Writing Reproducible Code

Literate, linear programming and modularity

Daniela Palleschi

Fri Aug 23, 2024

Table of contents

Learning objectives	
Modular analyses	2
Reproducible code	3
Writing linear code	3
Literate programming	3
Example R script	4
Dynamic reports	5
R v. Rmarkdown v. Quarto	5
YAML	6
Structure your reports	
Code chunks	7
Documenting package dependencies	7
Session info	8
Tips and tricks	8
Hands-on: working with Quarto	8
Session Info	12

Learning objectives

• learn about literate programming and modular analyses

- create and render a dynamic report with Quarto
- load data using here

Modular analyses

- recall our scripts folder (which you might've named analysis or something else)
- ideally, this would also contain subfolders, one for each stage of your analysis
 - or at least, multiple scripts
- this is the concept of modularity (Bowers & Voors, 2016; Nagler, 1995)
 - separating data cleaning, pre-processing, recoding, merging, analyses, etc. into files/scripts

Modularity

The concept of modularity relating to data analysis refers to saving different stages of data manipulation and analysis in separate scripts (Nagler, 1995). An important step is to also ensure the scripts are appropriately named so that the structure of your analysis/scripts folder "can be a map itself" (Bowers & Voors, 2016, p. 838).

A suggestion I have is to add numerical prefixes before scripts to explicitly state the order in which they must be run. For example, the script that reads in the raw data and cleans/preprocesses it (e.g., renaming or removing irrelevant/redundant variables) and exports a new data file that will be used in subsequent scripts could be called OO_cleaning.qmd or something of the sort. My current prefered analysis script structure for a given data set is as follows:

```
scripts/
00_cleaning.qmd
01_preprocessing-and-eda.qmd
02_analyses.qmd
03_figures.qmd
04_tables.qmd
```

Where EDA refers to Exploratory Data Analysis, where I look at things like distributions of variables and demographic information. You'll note the O3_figures.qmd and O4_tables.qmd scripts follow the O2_analyses.qmd script. This is because my figures and tables include model results or predictions, and so need to follow the analyses.

Reproducible code

- how you write your code is the first step in making it reproducible
- the first principle is that your code must be *linear*
 - this means code must be written in a linear fashion
 - this is because we typically run a script from top-to-bottom

Listing 1 Non-linear code

```
read_csv(here("data", "my_data.csv"))
library(readr)
library(here)
```

Writing linear code

- you need to load a package before you call a function from it
 - if we're just working in an R session, before means temporally prior
 - with linear code, before means higher up in the script
- such pre-requisite code must
 - a. be present in the script
 - b. appear above the first line of code that uses a function from this package
- missing pre-requisite code might not throw an error message
 - but might produce output we aren't expecting
 - e.g., forgetting to filter out certain observations
 - or forgetting that some observations have been filtered out

Literate programming

Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do.

```
— Knuth (1984), p. 97
```

• refers to writing and documenting our code so that humans can understand it

- important for us: we are (generally) not professional programmers, nor are our peers
- we need to not only know what our code is doing when we look back at it in the future/share it
- the easiest way: informative comments
 - the length and frequency of these comments is your choice

Example R script

```
# Analysis script for phoneme paper
# author: Joe DiMaggio
# date: Feb. 29, 2024
# purpose: analyse cleaned dataset

# Set-up ###

# load required packages
library(dplyr) # data wrangling
library(readr) # loding data
library(here) # project-relative file path

# Load-in data
df_phon <- read_csv(here("data", "phoneme_tidy_data.csv"))

# Explore data ###
summary(df_phone)</pre>
```

- begins with some meta-information about the document, including its purpose
 - aids in knowing which scripts to run in which sequence
- there are three hashtags after some headings (###)
 - this is helpful because it structures the outline of the document in RStudio
- the purpose of chunks of code are written above
 - $-\,$ description of specific lines of code are also given

Dynamic reports

- R scripts are useful, but don't show the code output
 - and commenting can get clunky
- dynamic reports combine prose, code, and code output
 - R markdown (.Rmd file extension) and Quarto (.qmd) are extensions of markdown
 * can embed R code 'chunks' in a script, thus producing 'dynamic' reports
 - produce a variety of output files which contain text, R code chunks, and the code chunk outputs all in one

i Task: New Quarto document

- 1. Navigate to File > New file > Quarto document
- 2. Write some title, your name (Author), make sure 'Visual markdown Editor' is unchecked
- 3. Click 'Create'
- 4. A new tab will open in R Studio. Press the 'Render' button above the top of the document, you will be prompted to save the document. Store it in a folder called scripts and save it as 01-literate-programming.qmd.
- 5. What happens?

R v. Rmarkdown v. Quarto

- .R files contain (R) source code only
- .Rmd files are *dynamic reports* that support
 - R-Code (and R-packages)
- .qmd files are dynamic reports (RStudio v2022.07 or later
 - R-Code (and R-packages)
 - native support for Python (and Jupyter-Notebooks)
 - native support for Julia

• Check your RStudio version

Run the following in the Console: RStudio.Version()\$version

- if the output is 2022.07 or higher you can use Quarto
- if not: update RStudio: Help > Check for updates

YAML

- the section at the very top fenced by ---
- contains all the meta information about your document
 - e.g. title, author name, date
 - also formatting information, e.g. type of output file
- there are many document formatting and customisation options that we won't cover in this course
- but for example I have many YAML formatting options in the source code of my slides

? YAML

- 1. change the title if you want to do so.
- 2. guess how to add a subtitle (hint: it is similar to adding a title)
- 3. add an author, author: 'firstname lastname' (see example below)
- 4. add a table of contents (Table of Contents = toc) by changing format so that it looks like this:

```
title: "Dynamic reports"
author: "Daniela Palleschi"
format:
  pdf:
    toc: true
```

5. Render the document. Do you see the changes?

Structure your reports

- remember to use (sub-)headings (e.g., # Set-up)
- describe the function/purpose at the beginning of the script
- document your train of thought and findings throughout the script
 - e.g., why are you producing this plot, what does it tell you?
- give an overview of the findings/end result at the end
- it's wise to avoid very long, multi-purpose scripts
 - rule of thumb: one script per product or purpose
 - e.g., data cleaning, exploration, analysis, publication figures, etc.

Code chunks

- the main benefit of dynamic reports: combining text with code (and code output)
- R code goes in code chunks:

```
```{r}
2+2
...
```

#### [1] 4

- to add a code chunk: Code > Insert Chunk
  - or use the keyboard shortcut: Cmd+Opt+I (Mac) / Ctrl+Alt+I (Windows)

### • Adding content

- 1. Use the example R script above to create a structured document
  - use headings (#) and subheadings (##) accordingly
- 2. Load in our dataset in a code chunk
- 3. Render the document. Do you see the changes?

### **Documenting package dependencies**

- R and R package versions are both open source, and are frequently updated
  - you might've run your code using dplyr version 1.1.0 or later, which introduced the .by per-operation grouping argument
  - what happens when somebody who has an older version of dplyr tries to run your code?
    - \* They won't be able to!
  - the reverse of this situation is more common:
    - \* a newer version of a package no longer supports a deprecated function or argument

#### Session info

- so, print your session info at the end of every script
  - this will print your R version, package versions, and more

sessionInfo()

### Tips and tricks

- when you start a new script make sure you always start with a clean R environment:
   Session > Restart R or Cmd/Ctrl+Shift+0
  - this means no packages, data, functions, or any other dependencies are loaded
- at the top of your script, always load packages required below
  - you can always add more packages to the list as you add to your script
- Render often: when you make changes to your script make sure you re-render your document
  - checks you haven't introduced any errors
  - easier to troubleshoot if smaller changes have been made
- if you can run your script manually from source but it won't render, restart your R session and see if you can still run it from source
  - often the problem is some dependency in your environment that is not linearly introduced in the script

### Hands-on: working with Quarto

If you've never worked with Rmarkdown or Quarto before, try the following task. If you have, then try looking at a current or past analysis project you have, and check to see if it has the following:

- a designated folder containing all required files
- an .RProj file in this folder
- a transparent and organised folder structure
- a consistent and clear folder and file naming convention
- a README file in the project folder, and any other large folders
- code that runs if you try it

### i Task: Editing your Quarto document

- 1. In the YAML (meta document info between ---):
  - Change the title
  - Add date: "08/21/2024" (MM/DD/YYYY format)
  - Change format: html to the following:

# format: html:

toc: true

Render the document. If it works, continue. If not, check that you aren't missing any punctuation (e.g., quotation marks).

- 2. Adding a heading and text
  - Remove all text below the YAML.
  - Add a heading, e.g., # Packages
  - Navigate two lines below your heading, and write Install the tidyverse and here packages. This will not run when we render the script.
- 3. Adding a code chunk
  - Navigate two lines below the text, and insert a new code chunk (either Code >
     Insert Chunk or the keyboard shortcut Cmd+Option+I on Mac or Ctrl+Alt+I
     on Windows)
  - in the very first line of the code chunk, add #| eval: false
  - below this line, write # install packages
  - below this line, write install.packages("tidyverse") and hit Cmd/Ctrl+Enter and wait for the package to install (this may take some minutes)
  - below this line, write install.packages("here") and hit Cmd/Ctrl+Enter
- 4. Loading packages
  - Add a new line of text: Print the project directory filepath using the here package.
  - Insert a new code chunk two lines below the first code chunk
  - below this line, write # load packages
  - below this line, write library(tidyverse)
  - below this line, write library(here)
  - hit Cmd/Ctrl+Shift+Enter, or click the little green triangle at the top right of the chunk

- 5. Printing our project directory path
  - In a new code chunk, add the comment # print project directory and the code here()
- 6. Render the document
- 7. In a code chunk: load in the dataset (e.g., df\_chromy <- read\_csv(here::here("data", "chromy\_et-al\_2023\_English\_final.csv")))
  - explore the dataset however you normally would (e.g., summary())
  - if you have some experience with R, try producing plots of the data
- 8. Add a section called Session Info at the bottom of your script
  - include a code chunk with sessionInfo()
- 9. Render the document

```
A Reveal Quarto syntax
title: "Literate programming"
author: "Daniela Palleschi"
date: "08/22/2024"
format:
 html:
 toc: true
Set-up
Set-up environment: load packages and data
Packages
Install the tidyverse and here packages. This will not run when we render the script.
```{r}
#| eval: false
# install packages
install.packages("tidyverse")
install.packages("here")
```

```
Load required packages.
```{r}
load packages
library(tidyverse)
library(here)
Print the project directory filepath using the `here` package.
```{r}
# print project directory
here()
## Load data
```{r}
df_chromy <- read_csv(here("data", "chromy_et-al_2023_English_final.csv"))</pre>
Data exploration
Take a look at the data.
```{r}
glimpse(df_chromy)
Summary statistics of self-paced reading times (all sentence regions).
```{r}
summary(df_chromy$ReadTimes)
Session Info
```{r}
sessionInfo()
```

Session Info

My session info.

sessionInfo()

[25] R6_2.5.1

[29] magrittr_2.0.3

```
R version 4.4.0 (2024-04-24)
Platform: aarch64-apple-darwin20
Running under: macOS Ventura 13.2.1
Matrix products: default
BLAS:
        /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;
locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
time zone: Europe/Berlin
tzcode source: internal
attached base packages:
[1] stats
              graphics grDevices datasets utils
                                                       methods
                                                                 base
loaded via a namespace (and not attached):
 [1] vctrs_0.6.5
                       cli_3.6.3
                                          knitr_1.48
                                                            rlang_1.1.4
 [5] xfun_0.47
                                                            glue_1.7.0
                       renv_1.0.7
                                          jsonlite_1.8.8
 [9] rprojroot_2.0.4
                                                            fansi_1.0.6
                       htmltools_0.5.8.1 hms_1.1.3
[13] rmarkdown_2.28
                       evaluate_0.24.0
                                         tibble_3.2.1
                                                            tzdb_0.4.0
[17] fastmap_1.2.0
                       yaml_2.3.10
                                          lifecycle_1.0.4
                                                            compiler_4.4.0
[21] pkgconfig_2.0.3
                       here_1.0.1
                                          rstudioapi_0.16.0 digest_0.6.36
```

Bowers, J., & Voors, M. (2016). How to improve your relationship with your future self. Revista de Ciencia Política (Santiago), 36(3), 829–848. https://doi.org/10.4067/S0718-090X2016000300011

utf8_1.2.4

pillar_1.9.0

Knuth, D. (1984). Literate programming. The Computer Journal, 27(2), 97–111.

readr_2.1.5

tools_4.4.0

Nagler, J. (1995). Coding Style and Good Computing Practices. PS: Political Science & Politics, 28(3), 488–492. https://doi.org/10.2307/420315