Building reproducible analytical pipelines with the R programming language

Jane Doe

1/6/23

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

1 Introduction

This book will teach how to structure your projects in such a way as to make them reproducible.

I think we should also explain that this book presents one way of doing things, and that it is by no means the only, canonic way, of solving the problems we will be talking about.

1.1 Prerequisites

You should be comfortable with the R programming language. This book will assume that you have been using R for some projects already, and want to improve not only your knowledge of the language itself, but also how to successfully manage complex projects.

1.2 What is reproducibility?

1.3 Are there different types of reproducibility?

Reproducibility is on a continuum.

2 Functional programming

This chapter will teach you the fundamentals of functional programming. *Functional programming* might sound scary, but we will focus on only a handful of concepts that are quite accessible while providing many benefits. Using these functional programming concepts will make your code more reliable, easier to test, document, share, and ultimately rerun.

3 Version control

What Miles said on the matter:

There are still a lot of people that find git intimidating and still potential for some things to go badly for a project if git is used in the wrong way. I once had a colleague who assured me they knew how to use git proceed to use a repo like their personal dropbox folder. Perhaps the details of git usage can be basically waved away, but some detail about good git workflow could be incorporated. For example: The branching model to use. IMHO trunk-based development works much better than gitflow for analysis teams. Version number discipline. Why you always bump the version number when making changes to your packages. Why keeping commits small and confined to just one target at a time if possible is useful when tracing problems with a pipeline.

4 Getting started with your project

We suggest to not start with an R script, but start from a Qmd file.

What Sébastien said on the matter:

*There seems to be two way of presenting such a workflow:

What does the complete project looks like at the end and what tools do I need? How do I start working on it?*

This questions should get answered in the intro?

Refer to https://github.com/b-rodrigues/rap4all/issues/1#issuecomment-1327627267

4.1 Literate programming

4.1.1 Why bother?

Allows you to explain what you're doing as you're coding. This file can later be inflated, if necessary, to make a package using `{fusen}`.

4.2 Quarto basics

Teach some Quarto basics

4.3 Parametrized reports

4.4 Your project is done (?)

So here the project is done, but actually it's just an Qmd file that gets compiled, so we would need to explain why this is not enough, and motivate the readers to go the full way: developing packages, using targets, and so on

5 Testing your code

5.1 Assertive testing (and defenvise programming?)

The analysis is still in Quarto, so how could the readers of this book test their code? Copying here what Miles wrote on the subject:

'Assertive programming' is a topic that might be missing from the book. I think of it as a kind of dual of unit testing. Unit testing is for more generally applicable packaged code. But when you have functions in your analysis pipeline that operate on a very specific kind of input data, unit testing becomes kind of nonsensical because you're left to dream up endless variations of your input dataset that may never occur. It's a bit easier to flip the effort to validating the assumptions you have about your input and output data, which you can do in the pipeline functions themselves rather than separate unit testing ones. This is nice because it ensures the validation is performed in the pipeline run, and so is backed by the same reproducibility guarantees.

I think at the end of the chapter we should hint at unit testing, but leave it as a subsection of the next chapter that deals with packaging code.

6 Packaging your code

In this chapter you're going to learn how to create your own package.

We should make clear that this does not mean publishing the package on CRAN.

6.1 Benefits of packages

6.2 Intro to packge dev

This is where fusen comes into play I guess; so we start from the Qmd file that was written before, containing the functions an the analysis, and see how we can now create a package from it, and use that file as a vignette? Copying here what Sébastien said on the matter

6.3 Document your package (?)

I guess fusen makes this process easy and leverages roxygen?

6.4 Managing package dependencies (?)

Discuss NAMESPACE and DESCRIPTION and all that. I think it's important to also discuss here how to define dependencies from remotes, not just CRAN.

6.5 Unit testing

This is where I think we should discuss unit testing

6.6 pkgdown

7 Build automation

Why build automation: removes cognitive load, is a form of documentation in and of itself, as Miles said

It is possible to communicate a great deal of domain knowledge in code, such that it is illuminating beyond the mere mechanical number crunching. To do this well the author needs to make use of certain styles and structures that produce code that has layers of domain specific abstraction a reader can traverse up and down as they build their understanding of the project. Functional programming style, coupled with a dependency graph as per {targets} are useful tools in this regard.

8 Introduction to reproducibility

Since we said in the intro to the book that reproducibility is on a continuum, I think that this chapter should focus on the bare minimum, which would culminate with renv

Then at the end, explain why renv is not enough (does nothing for R itself, nor the environment the code is running on)

9 Advanced topics in reproducibility

Now that the readers are familiar with renv, but also its shortcomings, we can go a step further and introduce Docker. I think some primer on the Linux command line could be included here

9.1 First steps with Docker

To write your own Dockerfile, you need some familiarity with the Linux cli, so here's...

9.2 A primer on the Linux command line

9.3 Dockrizing your project

10 Continuous integration and continuous deployment/delivery

References