

# Writing Reproducible Code

Literate, linear programming and modularity

Daniela Palleschi

Leibniz-Zentrum Allgemeine Sprachwissenschaft

Thu Oct 17, 2024



# Topics

- modular analyses and literate programming
- create and render a dynamic report with Quarto
- documenting your dependencies

# Reproducible code

# 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
  - i.e., our scripts should run from top-to-bottom

## ► Example

# 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

- our code should also be *literate*
- i.e., we should write and document our code so that humans can understand it
  - important for us: we are (generally) not professional programmers, nor are our peers
- we need to 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

analysis4.R

```
1 library(dplyr)
2 library(readr)
3 library(here)
4
5 df_phon <- read_csv(here("data", "phoneme_tidy_data.csv"))
6
7 summary(df_phon)
8
9 plot(df_phon$duration, df_phon$mean_f0)
```

phoneme\_analysis.R

```
1 # [example script]
2 # Analysis script for phoneme paper ①
3 # author: Joe DiMaggio
4 # date: Feb. 29, 2024
5 # purpose: analyse cleaned dataset
6
7 # Set-up ### ②
8
9 # load required packages ③
10 library(dplyr) # data wrangling ④
11 library(readr) # loading data
12 library(here) # project-relative file path
13
14 # Load-in data
15 df_phon <- read_csv(here("data", "phoneme_tidy_data.csv"))
16
17 # Explore data ###
18 summary(df_phon)
19
20 # scatterplot: phoneme duration by mean f0
21 plot(df_phon$duration, df_phon$mean_f0)
```

- the metadata, headings, and informative comments in `phoneme_analysis.R` make the second script much easier to follow
- this becomes more important with longer, more complex scripts



# 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

# Dynamic reports

# Dynamic reports: `.Rmd` and `.qmd`

- 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
- for example, we can look at the example script `phoneme_analysis.R`, but we have no idea what the scatterplot it produced looks like

### 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, checkout the [Quarto website](#) for more
- but for example I have many YAML formatting options in the source code of my slides

```
1 ---  
2 title: "My title"  
3 ---
```

# YAML

## 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:

```
1 ---
2 title: "Dynamic reports"
3 author: "Daniela Palleschi"
4 format:
5   pdf:
6     toc: true
7 ---
```

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



# Structure your reports

- remember to use (sub-)headings (e.g., `# Set-up`)
  - N.B., you don't need the 3 hashtags here (only in R scripts)
- 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:

```
1  ```{r}
2  2+2
3  ```
```

```
[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

## Adding structure and code chunks

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

```
1 sessionInfo()
```

- with dynamic reports: this will be produced the output
  - for R scripts: you can save the info as an object and save it as an RDS file (I recommend saving it alongside the relevant script, with the same name plus `session_info` or something of the like)

```
1 my_session <- sessionInfo()  
2 saveRDS(my_session, file = here("scripts", "03-analyses", "phoneme_analyses-session_info.rds"))
```

- or run it, copy-and-paste the output in the script, and comment it all out




# 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/Knit 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

Follow the instructions on the workshop website: [Hands-on: working with Quarto](#)

# Topics

- modular analyses and literate programming 
- create and render a dynamic report with Quarto 
- documenting your dependencies 



# Session Info

My session info.

```
1 sessionInfo()
```

R version 4.4.1 (2024-06-14)

Platform: aarch64-apple-darwin20

Running under: macOS Sonoma 14.6

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; LAPACK version 3.12.0

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 MASS datasets utils methods base

# References

- 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>
- Knuth, D. (1984). Literate programming. *The Computer Journal*, 27(2), 97–111.
- Nagler, J. (1995). Coding Style and Good Computing Practices. *PS: Political Science & Politics*, 28(3), 488–492. <https://doi.org/10.2307/420315>