain.

There are several options for depositing these components of the research compendia online that give DOIs for convenient citing and discovery (eg. figshare with github integration, zenodo also with github integration, and researchcompendia.org).

These recommendations will not be suitable for every occasion. For example, research that uses human subjects data obviously cannot make that data publicly available without substantial changes to protect identities. In these situations, some modifications will be required to maximize reproducibility while honoring obligations to stakeholders. In some cases, a solution might be releasing a sample of "dummy" data that contains similar qualities to the real data set. While this would not be fully 'reproducible research' as defined above, it would still be very useful to other researchers who could use a validated sample to inspect, evaluate and extend the research.

1.3.0.3 How can we make reproducible research the norm?

There are substantial costs of time and effort involved in become familiar with ways of making your research more reproducible. Until code and data sharing becomes common practice, it can be difficult for many researchers to see the point of adopting these procedures. There are many small informal things we can do to promote reproducibility and increase recognition of these efforts. Along these lines, Leveque et al. (2012) recommend we:

- **Train students** by putting homework, assignments & dissertations on the reproducible research spectrum
- **Publish examples** of reproducible research in our field
- **Request code & data** when reviewing
- **Submit to & review for journals** that support reproducible research
- **Critically review & audit** data management plans in grant proposals
- Consider reproducibility wherever possible in **hiring, promotion & reference letters**.

Similarly, Collberg et al. (2014) summarise the lessons they learned in attempting to reproduce 600 computer science projects:

- Unless you have compelling reasons not to, plan to release the code. It is the right thing to do, and if you start with this mindset from the beginning of the project, the amount of extra work will likely be negligible.
- Students will leave, plan for it. When building the system keep in mind that the code should outlive both you and the student.
- Create permanent email addresses. You and your students will most likely be changing jobs a few times during your career. While some schools will keep old email addresses around, or forward email, you cannot count on it. Create email addresses that you know will be permanent throughout your working life and use them in all professional correspondence.
- Create project websites. These are more likely to remain functional over time than email addresses. Put the URL in the paper. Be prepared to upload code and test data to your web sites at the same time as you upload the paper describing your system to a conference site for review.
- Use a source code control system. Whenever you submit or publish a paper, set a label on the corresponding code version so that you can easily recreate it.
- Backup your code.
- Resolve licensing issues. If you anticipate problems start the licensing process early, so that you are able to release the code at the same time as the paper is submitted for publication.
- Keep your promises. If your grant application states that you will be sharing code with the community, plan for keeping that promise.
- Plan for longevity. Projects may live on for a long time, with

many students building on the code. Plan for this at the onset of the project, by setting up the appropriate directory structures, plug-in architectures, etc., which will allow the project to grow.
- Avoid cool but unusual designs. Unless you have a compelling reason to do otherwise, stay with standard operating systems, programming languages, and tool chains.

1.3.0.4 What's next?

If this is all news to you, the next steps are to simply dive in and find the tools and workflow that best suit your research domain. Incrementally adding reproducibility to current projects is an excellent way to learn more, and then future projects can be planned for built-in reproducibility from the start. Becoming involved in the community of researchers in your domain who are using tools for reproducibility can also ease the transition to new tools and workflows.

Read through the other sections of this site for more specific instructions, details and ideas on reproducible research using R. Also check out our curated list of further readings which includes links to numerous excellent guides on the practical details of improving reproducibility with R.

Chapter 2

Resources

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

2.1 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

2.2 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)

2.3 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 3

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"
```

3.1 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?

3.1.1 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?

3.1.2 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?

3.1.3 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?

3.1.4 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?

3.1.4.1 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.

3.2 Coding groups

- Uni of Toronto Coders -R course

Chapter 4

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:

4.1 File building

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

4.1.1 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.

4.1.2 ProjectTemplate

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

4.2 Packages

Hadley wickham book on R Packages.

4.2.1 R

- rrtools

4.3 Version control

4.4 Markdown

Chapter 5

Additional resources for reference

5.1 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.

5.2 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.

5.3 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. 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.

5.3.1 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.

5.3.2 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.

5.3.3 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

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.

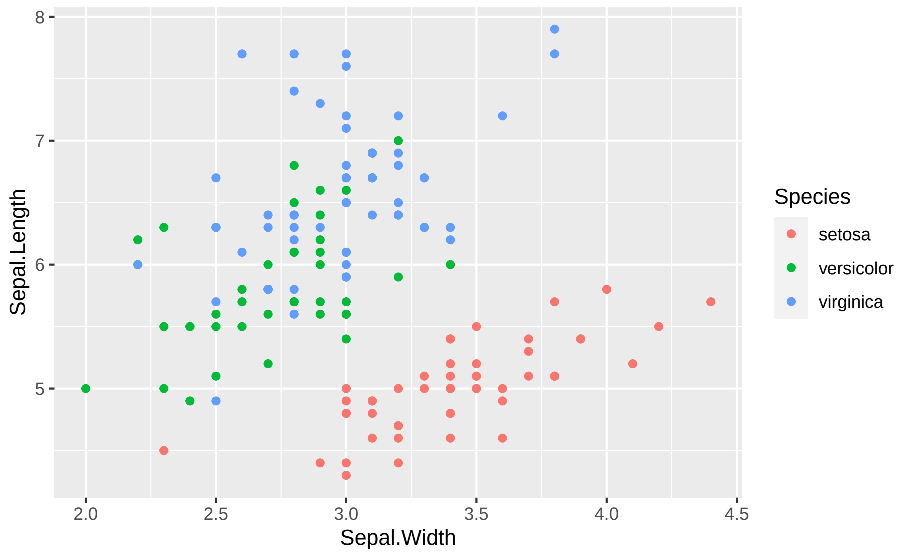
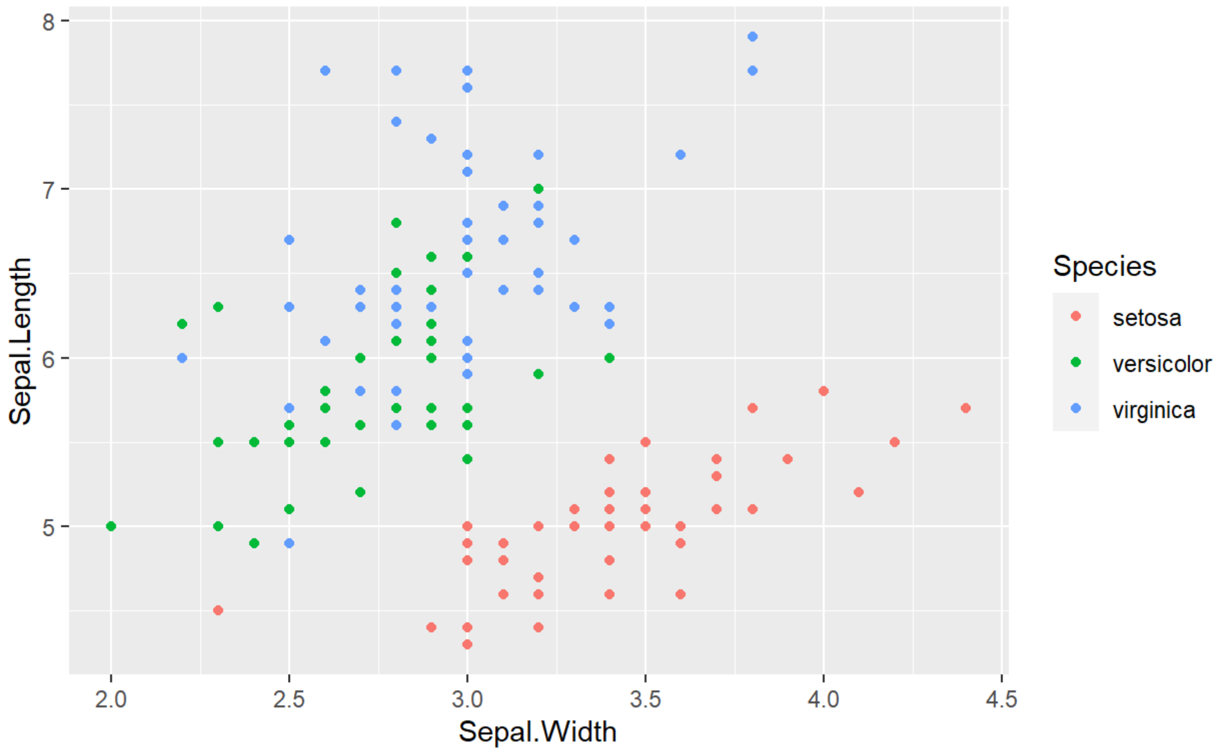


Figure 5.1: Relation between sepal width and length in three species of Iris.



5.3.4 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.

5.3.5 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.

5.4 Where can I find more information?

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

Chapter 6

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.

6.1 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.

6.2 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.

6.3 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.