Quarto for reproducible research workflows and academic publishing

A step-by-step tutorial

Elen Le Foll

26 January 2026

Table of contents

Learning objectives

Quarto is a modern open-source publishing system that turns plain text and code into publication-ready papers, slides, websites, books, and teaching materials. It unifies writing and computation, supports seamless reproducible workflows, and bridges the gap between conducting, writing, and communicating research.

This step-by-step introduction to Quarto for academic research and publication covers:

- the concept of literate programming
- why reproducibility matters
- how to use Quarto in RStudio
- how to write research reports, theses, and academic papers in Quarto
- how to insert tables, images, plots, and bibliographic references in Quarto documents
- how to export and share Quarto documents in different formats including HTML, PDF, LibreOffice Writer, and Microsoft Word.

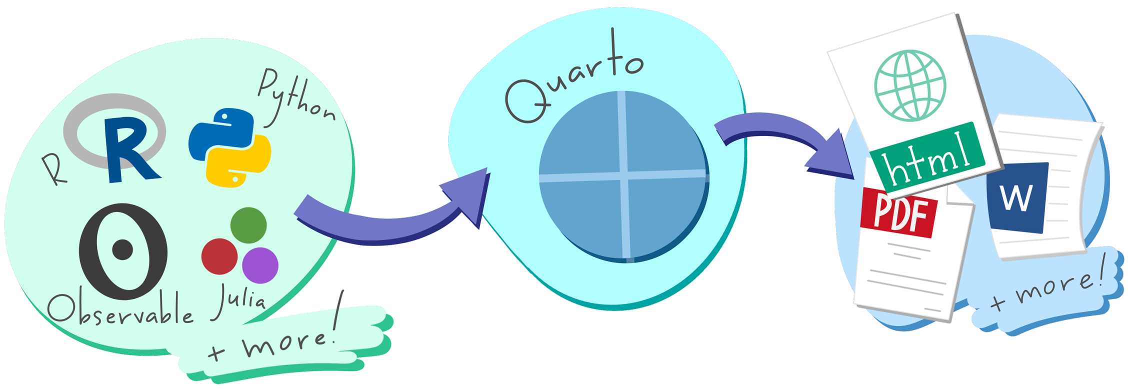


Figure 1: A schematic representation showing Quarto can understand multi-language input and produce multi-format outputs (Artwork CC BY 4.0 Allison Horst from the "Hello, Quarto" keynote by Julia Lowndes and Mine Çetinkaya-Rundel, first presented at the RStudio Conference 2022)

How to use

This introduction to Quarto may be used for self-study or as teaching material as part of a course or workshop.

A PDF and an online HTML version are available. You are encouraged to try things out and to answer the quiz questions (available in the online version only) to test your understanding as you go along. Once you've worked your way through this guide, you'll be ready to start writing your own term paper, dissertation, thesis, research report, or journal article in Quarto!

Educators are encouraged to adapt this Open Educational Resource (OER) to their students' needs and interests. The guide was itself written in Quarto and the source code is available for editing and reuse on Zenodo. The OER is shared under a CC BY 4.0 license. Any reuse or adaptation should include the following reference:

> Le Foll, Elen. 2026. Quarto for reproducible research workflows and academic publishing. Open Educational Resource. https://doi.org/10.5281/zenodo.18913131. https://elenlefoll.quarto.pub/quarto4research/ (last accessed DATE).

The present OER is an adaptation of "RepRoducible research and academic wRiting in Quarto" from Le Foll (2025: Chapter 14).

! Downloading the tutorial data

This tutorial is based on real research data analysed in:

> Dąbrowska, Ewa. 2019. Experience, Aptitude, and Individual Differences in Linguistic Attainment: A Comparison of Native and Nonnative Speakers. Language Learning 69(S1). 72-100. https://doi.org/10.1111/lang.12323.

We will work with the original datasets which correspond to data collected in England from English native speakers (L1 data) and speakers of English as a second language (L2 data). The author, Ewa Dąbrowska, generously shared her data on a public repository for applied linguistics research, IRIS.
Whilst you could download the two CSV files directly from the IRIS database, for the purposes of this tutorial, I recommend downloading a compressed (ZIP) file containing both datasets and the folder structure that we will use throughout this tutorial instead:

- Download `LearningQuarto.zip` from: https://doi.org/10.5281/zenodo.18913131.

Once you have downloaded the ZIP file, make sure that you decompress it (see Le Foll, 2025: Section 2.3) before proceeding. Do not open the CSV files in Microsoft Excel as this could potentially permanently damage the files (see e.g. Le Foll, 2025: Section 2.6 for further information)!

1 Literate programming

At its core, literate programming is about combining text, code, and code outputs (i.e. tables, statistics, and plots) within a single document that can be exported into different formats for sharing and publishing. If you've ever found yourself copying-and-pasting statistics, tables, or plots from different software into a text document containing your manuscript, you will know how error-prone that process is and how frustrating it is to have to repeat the operation every time something gets updated further up the research pipeline. Applying literate programming to academic writing avoids such situations altogether. It helps us to conduct and communicate our research in a more reliable and efficient manner.

Literate programming can be implemented in different authoring formats. Up until very recently, the most common format for R projects was R Markdown. For Python projects, Jupyter Notebooks remains the standard to date. This guide, however, focuses on Quarto, a relatively new open-source scientific and technical authoring and publishing system that has the advantage of supporting many different programming languages. This means that code in R, Python, Julia, and many other languages can be combined into one document, making project management and collaboration much easier. Quarto also allows us to readily export (or 'render') our documents to multiple formats such as HTML, PDF, and Word (see Figure 1 and Section 3).

Literate programming was popularised for data science, but is particularly well suited to academic research and teaching. Fun fact: This guide was written in Quarto! I chose this format because it allows for the seamless combination of explanations with nicely formatted code and code outputs. It also automatically generates consistent section and figure numbers, cross-references, bibliographic references, and much more. The `.qmd` source document is available on Zenodo and you are encouraged to explore it to learn more about the many possibilities that Quarto offers students, researchers, and data scientists.

Summary

Quarto documents are designed to:

1. Help you **collaborate** with other researchers (including your future self!) who are interested in both reproducing your results and understanding how you reached them (i.e. the code).
2. Provide you with a **convenient environment** in which to do research - a kind of "modern-day lab notebook where you can capture not only what you did, but also what you were thinking" (Wickham et al., 2023).
3. Help you **communicate** your research with others, including those who are not familiar with any programming languages.

1.1 Getting started with Quarto

We will be writing Quarto documents from the *RStudio* IDE[1], which conveniently ships with a version of Quarto, meaning that no additional installation[2] is required for you to use Quarto in *RStudio*. To get started with Quarto in *RStudio*:

[1]Many other IDEs (Integrated Developer Environments) support Quarto, including JupyterLab, Neovim, Positron, and VS Code. Feel free to pick the IDE that you are most comfortable with! The official Quarto guide offers a great starting point to author manuscript in Jupyter and VS Code, as well as RStudio.

[2]If you need to install Quarto, go to https://quarto.org/docs/get-started/ and download the latest Quarto version that is compatible with your operating system. Once the download is completed (which may take several minutes), double-click on the installer file that you downloaded and click your way through the installation process.

1. Install both R and *RStudio* on your computer. Detailed installation and recommended set-up instructions can be found in Le Foll (2025): Section 4.2.
2. Open *RStudio* and create a new Project by selecting *File > New Project...* in the main menu, or by clicking the "Create a project" button in the upper toolbar.
3. A dialogue menu will appear. You should already have downloaded and decompressed `LearningQuarto.zip` (see above), which means that we can select the second option "Existing Directory" (see Figure 2).

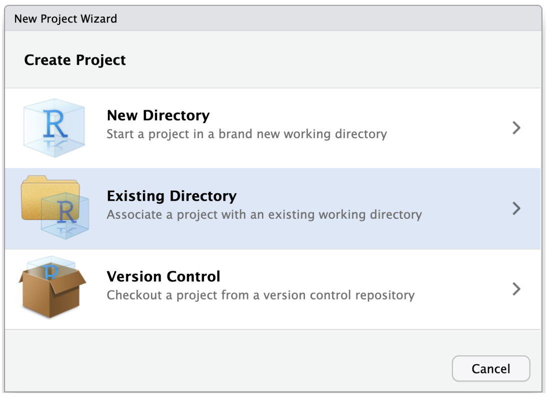
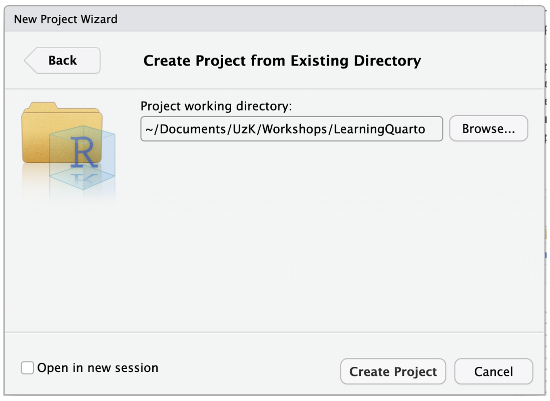

New Project Wizard
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

Figure 2: Selecting the project type



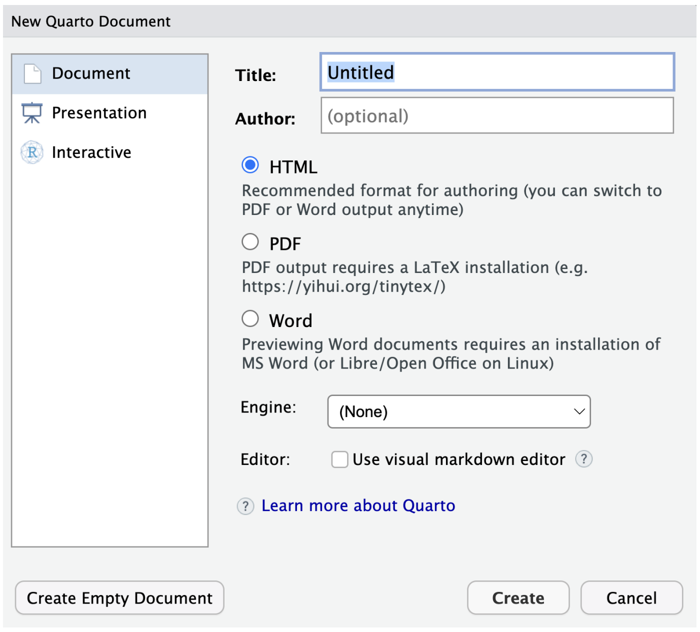
Project working directory:
~/Documents/UzK/Workshops/LearningQuarto

Figure 3: Defining the directory path

4. Click "Browse" to navigate to the location of the project directory "Learning Quarto" on your own computer and then click "Create Project" (Figure 3).
5. Then, create a new Quarto document by navigating to *File > New File > Quarto Document...*, or clicking on the "new document" button and selecting "*Quarto Document...*". A dialogue menu will appear (Figure 4). Leave everything as is and simply click on"Create" at the bottom.
6. *RStudio* has now opened a new, untitled Quarto file (`.qmd`). Depending on your settings, your new Quarto document may include some template material, which you can delete. Change the title of your Quarto document (which is not the same as its filename!) and add three further lines to the **document header** by copying and pasting the following lines at the top of the document, replacing the default header. Quarto document headers[3] are written in **YAML** which, I kid you not, stands for *Yet Another Markup Language*!

```
---
title: Learning Quarto
subtitle: "by reproducing the descriptive statistics of Dąbrowska's (2019)
 ↪  study"
author: Write your name here
```

[3]Note that, in YAML syntax, character strings that include special characters (e.g. `'`) need to be enclosed in quotation marks.



Title: Untitled
Author: (optional)

[✓ HTML]
Recommended format for authoring (you can switch to PDF or Word output anytime)

PDF
PDF output requires a LaTeX installation (e.g. https://yihui.org/tinytex/)

Word
Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux)

Engine: (None)

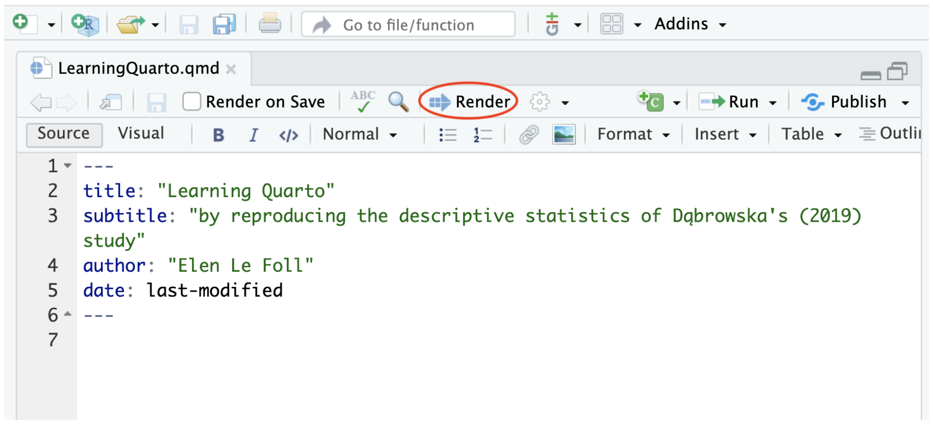
Editor: ☐ Use visual markdown editor

Learn more about Quarto

Figure 4: Creating a new Quarto document in *RStudio*

```
date: last-modified
---
```

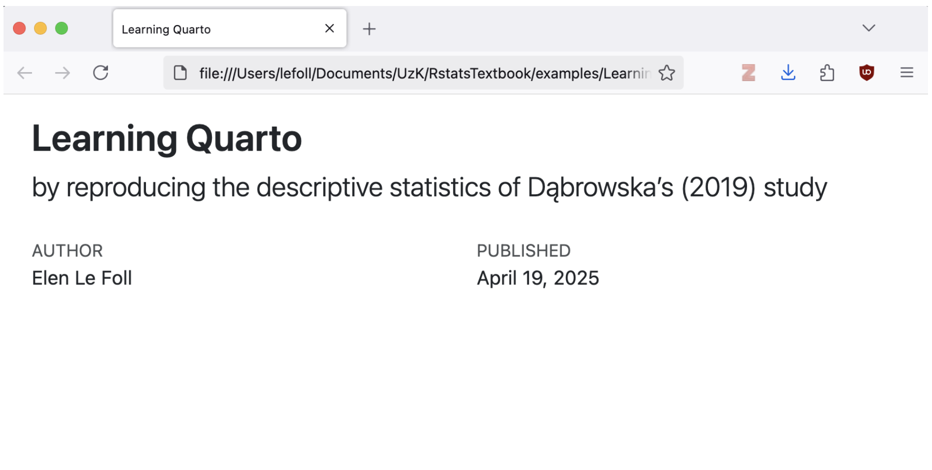
6. Click on the "save" button in the menu bar or navigate to File > Save to save your .qmd file. You will be prompted to give it a name. This could be `LearningQuarto.qmd` (see Navarro (2022) for excellent tips on how to name and organise files).
7. To check your Quarto installation, render your document by either selecting *File > Render Document* in the main menu, or clicking on "Render" button in the Quarto menu bar (see Figure 5). Your `.qmd` file will automatically be rendered to HTML (Quarto's default rendering format).



```
---
title: "Learning Quarto"
subtitle: "by reproducing the descriptive statistics of Dąbrowska's (2019)
study"
author: "Elen Le Foll"
date: last-modified
---
```

Figure 5: The `.qmd` file as opened in the Source pane of *RStudio*

8. Navigate to the folder where you saved your `.qmd` file to find the rendered HTML file. You can use a Finder (on macOS) or File Explorer window (on Windows) or go to the "Files" pane in *RStudio* to do this. The rendered version of your file will have the same filename as your Quarto document, but with the file extension `.html` (e.g. `LearningQuarto.html`). If you open on the file, it will appear in your default web browser (e.g. Firefox, Chrome, Safari). You should see that the HTML document features the title of your document, your name as the author, and today's date (see Figure 6). For now, the document is empty. In the next sections, you will learn how to add text, code, and code outputs to your Quarto document.


Learning Quarto

file:///Users/lefoll/Documents/UzK/RstatsTextbook/examples/Learnir

Learning Quarto

by reproducing the descriptive statistics of Dąbrowska's (2019) study

AUTHOR
Elen Le Foll

PUBLISHED
April 19, 2025

Figure 6: The `.html` file as opened in a web browser

1.2 Visual editor

You may have noticed that *RStudio* proposes two different modes in which Quarto documents can be edited: **Source** and **Visual** (see Figure 7).

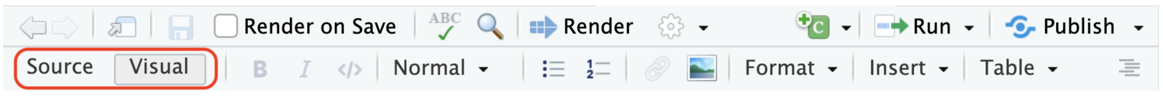


Figure 7: Source and Visual mode in RStudio

The Visual mode offers a WYSIWYM (What You See Is What You Mean) authoring experience. This means that, in Visual model, you will immediately see the effect of your formatting on screen. For example, to format a word in italics, you can click on the corresponding button in the toolbar (see Figure 7) or use the keyboard shortcut (Cmd/Ctrl + I) — just like you would in text-processing software — and this will immediately display the text in italics.

Note

To make writing in Quarto more convenient and less error-prone, you can switch on a **spell-checker** within *RStudio*. To do so, go to *Tools > Global Options... > Spelling*. You may need to restart *RStudio* for the change to take effect.

1.3 Markdown text

Writing and formatting text in *RStudio*'s Visual editor is very similar to writing in a word-processing software such as LibreOffice Writer or Microsoft Word. In the background, however, *RStudio* automatically converts your formatted text to Markdown in the underlying source code of your `.qmd` file. Markdown is a **plain-text format**. For example, in Markdown, words in italics are enclosed in asterisks like this: `*italics*`. Table 1 displays the Markdown syntax for other formatting options commonly used in academic writing.

The best way to get the hang of Markdown is simply to try things out. You will also find a handy cheatsheet under *Help > Markdown Quick Reference*. Remember that you can always go back to the **Visual** mode to format your text if that's easier for you. When it comes to debugging any Quarto syntax errors, however, it's usually easier to catch these in plain text, so you'll typically want to switch to the **Source** mode for that.

Markdown is gaining in popularity and is now widely supported across many platforms, from text editors to content management systems, ensuring that your formatting remains consistent and portable. It's increasingly used to write documentation, blog posts, or simply to take notes. Check out the Markdown Guide to learn more.

Table 1: Commonly used formatting options and their markdown syntax (adapted from the official Quarto Guide)

Markdown syntax	Rendered output
`*italics*`	*italics*
`**bold**`	**bold**
`***bold italics***`	bold italics
`superscript^2^ / subscript~2~`	superscript^2 / subscript$_2$
`~~strikethrough~~`	~~strikethrough~~
`` `verbatim code` ``	`verbatim code`
`# Heading 1`	**Heading 1**
`## Heading 2`	**Heading 2**
`### Heading 3`	**Heading 3**
`<https://quarto.org>`	https://quarto.org
`[Quarto guide](https://quarto.org)`	Quarto guide

Markdown syntax	Rendered output
`* bullet-point list`	• bullet-point list
`  + sub-item 1`	– sub-item 1
`  + sub-item 2`	– sub-item 2
`    - sub-sub-item 1`	∗ sub-sub-item 1
`1. numbered list`	1. numbered list
`2. item 2`	2. item 2
`   i. sub-item 1`	i. sub-item 1
`      A.  sub-sub-item 1`	A. sub-sub-item 1

1.1 Code chunks

To run code inside a Quarto document, we need to insert a code chunk. There are three ways to do so:

1. Using the keyboard shortcut Cmd/Ctrl + Option/Alt + i
2. Clicking on the green “Insert chunk” button icon in the editor toolbar
3. Manually typing the chunk delimiter ```` ```{r} to begin the code chunk and ``` ```` to close it.

In the code chunk below, `{r}` tells Quarto that this chunk is written in the programming language `R`. If we wanted to embed a chunk of Python code, we’d have to begin it with ```` ```{python} ```` instead.

~~~
```{r}
plot(1:10)
```
~~~

Using one of the three aforementioned options, insert the above R code chunk in your document. It is definitely worth learning the keyboard shortcut as it will save you a lot of time in the long run!

To run code within a Quarto document, we can either run:

- each individual line of code using the keyboard shortcut Cmd/Ctrl + Enter or
- the entire code chunk either by clicking the “Run” ▶ icon or with the shortcut Shift + Cmd/Ctrl + Enter.

Try running the R code chunk that you just inserted and see what happens.[4] *RStudio* will execute the code and display the outcome of the code either within your document (below the chunk) or in the Console pane, depending on your *RStudio* settings.[5]

Chunk output can be customised with **chunk options**. There are many options to choose from, but the most important options control whether a code block should be executed when the Quarto document is rendered and what results are inserted in the rendered version:

- `eval: false` prevents code from being evaluated. Given that the code is not run, no code outputs are generated either.
- `include: false` runs the code, but does not show the code or its outputs in the rendered document. This option is useful for code chunks that are not informative to the readers of your document.
- `echo: false` prevents the code from appearing in the rendered document, but displays the code outputs. This option is useful when you want to present the results of your analyses to people who are not interested in the underlying code.
- `echo: fenced` prints the code chunks in the rendered document together with the fenced code chunk delimiter (e.g. ` ```{r} `). This option is useful when creating pedagogical materials on how to write code chunks in Quarto (see e.g. source code of this tutorial).
- `message: false` or `warning: false` prevents messages or warnings from appearing in the rendered document.

It is also possible to label code chunks using the `label` option (see code chunk below). This can help to navigate long Quarto documents using the drop-down menu available in the bottom-left corner of the Source pane (see Figure 8). It also helps to quickly identify which code chunk is causing errors during rendering. Chunk labels should be short but meaningful.

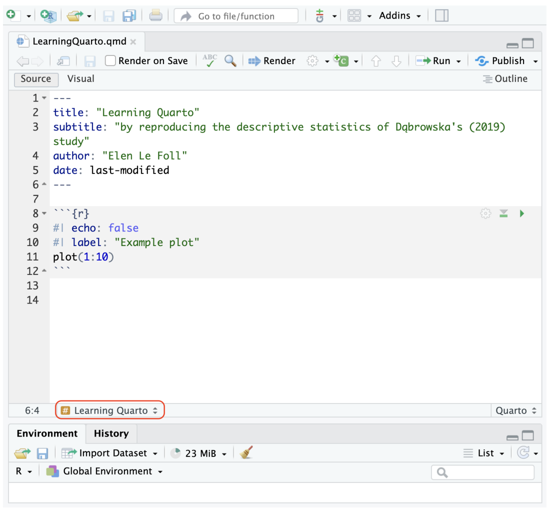
~~~
```{r}
#| echo: false
#| label: "Example plot"
plot(1:10)
```
~~~

The chunk above has two chunk options. The first means that only the chunk output, the plot, will be displayed in the rendered version of the Quarto document, not the code. The second is the chunk label.

[4]If you get the following error message `Error in plot.new() : figure margins too large`, this is because your Plots pane in RStudio is either hidden from view or too small for the plot to be printed there. Enlarge this window and then, re-type the command `plot(1:10)` in the Console pane and press enter again (see also Le Foll, 2025: Section 4.3.2).

[5]You can change this behaviour in your *RStudio* preferences under *Tools* > *Global Options* > R Markdown by selecting or unselecting the option: “Show output inline for all R Markdown documents”.

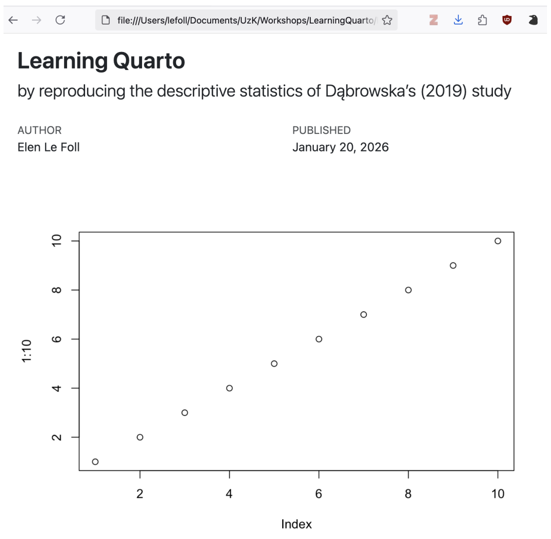
In *RStudio* the easiest way to set a chunk option is by clicking the gear icon in the top right corner of the chunk that you want to modify. This way, you can both choose a label and set chunk options. If you prefer to write code chunk options manually, these are placed at the top of the corresponding chunk following `#|`, as in the chunk above and Figure 8. As you can see in Figure 9, the `echo: false` chunk option means that the rendered document includes the chunk output, but not the code itself.



~~~
---
title: "Learning Quarto"
subtitle: "by reproducing the descriptive statistics of Dąbrowska's (2019)
study"
author: "Elen Le Foll"
date: last-modified
---

```{r}
#| echo: false
#| label: "Example plot"
plot(1:10)
```
~~~

Figure 8: Quarto document in Source mode in *RStudio*


file:///Users/lefoll/Documents/UzK/Workshops/LearningQuarto/
1:10
2
4
6
8
10
Index

Learning Quarto

by reproducing the descriptive statistics of Dąbrowska’s (2019) study

AUTHOR
Elen Le Foll

PUBLISHED
January 20, 2026

Figure 9: Rendered HTML document opened in a browser

1.2 Inline code

So far, we have seen how we can insert and format text in Quarto and how we can add code chunks with various options. But, to make the most of literate programming, we want to combine the two.

Copy the following two R chunks into your Quarto document to:

a. check that the packages required for this project are installed and then load them and
b. load the Dąbrowska (2019) data so that they may be used in your Quarto document.

The first chunk has been set to `include: false`, which means that the packages will be loaded, but nothing will appear in the rendered version of the document.

~~~
```{r}
#| label: "Package installation and loading"
#| include: false

# List of the packages necessary for this Quarto document:
packages <- c("here", "tidyverse", "xfun")

# Function to install the packages that are not yet installed:
installed_packages <- packages %in%
  rownames(installed.packages())

if (any(installed_packages == FALSE)) {
 ↪  install.packages(packages[!installed_packages], repos =
 ↪  "https://packagemanager.rstudio.com/all/latest") }

# Function to load all the packages at once:
lapply(packages, library, character.only = TRUE)
```
~~~

As the `import-data` chunk requires the `here()` function, it must come *after* the `setup` chunk because, when the document is rendered, code chunks are executed in the order that they appear. If the {here} library is not loaded *before* the data are imported, the rendering process will be aborted and an error message will be displayed in the Console. In the chunk below, the `here()` function tells R that it can find the two CSV files in a subfolder of the main project directory (i.e. folder) called "data". If you are running into errors when trying to load the data, make sure that this is where the files are stored on your own computer or adjust the relative paths, as necessary.

~~~
```{r}
#| label: import-data
#| include: false
~~~

```
L1.data <- read.csv(file = here("data", "L1_data.csv"))
L2.data <- read.csv(file = here("data", "L2_data.csv"))
```
```

To begin, we will reproduce the following basic descriptive statistics about the two datasets:

> "Ninety native speakers (42 male and 48 female) and 67 nonnative speakers of English (21 male and 46 female) were recruited through personal contacts, church and social clubs, and advertisements in local newspapers" (Dąbrowska, 2019: 5).

These are "tidy" datasets (Horst & Lowndes, 2020), which means that the number of native and non-native participants corresponds to the number of rows in the corresponding dataset:

```
nrow(L1.data)
```

```
[1] 90
```

```
nrow(L2.data)
```

```
[1] 67
```

In Quarto, we can use **inline code** to dynamically insert these numbers into our paragraph. Inline code in R begins with `` `{r} `` and ends with a single backtick `` ` ``.[6] It is best to use the Source mode to insert inline code. Using the Source mode, add the following section to your Quarto document and render it to HTML.

```
## Descriptive statistics about the participants

`{r} nrow(L1.data)` native speakers and `{r} nrow(L2.data)` nonnative
↪  speakers of English were recruited through personal contacts, church
↪  and social clubs, and advertisements in local newspapers.
```

The rendered version should read like this (if you are obtaining different numbers, this either means that you have tempered with the original data files or that they have been corrupted)[7]:

[6] If you look at the source code of this tutorial, you may notice that it sometimes makes use of double curly brackets `{r}` to introduce inline code. This syntax is only necessary in a tutorials: it means that the rendered version of the document will include the code fencing `{r}` in the output (similar to the `echo: fenced` option for code chunks, see Section 1.1).

[7] Using Microsoft Excel to open these `.csv` files can corrupt the files. This can happen even if you did not open Excel yourself (e.g. on some Windows computers, this is sometimes done automatically as part of the download process). To find out more, see Le Foll (2025: Section 2.6)
```

> **Descriptive statistics about the participants**
>
> 90 native speakers and 67 nonnative speakers of English were recruited through personal contacts, church and social clubs, and advertisements in local newspapers.

Inline code should only be used for very simple code, ideally with no more than one function, as in `` `{r} nrow(L1.data)` ``. To insert the output of more complex operations, it is best to write the code and temporarily save its output(s) in a hidden code chunk (using the option `#| include: false`, see Section 1.1).

~~~
```{r}
#| label: L1-gender
#| include: false

L1.males <- L1.data |>
  filter(Gender == "M") |>
  count()

L1.females <- L1.data |>
  filter(Gender == "F") |>
  count()
```
~~~

The saved objects (`L1.males` and `L1.females`) each contain one number. They can therefore be directly called within the text as inline code:

```
`{r} nrow(L1.data)` native speakers (`{r} L1.males` male and `{r}
↪ L1.females` female) and `{r} nrow(L2.data)` nonnative speakers of
↪ English were recruited through personal contacts, church and social
↪ clubs, and advertisements in local newspapers.
```

When rendered, the paragraph will read:

> 90 native speakers (42 male and 48 female) and 67 nonnative speakers of English were recruited through personal contacts, church and social clubs, and advertisements in local newspapers.

If we want to start our paragraph with “90” written in as a word rather than as digits, we can use the `numbers_to_words function()` function from the {xfun} package. First, let’s test that the function works by running the following line of code:

```
numbers_to_words(nrow(L1.data))
```

```
[1] "ninety"
```

To start our paragraph with a capital letter, we'll need to set the function's `cap` argument to `TRUE`.

```
`{r} numbers_to_words(nrow(L1.data), cap = TRUE)` native speakers (`{r}
↪ L1.males` male and `{r} L1.females` female) and `{r} nrow(L2.data)`
↪ nonnative speakers of English...
```

Next, we want to reproduce the following descriptive statistics about the L1 participants:

> "The L1 participants were all born and raised in the United Kingdom and were selected to ensure a range of ages, occupations, and educational backgrounds. The age range was from 17 to 65 years ($M = 38$, $SD = 16$)" (Dąbrowska, 2019: 5).

We can use the base R functions `min()`, `max()`, `mean()`, and `sd()` to compute these values.

```
The L1 participants were all born and raised in the United Kingdom and were
↪ selected to ensure a range of ages, occupations, and educational
↪ backgrounds. The age range was from `{r} min(L1.data$Age)` to `{r}
↪ max(L1.data$Age)` years (*M* = `{r} mean(L1.data$Age)`, *SD* = `{r}
↪ sd(L1.data$Age)`).
```

The rendered document will read:

> The L1 participants were all born and raised in the United Kingdom and were selected to ensure a range of ages, occupations, and educational backgrounds. The age range was from 17 to 65 years ($M = 37.5444444$, $SD = 16.148998$).

Whilst these values are correct, in practice, we want to round them off to the nearest integer. To this end, we can wrap the `round()` function around the `mean()` and `sd()` function (see Le Foll, 2025: Section 7.5).

```
The L1 participants were all born and raised in the United Kingdom and were
↪ selected to ensure a range of ages, occupations, and educational
↪ backgrounds. The age range was from `{r} min(L1.data$Age)` to `{r}
↪ max(L1.data$Age)` years (*M* = `{r} round(mean(L1.data$Age))`, *SD* =
↪ `{r} round(sd(L1.data$Age))`).
```

The rendered document will read:

> The L1 participants were all born and raised in the United Kingdom and were selected to ensure a range of ages, occupations, and educational backgrounds. The age range was from 17 to 65 years ($M = 38$, $SD = 16$).

ℹ Counting words

If you need to adhere to a specific word count, *RStudio* has a useful function to count the number of words you have written, excluding the YAML header and all code chunks. In *RStudio*'s top menu, click on "Edit" and select "Word Count" to find out how many words you've written so far.
You may also want to check out Andrew Heiss' Quarto extension that computes different types of word counts (e.g. including or excluding references) and optionally prints them in the rendered versions of your Quarto documents.

1.3 Tables

The easiest way to manually construct a table in a Quarto document in *RStudio* is to switch to Visual mode and click on *Insert > Table*. You can choose how many rows and columns you need and then fill in your table in the Visual editor.

Table 2: Terminology used in this chapter (see Section 2)

	Same data	**Different data**
Same analysis method	Reproducible	Replicable
Different analysis method	Robust	Generalisable

When you switch to the Source mode, you will see that, in Markdown (see Section 1.3), your table has been converted to a **pipe table**. Pipe tables allow for column alignment and captions.

```
|                               | **Same data** | **Different data** |
|-------------------------------|---------------|--------------------|
| **Same analysis method**      | Reproducible  | Replicable         |
| **Different analysis method** | Robust        | Generalisable      |

: Terminology used in this chapter (see @sec-Reproducibility)
```

Most of the time, however, you will want to display tabular results based on data that you have imported, wrangled, and/or analysed in `R`. If the output of a code chunk within your Quarto document is a table, it will be displayed in your rendered document by default (unless you specify a chunk option to hide its output, see Section 1.1).

```
L1.data |>
  count(OtherLgs,
        sort = TRUE)
```

Table 3: Example of a {gt} table

Additional language	N
None	84
German	3
French	2
Spanish	1

```
  OtherLgs  n
1     None 84
2   German  3
3   French  2
4  Spanish  1
```

However, this output is not particularly nicely formatted. There are several R packages designed to create tables that are ‘presentation-ready’. One of these is the **{gt}** package. Beyond its main function `gt()`, it offers many more functions to further style tables such as `cols_label()` to change the column headers. You will need to install this package before you can use it (see Section 2.1).

~~~
```{r}
#| label: tbl-L1-languages
#| tbl-cap: "Example of a {gt} table"
#| tbl-colwidths: [8,2]

#install.packages("gt")
library(gt)

L1.data |>
  count(OtherLgs,
        sort = TRUE) |>
  gt() |>
  cols_label(
    OtherLgs = "Additional language",
    n = "N")
```
~~~

In addition, Quarto also has a range of **chunk options** to customise the display of tables, including `tbl-cap` for the addition of a table caption and `tbl-cap-location` to determine where the caption is placed. Note that, in the above chunk, the table’s `label` chunk option begins with `tbl-`. This allows for in-text cross-referencing to the table with the insertion of `@tbl-L1-languages` within the text of the Quarto document, which will automatically be rendered as the following linked and numbered cross-reference: Table 3.

Going further

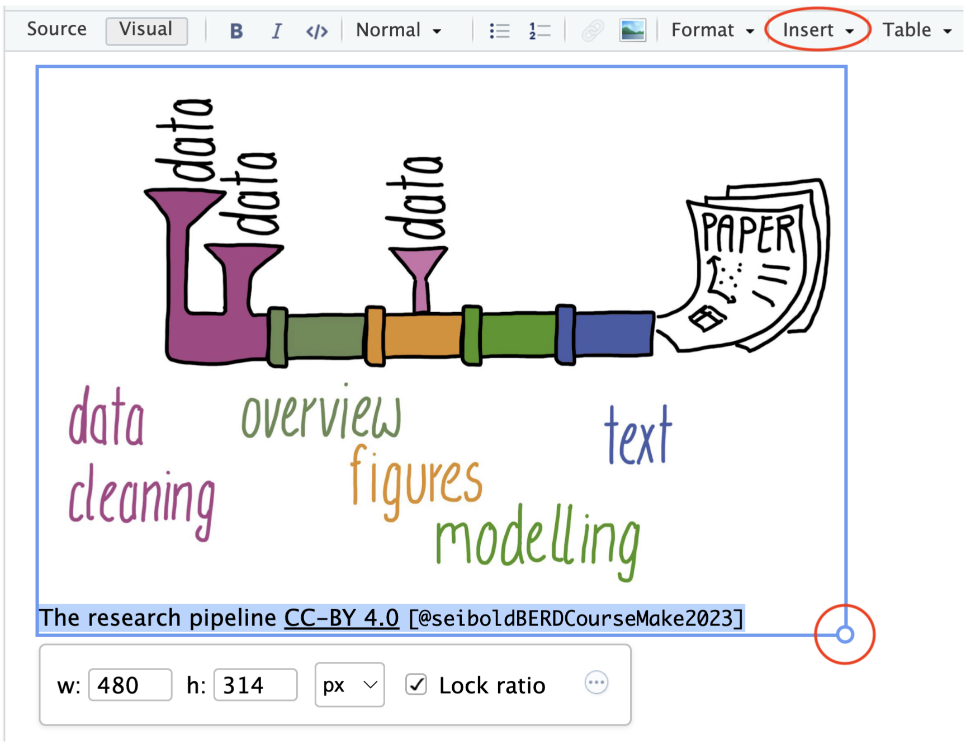
The Quarto guide provides further information about formatting tables.

1.4 Figures

In Quarto documents, figures can either be inserted from image files (e.g. `.png` or `.jpeg` files) or from the output of a code chunk (e.g. a plot).

1.4.1 Images

To embed an image from an external file, you can use the "Insert" menu in *RStudio*'s Visual editor and select "Figure / Image" (see Figure 10). This will open up a menu where you can select the image that you want to insert, as well as add **alt-text** W3C Web Accessibility Initiative (2022) and a **caption**. The easiest way to adjust the size of an embedded image is to click on the image and then adjust the size of the image with the blue circle in the bottom-right corner of the image (see Figure 10).



The research pipeline CC-BY 4.0 [@seiboldBERDCourseMake2023]

Figure 10: Adjusting the size of an image in Quarto

Below is the source code for Figure 11 in Markdown. The code includes the **relative path** to the image file relative to the **project directory** (see Section 1.1). In the example below, the image file `BERD_pipeline-real.jpg` is located in a subfolder called `images`. If you want to try this out yourself, you will need to create this subfolder within your own project directory and save Figure 11 to this subfolder.

```
![A more realistic research pipeline [[CC BY
↪  4.0](https://creativecommons.org/licenses/by/4.0/)
↪  @seiboldBERDCourseMake2023]](images/BERD_pipeline-real.jpg)
↪  {#fig-RealisticPipeline fig-alt="Cartoon drawing of a complex set of
↪  pipes with various entry points for \"data\" and a single output: a
↪  research paper with text, a table, and a plot. Sections of the pipe are
↪  coloured according to the processes that they correspond to. These
↪  include data cleaning, overview, figures, modelling, and text."
↪  width="480"}
```

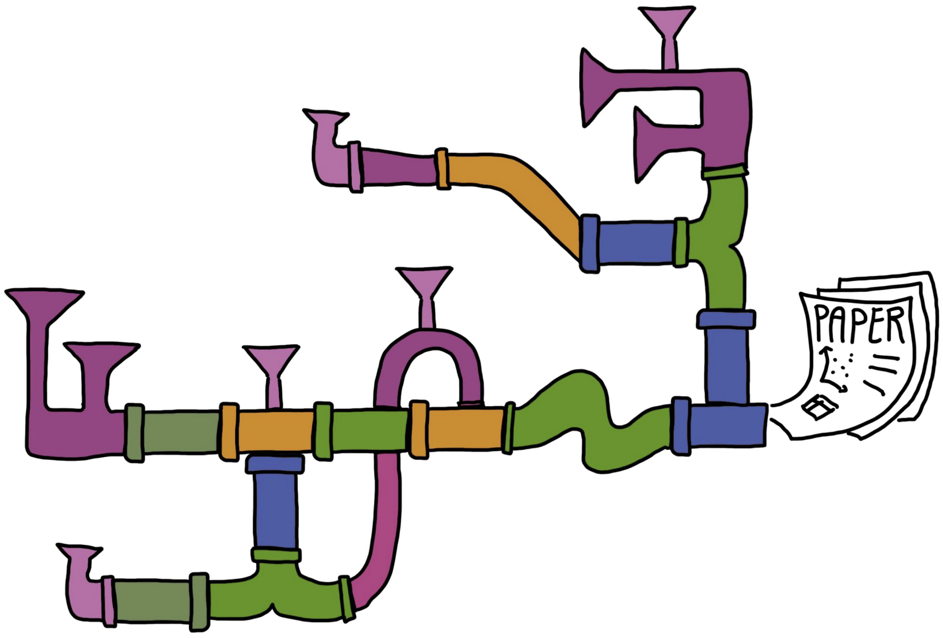


Figure 11: A more realistic research pipeline (CC BY 4.0 Seibold & Müller, 2023)

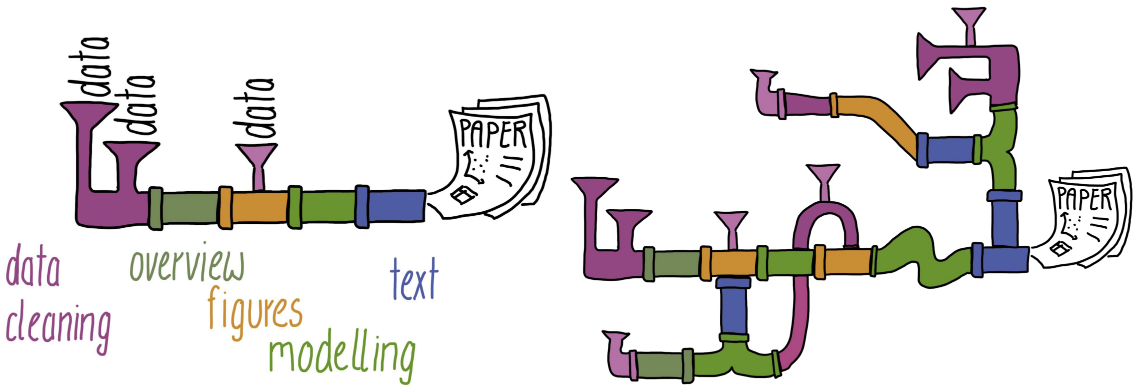
This example embedded image includes a caption (that, itself, includes a link), an alt-text, and a custom width in pixel. Note that, in the source code, special characters such as quotation marks need to be escaped using a backslash \. Tags beginning with `#fig-` can be used to cross-reference images by replacing the `#` with `@`. Hence, in this chapter, `@fig-RealisticPipeline` in the Quarto source code is rendered as Figure 11.

Figures can be arranged in many ways. The example below uses the `:::` **div syntax** to display two images side-by-side. This syntax also allows for subcaptions as shown in Figure 12.

```
::: {#fig-Pipelines layout-ncol="2"}
![An idealised research
↪  pipeline](images/BERD_pipeline-simple.jpg){#fig-IdealisedPipeline}

![A more realistic research
↪  pipeline](images/BERD_pipeline-real.jpg){#fig-RealisticPipeline2}

Research workflows as pipelines [[CC BY
↪  4.0](https://creativecommons.org/licenses/by/4.0/)
↪  @seiboldBERDCourseMake2023]
:::
```



(a) An idealised research pipeline (b) A more realistic research pipeline

Figure 12: Research workflows as pipelines (CC BY 4.0 Seibold & Müller, 2023)

ℹ Going further

To find out more about inserting and arranging figures, check out the detailed Quarto guide.

1.4.2 Plots

If your Quarto document includes code chunks that generate plots, they will automatically be integrated in your rendered document. Plots will either appear immediately after the corresponding code chunk or where the code chunk would be, if you chose to hide the code chunk that generated the plot with the `echo: false` option.

As with computed tables (see Section 1.3), various code chunk options can be added to customise the look of computed figures in rendered documents. Compare the code chunk options below and the generated output in Figure 13.

~~~
```{r}
#| label: fig-scatterplot
#| fig-cap: "L2 participants' lexical proficiency in English and their
↪ professional occupational group"
#| fig-height: 5
#| fig-asp: 0.618
#| message: false

L2.data |>
  ggplot(mapping = aes(x = VocabR,
                       y = CollocR)) +
  geom_point(aes(colour = OccupGroup),
             size = 2) +
  geom_smooth(method = "lm") +
  scale_colour_viridis_d() +
  labs(x = "Vocabulary test scores",
       y = "Collocation test scores",
       colour = "Occupational\ngroups") +
  theme_bw()
```
~~~

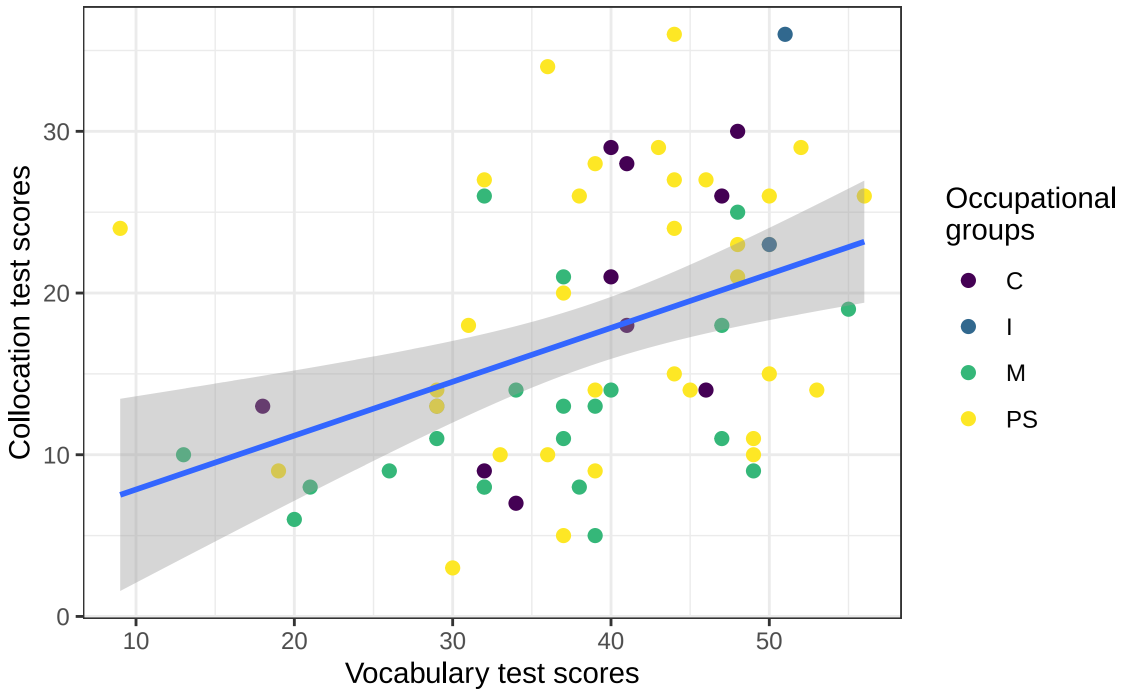


Figure 13: L2 participants' lexical proficiency in English and their professional occupational group

According to the authors of ‘R for Data Science’, figure sizing and scaling in R is “an art and science and getting things right can require an iterative trial-and-error approach” (Wickham et al., 2023). This is because there are five main options that control figure sizing: `fig-width`, `fig-height`, `fig-asp`, `out-width` and `out-height`. The first three control the size of the figure created by `R`, whereas the latter two control the size at which it is inserted in the rendered document.

If you are sharing your research analyses and results in HTML format, you can also embed **interactive plots** (see Le Foll, 2025: Section 10.2.8) in your Quarto documents. In HTML format, it is therefore possible to hover over Figure 14 to explore the data interactively.

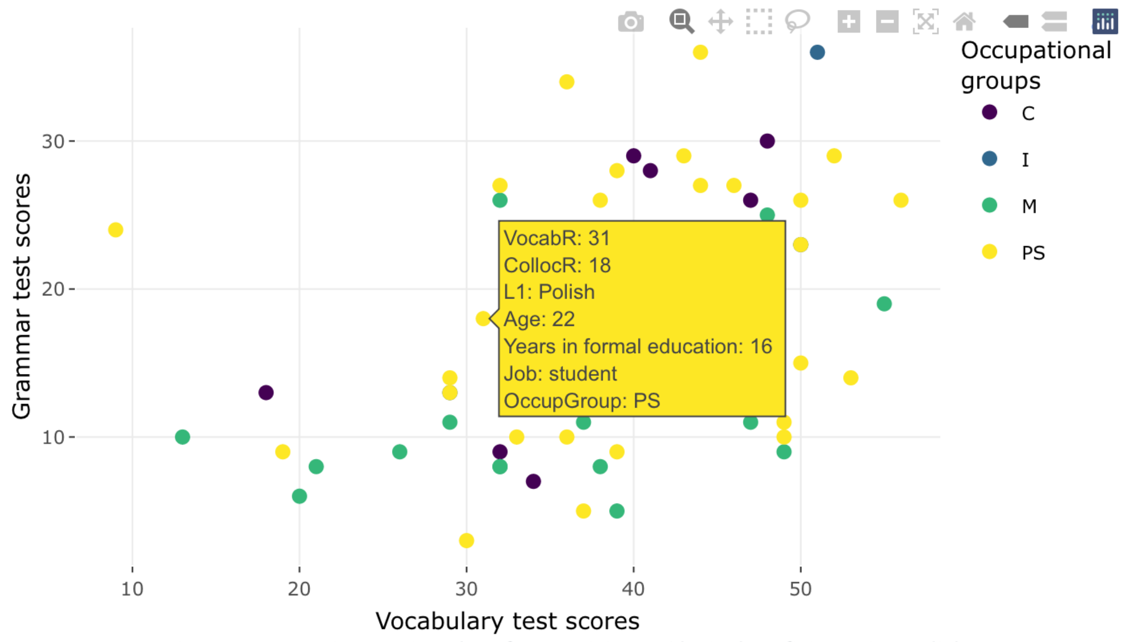


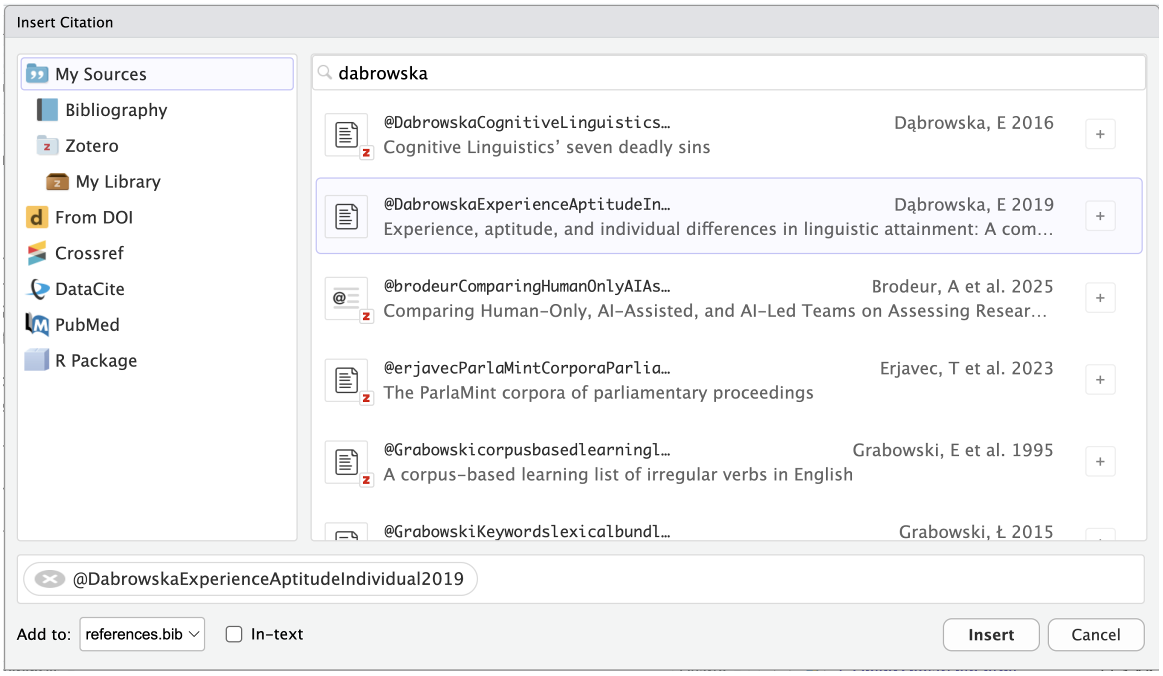
Figure 14.15: An interactive plot of L2 participants’ lexical proficiency in English

Figure 14: Screenshot of the interactive plot of L2 participants’ lexical proficiency in English

1.5 Citations and references

An important aspect of academic writing is the inclusion of in-text bibliographic references (**citations**) and a well-formatted list of references (also referred to as a **bibliography**). *RStudio*’s Visual editor makes inserting bibliographic references very convenient. To insert a reference, click on “Insert” and then select “Citation” or use the keyboard shortcut Cmd/Ctrl + Shift + F8. This opens up a menu (see Figure 15) giving you the option to search for the source that you’d like to cite on your own computer (e.g. in your own Zotero database, if you use Zotero), via the Crossref database, or directly using a DOI.

Alternatively, if you start typing `@` in the Visual editor, a quick reference menu will appear. Either way, any references that you add will be displayed as `@` followed by a **reference**



dabrowska

@DabrowskaCognitiveLinguistics… Dąbrowska, E 2016
Cognitive Linguistics' seven deadly sins

@DabrowskaExperienceAptitudeIn… Dąbrowska, E 2019
Experience, aptitude, and individual differences in linguistic attainment: A com…

@brodeurComparingHumanOnlyAIAs… Brodeur, A et al. 2025
Comparing Human-Only, AI-Assisted, and AI-Led Teams on Assessing Resear…

@erjavecParlaMintCorporaParlia… Erjavec, T et al. 2023
The ParlaMint corpora of parliamentary proceedings

@Grabowskicorpusbasedlearningl… Grabowski, E et al. 1995
A corpus-based learning list of irregular verbs in English

@GrabowskiKeywordslexicalbundl… Grabowski, Ł 2015

@DabrowskaExperienceAptitudeIndividual2019

Figure 15: *RStudio*'s search menu for bibliographic reference

identifier. For example, in the source code of this Quarto document, every reference to Dąbrowska (2019) is indicated as `@DabrowskaExperienceAptitudeIndividual2019`.

Note

For more information on how to format your in-text citations, see the Quarto guide.

When you insert your first reference in a Quarto document, *RStudio* will automatically create a `references.bib` file in your project folder. All references are automatically added to this new BibLaTeX file. As shown in the example `references.bib` file below, `.bib` files contain entries that begin with `@` followed by the type of reference (`article`, `book`, `manual`, `url`, etc.) and the reference identifier (e.g. `DabrowskaExperienceAptitudeIndividual2019`, `wickhamDataScienceImport2023`). The rest of the entries contains structured information about each reference including its title, date of publication, and DOI or ISBN.

Listing 1 `references.bib`

```
@article{
  DabrowskaExperienceAptitudeIndividual2019,
  title={Experience, Aptitude, and Individual Differences in Linguistic
   ↪  Attainment: A Comparison of Native and Nonnative Speakers},
  volume={69},
  ISSN={1467-9922},
  url={https://onlinelibrary.wiley.com/doi/abs/10.1111/lang.12323},
  DOI={10.1111/lang.12323},
  number={S1},
  journal={Language Learning},
  author={Dąbrowska, Ewa},
  year={2019},
  pages={72-100}
}

@book{
  wickhamDataScienceImport2023,
  place={Beijing, Boston, Farnham, Sebastopol, Tokyo},
  edition={2},
  title={R for Data Science: Import, tidy, transform, visualize, and model
   ↪  data},
  ISBN={978-1-4920-9740-2},
  url={https://r4ds.hadley.nz/},
  publisher={O'Reilly},
  author={Wickham, Hadley and Çetinkaya-Rundel, Mine and Grolemund,
   ↪  Garrett},
  year={2023}
}
```

In order to connect this `bibliography.bib` file with our Quarto document, we need to add a `bibliography` key to our YAML header (see Section 1.1). Provided that our `references.bib` file is located in the same folder as our Quarto document (which is what *RStudio* does by default), we can simply add the following line to our document header:

```
---
title: Learning Quarto
subtitle: "by reproducing the descriptive statistics of Dąbrowska's (2019)
 ↪  study"
author: Elen Le Foll
date: last-modified
bibliography: references.bib
---
```

With this modified YAML header, when the document is rendered, a bibliography will automatically be added to the end of the document. This means that, if you have citations in your document, it is a good idea to include a header section `# References` at the end of the document.

References

Dąbrowska, Ewa. 2019. "Experience, Aptitude, and Individual Differences in Linguistic Attainment: A Comparison of Native and Nonnative Speakers." *Language Learning* 69 (S1): 72-100. https://doi.org/10.1111/lang.12323.

Wickham, Hadley, Mine Çetinkaya-Rundel, and Garrett Grolemund. 2023. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data.* 2nd ed. O'Reilly. https://r4ds.hadley.nz/.

By default, Quarto will use the Chicago Manual of Style author-date citation format (as above). However, we can point to a different **citation stylesheet** in the form of a `.csl` (Citation Style Language) file in the YAML header. This allows us to determine exactly how our bibliography and in-text citations should be formatted. Many institutions, publishers, and journals have their own (sometimes annoyingly specific!) requirements. Luckily, the open-source research community has put together a large repository of citation stylesheets for you to choose from: https://www.zotero.org/styles. You can download any of these stylesheets (as a `.csl` file), place the file in your project folder, and then link it to your Quarto document by adding a `cls` key to your header.

```
---
title: Learning Quarto
subtitle: "by reproducing the descriptive statistics of Dąbrowska's (2019)
 ↪  study"
author: Elen Le Foll
date: last-modified
bibliography: references.bib
csl: international-journal-of-learner-corpus-research.csl
---
```

For example, if you wanted to submit your paper to the International Journal of Learner Corpus Research, you can download the corresponding CLS stylesheet from the Zotero styles database, save it in your project folder, and link to it in your YAML header as above. When rendered, your document's bibliography will then read:

References

Dąbrowska, E. (2019). Experience, Aptitude, and Individual Differences in Linguistic Attainment: A Comparison of Native and Nonnative Speakers. *Language Learning*, *69*(S1), 72-100. https://doi.org/10.1111/lang.12323.

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for data science: Import, tidy, transform, visualize, and model data* (2nd ed.). O'Reilly. Retrieved from https://r4ds.hadley.nz/.

Literature management

Managing the large number of references that we need to consult, read, and cite when doing research can be a real challenge. The good news is that **reference management software** are there to help you overcome this challenge! Whether you are working on a term paper, a Master's dissertation, PhD thesis, or post-doctoral project, it is *always* worth investing the time to learn to use a reference manager!

Zotero is a free and open-source bibliographic reference manager that will help you organise all your sources and generate beautifully formatted bibliographies for all your projects. It offers various browser extensions that enable you to quickly add references to your library directly from your web browser.

What's more, Zotero can be integrated in RStudio, making it very easy to include BibTeX-formatted references in your Quarto documents. Find out more in the *RStudio* documentation.

Combining Zotero and Quarto also allows you to generate annotated bibliographies. Benjamin Tjepkes explains how to do this in a detailed blog post.

2 Reproducible research

Not only is using Quarto (or any other literate programming format, see Section 1) highly convenient, it also helps us make our research more reproducible. Unfortunately, the terms **reproducible**, **replicable** and **repeatable** are often confused and, not helping matters, some definitions in the literature contradict each other.

This guide adopts the terminology of The Turing Way. We thus define **reproducibility** as the ability of an independent researcher or team to obtain the same results as in a study using the same data and methods as the original study (see Figure 16).

This is in contrast to **replicability**, where the same methods, but different data are used; and **robustness**, where the same data, but different methods are used. Finally, if a finding can be reliably observed across different datasets with different methods, then we can say that the finding is **generalisable**.

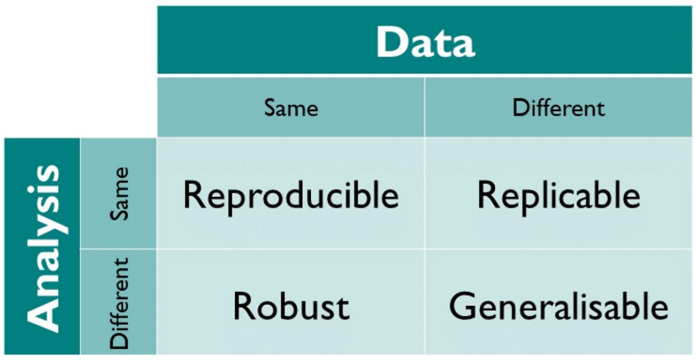


Figure 16: Defining *reproducibility* and related terms (CC BY 4.0 The Turing Way Community)

Given this definition, reproducibility might seem like a low bar to pass. You might be thinking: shouldn't it be obvious that we'll get the same results if we repeat a study using exactly the same data and method? Well, yes, it should be. But it very often isn't! For a start, to be able to even attempt to reproduce the results of a study, the underlying data must be available, which is not always the case even when the publication includes the phrase "data available upon (reasonable) request" (Hussey, 2025). Second, the data must be available in an accessible format and must be published together with enough documentation to be understandable to an independent researcher. Third, the author(s) of the original study need to have very diligently documented all their data wrangling and analyses steps. The best way to do this is undoubtedly to use code that does not require closed-source software (e.g. a researcher without a license for SPSS or Stata will not be able to run SPSS or Stata scripts). This open code must be shared in an accessible format, too. Fourth, independent researchers need to be able to run these scripts. To this end, it is important that they know exactly which tools were used. For example, if the analyses were conducted in R or Python, they need to know which R/Python version and which libraries and library versions were used (Section 2.1). They also need to know in which order the scripts were run and, finally, the scripts must run on their own computers without any errors. So now, reproducibility doesn't sound quite so easy, right? Luckily, if we apply the principles of literate programming in Quarto, we can go a long way towards ensuring that our research is reproducible.

Going further

To find out more about best practices for reproducible research, check out The Turing Way's excellent Guide for Reproducible Research.

2.1 Package versions and references

In addition to referencing academic papers, it is also very important that we reference which **`R` version** we used for our analyses and which **packages** and package versions. This serves two purposes:

1. Independent researchers (and our future selves!) know exactly what they need to be able to **reproduce** our analyses (see Section 2).
2. We give **credit** to the kind people who spent time and effort developing and sharing the R packages that we used for our analyses (LaZerte, 2021).

The easiest way to "give credit where credit is due" to `R` package developers is to use the {grateful} package. Its `cite_packages()` function will scan your project for all the `R` packages that are used and generate a BibTeX file called `grateful-refs.bib` that contains the package references. Having first installed the package, load the {grateful} library:

```r
#install.packages("grateful")
library(grateful)
```

Make sure that all the packages that your script relies on are loaded and then run the following command once to generate a bibliography of all loaded packages:

```r
cite_packages(out.dir = getwd(),
              omit = NULL)
```

The `.bib` text generated by the {grateful} library should now be in your Quarto project directory. Next, add a reference to this BibTeX file in your YAML header. This means that your Quarto document will now have be linked to two bibliography files, which is fine as long as you use the following YAML syntax to reference them both (watch the indentation!):

```yaml
---
bibliography:
  - references.bib
  - grateful-refs.bib
---
```

We can now call the `cite_packages(output = "paragraph")` function to generate a paragraph that mentions all the packages used in the document and add their references to the bibliography (either at the bottom of your rendered Quarto document or, in the case of this textbook, in the corresponding chapter, see Section 1.5).

```r
cite_packages(output = "paragraph",
              out.dir = getwd(),
              pkgs = "Session",
              omit = NULL)
```

We used R v. 4.5.2 (R Core Team, 2025) and the following R packages: checkdown v. 0.0.13 (Moroz, 2020), grateful v. 0.3.0 (Rodriguez-Sanchez & Jackson, 2025), gt v. 1.2.0 (Iannone et al., 2025), here v. 1.0.2 (Müller, 2025), tidyverse v. 2.0.0 (Wickham et al., 2019), xfun v. 0.55 (Xie, 2025).

Alternatively, `cite_packages()` can generate a table with all the package names, versions, and references. Table 4 lists the packages used in this chapter. To display functioning links and references, the table is rendered using the `kable()` function from the {knitr} package.

```
#install.packages("knitr")
pkgs <- cite_packages(output = "table",
                      out.dir = getwd(),
                      omit = NULL)
knitr::kable(pkgs)
```

Table 4

Package	Version	Citation
base	4.5.2	R Core Team (2025)
checkdown	0.0.13	Moroz (2020)
grateful	0.3.0	Rodriguez-Sanchez & Jackson (2025)
gt	1.2.0	Iannone et al. (2025)
here	1.0.2	Müller (2025)
tidyverse	2.0.0	Wickham et al. (2019)
xfun	0.55	Xie (2025)

2.2 Computing environment

Tracking the versions of the packages that your code relies on is important if you want your analysis code to be **reproducible** in the long-run (i.e. so that you or a colleague can run it next month or next year). However, manually installing these packages with these exact versions is hardly feasible. To simplify the process of re-creating the same **project environment**, consider using {renv} or {rix}.

The {renv} library (Ushey & Wickham, 2023) keeps track of the package versions that your project depends on, and ensures that those exact versions are installed whenever and wherever your project is opened. {renv} provides each project with its own isolated package library, ensuring that you can update packages in new projects without risking breaking older projects.

To create project-specific environments that additionally include system dependencies, you will need to check out the {rix} package (Rodrigues & Baumann, 2026). Both of these packages aim to make R projects more isolated, portable and therefore reproducible.

Going further

Accessible introductions to stabilising your computing environment can be found in the BERD course "Make Your Research Reproducible" (Seibold & Müller, 2023) and The Turing Way's guide to reproducible environments (Community, 2022).

2.3 Version control

Another powerful tool very much worth learning to improve your research workflows is version control with Git. Git can track changes to all our project documents over time, allowing us revert to previous versions whenever needed. For example, if we make a mistake or want to compare different versions of a Quarto document, Git can show us exactly what changes were made and when.

Git is essential when it comes to collaboration. It allows multiple project contributors to simultaneously work on the same document(s) without overwriting each other's edits. For instance, if you and a colleague are both editing a Quarto document, Git can help you merge your changes seamlessly. Conveniently, *RStudio* has built-in Git integration, facilitating the use of version control directly within our workflow. While a full hands-on introduction to Git is beyond the scope of this tutorial, learning Git has the potential to greatly improve your ability to manage and share your research effectively. There are many great resources to help you get started, e.g. the Software Carpertry' guide to Git for novices, Reproducibility with Git and Quarto, and Happy Git and GitHub for the useR.

Don't git muddled up!

Git and GitHub are often confused. Git is an open-source version control system. GitHub, by contrast, is a popular, proprietary web-based hosting service for Git repositories owned by Microsoft. In addition to easing collaboration, storing version-controlled projects in a (public or private) online repository is an excellent additional backup strategy for many research projects. Alternatives to GitHub include Codeberg and GitLab.

3 Sharing and publishing

The default rendering format for Quarto documents is HTML. HTML has many advantages and is ideally suited to online publications, but the beauty of Quarto is that you can share and publish your research in many other formats, too.

3.1 Sharing HTML documents

You may have noticed that, in addition to creating an `.html` file, rendering your Quarto document has also generated a folder containing any necessary data, images, stylesheets

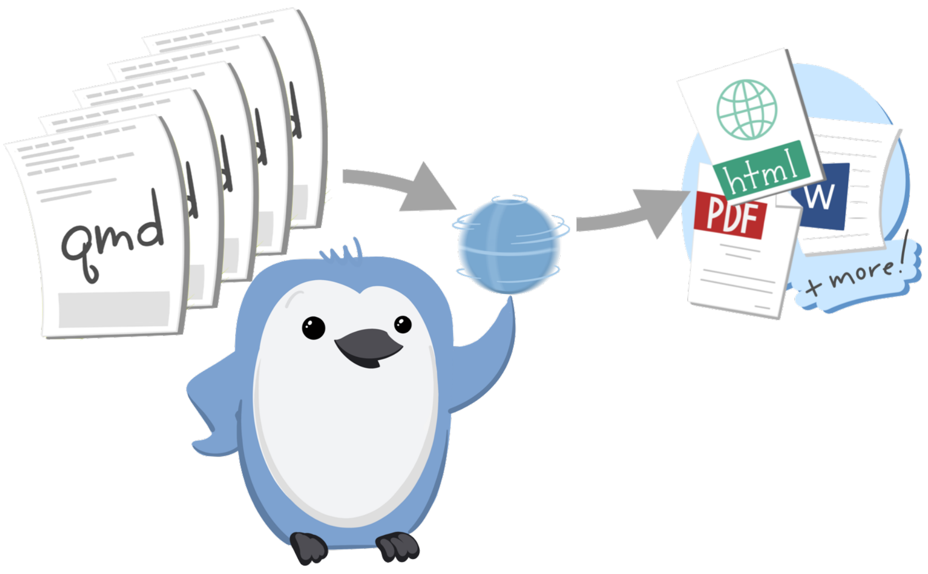


Figure 17: Artwork (CC BY 4.0) Allison Horst from the "Hello, Quarto" keynote by Julia Lowndes and Mine Çetinkaya-Rundel, presented at RStudio Conference 2022.

or other files required to display the HTML version of your document. This is because, by default, Quarto keeps external resources separate from the main HTML file. While this is advantageous for large documents and complex projects, it does mean that your HTML document can only be viewed if both the `.html` file and its associated folder are shared.

If you want to share a single, self-contained `.html` file with someone else, you will need to **embed** all the necessary files directly inside your HTML file. This is achieved by adding the following option at the end of your document's YAML header:

```
---
format:
  html:
    embed-resources: true
---
```

With this setting, Quarto will package all the necessary resources inside the HTML file, resulting in a self-contained document that is easy to share as it can be viewed in any web browser (e.g. Firefox, Google Chrome, Safari).

If you intend to share a longer Quarto document, it may be a good idea to number the headings and sub-headings (`number-sections`) and to include a table of content (`toc`). You can do this by adding the following two lines to the `format` section of your YAML header:

```
---
format:
  html:
    embed-resources: true
    number-sections: true
    toc: true
---
```

There are many ready-made HTML themes to choose from. The HTML version of this tutorial uses the `journal` theme in light mode, and `vapor` in dark mode.

3.2 Word, LibreOffice & co.

Your supervisor or colleague may request a **Microsoft Word** version of your Quarto document and, thankfully, this is no problem. You can change the rendering format to a `.docx` file by amending the format option in your YAML header:

```
---
format: docx
---
```

With this format option, rendering your Quarto document will generate a `.docx` file that includes your text, any code that you wanted to show in your document, and all of the code outputs that you wanted to share, such as your statistics, graphs, and tables. Some of the formatting options available for HTML also work in the `.docx` format:

```
---
format:
  docx:
    embed-resources: true
    number-sections: true
    toc: true
---
```

Note, however, that any options that are not available in the rendering format specified are ignored without warning or error messages.

⚠ Not rendering code chunks in specific formats

Dynamic code outputs, such as the **interactive** {plotly} graph displayed in Figure 14, cannot be meaningfully rendered to **static formats**, such as Microsoft Word or PDF. Attempting to do so can cause rendering errors such as:

```
Error: Functions that produce HTML output found in document
targeting docx output.
```

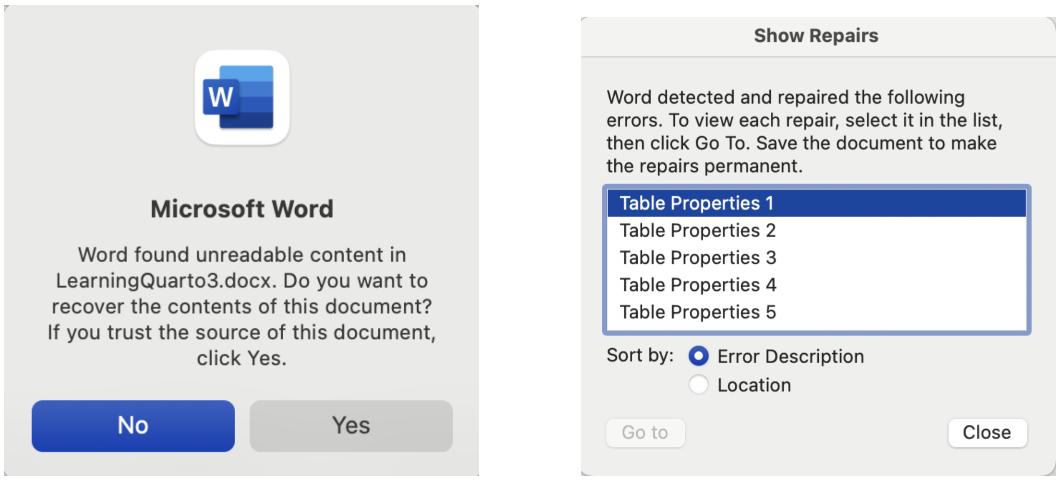
```
Please change the output type of this document to HTML.
```

To fix this, add the following options to any code chunk that generates content that only works in HTML:

~~~
```{r}
#| eval: !expr 'knitr::is_html_output()'
ggplotly(L2.scatter2)
```
~~~

These options ensure that the code chunk is ignored when the document is rendered to any format other than HTML.

When you open the `.docx` version of your Quarto document in Microsoft Word, you may get a number of warnings (e.g. Figure 18). You can safely click "Yes" or "Close" to get rid of these warnings and open up your Word file. If, for some reason, you cannot open a rendered document in Microsoft Word, try rendering to `.odt` instead (see below).


Microsoft Word

Word found unreadable content in LearningQuarto3.docx. Do you want to recover the contents of this document? If you trust the source of this document, click Yes.

Show Repairs

Word detected and repaired the following errors. To view each repair, select it in the list, then click Go To. Save the document to make the repairs permanent.

Table Properties 1
Table Properties 2
Table Properties 3
Table Properties 4
Table Properties 5

Sort by: [✓ Error Description] Location

(a) (b)

Figure 18: Examples of popup menus that may appear when opening the `.docx` version of a Quarto document.

To control more stylistic elements (e.g. font types, sizes, page margins, etc.) when rendering to Word, it is also possible to reference a Word template in the YAML (for details, consult the official Quarto Guide).

To share your work with **LibreOffice**, **OnlyOffice**, and **OpenOffice** users, use the `.odt` rendering option. This will generate an OpenDocument – an open standard file format that can be opened in any text-processing software, including Microsoft Word.

```
---
format: odt
---
```

By default, the quality of the images and graphs in rendered `.docx` and `.odt` files is low. This is to keep the file size reasonable. High-quality images can be rendered by specifying the **image definition** in the YAML option. To do so, replace the format line that you added above with the following lines. Make sure that you indent each line correctly as shown below; otherwise, you will get an error when you try to render your document.

```
---
format:
  odt:
    fig-dpi: 300
---
```

3.3 PDF

It is also possible to render Quarto documents to PDF; however, this requires you to have LaTeX installed on your computer. Alternatively, you can use Typst — a new open-source markup-based typesetting system designed to be as powerful as LaTeX but easier to use.

If you don't already have your favourite LaTex distribution, Quarto developers recommend that you use the TinyTeX distribution to render `.qmd` files to PDF. To **install** (or update) TinyTeX, go to the Terminal pane in *RStudio* and run the following command:

Listing 2 `Terminal`

```
quarto install tinytex
```

This is likely to take a few minutes but you will only need to do it once. Afterwards, you can add the following line to your Quarto YAML header and you're ready to render to PDF! If you run into any issues installing {tinytex}, consult the tinytex FAQ page.

```
---
format: pdf
---
```

HTML being the default format, some options available for HTML are not – at least by default – available in other publishing formats. Many of the basic options, however, work across different formats. The YAML header options below can be used to include a table of content with numbered sections at the start of the PDF version of your document. It also includes two options that are specific to the PDF format and which are particularly useful for academic writing: the first will print a list of figures (`lof`) and the second a list of tables (`lot`).

```
---
format:
  pdf:
    number-sections: true
    toc: true
    lof: true
    lot: true
---
```

3.4 Slides

In research, it's quite common that you will be working on a project that will be submitted as a paper or thesis (e.g. in PDF format) *and* that you'll also want to **present** in class, to your research group, or at a conference. Conveniently, we can turn any Quarto document into **presentation slides**. At the time of writing, there are three presentation formats to choose from.

Revealjs	An open-source HTML presentation framework.	`format: revealjs`
Power-Point	Microsoft Office's presentation editing software.	`format: pptx`
Beamer	A LaTeX class for producing presentations and slides in PDF format.	`format: beamer`

I recommend using **Revealjs**. The best way to get a sense of what is possible is to explore the demo presentation from the Quarto Guide.

4 Going further with Quarto

This chapter only just scratched the surface of what's possible in Quarto. The official Quarto Guide is very detailed, but highly accessible and well worth exploring to find out what else you can do in Quarto. Check out the Quarto Gallery to get a sense of what's possible. From books to interactive dashboards, the world's your oyster!

Figure 19: Artwork CC BY 4.0 Allison Horst from the "Hello, Quarto" keynote by Julia Lowndes and Mine Çetinkaya-Rundel, first presented at the RStudio Conference 2022

Further resources

- Quarto for Scientists by Nicholas Tierney is well worth checking out. There are many overlaps with this tutorial but, as a "living book", it is being regularly updated and expanded. The chapter on Common Problems with Quarto (and some solutions) is particularly useful.
- The latest edition of "R for Data Science" also has a great chapter on communicating the results of data science projects using Quarto.
- Quarto has many functionalities that are particularly attractive to those of us involved in higher education teaching and academic research. Watch Quarto for Academics (20 minutes) by Mine Çetinkaya-Rundel to find out more. In addition, the University of Utrecht has published a great resource on creating open textbooks with Quarto and GitHub Pages.
- Thinking of writing a term paper, thesis, dissertation, or book in Quarto? Cameron Patrick wrote his doctoral thesis in Quarto and has helpfully put together some great tips so that his "pain and suffering can help reduce yours". Similarly, Gina Reinhard wrote her M.A. thesis as a single Quarto document and has made her Quarto template for term papers and theses available to all in open access.
- Last but not least, Awesome Quarto provides a curated and regularly updated list of the many Quarto-related docs, talks, tools, examples, and articles that the internet has to offer.

References

Community, T. T. W. (2022). *The turing way: A handbook for reproducible, ethical and collaborative research (1.0.2)*. https://doi.org/10.5281/zenodo.3233853

Dąbrowska, E. (2019). Experience, aptitude, and individual differences in linguistic attainment: A comparison of native and nonnative speakers. *Language Learning*, *69*(S1), 72–100. https://doi.org/10.1111/lang.12323

Horst, A., & Lowndes, J. (2020). *Openscapes - tidy data for efficiency, reproducibility, and collaboration*. https://openscapes.org/blog/2020-10-12-tidy-data/

Hussey, I. (2025). Data is not available upon request. *Meta-Psychology*, *9*. https://doi.org/10.15626/MP.2023.4008

Iannone, R., Cheng, J., Schloerke, B., Haughton, S., Hughes, E., Lauer, A., François, R., Seo, J., Brevoort, K., & Roy, O. (2025). *gt: Easily create presentation-ready display tables*. https://doi.org/10.32614/CRAN.package.gt

LaZerte, S. (2021). How to cite r and r packages. In *rOpenSci*. https://doi.org/10.59350/t79xt-tf203

Le Foll, E. (2025). *Data analysis for the language sciences: A very gentle introduction to statistics and data visualisation in r*. Open Educational Resource. https://elenlefoll.github.io/RstatsTextbook/

Moroz, G. (2020). *Create check-fields and check-boxes with checkdown*. https://CRAN.R-project.org/package=checkdown

Müller, K. (2025). *here: A simpler way to find your files*. https://doi.org/10.32614/CRAN.package.here

Navarro, D. (2022). *Project structure*. https://doi.org/10.5281/zenodo.6976003

R Core Team. (2025). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Rodrigues, B., & Baumann, P. (2026). *Rix: Reproducible data science environments with "nix"*. https://docs.ropensci.org/rix/

Rodriguez-Sanchez, F., & Jackson, C. P. (2025). *grateful: Facilitate citation of R packages*. https://pakillo.github.io/grateful/

Seibold, H., & Müller, R. (2023). *BERD course: Make your research reproducible*. https://doi.org/10.17605/OSF.IO/RUPT7

South Carolina, U. of. (n.d.). Alternative text. In *Digital Accessibility*. https://sc.edu/about/offices_and_divisions/digital-accessibility/toolbox/best_practices/alternative_text/

Ushey, K., & Wickham, H. (2023). *Renv: Project environments*. https://CRAN.R-project.org/package=renv

W3C Web Accessibility Initiative. (2022). Images. In *Strategies, standards, resources to make the Web accessible to people with disabilities*. https://www.w3.org/WAI/tutorials/images/

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., … Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, *4*(43), 1686. https://doi.org/10.21105/joss.01686

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for data science: Import, tidy, transform, visualize, and model data* (2nd edition). O'Reilly. https://r4ds.hadley.nz/

Xie, Y. (2025). *xfun: Supporting functions for packages maintained by "Yihui Xie"*. https://doi.org/10.32614/CRAN.package.xfun