

Quarto: a Tool for Open Science

Course organized by Wageningen Graduate Schools

Tiny van Boekel, Food Quality & Design, WUR
Jos Hageman, Biometris, WUR

10/28/22

Table of contents

1	Introduction	3
1.1	Open Science: what does that mean?	3
1.2	Structure and learning goal of the course	4
1.3	Prerequisites	4
2	Anatomy of a Quarto/RMarkdown document	5
2.1	The YAML	6
2.2	The Text	6
2.3	Data handling, tables, figures	9
2.3.1	Quarto projects	9
2.3.2	Inline code and code chunks	14
2.3.3	Inserting tables and figures	17
2.3.4	Numbering sections, table of contents, list of tables and figures	34
2.4	Inserting equations	34
2.5	Inserting references and reference styles	39
2.5.1	Using existing Journal Formats	41
2.6	Producing Word documents	42
2.7	Inserting comments in Quarto files	43
3	Advanced formatting using latex packages	43
3.1	Inserting line numbers	43
3.2	Changing line spacing	44
3.3	Creating an index	45
4	Home work	45
4.1	Own data?	45

4.2	The pottery data set	45
4.3	The wine data set	47
5	Producing a thesis (book)	48
6	Wrapping up the Quarto Course	53
	References	53



1 Introduction

Quarto is a tool to practice Open Science. It weaves together text, code, tables, graphics and bibliographic content: it allows to produce dynamic documents (i.e., documents that update as soon as something is altered in the underlying data and/or code). It is an example of literate programming¹. The code is not limited to R, it can also be Python, Julia, Observable, but only R is discussed in this course. In other words, Quarto is more generic and not language-specific, whereas RMarkdown is an R package (note that Quarto is not an R package). Quarto is the next generation of RMarkdown and is easier to use (i.e., same functionality with less code); it combines code from earlier packages in one tool. Quarto can be used seamlessly with RStudio (also from the command line interface or other IDEs but that will not be discussed here)². To learn more about Quarto see <https://quarto.org>. The website offers some excellent tutorials. RStudio published a workshop on Getting Started with Quarto: [Get Started with Quarto - rstudio::conf 2022 Workshop \(rstudio-conf-2022.github.io\)](https://rstudio-conf-2022.github.io). There is also a living book (Quarto is just recently launched so it is in development): [Quarto for Scientists](#). With Quarto³ you can produce a multitude of outputs from the same source file: html, pdf, Word docx, presentations (powerpoint, revealjs), blogs. Here we limit ourselves to html, pdf and docx. Incidentally, this reader is also written in Quarto!

1.1 Open Science: what does that mean?

Open Science is a movement within the scientific community that wants to make scientific work more open and transparent than it is today. The ultimate goal of science is that work can be reproduced and criticized and debated. To do that, scientists must make their scientific activities and methods and the data obtained publicly available. More and more journals require this, but apart from that it should also be the normal attitude, and not unimportant, it will save you and your supervisors a lot of time in the end and will be a much more efficient way of working (Perkel 2022). There are two ‘technical’ aspects on Open Science: i) to report on a scientific finding (regardless whether it is in a scientific journal, on internet, via blogs or e-books), and ii) to communicate and allow for sharing, comments and criticism. Tools like Quarto and RMarkdown make it relatively easy to integrate data analysis, data wrangling, reporting, and publishing, which comprises mainly the first technical aspect. For the second technical aspect, there is another tool called Git in combination with Github. Briefly, it allows to work simultaneously on projects/documents, you can go back in time to see what has changed over time and people can take action (comments, corrections, suggestions, etc.), no need anymore to call a file revised_version_x. It is actually a perfect tool for a supervising

¹Literate programming is writing out the program logic in a human language with included (separated by a primitive markup) code snippets and macros. - [Wikipedia](#)

²RStudio will change its name in Posit by the end of 2022.

³Quarto is the format of a book or pamphlet produced from full sheets printed with eight pages of text, four to a side, then folded twice to produce four leaves. The format of the first published European book (in 1452) was quarto.

team of a PhD student or for group work of students in general. But is also a perfect tool to share data, methods, computer scripts with the scientific community and to allow for questions and criticism. However, Git and GitHub will not be discussed in this course because of time constraints; a reference for self-study is given at the end of this reader. Keep in mind, however, that RStudio, Quarto and RMarkdown can be linked easily to Git and Github.

In the following, we will focus on Quarto but almost everything we describe in this reader can also be done in RMarkdown, except from some new features in Quarto (but there will be dedicated packages to do the same thing in RMarkdown). If you are new to RMarkdown, our advice is to use Quarto. If you are already a little bit familiar with RMarkdown, you will recognize that many things are the same in Quarto, except for some new interesting and relevant features.

1.2 Structure and learning goal of the course

This course is primarily meant for PhD students and postdocs who are interested in using Quarto (or RMarkdown) in their scientific work. It is a perfect tool to publish papers and a PhD thesis within the framework of Open Science.

The structure of the course is that you will first learn about the basic anatomy of Quarto documents. Then you will apply this to build up a scientific paper step by step. We will provide you with instructions for each new step and you will then practice with an exercise. At the end of the first day of the course, the result will be a small mock paper containing all the basic elements of a scientific paper written in Quarto. In between the first and the second day of the course, we will ask you to do some homework by writing two more small mock papers, either with your own data or with data supplied by us. During the second day, we will focus on how the three mock papers can be combined in a mock thesis. In addition, some more advanced Quarto aspects will be discussed.

The learning goal is that, after this course, participants are knowledgeable about the basic principles of Quarto and how to apply this to their own work when writing scientific papers and compile those in a PhD thesis.

1.3 Prerequisites

We assume that you have basic knowledge of how to work in R and RStudio, i.e., how to install and load libraries, how to view data, how to handle dataframes and how to perform basic things like simple regression analysis. Some knowledge of the tidyverse ecosystem ([Tidyverse packages](#)) will also be very helpful. Though this course is not about the tidyverse, some elements of it are used in the exercises. Using the tidyverse will make your scientific life much easier and fits in the Open Science approach. It is about how to store, analyze and process data in a tidy way. It is also very useful for visualization with ggplot, which we will use here. Tidy data are not only useful for R users but also for excel users, for example, see Broman

and Woo (2018). In short, tidy data represent a data matrix in which each column represents a single measurement and each row a single object on which the measurement was made.

2 Anatomy of a Quarto/RMarkdown document

A Quarto/RMarkdown document has the following basic elements:

1. A YAML. This piece is the start of your document and contains meta-data for the whole of your document. It tells the software basic things like font size, graphics details, type of output desired. There are default settings but you can override them.
2. Text (the story you want to tell). For scientific articles this will be the usual Introduction, Materials and Methods, Results and Discussion and References. For books and PhD theses it would be the thesis chapters.
3. Code for data wrangling, calculations, tables and figures. There are two options to do something based upon data: inline code in the text, and separate code chunks. Sometimes it may be needed to insert figures that cannot be produced by R, which is why the option exists that Figures are imported from outside Quarto. While RMarkdown was basically developed for use with R, Quarto can now also be used with other programming languages such as Python and Julia. Which language to choose is up to the user, of course. In this course, we only use R.
4. Code to insert equations. Equations can be added to the text and these are coded in Latex language.
5. References. These will be automatically inserted once the references you want have been indicated by you and are present in a so-called bibtex file. It can be done using various styles; these are defined by so-called css files. However, with Quarto you can also refer directly to a DOI (digital object identifier) and once the reference has been found it will be added to your bib file.

The real strength of Quarto and RMarkdown is that it integrates all these parts. For instance, suppose you have made a draft report, and then you discover an error in a calculation somewhere, or some data have changed, or some additional data have become available. Once you have incorporated such changes, they will be automatically taken into account in all other related calculations, graphs and Tables. This is Open Science in optima forma! And if you use Git/GitHub you can actually also trace this back in history so that you yourself and your supervisors can see what has changed.

2.1 The YAML

The YAML (acronym for Yet Another Markup Language, or tongue-in-cheek: Yaml Ain't Markup Language) is where it all begins. Whatever the acronym, it is an essential part with which every Quarto document starts. It is separated from the main text by three dashes at the top and at the end. The YAML tells the Quarto software what to do in general terms with the content. You can specify options further into sub-options, in which case spacings matter in this respect: see Figure 1.

```
---  
format:  
  pdf:  
    toc: true  
    lof: true  
---
```

Figure 1: Note the use of spaces with sub-options, here illustrated with the format option

You can indicate the title, description of the document, date (today, last-modified, a fixed date), authors of the document, abstract, font size, type of desired output, table of contents, and many more things that you will learn by doing in this course.

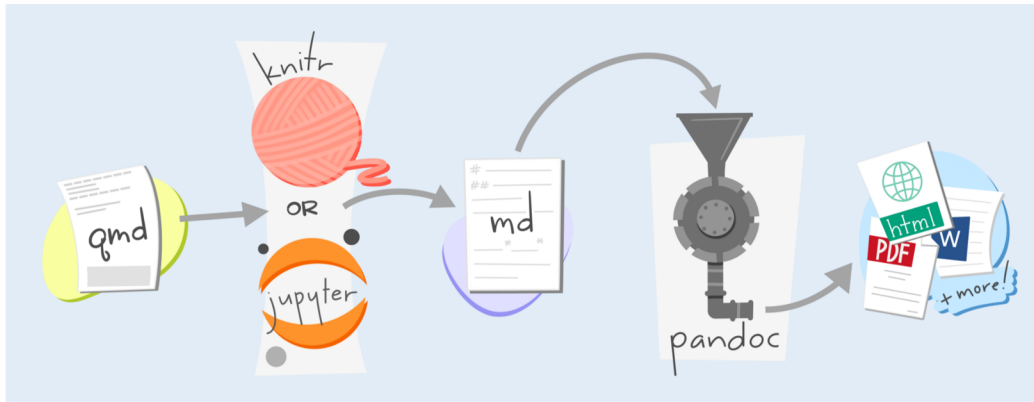
2.2 The Text

Markdown as such is a language, elements of which are used within Quarto. A Quarto file (extension .qmd) containing R code is processed by rendering (Quarto) or knitting (RMarkdown) via the R package `knitr` into plain markdown text (files with extension .md, qmd files with Python code use jupyter). A markdown file is then the basis for other document types like pdf, word, html; this processing is done by pandoc (a universal document conversion tool) in the background: see Figure 2. Normally, you do not notice this (except perhaps for some intermediate messages in the background) but it may be handy to know a little bit of what goes on behind the scenes (not further discussed here).

Quarto/RMarkdown/RStudio can be used like text editors. You can write text as you would do in Word or any other word processor with more or less the same functionality. There is also a spelling checker. Nowadays, you can use RStudio in two modes, the ‘visual editor mode’ and the ‘source mode’, see Figure 3.

Visual mode supports editing of all pandoc markdown features. Regardless of the mode you use, the result is the same for the output but the visual editor mode is probably easier to use than the source mode; the source mode requires more coding (you can toggle between the two modes without any problem, so you can choose your preference). There are shortcuts and clickable buttons in visual mode to achieve special formatting effects (see Figure 4), these are not available in Source mode, there you need to type code.

Some possible features are:



Source: mine çetinkaya-rundel, RStudio

Figure 2: Schematic overview of what happens upon rendering a qmd file

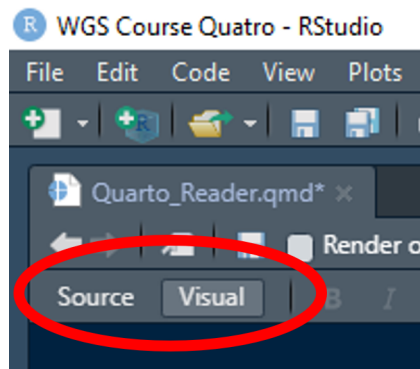


Figure 3: Choose Source or Visual Mode

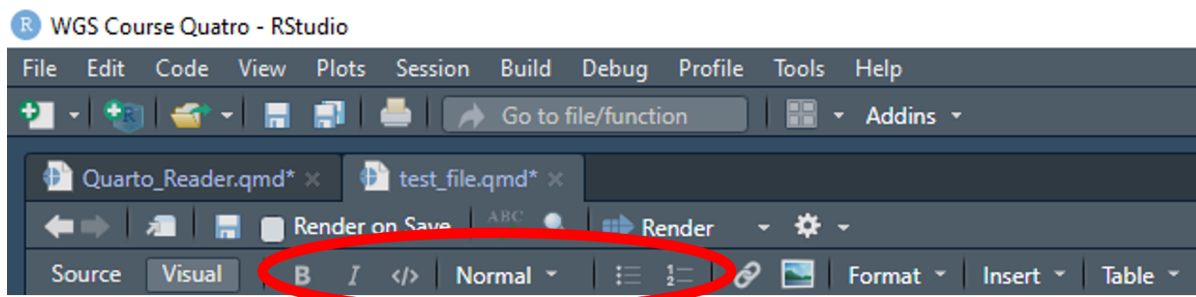


Figure 4: Buttons in Visual mode to format text

Bold text

Italic text

- Bullets (unordered list)
1. Numbered lists (ordered list)

(Lists can be with normal spacing (default) or tight where tight means: less vertical spacing between items, see [Quarto - Content Editing](#) for details).

Underlined text

Links in text are indicated by: `<https://quarto.org>`, or: `[Quarto](https://quarto.org)`

superscripts need to be enclosed between `^` on each side: `superscript`

subscripts need to be enclosed between `~` on each side: `subscript`

strikethrough need to be enclosed between two `~` on each side: `strikethrough`

Headings in various sizes, to be chosen via the button Normal in Visual mode, or by using the hashtag (`#`, `##`, `###`) in Source mode.

A hard pagebreak can be achieved with a so-called shortcode:

```
{{ <pagebreak> }}
```

In Visual mode, you can insert many things in the file by clicking on the button ‘Insert’, including hard line breaks, non-breaking spaces and special unicode characters, see Figure 5.

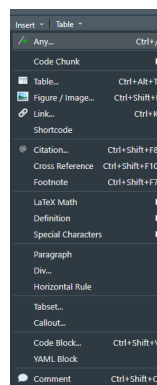


Figure 5: Many options after clicking the Insert button in Visual mode


Sections with headings are automatically numbered but these numbers are only shown in the output after adding in the YAML: `number-sections: true`. You can also refer to sections throughout the text by adding `{#sec-‘name of the section’}` after the section heading. If, for


instance, the section Introduction is labeled as `{#sec-intro}`, you can refer to this section in the text as `@sec-intro` and Quarto will then refer to it with the section number.

Everything you do in Visual mode can be done also in source mode but that requires more coding. If you are in the Visual editor mode, you can switch to the Source mode to see how exactly the code then looks like. Furthermore, It is possible to insert block quotes and call-outs, Figures and Photo's (if you use links, think about copyright!). An example of a block-quote is:

Every PhD student should use Quarto!

Two examples of a call-out:

 The learning curve for R is steep but rewarding

 Once you master Quarto, you can practice open science!

Of course, also clickable links to websites can be given where the format is: `[name](address of the website)`.

It is possible to write in columns, where you can determine the width of each column. A new feature in Quarto is that you can also make remarks on the side (and also figure and table captions can be placed there if so desired).

A very handy tool when using Quarto in RStudio is to type the slash `/` (if you are at the beginning of a line, or control-slash anywhere in a line) and you will be presented with a lot of options to choose from for formatting, inserting code chunks, equations, call-outs, emoji's , to name a few.

Basic elements of markdown can be found in the following link [Quarto-guide](#)

2.3 Data handling, tables, figures

2.3.1 Quarto projects

An essential element of Open Science is that you work in a structured way. This implies that you should organize your files in directories and subdirectories so that they become portable (not only for others but also for future you!). RStudio makes this very easy for you if you decide to work in projects. Our advice: make a new project for every new activity that you start. If you do so: RStudio will present you with the option to make a new directory, or to use an existing directory, it will write a small file with the extension `.RProj`. Make further subdirectories in this project directory, for instance, a subdirectory data, scripts, figures, tables.

And very importantly, do not point at those directories with absolute computer addresses, this will only work at your own computer (and even then, not anymore when you obtain a new computer).

💡 Use the R package here!

There is a very nice R Package called **here** that takes all sub-directories relative from the main project directory in which the file `.RProj` is present. Never use commands such as `getwd()` or `setwd()` in your code, that is not transparent! (If you do and it is discovered, Jenny Bryan from RStudio will come and set your computer on fire, so be warned!).

A practical exercise is now started by constructing a mock paper using an existing real data set. This data set is about measurements done on three penguin species living on various islands in the Palmer Archipelago in Antartica, data like body weight, bill- and flipper length, species type, island they lived on. There is even a special R package called **palmerpenguins** (Horst, Hill, and Gorman 2020). (M. Horst, Presmanes Hill, and B. Gorman 2022). (We chose this data set because quite a few internet tutorials use this nowadays, so you can practice with it further if you like.) Here we use it merely as a vehicle to illustrate some aspects of Quarto that you need to write a scientific paper. For the purpose of the exercise there is no need to dive into the background of the penguin research.

In fact, we as course leaders have already produced a mock paper to show you the end product we are aiming for; take a look at ‘penguin_paper.pdf’. We are going to build it up from scratch together with you as a practical exercise (our mock paper is only meant as an example, you do not need to rebuild it exactly like this). So, to begin the production of your mock article: start with defining an R Project.

Exercise 1: Setting up a Quarto project to produce a mock paper

It is assumed that you have installed the newest version of Rstudio (v2022.07.1 or later) which includes Quarto; with older versions of RStudio, you need to install Quarto from <https://Quarto.org> (always use the newest version!). Open RStudio and start by creating a new project in Quarto/RMarkdown from the RStudio menu:

File -> New Project

In a popup menu (Figure 6) select new directory (you can also use an existing directory, but start with a New Directory for this exercise).

In the next menu, select Quarto Project (there are lots of other options) and type a Directory Name in the next popup screen, you also have the option to create it as a subdirectory of

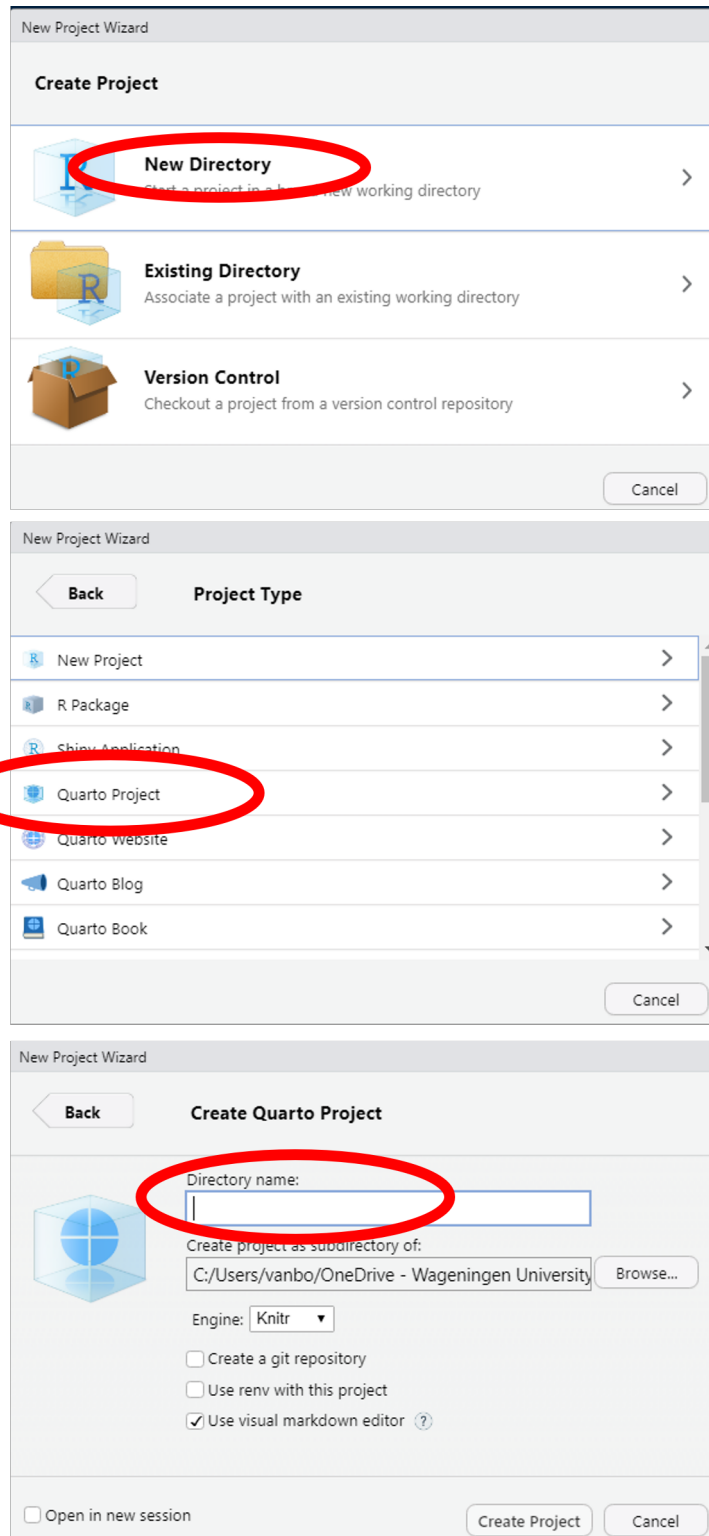


Figure 6: Screens appearing upon opening a new Quarto project

another folder). RStudio will consider this directory as the root directory for this project. Quarto returns a setup with a very small YAML. Modify the YAML as indicated below:

```
---
title: Document title
author: your name
date: last-modified
affiliations: your department
format: html
editor: visual
---
```

Give the document a suitable name and insert your own name as author and delete the text below the YAML, You can modify and expand the YAML any time. Give the file a name and save it from the RStudio menu:

File -> Save As..

Quarto will give the .qmd extension to the file (in RMarkdown it will be .Rmd).

If you want to produce a pdf, please note that you have to install latex (tinytex is recommended) on your computer if you did not do that before (you only need to do this once).

How to install tinytex on your computer

Go to the terminal mode in RStudio (click Tools > Terminal) and then type on the command line: quarto install tool tinytex.
(pdf's are produced from LaTeX with the xelatex engine)

When you stop working on your project and you have saved your file, close the project:

File -> Close Project

Whenever you are ready to resume working on it, open first the project again from the RStudio menu:

File -> Open Project..

(Or click on Open Project in the upper right corner of the RStudio menu). You can then continue working on it because RStudio will have opened the project in the right directory. Using the R package `here` will automatically take this directory as the root directory for the project, and absolute referencing for your specific file directories on your computer is not needed anymore (which would be very much against open science!).

If you want to open a new document in the same project, then go to:

File -> New File -> Quarto Document...

A pop-up screen appears as in Figure 7.

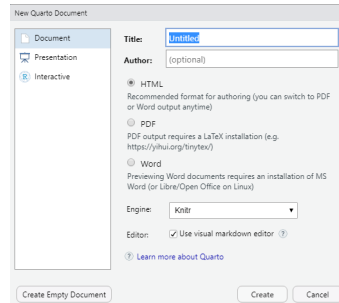


Figure 7: Pop-up screen to create a new Quarto document in an existing Quarto project

Fill in the details, click on Create and save the file.

End of exercise 1

The project you created, with a .qmd file in it, is now ready for further processing. At this stage, you may decide to add some other subfolders to the project, such as the subfolders ‘data’, ‘scripts’, ‘figures’. You can do that inside RStudio or outside, for instance by using Windows’ File Explorer. Once you have done that, use the package [here](#) to refer to files in these subdirectories. For instance, if you have a .csv file with data in the subdirectory data, and a .png file as a figure in the subdirectory figure, it could look like this:

```
file_name <- read.csv(here("data","file_name.csv"))
graph <- knitr::include_graphics(here("figures","figure.png"))
```

After having prepared the project in the previous exercise, the next exercise is to add some text.

Exercise 2: adding and formatting text to the mock paper

Resume working on the project saved in the previous exercise. Imagine for this assignment that you are going to write a scientific report on penguins using the data that are supplied. This exercise is to write a setup for such a report. Imagine that you are the one who did the measurements and you need to report on it. The format should be that of a standard scientific publication, i.e., Abstract, Introduction, Materials and Methods, Results and Discussion,

Acknowledgments, References. You can use the text provided in the mock paper supplied, or use your imagination to write some text in each of the sections. Use headings, bold and italic words here and there, an occasional footnote, ordered and unordered lists. At this stage of the exercise, it is only about text, other parts will be added later. After having done this part of the assignment, render the file and study the output.

i Render options

For scientific output you probably would use pdf in most cases. However, in the writing phase you may want to use html output at first. It is less sensitive to small mistakes (the pdf engine can be rather picky and comes with rather vague error messages) and html is much faster in producing output. Once you are happy with the output, switch to pdf output.

Play with adding a toc (table of contents) in the YAML:

```
---
title: Document title
author: your name
date: last-modified
affiliations: your department
format: html
editor: visual
toc: true
---
```

Switch between html and pdf in the format YAML field to see the difference in output. Do not forget to save your file once in a while while you are working on it!

End of exercise 2

2.3.2 Inline code and code chunks

Inline code

The option to insert inline code relieves you from the error-prone task to copy and paste data, or parameters derived from your data, into the text (for instance, to report a mean or a standard deviation). The big advantage is that, if for whatever reason data change, you do not need to copy new values all over again, which could potentially be a big source of errors. The way to insert R code is to start with a backtick ‘, followed by the letter r, then the code you need to calculate something, followed by another backtick ‘:

```
`r mean(penguins$body_mass_g)`
```

All kinds of expressions can be used in the inline code, for instance, if you want to round numbers to two decimals:

```
`r round(mean(penguins$body_mass_g),2)`
```

Exercise 3: insert inline code

Practice with inserting inline code using the penguin data set in your mock article. Import the data set `penguins.csv` into your `.qmd` file (because there are some missing values in the data the “`drop_na()`” command is added, `%>%` is the so-called pipe from the tidyverse, you could also use the RStudio pipe `|>`):

```
#| label: data

penguins <- read_csv("penguins.csv") %>% drop_na()
```

You will need to do some data exploration to be able to do this: study the data file in RStudio to see which variables are used. Use inline code to report in a sentence, for instance:

- how many penguins were measured on each island
- summary statistics: how many penguins were measured
- the average bill-length and bill-depth

Render the file and study the output.

End of exercise 3

Code chunks

The second option to weave R code into the text is to add code chunks, which can be interspersed throughout the text. Code chunks are used to do more complex calculations or to prepare graphs and tables. The output of the chunk may, or may not, be put inside the text (such as a graph or a table). A chunk can be labeled so that you can refer to it. In the case that a graph or table will be output, a caption can be supplied in the chunk. A chunk can be inserted via the RStudio menu

Insert -> Code Chunk -> R

or via the shortcut (Ctrl-Alt-i, in which case {r} is put between the curly brackets (can be changed into Python or Julia) or you can insert code chunks manually by enclosing the code in between three backticks at the beginning (followed by {r}) and three backticks at the end. **The by far easiest way to insert code chunks in Quarto in RStudio is to type the slash / at the beginning of a new line, which will give you a lot of options to choose from, including code chunks.** Always give a short name to a code chunk (syntax: #| label: name; the #| is called the hash pipe). Think of a name that gives a hint what the code chunk is supposed to do. That way you will be able to find it back, which is also handy if errors occurring in a code chunk are reported by the software (and errors will appear for sure!). Valid chunk names to be used with the chunk option label are, for instance:

- #| label: chunkname
- #| label: mychunkname
- #| label: my-chunkname
- #| label: mychunk1

Names that cannot be used:

- #| label: my__chunkname (do not put the character __ in a label)
- #| label: my chunk name (do not use spaces in a label)

There are many code chunk options, it could look like this:

```
```{r}
#| label: chunkname
#| echo: false
#| include: false
```
```

See for a list of possible options: [Quarto - Code Cells: Knitr](#). Code chunk options can override options mentioned in the YAML for that specific chunk. A code chunk that will be always needed is one that tells Quarto which packages need to be loaded in R. This depends obviously on the type of analysis you want to do. Some packages you will probably always need, such as the `tidyverse` and the package `here` and for the exercise you could load the package `palmerpenguins`:

```
```{r}
#| label: libraries
#| include: false

library(tidyverse)
```



```
library(here)
library(palmerpenguins)
```

```

Another chunk you will need is one that will load the data. Always keep your raw data separate in a subdirectory, for instance in csv format or excel format, and import them into your Quarto file to do further data wrangling if needed. Leave your raw data untouched.

```
```{r}
 #| label: data

 data_file1 = read.csv(here("data","filename.csv"))
 data_file2 = read.excel(here("data","filename.xlsx"))
```

```

(you can use any name for the file, it does not need to be data_file1).

Exercise 4: insert code chunks

Go to your mock article and insert a code chunk to load the libraries `here`, `tidyverse` (you may need to install them first using `install.packages()` from the R console). Also add a code chunk to load the penguin data. Label the code chunks with a name. For transparency, insert such general code chunks at the beginning of your file.

End of exercise 4

2.3.3 Inserting tables and figures

Tables

In RMarkdown, inserting tables has always been a somewhat more difficult task. The most basic way is in Markdown language via so-called pipe tables where the pipes indicate column boundaries but that is not so user-friendly. If you are interested in this way of working, study [Quarto - Tables](#). Fortunately, there are quite a few R packages to facilitate the setting up of tables (`gt`, `flextable`, `kable` and `kableExtra`, `stargazer`, `apa-table` from the R package `papaja`, ...). In Quarto, life has become much easier with respect to inserting Tables. You can refer to tables produced in a code chunk with the chunk option `#| label: tbl-tablename` and you can provide a caption to the table with the chunk option `#| tbl-cap:`

“caption text”. The prefix ‘tbl-’ tells Quarto that it is about a table and you can refer to it in the text by typing: @tbl-tablename and Quarto will then number the table for you. You can use various table packages inside the code chunk to produce tables from data. In this course we stick to **kable** and **kableExtra** and **gt** with its extensions.

If you want to produce a table from scratch, then there is the possibility to do this from the RStudio menu:

Insert -> Table

(Shortcut: Ctrl-Alt-T) and then you will have to fill in the table cells manually. This is not recommended for scientific work (avoid copy-paste actions!) but it can be handy for some simple stand-alone tables that are not based on data, or if you need to produce a table with purely text. The following Table was produced in this way and may serve as a simple example:

Table 1: Overview of measuring methods

| Measurement | Method | Dimension |
|-------------|----------------|-----------|
| weight | balance | kg |
| length | measuring tape | m |

(If you switch to Source mode when you have produced a table in this way, you can actually see a table in Markdown language.) Quarto will provide a Table number but you cannot cross-reference such tables, it seems. Quarto can even produce tables directly from dataframes without any additional editing. By using the function `head()` from the **tidyverse** package (`head` displays the first six rows of a dataframe or tibble), for instance, the following table is obtained for the R built-in data set `mtcars`.

i Information about the data set `mtcars`

For some exercises we will use the data set `mtcars` that is part of base R. It contains data about fuel consumption, aspects of design and performance, such as: `mpg` (miles per gallon), `cyl` (number of cyl), `disp` (displacement), `hp` (horsepower), `am` (0 = automatic, 1 = 1manual), `gear` (number of gears). By typing `str(mtcars)` you can see how the data look like.

```
head(mtcars)
```

```
      mpg  cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4   21.0    6  160 110 3.90 2.620 16.46  0  1    4    4
```

Table 2: Table produced by kable on the first 6 rows and columns of mtcars

| | mpg | cyl | disp | hp | drat | wt |
|-------------------|------|-----|------|-----|------|-------|
| Mazda RX4 | 21.0 | 6 | 160 | 110 | 3.90 | 2.620 |
| Mazda RX4 Wag | 21.0 | 6 | 160 | 110 | 3.90 | 2.875 |
| Datsun 710 | 22.8 | 4 | 108 | 93 | 3.85 | 2.320 |
| Hornet 4 Drive | 21.4 | 6 | 258 | 110 | 3.08 | 3.215 |
| Hornet Sportabout | 18.7 | 8 | 360 | 175 | 3.15 | 3.440 |
| Valiant | 18.1 | 6 | 225 | 105 | 2.76 | 3.460 |

| | | | | | | | | | | | |
|-------------------|------|---|-----|-----|------|-------|-------|---|---|---|---|
| Mazda RX4 Wag | 21.0 | 6 | 160 | 110 | 3.90 | 2.875 | 17.02 | 0 | 1 | 4 | 4 |
| Datsun 710 | 22.8 | 4 | 108 | 93 | 3.85 | 2.320 | 18.61 | 1 | 1 | 4 | 1 |
| Hornet 4 Drive | 21.4 | 6 | 258 | 110 | 3.08 | 3.215 | 19.44 | 1 | 0 | 3 | 1 |
| Hornet Sportabout | 18.7 | 8 | 360 | 175 | 3.15 | 3.440 | 17.02 | 0 | 0 | 3 | 2 |
| Valiant | 18.1 | 6 | 225 | 105 | 2.76 | 3.460 | 20.22 | 1 | 0 | 3 | 1 |

In general, however, you will most likely build tables from data and then it pays off to use dedicated packages because those will give you many options for formatting. With the kable function (from R package knitr) inside a code chunk the first six rows and columns of the mtcars data set look as displayed in Table 2:

```
#| label: tbl-two
#| tbl-cap: "Table produced by kable on the first 6 rows of mtcars"

knitr::kable(mtcars[1:6,1:6])
```

It is also possible to produce two tables side-by-side with the code chunk option #| layout-ncol, see Table 3:

```
#| label: tbl-three
#| tbl-cap: "Example of two tables next to each other"
#| tbl-subcap: ["mtcars", "Just cars"]
#| layout-ncol: 2

# table on the left
knitr::kable(head(mtcars[, 1:3]))

# table on the right
knitr::kable(head(cars))
```

Multiple tables produced within a code cell can be referred to as subtables by adding a tbl-subcap option, as in Table 3.

Table 3: Example of two tables next to each other

| (a) mtcars | | | | | | | (b) Just cars | |
|-------------------|------|-----|------|-----|------|-------|---------------|------|
| | mpg | cyl | disp | hp | drat | wt | speed | dist |
| Mazda RX4 | 21.0 | 6 | 160 | 110 | 3.90 | 2.620 | 4 | 2 |
| Mazda RX4 Wag | 21.0 | 6 | 160 | 110 | 3.90 | 2.875 | 4 | 10 |
| Datsun 710 | 22.8 | 4 | 108 | 93 | 3.85 | 2.320 | 7 | 4 |
| Hornet 4 Drive | 21.4 | 6 | 258 | 110 | 3.08 | 3.215 | 7 | 22 |
| Hornet Sportabout | 18.7 | 8 | 360 | 175 | 3.15 | 3.440 | 8 | 16 |
| Valiant | 18.1 | 6 | 225 | 105 | 2.76 | 3.460 | 9 | 10 |

Table 4: Using kable and kableExtra to show the first 6 rows and first 3 columns of mtcars

| | mpg | cyl | disp | hp | drat | wt |
|-------------------|------|-----|------|-----|------|-------|
| Mazda RX4 | 21.0 | 6 | 160 | 110 | 3.90 | 2.620 |
| Mazda RX4 Wag | 21.0 | 6 | 160 | 110 | 3.90 | 2.875 |
| Datsun 710 | 22.8 | 4 | 108 | 93 | 3.85 | 2.320 |
| Hornet 4 Drive | 21.4 | 6 | 258 | 110 | 3.08 | 3.215 |
| Hornet Sportabout | 18.7 | 8 | 360 | 175 | 3.15 | 3.440 |
| Valiant | 18.1 | 6 | 225 | 105 | 2.76 | 3.460 |

The R package `kableExtra` gives you many more options to format a table according to your wishes, see [Create Awesome HTML Table with knitr::kable and kableExtra \(r-project.org\)](#). For instance, Table 3 a could be changed as shown in Table 4 (see Figure 8 for the code).

```
{r}
#| label: tbl-four
#| tbl-cap: "Using kable and kableExtra to show the first 6
rows and first 3 columns of mtcars"
knitr::kable(mtcars[1:6,1:6], booktabs=TRUE) %>%
kable_styling(position="center", full_width = F)
```

Figure 8: Code combining kable and kableExtra

You can be more specific in the YAML about how to print using the `df-print` option (see [Quarto - Using R](#) for more details). This includes an option for `kable`. `kable` takes a dataframe as input and turns it into a markdown format in the background, which is then rendered in the required output (html, pdf, word).

Probably, the most advanced R package to produce tables is `gt` (which stands for the grammar of tables, from the RStudio team). See [Easily Create Presentation-Ready Display Tables • gt \(rstudio.com\)](#) and links therein for more information. Figure 9 gives a typical workflow for `gt` and shows schematically the many options to change the appearance of a table.

For instance, Table 5 is the table output with `gt`, shown here as the default setting of `gt`, but

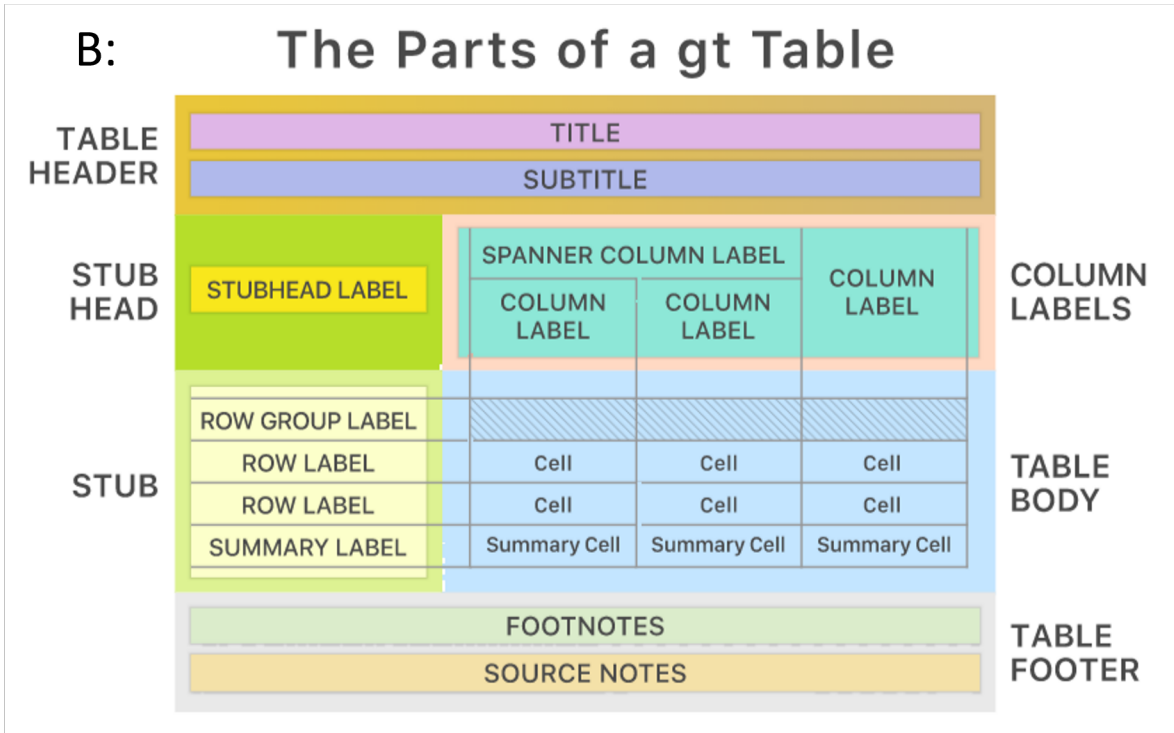
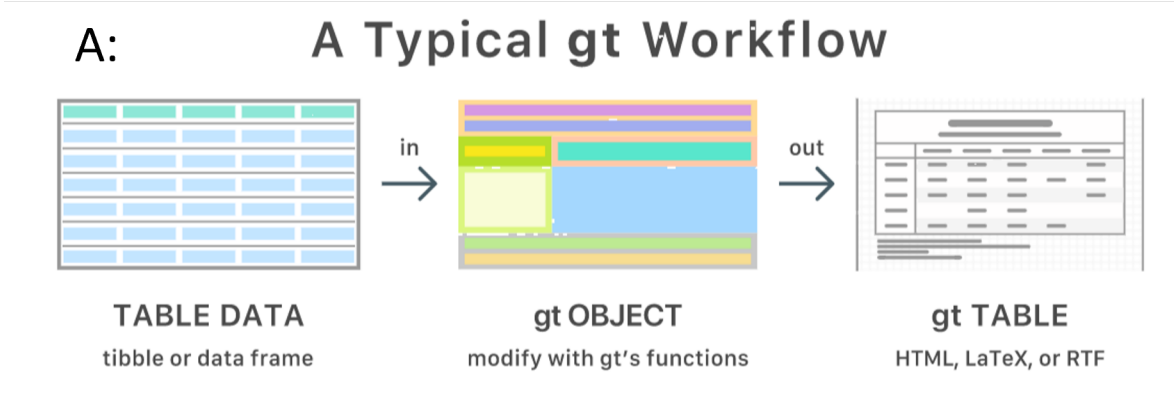


Figure 9: A typical gt workflow (A) and the parts of a gt table (B)

as mentioned there are many options to modify the format.

```
library(gt)

head(mtcars[,1:6]) %>%
  tibble::rownames_to_column("car") %>%
  gt()
```

Table 5: Table for the mtcars data produced using the package gt

| car | mpg | cyl | disp | hp | drat | wt |
|-------------------|------|-----|------|-----|------|-------|
| Mazda RX4 | 21.0 | 6 | 160 | 110 | 3.90 | 2.620 |
| Mazda RX4 Wag | 21.0 | 6 | 160 | 110 | 3.90 | 2.875 |
| Datsun 710 | 22.8 | 4 | 108 | 93 | 3.85 | 2.320 |
| Hornet 4 Drive | 21.4 | 6 | 258 | 110 | 3.08 | 3.215 |
| Hornet Sportabout | 18.7 | 8 | 360 | 175 | 3.15 | 3.440 |
| Valiant | 18.1 | 6 | 225 | 105 | 2.76 | 3.460 |

A very interesting package that builds upon `gt` is the R package `gtsummary`, see [Presentation-Ready Data Summary and Analytic Result Tables • gtsummary \(danielsjoberg.com\)](#). It produces ready-to-print tables, for instance with descriptive statistics, or the results of regression models. Table 6 gives an example for the mtcars data set using this package:

```
library(gtsummary)

tbl_summary(mtcars)
```

See also the cheat sheet `gtsummary` provided at Brightspace. Yet another extension to `gt` is `gtExtras` that makes the formatting of tables easy: see [Additional features for creating beautiful tables with gt • gtExtras \(jthomasmock.github.io\)](#).

Being able to format tables based on your own data in a publication-ready way is of course the ultimate goal, so let's practice with that now in the penguin mock article.

Exercise 5: inserting tables in the mock manuscript

Open your mock manuscript and add manually (via Insert Table) a similar table as in Table 1 shown above. Save and render to study the output.

Next, produce a Table from the penguins data set. Try to reproduce Tables 1, 2 and 3 from the mock manuscript using `kable` from `knitr` and `kableExtra` (see above for some options you

Table 6: Summary table of the mtcars data produced by gtsummary

| **Characteristic** | **N = 32** |
|---------------------------|----------------------|
| mpg | 19.2 (15.4, 22.8) |
| cyl | |
| 4 | 11 (34%) |
| 6 | 7 (22%) |
| 8 | 14 (44%) |
| disp | 196 (121, 326) |
| hp | 123 (96, 180) |
| drat | 3.70 (3.08, 3.92) |
| wt | 3.33 (2.58, 3.61) |
| qsec | 17.71 (16.89, 18.90) |
| vs | 14 (44%) |
| am | 13 (41%) |
| gear | |
| 3 | 15 (47%) |
| 4 | 12 (38%) |
| 5 | 5 (16%) |
| carb | |
| 1 | 7 (22%) |
| 2 | 10 (31%) |
| 3 | 3 (9.4%) |
| 4 | 10 (31%) |
| 6 | 1 (3.1%) |
| 8 | 1 (3.1%) |

can use), you will need to do some data wrangling. Feel free to experiment with other table producing R packages. Make sure that reference is made in the text to the tables. Render the document and study the output.

End of exercise 5

Figures

Figures can be incorporated in the text in two ways.

Importing figures

Importing figures (not produced by R code) goes via a link in Quarto; this needs not be done from a code chunk, it can be directly in the text. Syntax is: `![text](link)` (this is actually markdown code and is the same syntax as used for producing links but preceded with an exclamation mark). Another option is to insert an outside figure from a code chunk, using a function from knitr called `include_graphics()`.

```
```r
#| label: fig-knitrfunction
#| fig-cap: Use of knitr to include files from outside Quarto
knitr::include_graphics("filename.png")
```
```

Figures can be given captions (`#| fig-cap: caption-text`) and they are numbered automatically by referring to them in the text as `@fig-name` (this fig-name reference should obviously correspond to the label name of the chunk in which the figure is produced). The prefix ‘fig’ tells Quarto that it concerns a figure. You can also play with their location and alignment. If you would like to put figure captions in the margin, then write in the code chunk options: `#| cap-location: margin`. Other options are `fig-align`, `fig-height` and `fig-width` (the latter two with values in inches); the default values for figure width and height are 5.5 x 3.5 inch, but you can change that as indicated. For many more details on how to handle Figure details check out [Quarto - Figures](#). For positioning, aligning and combining graphs produced within code chunks, there is also the option to use the fantastic R package `patchwork` [The Composer of Plots • patchwork \(data-imaginist.com\)](#). Let’s practice with adding figures from an outside source in the mock manuscript.

Exercise 6: import figures from outside in the mock article

Two outside figures (as *.png files) are provided that explain some of the terms used in the penguin data set, namely penguin.png and culmen_depth.png. Insert them in the section Materials and methods, make reference to them in the text and make sure that they are numbered. Try both methods, with and without using a code chunk.

End of exercise 6

Producing Figures from data

Producing figures from available data can be done using baseR or in the tidyverse with ggplot. This will normally be done in a code chunk. ggplot is a fantastic tool to produce very fancy graphical output (Wickham 2016). Before we practice this in our mock article, let's first do some simple exercises with ggplot; ggplot is part of the R package **tidyverse**. See also the cheat sheets provided on data visualization, data-import, data-transformation, and tidyr collected for you in Brightspace.

Exercise 7: practicing ggplot

Open a new Quarto document (to practice with ggplot):

File -> New File -> Quarto document

and give the file a name, e.q., ggplot_exercise and save it.

ggplot builds up a graph with layers. To do so, it needs three basic elements:

- data
- aesthetics (indicating x and y values, possibly with additional information such as color, size, shape)
- geometry: what type of graphics

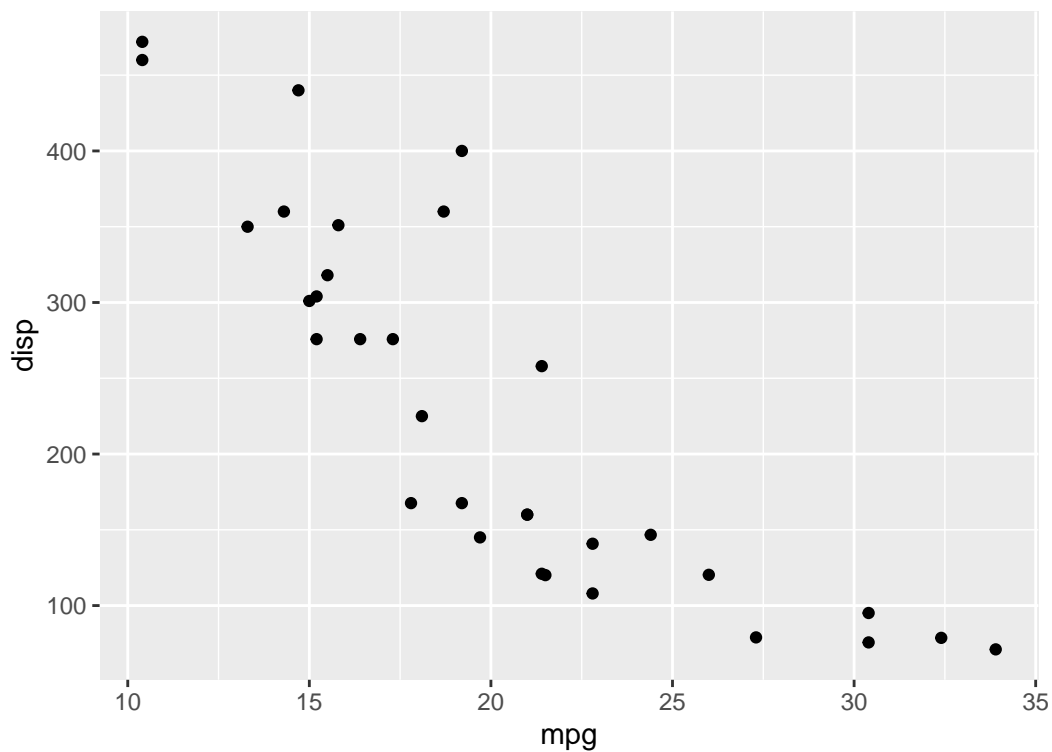
In addition, ggplot can subset data and identify them by color and shape, or it can produce various graphs in one plot via facetting. Annotation can be used to add text. Some basic statistical elements can be added as well. The following table gives an overview of possible 'geoms':

Table 7: Overview of ggplot geoms

| Type of plot | geom |
|--------------|--|
| scatterplot | geom_point(), geom_smooth(), stat_smooth() |
| bar chart | geom_bar(), geom_errorbar() |
| histogram | geom_histogram() |
| box plot | geom_boxplot() |
| line plot | geom_line() |
| pie chart | coord_polar() |

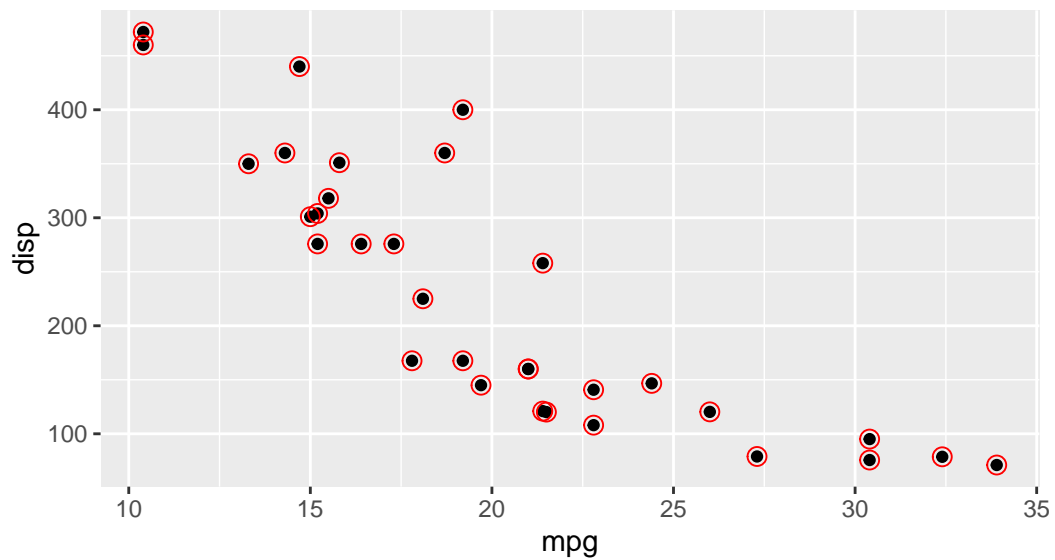
Let's start by making a very basic scatterplot of disp versus mpg from the data set mtcars:

```
plot1 <- ggplot(data=mtcars, aes(x=mpg, y=disp))+
  geom_point()
plot1
```



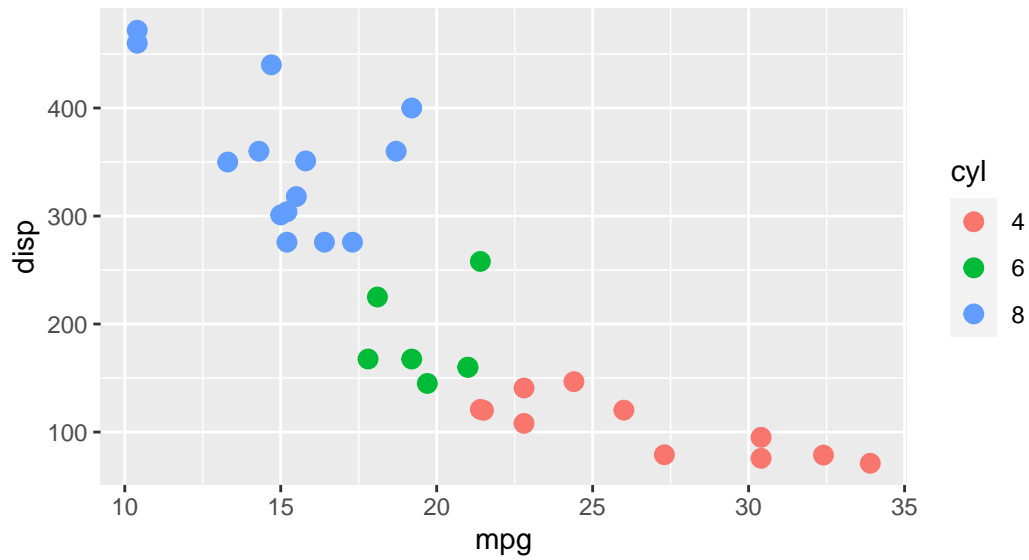
You can add labels, title and subtitles, different colors and shapes to the data if so desired. To change size and color, do the following (many more options are available!):

```
plot1 + geom_point(size = 3, shape=21, colour="red")
```



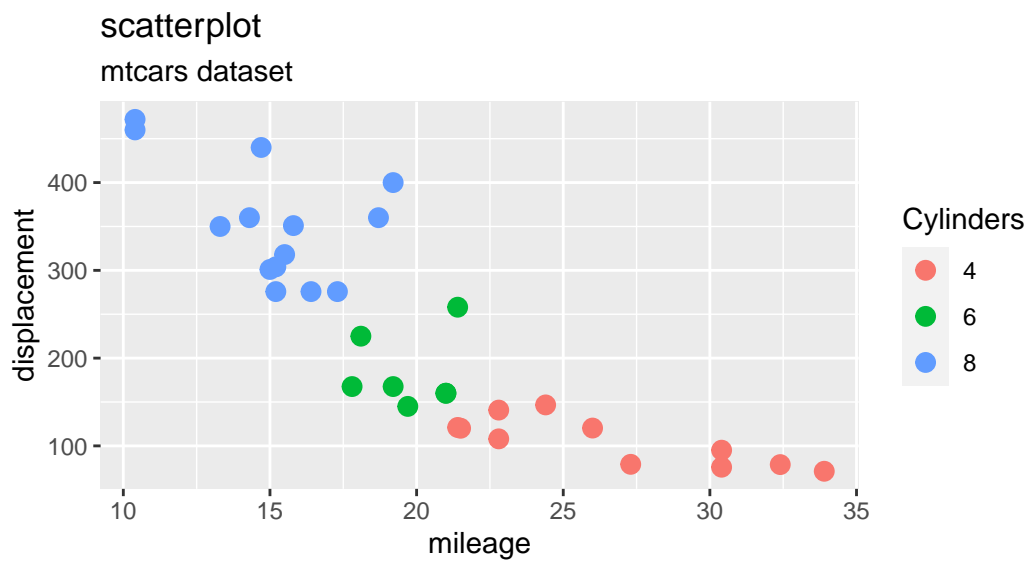
Suppose, you would like to find out which cylinders are present in which data, then to do that we first must turn the variable `cyl` from being numeric into a factor.

```
mtcars$cyl <- as.factor(mtcars$cyl)
plot2 <- ggplot(data=mtcars, aes(x=mpg, y=disp, color=cyl))+
  geom_point(size=3)
plot2
```



Titles, subtitles and captions can be added in ggplot:

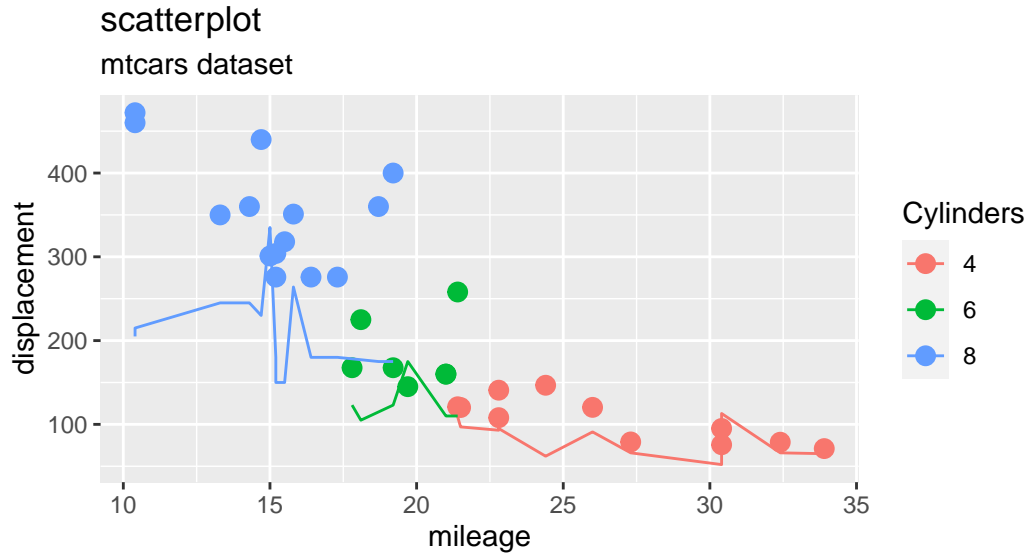
```
plot3 <- plot2 + labs(color="Cylinders") +
  labs(x="mileage", y="displacement")+
  ggtitle("scatterplot", subtitle = "mtcars dataset")
plot3
```



However, for figure captions in a publication it is best to use the code chunk option “#|fig-cap:” instead of the ggplot option because automatic numbering is then a piece of cake.

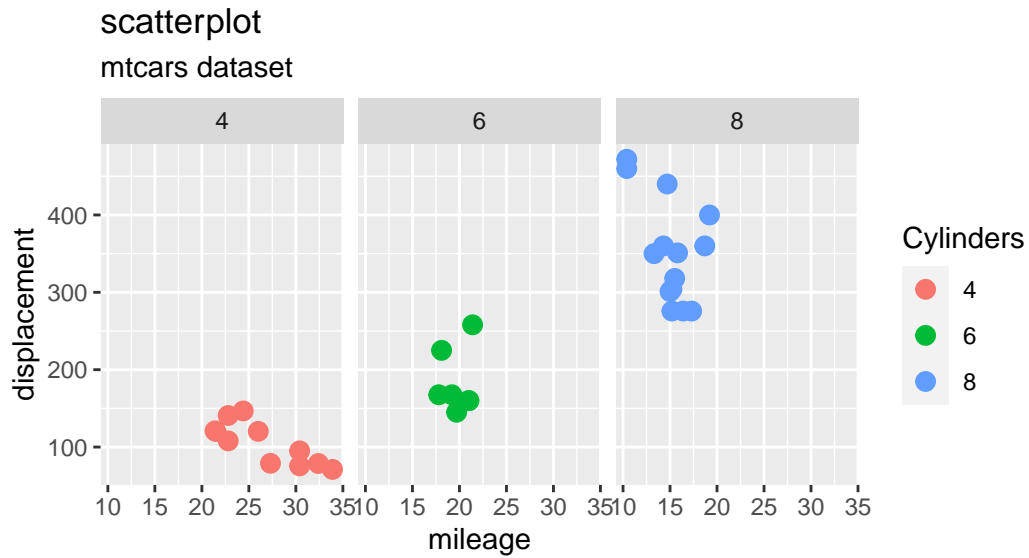
Several geoms can be added to an already existing plot, in the example below by adding a lineplot representing hp as a function of mpg:

```
plot4 <- plot3 + geom_line(aes(y=hp))
plot4
```



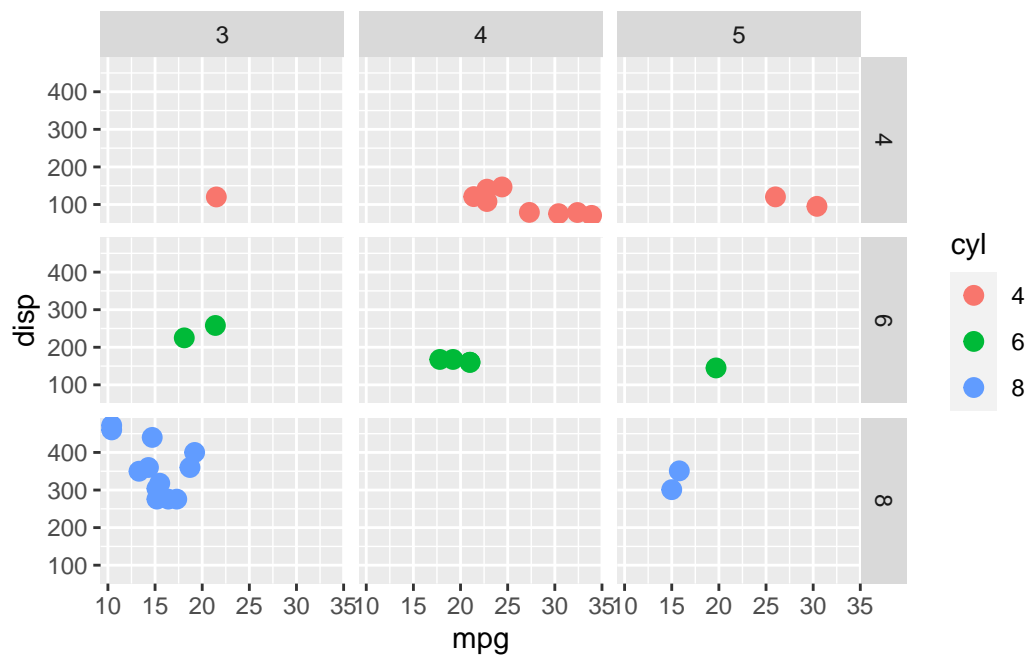
Faceting is a strong point of ggplot, here is an example showing scatterplots for the three cylinders separately in one graph:

```
plot3 + facet_wrap(~cyl)
```



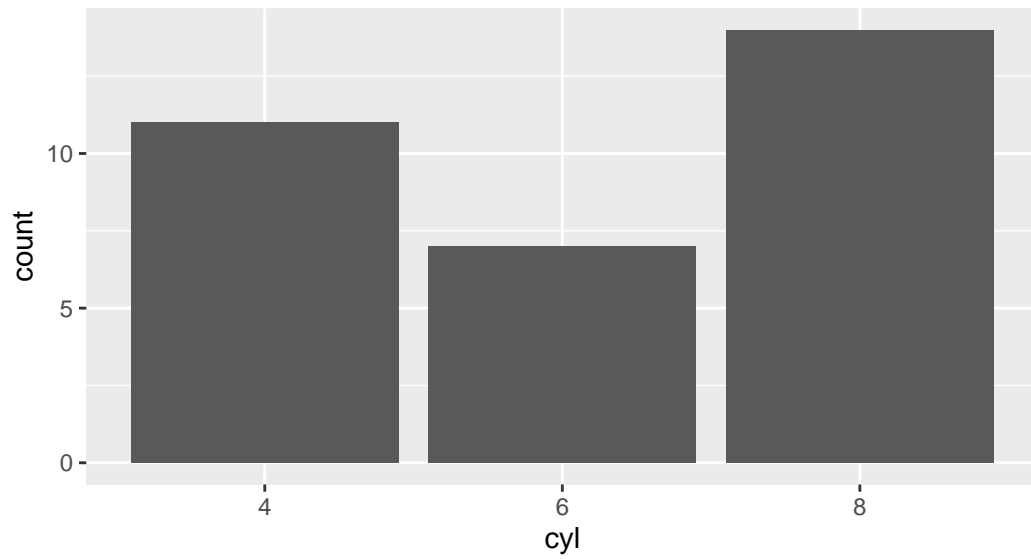
Facets can also be two-dimensional:

```
plot2 + aes(x=mpg, y=disp)+ facet_grid(cyl~gear)
```



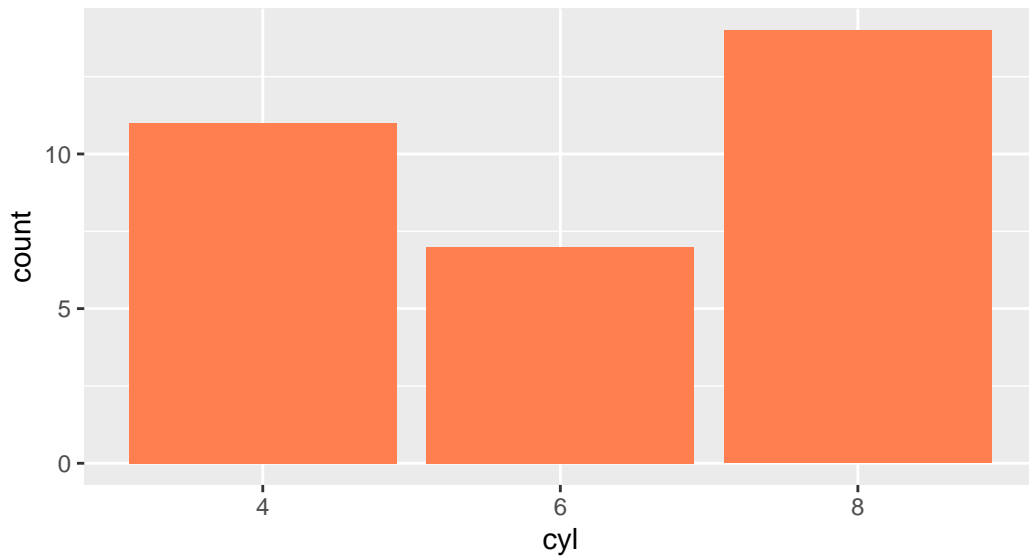
Example of a bar chart, showing the number of cars present in the category cylinders:

```
bar1 <- ggplot(mtcars, aes(x = cyl)) +  
  geom_bar()  
bar1
```



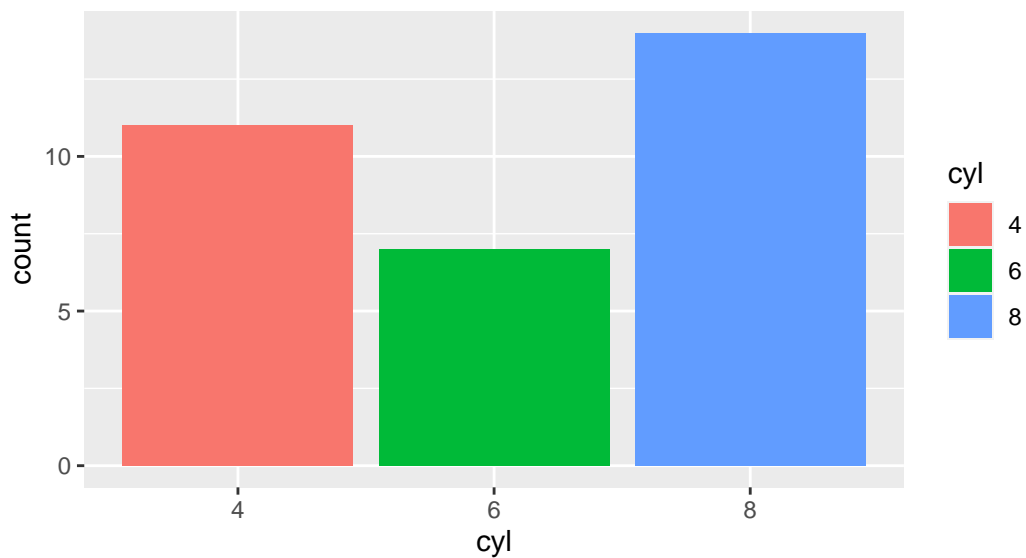
Bars can be filled with a color:

```
bar2 <- bar1+geom_bar(fill="coral")  
bar2
```



The three factors can be given different colors:

```
bar3 <- bar1+aes(fill=cyl)
bar3
```



There are endless possibilities in using ggplot to visualize data. A website describing extensively such possibilities is [ggplot tutorial](#). Besides, there are numerous internet sites you can consult, or fora where you can ask for help if you are stuck.

The R package `patchwork` makes it easy to play with combining and positioning of graphs and annotation. For instance, to combine the three bar charts, try this (you may need to install ‘patchwork’ and load it if you did not do that before):

```
bar1+ bar2 + bar3
```

Or:

```
bar1/bar2/bar3
```

Or:

```
(bar1 + bar2)/bar3
```

We hope that this exercise has given you a basic idea of how ggplot works and return now to our mock article.

End of exercise 7

Now that you have practiced a little with ggplot and patchwork, let’s apply this knowledge to produce graphs in our mock paper using the penguin data.

Exercise 8: producing graphs from data in the mock article

There are quite a few options to produce graphs from the penguin data set. Let’s practice a few.

- Reproduce Figures 2, 3, 4, 5, 6 of the mock paper using ggplot and patchwork and write some text to explain what the figures are showing.

End of exercise 8

It is possible to produce multiple figures in a code cell and reference them as subfigures (e.g., Figure 1a and Figure 1b) by using the option ‘fig-cap’ for the main caption and ‘fig-subcap’ for subcaptions. See Figure 10 for how to achieve this for two figures in one. You can then refer to the whole figure (@fig-plots) but also to a subfigure, e.g., the second figure (@fig-plots-2). See [Quarto - Cross References](#) for details.

```
{r}
#| label: fig-plots
#| fig-cap: "Plots"
#| fig-subcap:
#|   - "Plot 1"
#|   - "Plot 2"
#| layout-ncol: 2
```

Figure 10: Code to produce two sub-figures in one figure

2.3.4 Numbering sections, table of contents, list of tables and figures

Some journals may ask to add a list of tables (lot) and figures (lof) used in a manuscript, or you may want to show that in the thesis. You may also want to include a table of content (toc). If you want sections to be numbered, you have to indicate that by the command: `number-sections`. This is easily achieved by adding to the yaml:

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

Exercise 9: introduce numbered sections and lists of tables and figures and a table of contents in the mock article

Open the mock penguin_paper and modify the yaml as indicated above to produce a table of contents, a list of figures and a list of tables. Render and observe the output.

End of exercise 9

2.4 Inserting equations

Depending on your field of research, you may want to show mathematical/statistical equations in your paper. There are three ways to show equations in a Quarto document:

- inline equations, enclosed by a $\$$ sign at each side of the equation (such equations will not be numbered)
- unnumbered equations separate from the text, enclosed by two $$$$ signs at each side of the equation (such equations will not be numbered if you do not indicate that)
- numbered equations enclosed by two $$$$ signs at each side of the equation to which you can refer by giving them a name

In all cases LaTeX syntax is used, so you will have to learn that first. There are many cheat sheets available on internet where you can look up codes, one of them is provided as course document in Brightspace (latexcheatsheet.pdf). Though it may look intimidating at first, once you get to know it, it is actually quite intuitive and not more difficult than, for instance, the equation editor in Word. Moreover, when in Visual mode, Quarto makes life easy by showing the resulting equation while you type LaTeX code, so you can immediately see how the output will look like. Start with an equation by clicking on Insert: see Figure 11 and choose either Inline Math (will produce an equation inline) or Display Math (will produce an equation on a separate line).

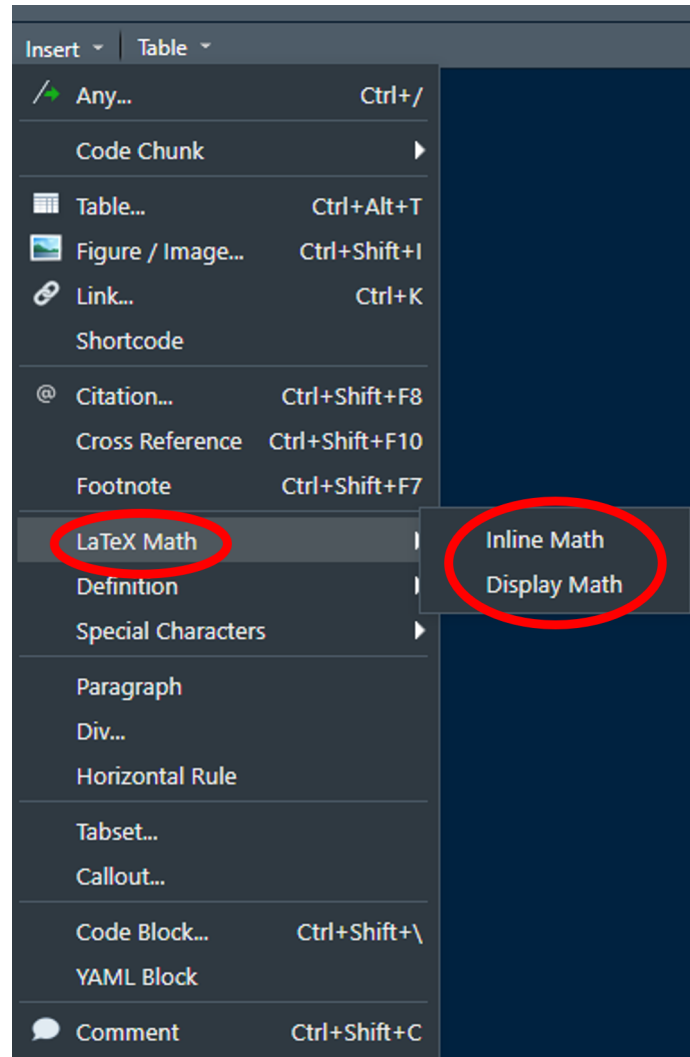


Figure 11: Use Insert for adding equations

The best way to learn is to just do it, so here we go.

Exercise 10: practising writing equations LaTeX style

Open a new Quarto document (to practice with equations):

File -> New File -> Quarto document

and give it a name, e.g., equation_exercise, and save it. To start an equation, click on

Insert -> LaTeX Math -> Inline Math (for an equation in the text)

or:

Insert -> LaTeX Math-> Display Math (for stand alone equations)

LaTeX code you will often need:

\wedge for superscripts

$_$ for subscripts

$\{ \}$ to keep expressions together

\backslash followed by the name of a greek letter will display greek letters in the equation

\backslash followed by `frac` will give fractions with the numerator in $\{ \}$ followed by the denominator in $\{ \}$.

Quarto will put symbols that represent quantities in italics by default, if you want it differently:

\backslash followed by `mathbf`, followed by $\{symbol\}$ will give you a boldface symbol

\backslash followed by `mathrm`, followed by $\{symbol\}$ will give you a roman symbol

Operators such as logarithms, exponentials need to be preceded by a backslash \backslash to prevent them from being put in italics

Try to introduce the following equations, inline:

The famous Einstein equation is $E = mc^2$ that everyone knows.

Or as stand-alone:

The famous Einstein equation is:

$$E = mc^2$$

Everyone knows this equation (but may not understand its implications).

The following equation is a thermodynamic relation:

$$\mu_w = \mu_w^0 + RT \ln X_w$$

The symbol μ_w represents the thermodynamic potential of water as a function of the mole fraction of water X_w . R is the gas constant and T absolute temperature. μ_w^0 is the standard chemical potential of water at 25 °C and 1 bar. An equation for X_w is:

$$X_w = \frac{1}{1 + m_i M_w}$$

An exercise with some Greek symbols:

$$\xi = \frac{n_i - n_0}{\nu_i} = \frac{\Delta n_i}{\nu_i}$$

An equation with an exponent and a fraction:

$$r = r_f \left(1 - \exp \left(-\frac{A_f}{RT} \right) \right)$$

A covariance matrix looks like this:

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{pmatrix} \tag{1}$$

An integral can also be exercised:

$$\begin{aligned} \tau &= \int_{c_{A,\text{in}}}^{c_{A,\text{out}}} \frac{dc_A}{-k} = -\frac{1}{k} (c_{A,\text{out}} - c_{A,\text{in}}) \\ &\rightarrow c_{A,\text{out}} = c_{A,\text{in}} - k\tau \end{aligned}$$

End of exercise 10

Quarto will number equations, if they are on a separate line and when you give them a name. In Visual mode, click on the three small dots on the right hand side, edit attributes, and fill in a name preceded by #: see Figure 12 and refer to them in the text as @eq-name (the prefix ‘eq-’ tells Quarto it is about an equation).

(In Source mode, you can achieve the same by adding at the end of the equation {#eq-name} after the two dollar signs .) We continue now with our mock article by adding an equation or two.

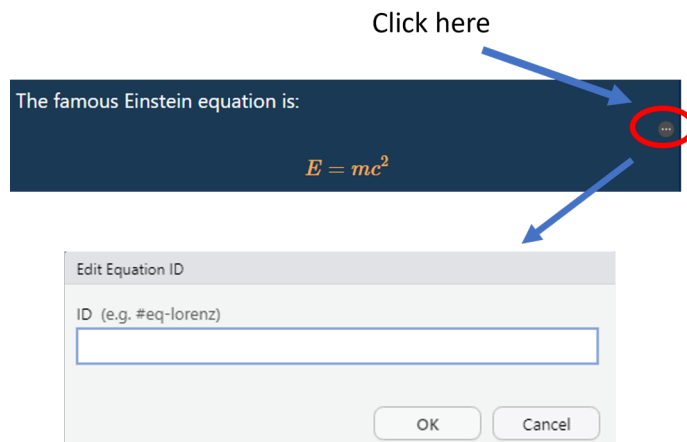


Figure 12: Screenshot showing how to label equations

Exercise 11: introduce equations in the mock article

Open the mock article and add the equation as it is written in equation (1), the linear regression model. Make sure that it is numbered. Render your article and check that it functions as you desire.

As an additional exercise (not mentioned in the mock article) you can add a second equation. Imagine that you want to test a quadratic equation as well (a polynomial of order two), next to the linear model. Write down the equation for such a model and make sure that it will be labeled as equation 2.

End of exercise 11

Mathpix: from image to LaTeX

There is a very handy snip tool called Mathpix with which you can copy equations, printed or even hand-written, from a digital source; the snip tool will translate it into a LaTeX equation that you can insert in a document. See [Image to LaTeX \(mathpix.com\)](https://mathpix.com). You have to sign up before you can use it but it is free.

2.5 Inserting references and reference styles

A BibTeX file is needed to manage bibliographic references⁴. BibTeX files have a specific format that can be created manually with a text editor but there are also many citation management tools such as Mendeley (available for WUR users), Zotero (free software), Google Scholar (free software). These software tools can all produce a BibTeX file (extension: *.bib). For your own research, you will, of course, have to build your own BibTeX file with your specific references, either manually or with one of the citation management tools; here we provide you with a bib file to do the exercise with the mock article that you can use directly (exercise.bib). You need to refer to a .bib file in the YAML using the command:

```
bibliography: filename.bib
```

The default citation style in Quarto is: author-year (Chicago Manual of Style). You can change this, of course, because the style of reference depends on the scientific journal and is defined in a so-called csl (citation style language) file. There are several formats you can download directly from journal websites. You can also provide a custom csl if needed. Refer to a specific csl file (custom made or provided by a journal) by mentioning it in the YAML using the command:

```
csl: filename.csl
```

Producing a bibtex file

A bibtex entry consists of the type (the word after @, article, book, book chapter, misc), a citation key, and a number of tags such as author, title, year, DOI. See the exercise.bib for how it looks like.

In Zotero, choose:

File -> Export Library -> select BibTeX format

see: [Zotero and LaTeX - Zotero - LibGuides at Graduate Institute of International and Development Studies](#)

See for a whole list of Zotero csl files: [Zotero Style Repository](#)

In Mendeley, choose:

Mendeley Preferences -> BibTeX tab

see: [Bibtex – Mendeley Blog](#)

When using Google Scholar:

⁴Quarto uses the default pandoc citation handling based on citeproc but you could also use natbib or BibLaTeX, see [Quarto - Citations & Footnotes](#).

Go to Google Scholar and search for a publication. Click on the site button and then on the link “Import into BibTeX”.

see: [Using Google Scholar to download BibTeX citations – texblog](#)

If you want to stick to the default style then you do not need to specify a csl in the YAML.

Scientific references can subsequently be inserted in Quarto documents in various ways:

- First, make sure that the name of the bibliographic file is mentioned (and its location in subfolders if it is not in the root of your project). Then, you can search for a specific reference in that *.bib file by clicking on : Insert > Citation. The reference will then be added in the text.
- you can also search for a DOI if you have that available. When Quarto is able to find it, it will insert that reference in the text and add that reference automatically into the *.bib file
- you can also enter a reference manually in the text if it is present in the bib-file. The syntax is @authoryear (no parentheses around the reference) or [@authoryear] (parentheses around the reference).

The section References will be automatically added by Quarto to your manuscript as soon as references are discovered in the text when the format is html, when you want to produce a pdf, add a section References at the very end of your manuscript.

Exercise 12: insert references in the mock article

Open the mock penguin_paper.pdf file and locate its references. We have already provided you with a bibtex file where these references are present. You need to refer to this bibtex file in the YAML, otherwise the references will not be recognized. Try to insert the references as they are mentioned in the penguin_paper example:

Insert -> Citation

Choose from the options shown in the pop-up screen (see Figure 13)

A very handy option is to insert a reference from a DOI. Search the internet for a scientific article related to penguins, find the DOI and insert it. If Quarto recognizes it and finds the reference it will insert it in the text and add it to the bibtex file as well.

End of exercise 12

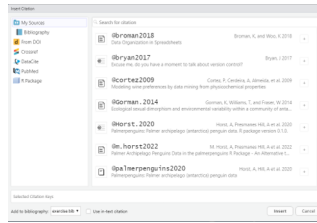


Figure 13: Options to choose from for adding a reference

It may happen that you want to cite references in the Reference section that are not cited in the text. To achieve that write in the YAML:

2.5.1 Using existing Journal Formats

If you want to publish in a specific journal format, it is good to know that there are already quite a few formats available for Quarto. Of course, you can also make your own template, or perhaps obtain one from the journal itself. Suppose you want to publish in PLOS, you can enter the following command in the Terminal Mode:

Tools > Terminal > New Terminal

type:

```
quarto use template quarto-journals/plos.
```

To start an article from scratch in a specific format, you can use in the Terminal mode the command:

```
quarto use template
```

Currently, there are templates for ACM, PLOS, ASA, Elsevier, Biophysical, ACS, JSS. If you want to use one of them, type in the Terminal mode (choose the one you need):

```
quarto use template quarto-journals/acm
```

```
quarto use template quarto-journals/plos
```

```
quarto use template quarto-journals/elsevier
```

```
quarto use template quarto-journals/acs
```

```
quarto use template quarto-journals/jss
```

This will start a new article with default contents.

If you have already an existing document for which you want to use a different format, then type in Terminal mode:

quarto install extension quarto-journals/plos

These templates will be written to a folder ‘__extensions’.

Finally, add in your document yaml:

```
format:
  pdf: default
  plos-pdf:
    keep-tex: true
```

This will then produce a pdf with the desired format. it is also possible to create an own template in case the one you need is not available. See for more information the Quarto guide [Quarto - Journal Articles](#)

2.6 Producing Word documents

Though it is not the best way to practice Open Science, Word docs are still very popular (where you then probably have to copy and paste data from excel, or a statistical program). Some scientific journals even require that you submit manuscripts as Word files. Also some supervisors may ask you to produce Word files for them (but maybe you should do an attempt to persuade them to also learn Quarto...). In any case, Quarto has the option to produce Word output. So you can still practice Open Science in Quarto and then produce Word files. So, reluctantly, we spend some attention to it (see [How to create Word docs from R or Python with Quarto | InfoWorld](#)). There is also a Quarto guide: [Quarto - Word Basics](#). You can even create your own custom MS Word styles reference docx: see [Quarto - Word Templates](#).

Exercise 13: produce a word file from the mock article

Open the penguin_paper.qmd file and change the output in the YAML to docx. Render and observe the output.

End of exercise 13

2.7 Inserting comments in Quarto files

The option to work with track changes in Word is well-known and very useful: you and your supervisors can write comments, make corrections and change text such that everyone can see what changes are desired. Such an option also exists in Quarto:

Insert -> Comment

Use this option if there is text that you want to change or to add a comment. The result can be seen in the .qmd file but not in the output (it will be neglected in the rendering process). The # prefix that is added by Quarto if you use this option highlights the text and can be omitted if desired.

3 Advanced formatting using latex packages

When you want to produce pdf output, then you have the option to use the many latex packages that are available, for instance, to insert line numbers, change line spacing, different fonts, and many more. This can be achieved by adding to the yaml the command: include-in-heading. See [Quarto - PDF Basics](#) for details. You can find such packages by doing an internet search and locating the one you need. See the following section for an example.

3.1 Inserting line numbers

If you want to submit an article to a scientific journal you will be asked frequently to insert line numbers in the manuscript so that reviewers can easily refer to parts of the text. There is a latex package that you can use for this purpose called `lineno`. You will have to adapt the yaml in this way to make this happen:

```
format:
pdf:
  include-in-header:
    text: |
      \usepackage{lineno}
      \linenumbers
```

Note the use of the character | after the text subkey, it indicates that the value is a multi-line string. Note also the two spaces used to indent in the yaml, this is essential, otherwise you will get an error message! Let's practice with the mock article.

Exercise 14: introduce line numbers in the mock article

Open the mock penguin_paper and introduce line numbers in it as indicated above. Render and observe the output.

End of exercise 14

It is also possible to produce pdf output where the R code you use is line-numbered (not the text itself). In order to achieve that, adapt the yaml like this:

```
format:
  pdf:
    code-line-numbers: true
```

3.2 Changing line spacing

When you submit an article, journals may ask you to submit your manuscript with a specific line spacing. There is a latex package called ‘setspace’ that has three options: single line spacing, one-and-half spacing and double spacing. If you want to change line spacing, for instance, to double spacing, adapt the YAML as follows:

```
format:
  pdf:
    include-in-header:
      text: |
        \usepackage{setspace}
        \doublespacing
```

The other two possible commands are ‘singlespacing’ and ‘onehalfspacing’. Try to modify your mock article so that it results in double spacing.

Exercise 15: introduce double line spacing in the mock article

Open the mock penguin_paper and introduce double spacing as indicated above. Render and observe the output.

End of exercise 15

3.3 Creating an index

For books, an index can be handy, for stand-alone documents this will normally not be done. You can create an index for pdf output using the latex package `makeidx`:

```
pdf:
  include-in-header:
    text: |
      \usepackage(makeidx)
      \makeindex
  include-after-body:
    text: |
      \printindex
```

The index will then be printed at the end of your book. The words you would like to be indexed need to be marked like this, for instance, `Quarto\index{Quarto}` and it will mark the word Quarto to be placed in the index. For non-pdf output the command will be ignored.

4 Home work

Based upon what you have learned today, you should be able to produce two more mock articles on your own. That is the assignment for home work so that we can bind the three mock papers together in a mock PhD thesis in the second day of the course.

4.1 Own data?

If you have own data available that you would like to use in a future manuscript, then you can also practice with that. If you do not have such data, we have provided you with two data sets to practice with, the pottery data set and the wine data set.

4.2 The pottery data set

Pottery info and data

Data were collected from archaeological sites in the UK with the research objective to investigate if a particular chemical composition of pottery discovered at various sites could be linked to a particular site. The original paper in which the data were published is to be found in TUBB, PARKER, and NICKLESS (1980). (The data are also used in and part of the R package `car` (companion to applied regression) and SAS). You can

find the data in a file called Pottery_data.csv on Brightspace. What is reported is actually the concentration of the oxides of the elements. You can read a screendump of the Introduction of the original article in the file Pottery_intro.docx, also available on Brightspace.

Exercise 16: produce a mock article on the pottery data

Write a small paper using the data provided with the usual scientific format. Then, import the data into a dataframe called Pottery and explore the data: what are in the rows and what in the columns? Indicate in the Introduction *using inline R code* how many sites there are, how many elements were analyzed and what the ranges of the concentrations are. |

1. Create boxplots for the element Al for each site using ggplot.
2. Do the same for the other elements
3. You may have noticed that this is not a very efficient way of working: repeating the same action five times. There is a much more efficient way but then you need tidy data. The data in Pottery_data.csv is in the so-called wide format, i.e., one observation row per subject (site) with each measurement (chemical element) present in a different column. You can change a wide format in a long format so as to make the data tidy, i.e., one observation row per measurement, i.e., multiple rows per subject (site). The **tidyverse** package contains a command to do this: `pivot_longer()`. Use this command as follows and observe the output in the new dataframe Pottery_long:

```
Pottery_long <- Pottery %>% pivot_longer(!Site, names_to="elements", values_to="concentration")
```

4. Once you have the data in the long format, make use of the ggplot command `facet_grid()` to produce faceted plots and add some labels to the plots.
5. Apply a linear model for the differences in concentrations and relate them to the different sites and elements. This is achieved by a full factorial ANOVA model with Site and elements as factors with interaction. Write the equation for such a model and make sure the equation will be numbered.
6. Perform linear regression using the `lm` function from base R and then do an ANOVA analysis and make a Table of the ANOVA results. Make sure that the table is numbered. Comment on the results.
7. A visualisation of the interaction effect can be made using the function `emmip()` from the R package **emmeans** (you may have to install the package first). Show this in a figure and make sure it is numbered in the text.

8. Perform pairwise comparisons to test which combinations are significant. Use Tukey's method to correct for false positives (type 1 errors). Use the `emmeans()` function from the `emmeans` package for this. Show the results in a table and make sure it is numbered in the text.
9. Show the results and write some text in which you refer to the figures and tables. Add some references from the `exercise.bib` file, or other relevant articles you may find on the internet.

End of exercise 16

4.3 The wine data set

i Info about the wine data set

A large data set is available on quality of wine in relation to several chemical parameters. This dataset is related to Portuguese “Vinho Verde” wine. For more details, consult the reference \[Cortez et al., 2009\]. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

Exercise 17: produce a mock article on wine data

Write a short note on the prediction of quality of wine using the variables investigated with the usual format of Introduction, Materials and Methods, Results and Discussion and References.

1. Use inline code to indicate how many different samples of wine are available and how many variables.
2. Spend some words on exploratory data analysis. Create boxplots to show how the variables are distributed and comment on it.
3. Perform a simple linear regression of the dependent variable Sweet taste various the other variables to see which metabolites are significantly related with sweet taste.
4. Before putting some trust in this model, you need to check if there are collinearity problems. One way of doing this is by calculating the VIF values for each variable: as long as the VIF values are below 10, there is no collinearity problem.
5. Not all independent variables are relevant. Produce a table that shows which variables have a significant effect and which not.

6. It is also important to get an impression whether or not the assumption of normality is reasonable. Make a QQ plot of the residuals to check this.
7. Use inline code in the text to report an R^2 value.

End of exercise 17

5 Producing a thesis (book)

In the first part of the course and in the homework, you have produced three mock articles that will now be used to put them together. You learned the basic things to put a scientific article together. Of course, a proper thesis goes a step further than just binding articles together and consists of an Introduction and a General Discussion, and perhaps appendices, so you need to write them as well. When all the elements are there, the next thing to do is to bind it all together into one Quarto document.

To start with a book, click on

File -> New Project

and you will be faced with options as in Figure [Figure 14](#):

Click on New Directory and then on Quarto Book, then give a name to the project (e.g., mythesis). A minimal number of files will be created that will form the chapters of a book:

- an index file (index.qmd), containing sections such as preface, acknowledgements, headings (will be unnumbered)
- a yaml file (_quarto.yml), containing rendering options, the initial configuration for the book (will be unnumbered)
- an introduction (intro.qmd) (will be numbered as a book chapter)
- a summary (summary.qmd) (will be numbered as a book chapter)
- references (references.qmd) (will be unnumbered)
- a cover (cover.png) (will be unnumbered)

You can take these files as a starting point, rename them, and replace their (minimal) contents with your own text and code. All chapters are numbered by default: all headings create a numbered section. You can customize numbering depth using the number-depth option in the yaml. Chapter numbering can be undone, if you so wish, by adding `{.unnumbered}` after the chapter title. Numbered and unnumbered chapters can be mixed and you can refer to both of them but you cannot cross-reference to figures and tables in unnumbered chapters.

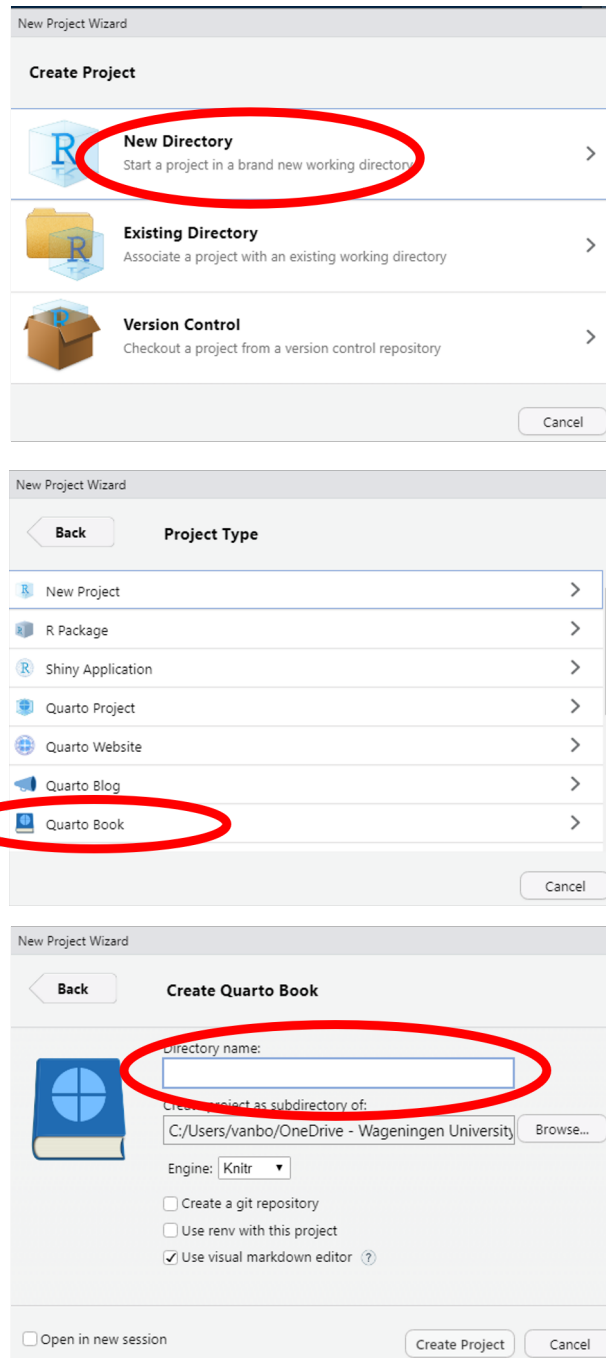


Figure 14: Options to choose from to start a Quarto book project

Cross-references to figures and tables in numbered chapters will result in, for example, Figure 2.3 and Table 3.4. In other words, figures and tables get automatically the chapter number first in books.

Take a look at the file ‘references.qmd’ that is produced upon starting a Quarto book and you will notice a so-called div (three colons at the beginning and three at the end). If you want to have references at a certain spot in your thesis (e.g., at the end of each chapter) you have to use such a div with the id `{#refs}` at the place where you would like the references to be generated.

i divs and spans

divs are special entities that allow to apply identifiers and styles to a region of a document, while spans do the same but then for a selected text (for instance to give a word or a line a specific color). Call-outs like this one are actually divs.

To incorporate your own articles into a thesis, you can use them as such but you have to remove the yaml from each article because the `_quarto.yml` will be valid for the whole of the book, and the chapters do not need a yaml. Then, give a heading to each chapter, starting with `#`, and the subsequent sections need to have two `##`; this will ensure the right numbering of the chapters and its sections. The headings that you give to each chapter should be added to the `_quarto.yml` file; the numbering of the chapters will correspond to the order in which you feed them to the `_quarto.yml`. To make life a bit easier you could name them (just a suggestion, you may choose your own names):

01-Intro

02-Chapter2

03-Chapter3

etc.

The index file is always needed.

Once you have copied the articles to the directory that you have created as a Quarto book (don’t forget to copy the data and figures as well that you have incorporated in the papers!), you will see a new button in the Environment pane named Build: see Figure 15. Click on it, then on Render Book and you will have the choice to render different formats or a pdf book. When you made the choice, rendering will start and if everything goes well, your book will appear in the viewer pane.

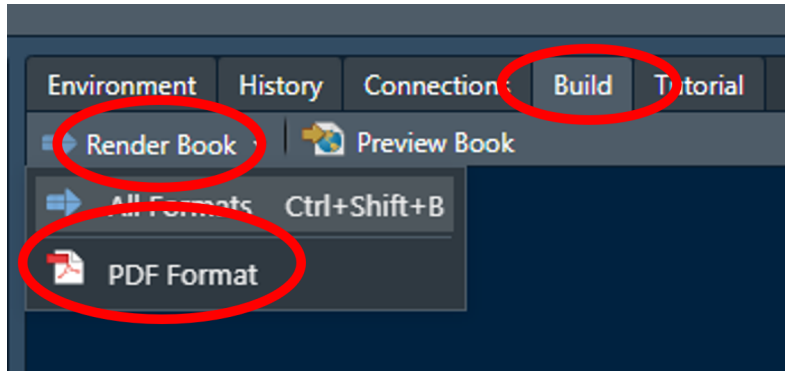


Figure 15: Screenshot showing how to render a book

Exercise 18: produce the backbone of a thesis

Start a Quarto book project as indicated above and take a look at the content of the files that are produced by Quarto, especially also the yaml file: you can change the default settings into your specific requirements, if needed. The `__quarto.yml` could look like this:

```
project:
  type: book

book:
  title: "Thesis"
  author: "Your name"
  date: last-modified
  chapters:
    - index.qmd
    - intro.qmd
    - penguin_paper.qmd
    - wine_paper.qmd
    - pottery_paper.qmd
    - summary.qmd
    - references.qmd

bibliography: exercise.bib
format:
  pdf:
    documentclass: scrreprt
editor: visual
execute:
  echo: false
```

```
warning: false
message: false
```

Then, delete the text in the index.qmd file below “# Preface” and write some text here (you may give another name to this section than Preface if you prefer). Do the same for the file intro.qmd file below “# Introduction” and the file summary.qmd below “# Summary”. Then, add the three mock articles that you produced as .qmd files to the directory. Make sure that the headings of the chapters correspond to the ones in the yaml file. To make it look a bit like a thesis, you could write an additional introduction chapter, a general discussion and an Acknowledgement. Perhaps you want to add Appendices as well. That is all possible.

End of exercise 18

Not discussed here, but you can generate an index of keywords for pdf output by making use of a Latex package called `makeidx`, see [Quarto - Book Structure](#). Also in the same guide, you can find information about how to add appendices.

You can include appendices by adding an appendix key in the yaml:

```
book:
  appendices:
    • filename1.qmd
    • filename2.qmd
```

Appendices are numbered using uppercase letters, for instance, “Appendix A: filename1”, “Appendix B: filename2”. For more information see [Quarto - Book Structure](#).

To make references to other chapters possible, you will have to add a `#sec` id, for instance, to refer to the Introduction from another chapter, you will have to label the introduction chapter like this: `# Introduction {#sec-intro}`. You can then refer to the introduction chapter from anywhere in the book as `@sec-intro` (i.e, the same syntax as for figures, tables and equations). This will then produce the number of that chapter. If you want not only the number but also the prefix Chapter you can write `[Chapter -@sec-intro]`, see [Quarto - Book Crossrefs](#). If you want a page-footer for all pages, this is possible by adding this to the yaml:

```
book:
  page-footer: “your text here”
```

Cover images can be included for Epub and html output but for pdf output it is a little more complicated but not impossible, see [GitHub - nmfs-opensci/quarto_titlepages: A Quarto extension for making title and cover pages for PDF output](#).

Book output is written to the `_book` directory of the project you have defined.

6 Wrapping up the Quarto Course

In conclusion, we have tried to offer you basic aspects that should help you on the way to write scientific articles with Quarto as the tool and to compile them together in a thesis, or a book in general. There are many more options than we were able to discuss here. There are excellent tutorials and guides that we highly recommend see: [Quarto - Guide](#). Furthermore, there are Quarto extensions, a way to modify and extend the behaviour of Quarto. Check out this website for an overview of already existing extensions: [quarto-ext · GitHub](#) and [Quarto - Quarto Extensions](#).

Also, consider to use Git and Github in your work as part of your endeavor to contribute to Open Science. Unfortunately, we were not able to cover that in this course, it would require another day. But there are excellent tutorials and manuals on internet. One of the best is from Jenny Bryan [Let's Git started](#). She also has written an article on the topic (Bryan 2017).

Another aspect we did not cover is the use of Quarto to produce slides. The nice thing is that you can couple it directly to your manuscript, i.e., using the same code and text. Especially, the possibility to produce html reveals slide presentations works fantastic, but you can also have powerpoint presentations as output. A nice example can be seen in the presentation of Julia Muller (R Ladies Freiburg): [\(1341\) R-Ladies Freiburg \(English\) - Getting to know Quarto - YouTube](#). She also uses the palmer penguins data to illustrate what is possible with Quarto. Since Quarto is released only very recently, you may expect many more applications on the internet in the very near future.

References

- Broman, Karl W., and Kara H. Woo. 2018. "Data Organization in Spreadsheets." *The American Statistician* 72 (1): 2–10. <https://doi.org/10.1080/00031305.2017.1375989>.
- Bryan, Jennifer. 2017. "Excuse Me, Do You Have a Moment to Talk about Version Control?" <http://dx.doi.org/10.7287/peerj.preprints.3159v2>.
- Horst, Allison Marie, Alison Presmanes Hill, and Kristen B Gorman. 2020. *Palmerpenguins: Palmer Archipelago (Antarctica) Penguin Data*. <https://doi.org/10.5281/zenodo.3960218>.
- M. Horst, Allison, Alison Presmanes Hill, and Kristen B. Gorman. 2022. "Palmer Archipelago Penguins Data in the Palmerpenguins R Package - An Alternative to Anderson's Irises." *The R Journal* 14 (1): 244–54. <https://doi.org/10.32614/rj-2022-020>.

- Perkel, Jeffrey M. 2022. “Cut the tyranny of copy-and-paste with these coding tools.” *Nature* 603 (7899): 191–92. <https://doi.org/10.1038/d41586-022-00563-z>.
- TUBB, A., A. J. PARKER, and G. NICKLESS. 1980. “THE ANALYSIS OF ROMANO-BRITISH POTTERY BY ATOMIC ABSORPTION SPECTROPHOTOMETRY.” *Archaeometry* 22 (2): 153–71. <https://doi.org/10.1111/j.1475-4754.1980.tb00939.x>.
- Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. <https://ggplot2.tidyverse.org>.