Nicholas Tierney

Quarto for Scientists

Table of contents

About this

This is a book on using Quarto for writing and document preparation, aimed for scientists. It was initially developed as a half day workshop, "Rmarkdown for scientists". This focusses on Quarto, which is a next-generation rmarkdown. It is now developed into a resource that will grow and change over time as a **living book**.

This book aims to teach the following:

- Getting started with your own Quarto document
    - Using Rstudio
    - Visual Studio Code
- Improve workflow:
    - RStudio
        * Demonstrate rstudio projects
        * Using keyboard shortcuts
    - Quarto projects
- Export your Quarto documents to PDF, HTML, and Microsoft Word
- Better manage figures and tables
    - Reference figures and tables in text so that they dynamically update
    - Create captions for figures and tables
    - Change the size and type of figures
    - Save the figures to disk when rendering a document
- Work with equations
    - Inline and display
    - Caption equations
    - Reference equations
- Manage bibliographies
    - Cite articles in text
    - Generate bibliographies
    - Change bibliography styles
- Debug and handle common errors with Quarto
- Next steps in working with Quarto:
    - How to extend yourself to other formats, such as slides, websites, books, and more

Why write this as a book?

This book started out its first life being around rmarkdown. There are many great books on R Markdown and it's various features, such as "Rmarkdown: The definitive guide", "bookdown: Authoring Books and Technical Documents with R Markdown", and "Dynamic Documents with R and knitr, Second edition", and Yihui Xie's thesis, "Dynamic Graphics and Reporting for Statistics".

With the release of Quarto, I wanted to translate the materials I developed in "Rmarkdown for scientists" to cover the same material. Here are some resources that I really liked for learning Quarto:

- The Quarto "get started" guide
- The Quarto guide "Quarto manuscripts"
- The Quarto chapter in "R for data science"
- Making shareable documents with Quarto from, from OpenScapes
- Alison Hill's blog post: "we don't talk about Quarto"
- Mine Çentinkaya-Rundel's talk "Quarto for academics"

While the Quarto guide is extensive, and indeed their "Quarto manuscripts" guide covers a lot of the ground in this book.

So, why write a book?

Good question. The answer is that writing this as a book provides a nice way to structure the content in the form of a workshop, in a way suitable for learning in a few hours. It is not to say that there aren't already great resources out there; there are. It is instead adding to the list of other useful information out there on the internet. I considered the Rmarkdown for Scientsts book and course a success, and it helped myself and others understand and better use rmarkdown. So I guess, to answer a question with another question:

Why NOT write this as a book?

How to use this book

This book was written to provide course materials for a half day course on Quarto

We worked through the following sections in the book in a half day:

- why use Quarto
- installation

- what is RStudio?
- suggested workflow and hygiene
- how to use Quarto
- using Quarto with pdf, html, and Word
- what are some useful keyboard shortcuts
- adding captions to tables and figures
- changing figures
- adding mathematics
- citing figures and tables
- changing citations and styles

With the remaining sections being used as extra material, or have since been written after the course:

- fixing some common problems in Quarto
- what are some alternative outputs of Quarto?
- where to go next?
- suggested references

Course materials can be downloaded by using the following command from the `usethis` package:

```r
usethis::use_course("njtierney/qmd4sci-materials")
```

Licence

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

License

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

1

Installation

In this section, the aim is to have everyone setup with R, RStudio, and Quarto

1.1 Overview

- **Duration** 15 minutes

1.2 Questions

- How do I install R?
- How do I install Quarto?
- How do I install LaTeX in a sane way?

1.3 Software Setup

1.3.1 R

1.3.1.1 Windows

https://cloud.r-project.org/bin/windows/

1.3.1.2 MacOS

https://cloud.r-project.org/bin/macosx/

1.3.1.3 Linux

https://cloud.r-project.org/bin/linux/

1.3.2   RStudio

https://posit.co/download/rstudio-desktop/#download

1.3.3   Quarto

Quarto installation page

1.4   Checking you are up to date

To ensure you are up to date, run the following script to install the packages.

```
install.packages("quarto")
install.packages("knitr")
install.packages("here")
install.packages("tidyverse")
install.packages("broom")
install.packages("fs")
install.packages("usethis")
```

1.5   A note on PDF

Quarto documents can be compiled to PDF, which is a great feature. In order to convert the documents to PDF, they use a software called LaTeX (pronounced la-tek or lay-tek).

Installing LaTeX is thankfully handled when you install Quarto, as Quarto includes a built-in Latexmk engine.

1.5.1   PDF / LaTeX Pain

Installing LaTeX can be a pain, but thankfully Yihui Xie has put a lot of time and energy into making an easier way to install it - `tinytex`. `tinytex` is an R package that installs a sane, lightweight (<200Mb) version of LaTeX.

If you are running into issues rendering a PDF, you can try the following:

```
tinytex::install_tinytex()
```

If you get the following error, **this is good**! As it means that TeX has already been installed:

```
Error: Detected an existing tlmgr at /usr/local/bin/tlmgr. It seems TeX Live has been inst
```

Alternatively, you can run the following from the terminal

```
quarto install tinytex
# follow the prompts from here
```

1.5.2   Problem solving with LaTeX

If you have any problems with installing `tinytex`, I recommend you check out
the tinytex FAQ page.

1.6   Test Script

You should be able to run the following code on your machine

```
library(quarto)
library(knitr)
library(here)
library(tidyverse)
library(broom)
library(fs)
library(usethis)
```

2

Why Quarto

The goal of this section is to briefly discuss why we want to learn Quarto, the benefits, and the barriers to using it.

2.1 Overview

- **Teaching** 3 minutes
- **Exercises** 5 minutes

2.2 Questions

- What is the value in a reproducible report?
- What is Markdown?
- Can I combine my software and my writing?

2.3 Objectives

- Learn how to use Markdown
- Think about why you want to use Markdown

> **i** Your Turn
>
> 1. **Why are we here** Form small groups of 2-4 with your neighbours and discuss how you expect learning Quarto might benefit you.

2.4 Reproducibility is a problem

It is unfortunately a common, seemingly evergreen problem that a lot of people cannot reproduce scientific work. This might appear to be a "current" problem, but it has indeed been a problem throughout a lot of scientific history. To illustrate this, here's a nice article by Rich FitzJohn, Reproducible research is still a challenge, which was written 10 years ago, in 2014, and provides a list of the challenges and lessons learned in making research reproducible. The list is still relevant. This problem isn't completely solved. But, we can make it easier to solve, to get further.

Reproducibility isn't just something that impacts a few people, and it's not cheap. A 2010 estimate stated that in the biomedical industry, in the USA, irreproducibility (not being able to reproduce a given piece of work) costs \$28 Billion dollars annually [1]. That was one country, one field, and one year.

So what can we do about it?

2.5 Literate programming is a partial solution

The idea of literate programming shines some light on this dark area of science. This is an idea from Donald Knuth where you combine your text with your code output to create a document. This is a *blend* of your literature (**text**), and your programming (**code**), to create something that you can read from top to bottom. Imagine your paper - the introduction, methods, results, discussion, and conclusion, and all the bits of code that make each section. With Quarto, you can see all the pieces of your data analysis all together.

> 🔥 Some history: Quarto, Rmarkdown, Sweave
>
> Literate programming was a popular idea, and it has had some interesting discussion and contributions over the years. Notably, in the R ecosystem, the *Sweave*, (The **S language** + weaving together text and code) program provided a way to write text and code together. As with any technology, there were some speedbumps with using Sweave, and some of the reasons we are not teaching it now is because:
> - It uses a form of LaTeX, which provides great flexibility at the cost of complexity.

---

[1] The article, Freedman, 2010, Heard via Garret Grolemund's great talk

> - Printing figures involves additional work
> - There isn't a way to save (cache) your work. Every analysis has to be repeated from start to finish. This was time consuming.

2.6 Markdown as a new player to legibility

In 2004, John Gruber, of daring fireball created Markdown, a simple way to create text that rendered into an HTML webpage. The core idea was that you could write plain text (not text inside a MS Word/WordPerfect/Pages/Proprietary Format Document), and it would look readable, then get rendered into HTML.

The idea took off.

2.6.1 A brief example of markdown

2.6.1.1 Markdown text

```
- bullet list
- bullet list
- bullet list

1. numbered list
2. numbered list
3. numbered list

__bold__, **bold**, _italic_, *italic*

> quote of something profound


```r
# computer code goes in three back ticks
1 + 1
2 + 2
image(volcano)
```
```

2.6.1.2 Rendered result

- bullet list
- bullet list
- bullet list

1. numbered list
2. numbered list
3. numbered list

bold, **bold**, *italic*, *italic*

> quote of something profound

```
# computer code goes in three back ticks
1 + 1
```

```
[1] 2
```

```
2 + 2
```

```
[1] 4
```

```
image(volcano)
```


With very little marking up, we can create rich text, that **actually resembles** the text that we want to see.

Some other nice features of Markdown include:

Feature	Markdown	rendered
superscript	`2^nd^`	2nd
subscript	`CO~2~`	CO_2
strikethrough	`~~mistake~~`	~~mistake~~
links	`[text](https://quarto.org/)`	text
links	`<https://quarto.org/>`	https://quarto.org/)

Feature	Markdown | rendered
images	`![alternative text](link-to-image)` | (cannot render in a table)

For more examples and details of using markdown in Quarto, see the official Quarto "Markdown Basics" documentation.

3

What about Rmarkdown?

Issues around Sweave led to the development of knitr, and subsequently Rmarkdown, which used the knitr engine. You could run more than R code in rmarkdown, in fact there are over 60 engines available, from awk and bash, to haskell, perl, php, sql, scala, stata, javascript, python, julia, and even C.

- Through rmarkdown there were many approaches to document processing, such as bookdown for books, blogdown for blogs and websites, xaringan for slide decks.

However, there are a few points of friction:

- You need to call it from R to use it. No problem for R users, but what if you use python? Or Julia? Or javascript? If you are a python user, using R to use python just might not be in your workflow.
- There are great packages that provide extensions, such as blogdown for blogs, bookdown for books, and xaringan for slides. Each of these systems is an iteration towards something awesome, and it's only natural they might be a little bit different. Overall, there are differences between these systems that might cause stumbles.
- Quarto, instead of being an R package, is a separate piece of software, that you can call from the command line (terminal). This means other pieces of software can use it to create their own literate programming documents. Well, that's my understanding.

Here are some diagrams to illustrate this point:

Figure 3.1: Rmarkdown can talk to Python, but it works from within R

In rmarkdown, we are working in rmarkdown, and that uses knitr to talk to R and handle the document generation:

But with Quarto, we have this general interface, where Quarto can talk to different programming languages. Not pictured, but the "R engine" is in fact, `knitr`:

Figure 3.2: Quarto is a separate program

> 💡 Converting Rmarkdown to Quarto
>
> One of the big things that is different between Rmarkdown and Quarto are the chunk options. In Rmarkdown they are in the chunk header, and in quarto they are in the body of the function. To help change these over appropriately, you can run this code, which will rewrite the chunk headers.
>
> ```r
> knitr::convert_chunk_header(
>   input = "paper.qmd",
>   output = identity
> )
> ```
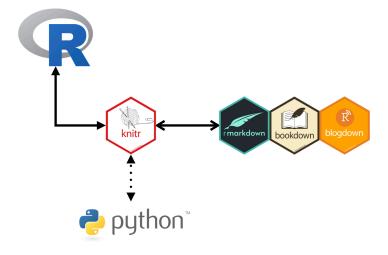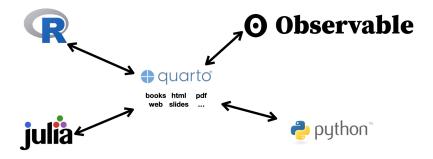
> ℹ Your Turn
>
> 1. **Learn to use Markdown** In your small groups, spend five minutes working through this brief online Markdown tutorial

4

Quarto helps complete the solution to the reproducibility problem

So, how do we combine this with our R code, into a literate programming environment?

Quarto provides an environment where you can write your complete analysis. It weaves your text, and code, and its output together into a single document.

For example, look at the following report:

Exploring gapminder
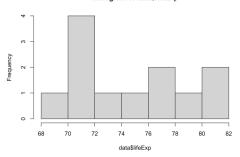
AUTHOR
Your Name

```
library(tidyverse)
library(broom)
```

```
data <- read_csv(here::here("data/gapminder_oz.csv"))
```

Introduction

let's look at the lifespan

```
hist(data$lifeExp)
```

Histogram of data$lifeExp

Let's fit a simple linear model of the effect of year on life expectancy

```
fit <- lm(lifeExp ~ year, data = data)
fit
```

```
Call:
lm(formula = lifeExp ~ year, data = data)

Coefficients:
(Intercept)          year
  -376.1163        0.2277
```

And let's look at the coefficient table:

```
library(broom)
fit_coef <- tidy(fit)
knitr::kable(fit_coef,
             caption = "A table of the coefficients")
```

term	estimate	std.error	statistic	p.value
(Intercept)	-376.1162984	20.5471585	-18.30503	0
year	0.2277238	0.0103796	21.93960	0

A table of the coefficients

The effect of year on life expectancy is 0.2277238.

How did we generate it?

```
---
title: "Exploring gapminder"
author: "Nicholas Tierney"
format: html
---
```

````
```{r}
#| label: library
#| message: false
library(tidyverse)
library(broom)
```


```{r}
#| label: data-read-in
data <- read_csv(here::here("data/oz_gapminder.csv"))
```



# Introduction

let's look at the lifespan

```{r}
#| label: hist-life-exp
hist(data$lifeExp)
```



Let's fit a simple linear model of the effect of year on life expectancy

```{r}
#| label: example-lm
fit <- lm(lifeExp ~ year, data = data)
fit
```



And let's look at the coefficient table:

```{r}
#| label: coef-table
library(broom)
fit_coef <- tidy(fit)
knitr::kable(fit_coef,
             caption = "A table of the coefficients")
```

The effect of year on life expectancy is `{r}  fit_coef$estimate[2]`.
````

We *render* this code and it creates this report!

It has a plot, it has a table, we even refer to some of the values in the text - the last line of the report looks at the effect of year.

But what if the data changes? At the moment we are looking at only Australia - say we get the full dataset, what happens then?

Say you'd created your report by hand in microsoft word, and with a graphical user interface software, you would need to:

1. Go back to the GUI, re run the analysis
2. Import the results into Excel
3. Create your graph
4. Copy the graph into Word
5. Copy the results of the coefficients into the text
6. Copy the results of the coefficient table into the text.

This is painful.

And what if someone wants to know *exactly* how you did your analysis?

This process isn't exactly sharable.

But if you did it in Quarto?

Just update the data, and render the document again, and get an updated document:
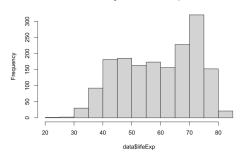
Exploring gapminder

AUTHOR
Your Name

```
library(tidyverse)
library(broom)
```

```
data <- read_csv(here::here("data/gapminder.csv"))
```

Introduction

let's look at the lifespan

```
hist(data$lifeExp)
```

Histogram of data$lifeExp

Let's fit a simple linear model of the effect of year on life expectancy

```
fit <- lm(lifeExp ~ year, data = data)
fit
```

```
Call:
lm(formula = lifeExp ~ year, data = data)

Coefficients:
(Intercept)         year
  -585.6522        0.3259
```

And let's look at the coefficient table:

```
library(broom)
fit_coef <- tidy(fit)
knitr::kable(fit_coef,
             caption = "A table of the coefficients")
```

term	estimate	std.error	statistic	p.value
(Intercept)	-585.6521874	32.3139645	-18.12381	0
year	0.3259038	0.0163237	19.96509	0

A table of the coefficients

The effect of year on life expectancy is 0.3259038.

The results are updated!

And we just pointed it to some different data. Then re-rendered it.

That's it.

That is why we use Quarto

4.1 Summary

In this section we've learned about:

- What the value is in a reproducible report
- What is Markdown
- How to combine software and writing
- How to use Markdown

4.2 Learning more

- Posit's Quarto cheatsheet

5

RStudio, What and Why

5.1 Overview

- **Teaching** 5 minutes
- **Exercises** 2 minutes

5.2 Questions

- What is RStudio?
- Why should I use RStudio?
- What features should I change?

5.3 Objectives

- Get familiarised with RStudio
- Get set up with not storing the RStudio workspace
- Download the course materials for the workshop

5.4 What is RStudio, and why should I use it?

> If R is the engine and bare bones of your car, then RStudio is like *the rest of the car*.

The engine is super critical part of your car. But in order to make things properly functional, you need to have a steering wheel, comfy seats, a radio, rear and side view mirrors, storage, and seatbelts. RStudio is all those niceties

The RStudio layout has the following features:

- On the upper left, the Quarto script
- On the lower left, the R console
- On the lower right, the view for files, plots, packages, help, and viewer.
- On the upper right, the environment / history pane
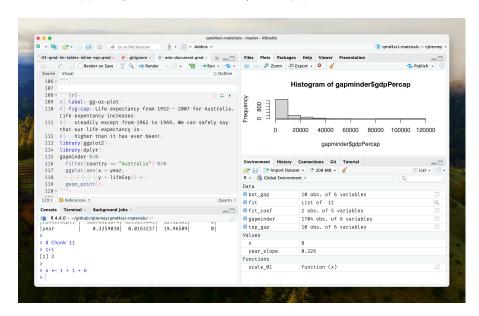
Figure 5.1: A screenshot of the RStudio working environment.

We saw a bit of what an Quarto script does.

- The R console is the bit where you can run your code.
- The file/plot/package viewer is a handy browser for your current files, like Finder, or File Explorer.
- Plots are where your plots appear, you can view packages, see the help files.
- The environment / history pane contains the list of things you have created, and the past commands that you have run.

> **ⓘ Your Turn: RStudio default options**
>
> To first get set up, I highly recommend changing the following setting
> Tools > Global Options (or `Cmd + ,` on macOS)
> Under the **General** tab:
> - For **workspace**:
>   - Uncheck restore .RData into workspace at startup.
>   - Save workspace to .RData on exit : "Never".

- For **History**:
  - Uncheck "Always save history (even when not saving .RData).
  - Uncheck "Remove duplicate entries in history".

Figure 5.2: Setting the options right for RStudio, so you don't restore previous sessions work, and don't save it either.

This means that you won't save the objects and other things that you create in your R session and reload them. This is important for two reasons

1. **Reproducibility**: you don't want to have objects from last week cluttering your session
2. **Privacy**: you don't want to save private data or other things to your session. You only want to read these in.

Your "history" is the commands that you have entered into R. Additionally, not saving your history means that you won't be relying on things that you typed in the last session, which is a good habit to get into!

5.5 Learning more

- [RStudio IDE cheatsheet](#)

6

Workflow

Before we start with Quarto, we need to make sure that you understand *file storage hygiene.*

We can prevent **unexpected problems** if we can maintain an order to your files, paths, and directories. A common problem that arises is R not knowing where a certain file is. For example, we get the error:

```r
read.csv("my-very-important-data-file-somewhere.csv")
```

```
Warning in file(file, "rt"): cannot open file
'my-very-important-data-file-somewhere.csv': No such file or directory
```

```
Error in file(file, "rt"): cannot open the connection
```

Because R doesn't know where `"my-very-important-data-file-somewhere.csv"` is.

Practicing *good file storage hygiene* will help maintain an order to files, paths, and directories. This will make you more productive in the future, because you'll spend less time fighting against file paths.

Not sure what a file path is? We explain that as well.

6.1   Overview

- **Teaching** 10 minutes
- **Exercises** 10 minutes

6.2   Questions

- Where should I put all my files?
- What is an RStudio project, anyway?
- What is a file path?

6.3 Objectives

- Understand what a file path is
- Set up an RStudio Project to organise your work
- Put some data in your project to set up the next tasks

> **i** Your Turn
>
> In groups of 2-4 discuss:
>
> 1. What your normal "workflow" is for starting a new project
> 2. Possible challenges that might arise when maintaining your project

6.4 When you start a new project: Open a new RStudio project

This section is heavily influenced by Jenny Bryan's great blog post on project based workflows.

Sometimes this is the first line of an R Script or R markdown file.

```
setwd("c:/really/long/file/path/to/this/directory")
```

> **Q** Question
>
> What do you think the `setwd` code does?

6.4.1 So what does this do?

This says, "set my working directory to this specific working directory".

It means that you can read in data and other things like this:

```
data <- read_csv("data/mydata.csv")
```

Instead of

```
data <- read_csv("c:/really/long/file/path/to/this/directory/data/mydata.csv")
```

So while this has the effect of **making the file paths work in your file**, it

is a problem. It is a problem because, among other things, using `setwd()` like this:

- Has 0% chance of working on someone else's machine (**this could include you in 6 months!**)
- Your file is not self-contained and portable. (Think: *"What if this folder moved to /Downloads, or onto another machine?"*)

So, to get this to work, you need to hand edit the file path to your machine.

This is painful.

When you do this all the time, it gets old, fast.

6.5   What is a file path?

This might all be a bit confusing if you don't know what a file path is. A file path is the machine-readable directions to where files on your computer live. So, the file path:

`/Users/njtierney/Desktop/qmd4sci-materials/demo.R`

Describes the location of the file "demo.R". This could be visualised as:

```
users
    njtierney
        Desktop
            qmd4sci-materials
                demo.R << THIS IS THE FILE HERE
                exercises
                exploratory-data-analysis
                    eda-document.qmd
                    eda-script.R
                 data
                    gapminder.csv
```

So, if you want to read in the `gapminder.csv` file, you might need to write code like this:

```
gapminder <- read_csv("/Users/njtierney/Desktop/qmd4sci-materials/data/gapminder.csv")
```

As we now know, this is a problem, because this is not portable code. It is unlikely someone else will have the `gapminder.csv` data stored under the folders, `"Users/njtierney/Desktop"`.

If you have an RStudio project file inside the **qmd4sci-materials** folder, you can instead write the following:

```
gapminder <- read_csv("data/gapminder.csv")
```

> **i** Your Turn
>
> - (1-2 minutes) Imagine you see the following directory path:
>   `"/Users/miles/etc1010/week1/data/health.csv"` what are the
>   folders above the file, `health.csv`?
> - What would be the result of using the following code in
>   `demo-gapminder.qmd`, and then using the code, and then moving this
>   to another location, say inside your C drive?
>
> ```
> setwd("Downloads/etc1010/week1/week1.qmd)
> ```

6.6   Is there an answer to the madness?

This file path situation is a real pain. Is there an answer to the madness?

The answer is **yes**!

I highly recommend when you start on a new idea, new research project, paper.
Anything that is new. It should start its life as an **rstudio project**.

An rstudio project helps keep related work together in the same place.
Amongst other things, they:

- Keep all your files together.
- Set the working directory to the project directory.
- Starts a new session of R.
- Restore previously edited files into the editor tabs.
- Restore other rstudio settings.
- Allow for multiple R projects open at the same time.

This helps keep you sane, because:

- Your projects are each independent.
- You can work on different projects at the same time.
- Objects and functions you create and run from project idea won't impact
  one another.

- You can refer to your data and other projects in a consistent way.

And finally, the big one:

RStudio projects help resolve file path problems, because they auto-
matically set the working directory to the location of the rstudio project.

Let's open one together.

> **i** Your Turn Use your own rstudio project
>
> 1. In RStudio, and run the following code to start a new rstudio project called "qmd4sci-materials".
>
> ```
> usethis::use_course("njtierney/qmd4sci-materials")
> ```
>
> 2. Follow the prompts to download this to your desktop and then run the rstudio project. (You can move it later if you like!)
> 3. You are now in an rstudio project!

> **i** Your Turn: open the `demo.R` file
>
> 1. Run the code inside the `demo.R` file
> 2. Why does the `read_csv` code work?
> 3. Run the code inside the `exploratory-data-analysis` folder - `eda-script.R`.
> 4. Does the `read_csv` code work?
> 5. Run the code inside the `exploratory-data-analysis` folder - `eda-document.qmd`, by clicking the "render" button (we'll go into this in more detail soon!)
> 6. Does it work?

6.7 The "here" package

Although RStudio projects help resolve file path problems, in some cases you might have many folders in your r project. To help navigate them appropriately, you can use the `here` package to provide the full path directory, in a compact way.

```
here::here("data")
```

returns

```
[1] "/Users/nick/github/njtierney/qmd4sci-materials/data"
```

And

```
here::here("data", "gapminder.csv")
```

returns

```
[1] "/Users/nick/github/njtierney/qmd4sci-materials/data/gapminder.csv"
```

(Note that these absolute file paths will indeed be different on my computer compared to yours - super neat!)

You can read the above `here` code as:

> In the folder `data`, there is a file called `gapminder.csv`, can you please give me the full path to that file?

This is really handy for a few reasons:

1. It makes things *completely* portable
2. Quarto documents have a special way of looking for files, this helps eliminate file path pain.
3. If you decide to not use RStudio projects, you have code that will work on *any machine*

6.8   Remember

> If the first line of your R script is

```
setwd("C:\Users\jenny\path\that\only\I\have")
```

> I will come into your office and SET YOUR COMPUTER ON FIRE .

– Jenny Bryan
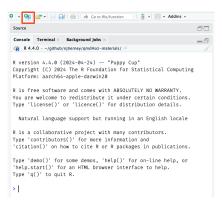
> 🔥 Aside: Creating an RStudio project
>
> You can create an Rstudio project by going to:
> file > new project > new directory > new project > name your project > create project.
> You can also click on the create project button in the top left corner
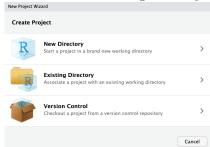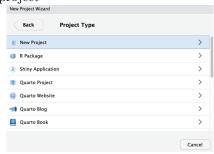
Then go to new directory, if it is a new folder - otherwise if you have an existing folder you have - click on existing directory.

Then go to new project
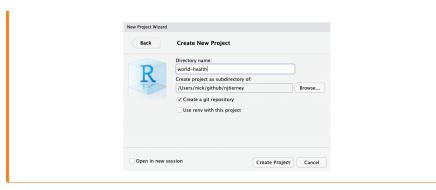
Then write the name of your project. I think it is usually worthwhile spending a bit of time thinking of a name for your project. Even if it is only a few minutes, it can make a difference. You want to think about:

- Keeping it short.
- No spaces.
- Combining words.

For example, I had a project looking at bat calls, so I called it `screech`, because bats make a screech-y noise. But maybe you're doing some global health analysis so you call it "world-health".

And click "create project".

7

Summary

In this lesson we've:

- Learnt what file paths are
- How to setup an rstudio project
- How to construct full file paths with the `here` package

8

Using Quarto

So far we have covered:

- How to **organise** your project (RStudio projects!)
- **Appropriately** refer to data (file storage hygiene!)
- A brief intro into **what** Quarto is

Now, let's talk about **using** Quarto.

8.1 Overview

- **Teaching** 10 minutes
- **Exercises** 10 minutes

8.2 Questions

- How should I start an Quarto document?
- What do I put in the YAML metadata?
- How do I create a code chunk?
- What sort of options to I need to worry about for my code?

8.3 Objectives

- Create a Quarto document, do some basic exploration

8.4 The anatomy of a Quarto document

This is a Quarto document (demo). It has three parts:

1. Metadata (YAML)
2. Text (markdown formatting)
3. Code (code formatting)

8.4.1 Metadata

The metadata of the document tells you how it is formed - what the title is, what date to put, and other control information. If you're familiar with LaTeX, this is kind of like how you specify the many options, what sort of document it is, what styles to use, and so on at the front matter.

Quarto documents use YAML (YAML Ain't Markup Language) to provide the metadata. It looks like this.

```
---
title: "An example document"
author: "Nicholas Tierney"
format: html
---
```

It starts and ends with three dashes `---`, and has fields like the following: `title`, `author`, and `format`.

`title` and `author` are special inputs which place the title and author information at the top of the document in large font. They are optional!

`format: html` tells us we want this to be a HTML formatted document - you'll see what this looks like in a moment!

8.4.2 Text

Is markdown, as we discussed in the earlier section,

It provides a simple way to mark up text

8.4.2.1 Markdown

```
- bullet list
- bullet list
- bullet list

1. numbered list
2. numbered list
```

```
3. numbered list

__bold__, **bold**, _italic_, *italic*

> quote of something profound

```r
# computer code goes in three back ticks
1 + 1
2 + 2
```
```

8.4.2.2 Rendered text

- bullet list
- bullet list
- bullet list

  1. numbered list
  2. numbered list
  3. numbered list

bold, **bold**, *italic*, *italic*

> quote of something profound

```
# computer code goes in three back ticks
1 + 1
```

```
[1] 2
```

```
2 + 2
```

```
[1] 4
```

8.4.3 Code

We refer to code in an Quarto document in two ways:

1. Code chunks, and
2. Inline code.

8.4.3.1 Code chunks

Code chunks are marked by three backticks and curly braces. We put the letter `r` inside them to denote them as "r" code chunks, but you can instead use "python" and "julia" instead:

8.5   R

```{r}
#| label: r-chunk-name
# a code chunk
```

8.6   python

```{python}
#| label: py-chunk-name
# a code chunk
```

8.7   julia

```{julia}
#| label: julia-chunk-name
# a code chunk
```

> **!** This book currently focusses only on R
>
> Quarto provides support for R, Python, Julia, and Observable, which
> are all very powerful and awesome languages! However currently we will
> only be focussing on using R in this book. But I want to make sure that
> you know you can use python, or Julia, or Observable! More languages
> will be supported into the future, I believe.

a backtick is a special character you might not have seen before, it is typically
located under the tilde key (~). On USA / Australia keyboards, is under the
escape key:
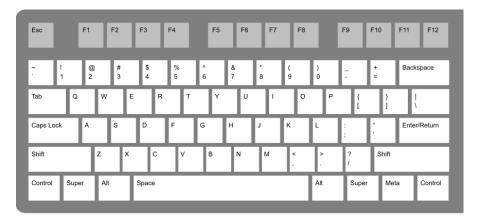
Figure 8.1: image from https://commons.wikimedia.org/wiki/File:ANSI_Keyboard_Layout_Diagram_with

8.7.1 Chunk names

Every chunk should ideally have a name. As I've mentioned earlier, naming things is hard, but follow these rules and you'll be fine:

- one word that describes the action (e.g., "read")
- one word that describes the thing inside the code (e.g, "gapminder")
- separate words with "-" or "_" (e.g., `read-gapminder`)

8.8 Code chunk options

You can control how the code is output by changing the code chunk options, which are written with a `#|`, called a "hash-pipe", since `#` is "hash", and `|` is "pipe", but might sometimes be called "bar" or "v-bar".

```{r}
#| label: read-gapminder
#| eval: false
gap <- read_csv("gapminder.csv")
```
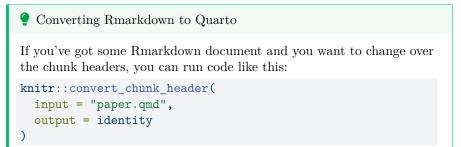
A nice feature of Quarto + Rstudio is that they provide code completion when you start writing the code chunk options, and they will provide options when hitting "tab".

In the past Rmarkdown required "TRUE" and "FALSE", but note that Quarto **always** uses `true` or `false` in lowercase, and **never** "yes" or "no".

The code chunks you need to know about right now are:

- `eval`: `true`/`false` Do you want to evaluate the code?
- `echo`: `true`/`false` Do you want to print the code?
- `cache`: `true` / `false` Do you want to save the output of the chunk so it doesn't have to run next time?
- `include`: Do you want to include code output in the final output document? Setting to `false` means nothing is put into the output document, but the code is still run.

You can read more about the options at the official documentation: https://quarto.org/docs/computations/execution-options.html

> 💡 Converting Rmarkdown to Quarto
>
> If you've got some Rmarkdown document and you want to change over the chunk headers, you can run code like this:
>
> ```
> knitr::convert_chunk_header(
>   input = "paper.qmd",
>   output = identity
> )
> ```

8.8.1 Inline code

Sometimes you want to run the code inside a sentence. When the code is run inside the sentence, it is called running the code "inline".

You might want to run the code inline to name the number of variables or rows in a dataset in a sentence like:

> There are XXX observations in the airquality dataset, and XXX variables.

You can call code "inline" like so:

```
```{r}
r_heights <- c(153, 151, 156, 160, 171)
r_mean <- mean(r_heights)
```

The mean of these heights is `{r} r_mean`
```

Which will produce the following sentence:

> The mean of these heights is 158.2

Essentially, instead of using **three backticks** to write multiple lines of code, you use **a single backtick**. You can think of this as a backtick being used inside text for a one liner, whereas creating a code fence with three backticks indicates something longer.

```
There are `{r} nrow(airquality)` observations in the airquality dataset,
and `{r} ncol(airquality)` variables.
```

Which gives you the following sentence

There are 153 observations in the airquality dataset, and 6 variables.

What's great about this is that if your data changes upstream, then you don't
need to work out where you mentioned your data and change that bit of text.
You just render the document and it takes care of these details.

8.9 Creating a Quarto document

- Rstudio menu system
- Explore the template provided by Rstudio
- Compile an Quarto document

8.10 Working with a Quarto document

Demo: Create a Quarto document in rstudio.

> **i** Your Turn
>
> 1. Use the rstudio project you previously created,
>   `qmd4sci-materials`, and open the "01-qmd-examples.qmd"
>   file.
> 2. Run some brief summaries of the data in the Quarto docu-
>   ment:
>   - hist(data$)
>   - How big is the data?
>   - How many countries are there?
>   - What was the lowest life expectancy in Australia's His-
>   tory?
>   - How about the lowest GDP for Australia?
>   - Where does Australia rank in GDP in 1997?

8.11   Nick's Quarto starter pack

I highly recommend that each document you write has a certain structure:

- Sets global options in the YAML
- First code chunk manages libraries
- Second code chunk manages functions

For example

```
---
title: example
format:
  html:
    fig-align: center
    fig-width: 4
    fig-height: 4
    fig-format: png
execute:
  echo: false
  cache: true
---


```{r}
#| label: library
library(tidyverse)
```


```{r}
#| label: functions
# A function to scale input to 0-1
scale_01 <- function(x){
  (x - min(x, na.rm = TRUE)) / diff(range(x, na.rm = TRUE))
}
```


```{r}
#| label: read-data
gapminder <- read_csv(here::here("data", "gapminder.csv"))
```
```

In the YAML chunk under `execute`, you set the options that you want to define globally. In this case, I've told Quarto:

- `fig-align: center` Align my figures in the center

- `fig-width: 4` & `fig-height: 4`. Set the width and height to 4 inches.
- `fig-format: png`. Save the images as PNG
- `cache: true`. Save the output results of all my chunks so that they don't need to be run again.
- `echo: false`: I don't want any code printed by setting `echo: false`.

In the `library` chunk, you put all the library calls. This helps make it clearer for anyone else who might read your work what is needed to run this document. I often go through the process of moving these `library` calls to the top of the document when I have a moment, or when I'm done writing. You can also look at Miles McBain's `packup` package to help move these library calls to the top of a document.

In the `functions` chunk, you put any functions that you write in the process of writing your document. Similar to the `library` chunk, I write these functions as I go, as needed, and them move these to the top when I get a moment, or once I'm done. The benefit of this is that all your functions are in one spot, and you might be able to identify ways to make them work better together, or improve them separately. You might even want to move these into a new R package, and putting them here makes that easier to see what you are doing.

In the `readr` chunk, you read in any data you are going to be using in the document.

Now, this is my personal preference, and there are definitely other ways to organise things! But, I find the following benefits:

1. The "top part" of your document contains all the metadata / setup info. Your global options. You don't need to specify every single code chunk.
2. It helps another person get oriented with your work - they know the settings, the functions used, and the special things that you wrote (your functions)
3. Remember, "another person" includes yourself in 6 months. You are always collaborating with your future self. **You are always collaborating with your future self.** Say it with me.

---

i Your Turn

1. Update the "01-qmd-example.qmd" Quarto document you just created, based on the aforementioned steps discussed above.

9

HTML, PDF, and Word (and more!)

One of the great things about Quarto is that we can convert it to many different output types. The top three that you might be most likely to use are HTML, PDF, and Microsoft Word. There are other formats! But we can discuss later.

In this section, we are going to briefly discuss how to render to these output formats, and some things that you might want to do for each of them.

9.1 Overview

- **Teaching**: 10 minutes
- **Exercises**: 15 minutes

9.2 Questions

- How do I convert to HTML, PDF, or Word?
- How do I set options specific to each of these?
- How can I include a screenshot of an interactive graphic in PDF or Word?
- How to make a HTML page "self contained"?

9.3 Objectives

9.4 How do I convert to HTML, PDF, or Word?

Here are three ways to do this:

1. You can control this in the "render" button

You might notice that depending on the option you select, this changes things in the YAML - which is another way to control which output you have:

2. You can change the YAML option

```
title: "Exploring gapminder"
format: html
```

```
title: "Exploring gapminder"
format: pdf
```

```
title: "Exploring gapminder"
format: docx
```

3. You can call the `quarto render` function - from the terminal if you wish

```
quarto render example.qmd --to html
quarto render example.qmd --to docx
```

9.4.1 A note on workflow with Quarto: HTML first, PDF/word later

It can be easy to get caught up with how your document looks. I highly recommend avoiding compiling to PDF or word until *you really need to*. This is also recommended by the author of rmarkdown and knitr, Yihui Xie. Because HTML doesn't have page breaks, this means that you can spend time working on generating content, and not trying to get figures to line up correctly.

This was a minor revelation to me, to understand that page breaks were the cause of so much angst and pain. Embrace HTML, I say!

9.5 How to make a HTML page "self contained"

PDFs are great because they are a very Portable Document Format (PDF!). But HTML can be really awesome as well because:

- no page breaks
- awesome online interactive data vis
- did I mention no page breaks?

But in order to share a HTML document you'll need to also typically share a folder of other figures/files etc.

You can have everything crammed into a *single HTML page* by setting the option `embed-resources` to `true`:

```
title: HTML is great
format:
  html:
    embed-resources: true
```

> **i** Your Turn
>
> 1. Using your "01-qmd-examples.qmd" Quarto document you were using earlier, generate three reports, one as HTML, one as PDF, and one as microsoft word. Remember, if you are having PDF problems, see the installation chapter note on installing LaTeX with the R package, `tinytex`.

10

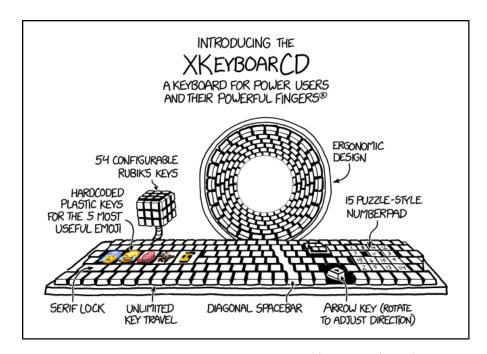## *Keyboard Shortcuts*

Figure 10.1: Imaged sources from https://xkcd.com/2150/

Keyboard shortcuts tend to make our lives easier. Some that you might already be familiar with in day to day life are: quickly saving (Cmd + S or Ctrl + S), or Undo (Cmd + Z or Ctrl + Z).

There are many keyboard shortcuts you can access in R, this section provides a brief tour of them, and why you might want to use them.

10.1 Overview

- **Teaching** 5 minutes
- **Exercises** 5 minutes

10.2 Questions

- What sort of keyboard shortcuts should I care about?

10.3 Objectives

10.4 Table of Common Shortcuts

Below is a small table of tasks you can perform with keyboard

Action	Windows/Linux	Mac
Render document	Ctrl + Shift + K	Cmd + Shift + K
Insert Chunk	Ctrl + Alt + I	Cmd + Option + I
Run Current Chunk	Ctrl + Alt + C	Cmd + Option + C
Jump to	Cmd+Shift+Option+J	
Shift+Alt+J		
Show Keyboard Shortcut Reference	Alt+Shift+K	Option+Shift+K
Create multiple cursors	Ctrl + Alt + Up/Down	option + control + Up/Down
Delete the current line	Ctrl + D	Cmd + D
Un/Comment out a line	Ctrl + Shift + C	Cmd + Shift + C
Reformat Section	Ctrl + Shift + A	Cmd + Shift + A

10.5   Further Reading

- The Rstudio Cheat Sheet has an index of shortcuts.
- This help file has a guide to customising keyboard shortcuts.
- The shrtcts R package for creating new shortcuts.

---

i Your Turn

- In your **awesome** quarto document
  - Practice using the Keyboard Shortcut Reference and find the keyboard shortcut for inserting a pipe character (`%>%` or `|>`)
  - Spend 3 minutes practicing these commands.
- Ask Nick to demonstrate some of his favourite shortcut combos
  - e.g., multiline cursor, yank lines, copy lines, go to file/function

11

Figures, Tables, Captions.

You need figures and tables in your own writing, whether it be a journal paper, an internal document, or some documentation. In this section, we discuss how to add figures and tables into your Quarto document, and how to provide captions for them.

11.1 Overview

- **Teaching** 10 minutes
- **Exercises** 10 minutes

11.2 Questions

- How do I create a figure in Quarto?
- How do I create a table in Quarto?
- How do I add captions for figures and tables?

11.3 Objectives

11.4 Tables

To produce a table, I recommend you use the `kable` function from the `knitr` package.

> 🔥 Other table R packages
>
> There are many other table making pieces of R packages, such as `gt`, `formattable`, `reactable`, and `flextable`). But I think you can get 90% of the way there with `kable` from `knitr`, and for the

11.4.1 Demonstrating using tables

`kable` takes a `data.frame` as input, and outputs the table into a `markdown table`, which will get rendered into the appropriate output format.

For example, let's say we wanted to share the first 6 rows of our gapminder data.

This gives us the following output

```
top_gap <- head(gapminder)

knitr::kable(top_gap)
```

country	continent	year	lifeExp	pop	gdpPercap
Afghanistan	Asia	1952	28.801	8425333	779.4453
Afghanistan	Asia	1957	30.332	9240934	820.8530
Afghanistan	Asia	1962	31.997	10267083	853.1007
Afghanistan	Asia	1967	34.020	11537966	836.1971
Afghanistan	Asia	1972	36.088	13079460	739.9811
Afghanistan	Asia	1977	38.438	14880372	786.1134

So how does that work? `kable` prints out the following:

```
|country     |continent | year| lifeExp|       pop| gdpPercap|
|:-----------|:---------|----:|-------:|--------:|---------:|
|Afghanistan |Asia      | 1952|  28.801|  8425333|  779.4453|
|Afghanistan |Asia      | 1957|  30.332|  9240934|  820.8530|
|Afghanistan |Asia      | 1962|  31.997| 10267083|  853.1007|
|Afghanistan |Asia      | 1967|  34.020| 11537966|  836.1971|
|Afghanistan |Asia      | 1972|  36.088| 13079460|  739.9811|
|Afghanistan |Asia      | 1977|  38.438| 14880372|  786.1134|
```

And this then gets *rendered* as a table. This works for HTML, PDF, and word!

11.4.1.1 Adding captions to a table

Now, say that we wanted to include a caption? We use the `caption` argument. This will also automatically number the table (woo! We'll cover this later).

```
knitr::kable(top_gap,
             caption = "The first 6 rows of the dataset, gapminder")
```

Table 11.2: The first 6 rows of the dataset, gapminder

country	continent	year	lifeExp	pop	gdpPercap
Afghanistan	Asia	1952	28.801	8425333	779.4453
Afghanistan	Asia	1957	30.332	9240934	820.8530
Afghanistan	Asia	1962	31.997	10267083	853.1007
Afghanistan	Asia	1967	34.020	11537966	836.1971
Afghanistan	Asia	1972	36.088	13079460	739.9811
Afghanistan	Asia	1977	38.438	14880372	786.1134

Some other useful features of `kable` include setting the rounding number, with the `digits` option.

For example, we could present the first 2 digits of each number like so:

```
knitr::kable(top_gap,
             caption = "The first 6 rows of the dataset, gapminder",
             digits = 2)
```

Table 11.3: The first 6 rows of the dataset, gapminder

country	continent	year	lifeExp	pop	gdpPercap
Afghanistan	Asia	1952	28.80	8425333	779.45
Afghanistan	Asia	1957	30.33	9240934	820.85
Afghanistan	Asia	1962	32.00	10267083	853.10
Afghanistan	Asia	1967	34.02	11537966	836.20
Afghanistan	Asia	1972	36.09	13079460	739.98
Afghanistan	Asia	1977	38.44	14880372	786.11

There are other options that you can set in `kable`, but for these options will get you through a large majority of what you need. For more information on what `kable` can provide, see `?knitr::kable`.

There are many different ways to produce tables in R. We have chosen to show `kable` today because kable is minimal, but powerful. If you want to extend `kable` to do more, look at kableExtra. For PDF/LaTeX output, I found the option `kableExtra::kable_styling(latex_options = c("hold_position"))` particularly nice to just **put the table where it should be, goshdarnit**.

> **i** Your Turn
>
> 1. Using the "02-qmd-figures-chunks.qmd"
> 2. Create a summary of your gapminder data, put it into a table.
> 3. Add a caption to this table.
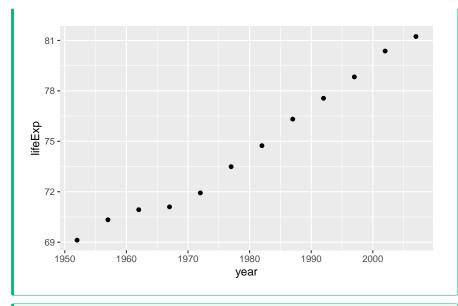> 4. Set the number of decimals to 2.

11.5   Figures

Printing figures is probably my favourite feature of Quarto. It is actually relatively straightforward in the case of plots. You provide the plot you want to show in a code chunk!

> **💡** Demo using gapminder
>
> For example, I can print a plot of the gapminder data for Australia like so:
>
> ```r
> options(tidyverse.quiet = TRUE)
> library(tidyverse)
>
> gapminder |>
>   filter(country == "Australia") |>
>   ggplot(aes(x = year,
>              y = lifeExp)) +
>   geom_point()
> ```

💡 Demo: Captions for figures

Inserting a caption for a figure is a little bit different. The caption argument is controlled in the chunk option, under the option, `fig-cap`.
So to insert a figure, we do the following.

````
```{r}
#| label: gg-oz-gapminder
#| fig-cap: "Life expectancy from 1952 - 2007 for Australia. Life expentancy increases s
library(ggplot2)
library(dplyr)

gapminder |>
  filter(country == "Australia") |>
  ggplot(aes(x = lifeExp,
             y = year)) +
  geom_point()
```
````

Which would produce the following output

```
library(ggplot2)
library(dplyr)

gapminder |>
  filter(country == "Australia") |>
  ggplot(aes(x = lifeExp,
             y = year)) +
  geom_point()
```
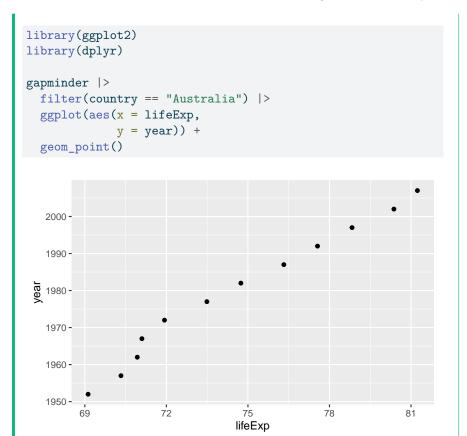

Figure 11.1: Life expectancy from 1952 - 2007 for Australia. Life expentancy increases steadily except from 1962 to 1969. We can safely say that our life expectancy is higher than it has ever been!

i Your Turn

1. Using 02-qmd-figures-chunks.qmd
   1. Create a plot
   2. Add a figure caption

11.5.1 Adding multiple (sub) figures and (sub) captions

Sometimes you want to add multiple figures that are linked, or slightly different views of similar data and then reference them as Figure 1A and Figure 1B. You can do this with `layout-ncol` and `fig-cap`, and reference the figures with `@fig-<chunk-name>-1` `@fig-<chunk-name>-2`. For example:

```{r}
#| label: fig-volcanos
#| layout-ncol: 2
#| fig-cap:
#|   - "An image plot of Auckland's Maunga Whau Volcano"
#|   - "A contour plot of Auckland's Maunga Whau Volcano"

image(volcano)
contour(volcano)
```
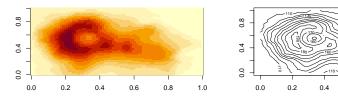

Figure 11.2: An image plot of Auck-Figure 11.3: A contour plot of Auckland's Maunga Whau Volcano land's Maunga Whau Volcano

We can see the image plot as (`@fig-volcanos-1`) Figure 11.2 and the contour plot as `@fig-volcanos-2` Figure 11.3.

For more information on this see https://quarto.org/docs/authoring/figures.html#layout and https://quarto.org/docs/authoring/cross-references.html

11.6 Inserting images

We cannot always generate the graphics that we want - for example, we might have an image of something that we want to show, or perhaps a nice flowchart someone else made.

In our case, say we wanted to insert the Statistical Society of Australia logo into our document, there are two ways we can do this.

1. With `markdown` syntax
2. with `knitr::include_graphics()`

Markdown syntax

The   markdown   syntax   to   insert   an   image   is:
`![caption](path/to/image){options}`

> 💡 Demo: inserting a logo
>
> So we could insert the new SSA vic logo by doing the following:
> ```
> ![The new, gorgeous SSA Logo has a hidden element, can you see it?](https://qmd4sci.njti
> ```
> Which would give us the following output:
>
>
>
> Figure 11.4: The new, gorgeous SSA Logo has a hidden element, can you see it?
>
> But say that we want more control over the output, like we want to center the image, and we want to make it smaller? Then you can use `knitr::include_graphics()`, and control the figure alignment using the options fig-align, and add a caption with `fig-cap`.

````
```{r}
#| label: ssa-logo
#| fig-align: center
#| fig-cap: "The new SSA logo, which is actually a scatterplot, which is super neat!"
knitr::include_graphics(here::here("figs", "ssa-logo.png"))
```
````

```r
knitr::include_graphics(here::here("figs", "ssa-logo.png"))
```


Figure 11.5: The new SSA logo, which is actually a scatterplot, which is super neat!

> 🔥 Controlling image output with css
>
> You can control more features of figures, e.g., sizing, alignment, alt text, etc., by using CSS type styling, which you can read more about here: https://quarto.org/docs/authoring/figures.html#figure-sizing%60

> ℹ Your Turn
>
> 1. Download the gapminder logo
> 2. Put it into a new directory called "figs" (at the top level)
> 3. Insert the image into your "02-qmd-figures-chunks.qmd" Quarto document around where you introduce gapminder.
> 4. Try using both markdown syntax, and `knitr::include_graphics()`.
> 5. Hint: Remember to refer to the image in the right spot using `here`!

11.7 Summary

We've now learned how to insert tables, plots, and images into our documents!

12

Customising your figures

When you produce figures, you usually want to tweak them a little bit. A bit wider, perhaps a bit taller. Perhaps a different image type other than "png", because the journal requires "svg" or "jpg". Maybe you need 600dpi because you're going to print it really big.

We can control these features with code chunk options.

In this section, we are going to talk more specifically about how to customise your figures.

12.1   Overview

- **Teaching** 10 minutes
- **Exercises** 10 minutes

12.2   Questions

- How do I change the height and width of a figure?
- How to I change the type of output of a figure? (e.g., PDF, PNG, JPG, SVG)
- Can I set all the figure features globally?
- How do I save the figures?

12.3   Objectives

- Learn how to set individual figure height, width, aspect, and print size
- Learn how to set global parameters for your chunks
- Get a copy of all of your figures

12.4 Which chunk options should you care about for this?

There are many chunk options that control your output, but only a few that you really need to worry about for your figures:

- `fig-align`: How do you want your figure aligned? Takes one of the following inputs: "default", "center", "left", or "right"? (**demo**)
- `fig-cap`: Would you like a caption for your figure? It takes a character vector as input: "My Amazing Graph"
- `fig-height` & `fig-width`: How tall and wide would you like your figure in inches? Each takes one number (e.g., 7, or 9) [Note: these numbers are not quoted]

For demonstration purposes, let's take a plot from earlier and show how it's output can change.

- with `fig-height`, `fig-width`, `fig-format`:

> **i** Your Turn
>
> 1. Open exercise `exercises/02-qmd-figures-chunks/02-qmd-figures-chunks.qmd`
>   Create three figures, with the respective dimensions
>   (`fig-height` and `fig-width`)
>   - 2x2
>   - 10x10
>   - 4x7
> 2. Now add to those figures, the following:
>   - `fig-align = "center"`
> 3. Now change the output type to be "svg"

12.5 Setting global options

If we repeat adding the same chunk options for each figure, we might want to consider setting them globally. We can do this by changing the options in the YAML:

```
---
title: "02-qmd-figures-chunks"
author: "Your Name"
date: 2024/06/23
format:
```

```
  html:
    fig-height: 7
    fig-width: 7
    fig-format: png
    fig-dpi: 300
---
```

> **ℹ Your Turn**
>
> 1. Set the global options in the "02-qmd-figures-chunks.qmd"
>   document to set:
>
> - `fig-height`
> - `fig-width`
> - `fig-format`

> **♥ Demo: Keeping your markdown**
>
> You can set the options for your figures, which will change how they
> appear on the page, but this won't save the figures anywhere. In order
> to save the figures to file, you need to edit the YAML option `keep-md:`
> `true`:
>
> ```
> ---
> title: "Awesome report"
> author: "You"
> format:
>   html:
>     keep-md: true
> ---
> ```

12.6 Altering where figures are saved to

By default, the figures are saved in a folder named after the file, e.g.,

`02-qmd-figures-chunks_files/figure-html`

If you want to change this location, you can control the specific name of the
folder by setting `fig.path` like so in the YAML

```
---
title: "Awesome report"
author: "You"
```

```
format:
  html:
    keep-md: true
knitr:
  opts_chunk:
    fig.path: folder/for/figures/prefix-
---
```

If you do **not** want a prefix specified, you must end this part with a slash, e.g.,

```
---
title: "Awesome report"
author: "You"
format:
  html:
    keep-md: true
knitr:
  opts_chunk:
    fig.path: figures/
---
```

(reference: https://github.com/quarto-dev/quarto-cli/discussions/4254)

> **i** Your Turn
>
> 1. Save your images to a specific directory of your choice

12.7 Further Reading

- Official Quarto documentation on figures

> **⚫** Managing EPS/TIFF/Other multiple image formats
>
> Unfortunately (currently, as far as I can tell) in Quarto it seems you cannot save to other image formats such as "eps", "tiff", and cannot save to multiple formats at the same time.
> If you would like to convert images to a specific format, you could try using code like the following.

```r
library(magick)
library(fs)

# List existing file paths matching "png" extension
figures_ls <- dir_ls(
  path = "exercises/02-qmd-figures-chunks/02-qmd-figures-chunks_files/",
  recurse = TRUE,
  glob = "*.png"
  )

# read images in
library(purrr)
figures <- map(
  figures_ls,
  \(x) image_read(path = x)
)

# create new paths with .TIFF extension
# substitute out for another image format like "bmp", "
new_paths <- xfun::with_ext(figures_ls, "tiff")

# write new images
walk2(
  .x = figures,
  .y = new_paths,
  \(x, y) image_write(image = x, path = y)
)
```

13

Math
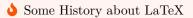
Want to include equations in your writing? Easy. Quarto supports LaTeX style equation writing. This section introduces the two types equations, inline, and display form, as well as numbering equations.

13.1 Overview

- **Teaching**: 10 minutes
- **Exercises**: 10 minutes

13.2 Questions

- How to I create an equation?
- LaTeX is funky, what are the basic math commands?

> 🔥 Some History about LaTeX
>
> Equation editing was first made available in TeX, which later become LaTeX, named after Leslie Lamport.

13.3 Anatomy of Equations

This section shows you some basic equations types that you want to be familiar with.

Inline equations are referenced by a pair of dollar signs: $.

So this text would have an equation here: $E = mc^2$

Generates:

So this text would have an equation here: $E = mc^2$

Display equations are referenced by two pairs of dollar signs:

```
$$
E = mc^2
$$
```

Gives:

$$E = mc^2$$

13.3.0.1   Viewing equations

Understanding whether or not you have created the right equation can be difficult. Rstudio provides previews of your equations in text (**demo**).

13.4   Example math commands

LaTeX is an amazing language, but understanding how to create the equations can be (more than) a bit confusing at times. This section demonstrates some example equations that you might be familiar with.

13.4.0.1   Fractions

```
$$
\frac{1}{2}
$$
```

$$\frac{1}{2}$$

13.4.0.2   Sub and Super Scripts

```
$$
Y = X_1 + X_2
$$
```

$$Y = X_1 + X_2$$

```
$$
a^2 + b^2 = c^2
$$
```

$$a^2 + b^2 = c^2$$

13.4.0.3 Square roots

```
$$
\sqrt{p}
$$
```

$$\sqrt{p}$$

```
$$
x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
$$
```

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

13.4.0.4 Summations

```
$$
\sum_{i = 1}^{n}{(\bar{x} - x_i)^2}
$$
```

$$\sum_{i=1}^{n} (\bar{x} - x_i)^2$$

13.4.0.5 Bayes Rule

```
$$
Pr(\theta | y) = \frac{Pr(y | \theta) Pr(\theta)}{Pr(y)}
$$
```

$$Pr(\theta|y) = \frac{Pr(y|\theta)Pr(\theta)}{Pr(y)}$$

```
$$
Pr(\theta | y) \propto Pr(y | \theta) Pr(\theta)
$$
```

$$Pr(\theta|y) \propto Pr(y|\theta)Pr(\theta)$$

13.4.0.6   Linear Model

```
$$
Y \sim X\beta_0 + X\beta_1 + \epsilon
$$
```

$$Y \sim X\beta_0 + X\beta_1 + \epsilon$$

```
$$
\epsilon \sim N(0,\sigma^2)
$$
```

$$\epsilon \sim N(0, \sigma^2)$$

> **i** Your Turn
>
> 1. Add some math to your "02-qmd-figures-chunks.qmd" document

13.5   Further Reading:

https://quarto.org/docs/visual-editor/technical.html#equations   https://oeis.org/wiki/List_of_LaTeX_mathematical_symbols

14

Citing Figures, Tables & Sections

When you're writing a report, you often refer to a table or figure in text.

Australia's life expectancy has increased a great deal over the past 50 years (Figure 1)
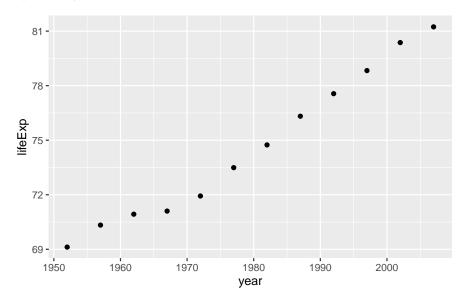
Figure 1. Life expectancy from 1952 - 2007 for Australia. Life expectancy increases steadily except from 1962 to 1969. We can safely say that our life expectancy is higher than it has ever been!

And sure, this is figure 1...for now.

But what happens if actually, that figure should be moved later in the paper?

You need to do the following:

1. Update the reference to figure 1 in the text.
2. Update the figure 1 caption to not say figure 1.

This is fine.

Once.

But it is **never** once. After this, it is frustrating, and error prone.

There is a way to solve this, using figure citations, which this lesson discusses.

14.1   Overview

- **Teaching** 10 minutes
- **Exercises** 15 minutes

14.2   Questions

- How do I refer to the table or figure in text and link to it?

14.3   Objectives

- Link to tables or figures in text.
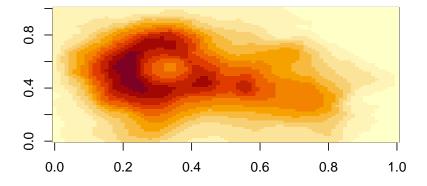
14.4   How to refer to tables and figures in text? (demo)

- Tables are referenced in text with `@tbl-label`
- Figures are references in text with `@fig-label`

Importantly here, for these two above examples to work, the things that they are referring to must have the exact label `tbl-label` and `fig-label`, respectively. That is, they must have the `tbl` or the `fig` part in there!

So, in order to use this referencing style, you must use specific labelling of your code chunks. For example, if you have some code like this:

```{r}
#| label: example-figure
image(volcano)
```

Then you cannot reference this figure in text - `@example-figure` does not work: **?**. We get the error: "example-figure?".

It would need to have a label like:

```{r}
#| label: fig-example
image(volcano)
```
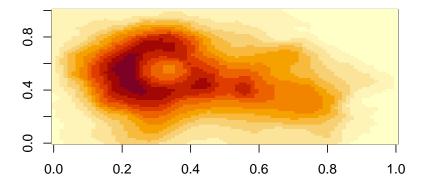
Figure 14.1

Then you can reference it with: `@fig-example` - Figure 14.1.

> 💡 Hover your mouse on a reference!
>
> Note the awesome on-hover behaviour for hovering over a reference!

Also note that there are a variety of ways to specify the figure:

- `@fig-example`: Figure 14.1
- `@Fig-example`: Figure 14.1
- `[Fig @fig-example]`: Fig 14.1
- `-@fig-example`: Figure 14.1

(reference here: https://quarto.org/docs/authoring/cross-references.html#references)

> ℹ️ Your Turn
>
> 1. Using "03-qmd-lm-tables-inline-qmd"
>   1. Convert your output to use `fig` prefixes to reference figures

💡 Demo

````
```{r}
#| label: fig-gg-oz
#| fig-cap: Life expectancy from 1952 - 2007 for Australia. Life expectancy increases
#|   steadily except from 1962 to 1969. We can safely say our life expectancy is
#|   higher than it has ever been!
library(ggplot2)
library(dplyr)
gapminder %>%
  filter(country == "Australia") %>%
  ggplot(aes(x = year,
             y = lifeExp)) +
  geom_point()
```
````


Figure 14.2: Life expectancy from 1952 - 2007 for Australia. Life expectancy increases steadily except from 1962 to 1969. We can safely say our life expectancy is higher than it has ever been!

Australia's life expectancy has increased a great deal over the past 50 years (See Figure 14.2).

> **i** Your Turn
>
> 1. Using "03-qmd-lm-tables-inline-qmd":
>   1. Add a new plot in your document and reference it

14.5   Referencing a table

To cite a table, you write the following:

`@tbl-chunk-name`

```{r}
#| label: tbl-gg-oz-tab
gapminder %>%
  filter(country == "Australia") %>%
  knitr::kable(caption = "Raw gapminder data for Australia.")
```

Table 14.1: Raw gapminder data for Australia.

country	continent	year	lifeExp	pop	gdpPercap
Australia	Oceania	1952	69.120	8691212	10039.60
Australia	Oceania	1957	70.330	9712569	10949.65
Australia	Oceania	1962	70.930	10794968	12217.23
Australia	Oceania	1967	71.100	11872264	14526.12
Australia	Oceania	1972	71.930	13177000	16788.63
Australia	Oceania	1977	73.490	14074100	18334.20
Australia	Oceania	1982	74.740	15184200	19477.01
Australia	Oceania	1987	76.320	16257249	21888.89
Australia	Oceania	1992	77.560	17481977	23424.77
Australia	Oceania	1997	78.830	18565243	26997.94
Australia	Oceania	2002	80.370	19546792	30687.75
Australia	Oceania	2007	81.235	20434176	34435.37

See above in Table **?@tbl-tbl-gg-oz-tab**.

If you want to move the caption location you can use `tbl-cap-location:`
`<position>` to specify the location. By default it is the top.

```{r}
#| label: tbl-gg-oz-tab-bottom
#| tbl-cap-location: bottom
```

```
gapminder %>%
  filter(country == "Australia") %>%
  knitr::kable(caption = "Raw gapminder data for Australia.")
```

country	continent	year	lifeExp	pop	gdpPercap
Australia	Oceania	1952	69.120	8691212	10039.60
Australia	Oceania	1957	70.330	9712569	10949.65
Australia	Oceania	1962	70.930	10794968	12217.23
Australia	Oceania	1967	71.100	11872264	14526.12
Australia	Oceania	1972	71.930	13177000	16788.63
Australia	Oceania	1977	73.490	14074100	18334.20
Australia	Oceania	1982	74.740	15184200	19477.01
Australia	Oceania	1987	76.320	16257249	21888.89
Australia	Oceania	1992	77.560	17481977	23424.77
Australia	Oceania	1997	78.830	18565243	26997.94
Australia	Oceania	2002	80.370	19546792	30687.75
Australia	Oceania	2007	81.235	20434176	34435.37

Table 14.2: Raw gapminder data for Australia.

Reference: https://quarto.org/docs/authoring/cross-references.html#tables
and https://quarto.org/docs/authoring/tables.html#caption-location

14.6   Other things you can cross/reference

You can also reference theorems, code, proofs, and equations! See https://quarto.org/docs/authoring/cross-references.html#theorems-and-proofs for more details.

> **i** Your Turn
>
> 1. Using "03-qmd-lm-tables-inline-qmd"
>   1. Create a table in your document and refer to it in text

14.7　Referencing a section

You can even reference a section in your report: `@sec-slug`

However, in order to write this, you need to include `sec-slug` in your markdown header, like so:

`## your amazing header {#sec-slug}`

You must also include `number-section` in your YAML:

```
---
title: "example"
number-sections: true
---
```

For example, I can refer to the first section (Section Chapter 14) in this document by referring to the section as

`(Section @sec-start)`

because it was written as:

`# Citing Figures, Tables & Sections {#sec-start}`

> 💡 Demo: Using visual mode
>
> Visual mode is this pretty neat feature, it's best demonstrated live!

> ℹ️ Your Turn
>
> 1. Using "03-qmd-lm-tables-inline-qmd"
>   1. Reference a section in the report.
> 2. Open up "04-qmd-global-opts-device.qmd" and work through the tasks there

15

Citing Articles & Bibliography Styles

Now that you are near the end of your data analysis, you want to make sure that you've ~~plugged in the gaps of REF1 REF2 and so on~~ correctly cited the articles and software you wanted to mention.

15.1 Overview

- **Teaching**
- **Exercises**

15.2 Questions

- What sort of things can I cite?
- How do I manage my `.bib` file?
- How do I change the citation style?

15.3 Objectives

- Provide a bibliography at the end of the document
- Cite articles and packages during the document
- learn how to manage citation styles

15.4 How to cite things

Citing things in a Quarto document is straightforward, you refer to articles you want to cite using `[@article-handle]`. Here, `article-handle` matches the article handle in your `.bib` file.

This `.bib` file is referred to in the YAML of your document, under the option `bibliography: filename.bib`:

```
---
title:
author:
output: html_document
bibliography: references.bib
---
```

15.4.1 What is a .bib file?

Good question.

`.bib` is a format for storing references from the heyday of LaTeX. It contains plain text with reference information for the article. Here's an example one:

```
@Book{ggplot2,
  author = {Hadley Wickham},
  title = {ggplot2: Elegant Graphics for Data Analysis},
  publisher = {Springer-Verlag New York},
  year = {2016},
  isbn = {978-3-319-24277-4},
  url = {http://ggplot2.org},
}
```

15.4.2 And how do I generate these .bib files?

You can use the `citation` function in R for R itself, and for specific R packages.

We can get the citation for R with:

```
citation()
```

```
To cite R in publications use:

  R Core Team (2025). _R: A Language and Environment for Statistical
  Computing_. R Foundation for Statistical Computing, Vienna, Austria.
  <https://www.R-project.org/>.
```

```
A BibTeX entry for LaTeX users is

  @Manual{,
    title = {R: A Language and Environment for Statistical Computing},
    author = {{R Core Team}},
    organization = {R Foundation for Statistical Computing},
    address = {Vienna, Austria},
    year = {2025},
    url = {https://www.R-project.org/},
  }


We have invested a lot of time and effort in creating R, please cite it
when using it for data analysis. See also 'citation("pkgname")' for
citing R packages.
```

And for ggplot2 with

```
citation("ggplot2")
```

```
To cite ggplot2 in publications, please use

  H. Wickham. ggplot2: Elegant Graphics for Data Analysis.
  Springer-Verlag New York, 2016.

A BibTeX entry for LaTeX users is

  @Book{,
    author = {Hadley Wickham},
    title = {ggplot2: Elegant Graphics for Data Analysis},
    publisher = {Springer-Verlag New York},
    year = {2016},
    isbn = {978-3-319-24277-4},
    url = {https://ggplot2.tidyverse.org},
  }
```

For journals or books, you'll need to get a specific .bib file. Yes, this can be a bit of a pain, but this is where you need to use a reference management software like Zotero, Mendeley, papers, or paperpile. The important thing to to **use something**. These all allow you to get .bib files of your articles, which you can then placec in your `references.bib` file.

> **i** Your Turn
>
> 1. Using "05-qmd-bib-polish.qmd"
>   1. Generate a references.bib file to place your citations

2. Using the `citation()` function, generate citations for the packages we have used, "dplyr", "ggplot2", "gapminder", and for the R software, place these in your `references.bib` file
3. Reference these in your document
4. Add a final heading in your file called `#bibliography`
5. Cite the packages you use when you use them
6. hint: https://quarto.org/docs/authoring/footnotes-and-citations.html
7. hint: look at the names in packages.bib

2. Tidy up the structure, provide some more words around the introduction and results
3. Render the document

15.5 How to change the bibliography style

OK so now you've got your bibliography, but you now need to change it to *a specific journal format.* Luckily, this is now pretty easy. You can change your citation style from the citation style language

Similar to how you referred to your `.bib` file with `bibliography: ref.bib`, you do something similar:

```
---
title:
author:
output: html_document
bibliography: references.bib
csl: my_journal.csl
---
```

> **i** Your Turn
>
> In "05-qmd-bib-polish.qmd":
>
> 1. select your bibliography style to be one from your favourite journal at the CSL github repo here: https://github.com/citation-style-language/styles (> 2,600 citations and counting)
> 2. place this in your rstudio project
> 3. refer to it in the YAML
> 4. Render your document and observe your greatness

15.6   How to move the bibliography location

The bibliography is typically placed at the end of the document, so your last heading should be something like `# References`. However, if you want to move it, place the following piece of text in the reference section. For example.

```
# Introduction

# References

::: {#refs}
:::

# Appendix
```

This is taken from this section of the Quarto documentation. Note they also state:

> If your bibliography is being generated using BibLaTeX or natbib...the bibliography will always appear at the end of the document and the #refs div will be ignored.

15.7   How to not print / suppress the bibliography?

The bibliography can be suppressed with the YAML option `suppress-bibliography`

```
title: "document"
output: html
bibliography: file.bib
suppress-bibliography: true
```

> **i** Your Turn
>
> In "05-qmd-bib-polish.qmd":
>
> 1. Generate a bibliography and an appendix that follows it

💡 Demo: Use the Visual Editor mode of RStudio

Show off the citation auto-complete magic!
- search for DOIs
- search for R packages
- search pubmed/datacite/more!

16

Captioning and referencing equations

This section introduces how to add captions to equations, and reference them in text.

16.1 Overview

- **Teaching**: 5 minutes
- **Exercises**: 5 minutes

16.2 Questions

- How do I caption an equation?
- How do I reference an equation?

16.2.1 Numbering equations

You can make reference an equation by adding a label starting with `#eq-` after the equation `$$`. For example:

```
$$
Y \sim X\beta_0 + X\beta_1 + \epsilon
$$ {#eq-linear}
```

Gives

$$Y \sim X\beta_0 + X\beta_1 + \epsilon \tag{16.1}$$

You can then refer to the equation in text using `@eq-linear`:

Our model is given in Equation 16.1.

16.3   Other equation-adjacent referencing

You can also use and reference theorems, lemmas, conjectures, and many more - to see these, see the Quarto documentation: theorems and proofs documentation.

17

Common Problems with Quarto (and some solutions)

There are some things that I run into fairly frequently (and some not so much) when I'm rendering my Quarto documents. This section details some the common problems, and the solution that I have found works for me.

> **i** Your Turn
>
> 1. Using "06-qmd-find-errors.qmd"
>
> - Fix all the problems and get the document to render
>
>   1. Go to this repo njtierney/qmd-errors, and give debugging some of these common Quarto errors a go.
>
> You can download this repository by running this code:
>
> ```
> usethis::use_course("njtierney/qmd-errors")
> ```

17.1 More practice?

If you want to practice fixing broken Quarto documents, check out some pathologically broken examples on github at github.com/njtierney/qmd-errors.

17.2 Avoiding problems

To avoid problems in the first place, I try and do the following:

- Develop code in chunks and execute the chunks until they work, then move on.
- Render the document regularly to check for errors.

Then, if there is an error:

- Recreate the error in an interactive session:
  - restart R
  - run all chunks below
  - find the chunk that did not work, fix until it does
  - run all chunks below
  - explore working directory issues
    * remember that the Quarto directory is where the .qmd file lives

17.3 The errors

What follows from here are all the errors you might in an Quarto document, with the following structure:

- What they might look like
- What the error message might appear to be, and
- How to solve them

17.3.1 Python not found

An error like:

```
Error:
! /Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/config-3.7m-darwin/libpy

Backtrace:
  1. global .main()
  2. execute(...)
  3. rmarkdown::render(...)
  4. knitr::knit(knit_input, knit_output, envir = envir, quiet = quiet)
  5. knitr:::process_file(text, output)
     ...
 17. reticulate:::initialize_python()
 18. base::tryCatch(...)
 19. base (local) tryCatchList(expr, classes, parentenv, handlers)
 20. base (local) tryCatchOne(expr, names, parentenv, handlers[[1L]])
 21. value[[3L]](cond)
```

This error went away when I installed python - I went to https://www.python.org/downloads/ and followed the prompts

17.3.2 No julia

```
Error in `loadNamespace()`:
! there is no package called 'JuliaCall'
Backtrace:
  1. global .main()
 14. base::loadNamespace(x)
 15. base::withRestarts(stop(cond), retry_loadNamespace = function() NULL)
 16. base (local) withOneRestart(expr, restarts[[1L]])
 17. base (local) doWithOneRestart(return(expr), restart)
```

Fix: Install `JuliaCall` R package:

```r
install.packages("JuliaCall")
```

17.3.3 "Duplication": Duplicated chunk names

What it might look like

Chunks like this:

````
```{r}
#| label: repeated
1+1
```


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

```
2+2
```
```

The error message

This is caught before the document compiles with a warning like:

```
processing file: duplicated-chunk-names.qmd
Error in parse_block(g[-1], g[1], params.src, markdown_mode) :
  Duplicate chunk label 'repeated', which has been used for the chunk:
1+1
Calls: .main ... process_file -> split_file -> lapply -> FUN -> parse_block
Execution halted
```

The important part to note is the start:

```
.
.
.
Duplicate chunk label 'repeated', which has been used for the chunk:
1+1
.
.
.
```

How to solve

• In our case we have the same chunk name twice: 'repeated'. Change the
  chunk name of one of them!

17.3.4   "Not what I ordered": Objects not created in the right order

What it might look like

```
plot(my_table)

my_table <- table(mtcars$cyl)
```

The error message

```
processing file: wrong-order.qmd
  |....................................                          |  67% [plot-table]Error:
! object 'my_table' not found
Backtrace:
 1. base::plot(my_table)
```

```
Quitting from lines 8-12 [plot-table] (wrong-order.qmd)
```

How to solve

There is a good clue at the end of this message here, stating:

.
.

```
Quitting from lines 8-12 [plot-table] (wrong-order.qmd)
```

.
.

The template here is:

```
Quitting from lines START-END [CHUNK LABEL] (QUARTO-DOCUMENT-NAME.qmd)
```

I would then navigate to those lines of code in the Quarto document, or search for the chunk label, and see if I can run that code interactively. A common problem with these kinds of errors is that they *just might* work interactively, because you might have run them out of sequence. However because a Quarto document goes from a fresh R session and runs the code from the top to the bottom in that order, the objects might not yet exist. So:

1. Restart R - ensure it is a clean session without objects from the previous session hanging around
2. Run code above the problem chunk
3. Run code in the problem chunk
4. Hopefully this reproduces your error
5. If it doesn't reproduce your error, consider trying to make a small reprex of the problem so you have try and figure out the bug.

17.3.5 Forgotten Trails I: Missing "(", or "}"

What it might look like

```{r}
#| label: fig-volcano
#| eval: false
image(volcano
```

The error message

```
processing file: forgotten-trails-i.qmd
  |...................................                         |  67% [fig-volcano]Error in `p
! <text>:2:0: unexpected end of input
1: image(volcano
    ^
Backtrace:
```

```
 1. global .main()
 2. execute(...)
 3. rmarkdown::render(...)
 4. knitr::knit(knit_input, knit_output, envir = envir, quiet = quiet)
 5. knitr:::process_file(text, output)
    ...
11. knitr:::eng_r(options)
14. knitr (local) evaluate(...)
15. evaluate::evaluate(...)
17. evaluate:::parse_all.character(...)
18. base::parse(text = x, srcfile = src)
```

```
Quitting from lines 7-9 [fig-volcano] (forgotten-trails-i.qmd)

Execution halted
```

How to solve

In this case the key part to look at is

```
|...................................                     |  67% [fig-volcano]Error in `par
! <text>:2:0: unexpected end of input
1: image(volcano
```

Here the error message even points to the `image` code, and the message, "unexpected end of input" is referring to the missing closing parenthesis.

What it might look like

````
```{r
#| label: fig-volcano
image(volcano)
```
````

The error message

There **is no error message** here, but your output might look like this:

```
Forgotten Trails I: Missing "(", or "}"
{r #| label: fig-volcano image(volcano)
```

How to solve

The clue here for me is that we are getting this part of the code:

```
{r #| label: fig-volcano image(volcano)
```

When we would normally not see any information about the code chunk option, `#| label`.

When you come across some funky looking text like that, look for a missing `}`.

17.3.6 "Forgotten Trails II": Chunk option with trailing ", or not input

What it might look like

The error message

```
ERROR: YAMLException: unexpected end of the stream within a double quoted scalar (forgotte
7: #| label: fig-volcano
8: #| fig-cap: "An elevation plot of Maunga Whau (Mt Eden). Darker colours indicate higher

9: image(volcano)
```

How to solve it?

- The clue here is `unexpected end of the stream within a double quoted scalar`. Which, although it might sound a bit obtuse, is referring to a missing quote. The other clue is that the error captures the code chunk information and prints it out to the console.
- When you are working with this code in RStudio it also highlights the error:

17.3.7   "The Path Not Taken" File path incorrect

What it might look like

```{r}
#| label: read-data
#| echo: fenced
#| eval: false
library(readr)
penguins <- read_csv("the_penguins.csv")
```

The error message

```
processing file: path-not-taken.qmd
  |..................................                               |  67% [read-data]Error:
! 'the_penguins.csv' does not exist in current working directory ('/Users/nick/github/njti
Backtrace:
 1. readr::read_csv("the_penguins.csv")
 4. vroom (local) `<fn>`("the_penguins.csv")
 5. vroom:::check_path(path)



Quitting from lines 7-10 [read-data] (path-not-taken.qmd)

Execution halted
```

How to solve

The key part to pay attention to here is:

```
Error:! 'the_penguins.csv' does not exist in current working directory ('/Users/nick/githu
```

In this case, we need to ensure that the data file is in the right spot - in our example we have a typo - the data should be "data/penguins.csv".

17.3.8   "Spolling I" Incorrectly spelled chunk options

These are often not an error, but you just won't get the behaviour that you expect.

What it might look like

```{r}
#| label: fig-penguins
#| fig-caption: "Penguin Bill Length against Flipper Length. coloured by species. Each dot
#| echo: fenced
library(palmerpenguins)
library(ggplot2)
```

```
ggplot(penguins,
       aes(x = flipper_length_mm,
           y = bill_length_mm,
           colour = species)) +
  geom_point() +
  scale_color_brewer(palette = "Dark2") +
  theme_minimal() +
  facet_wrap(~island)
```

Above we have @fig-counts...

In this case we do not get a rendered caption at all:

```
library(palmerpenguins)
library(ggplot2)
ggplot(penguins,
       aes(x = flipper_length_mm,
           y = bill_length_mm,
           colour = species)) +
  geom_point() +
  scale_color_brewer(palette = "Dark2") +
  theme_minimal() +
  facet_wrap(~island)
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_point()`).
```
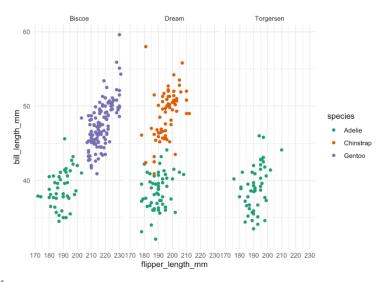

Figure 1

Above is Figure 1

The root cause of this is that we have specified `fig-caption` instead of `fig-cap`.

This once caused me to rewrite a lot of code and an entire section of a paper until I realised the problem.

The error message

There is no error message for this! It is a silent error.

How to solve

You can resolve this issue by using `fig-cap` instead of `fig-caption`.

17.3.9 "Spolling II" Incorrectly spelled chunk option inputs

So this is when you provide the wrong input to your chunk options. Like something that requires `true` gets "yes", or something that needs `"100%"` instead gets `100`

What it might look like

```
 6 ▾   ```{r}¬
 7     #| ·label:·fig-volc¬
✖ 8     #| ·cache:·yes¬
 9     image(volcano)¬
10 ▾   ```¬
```

The error message

```
==> quarto preview spolling-ii.qmd --to html --no-watch-inputs --no-browse

(line 8, columns 11--14) Field "cache" has value yes, which must instead be `true` or `fal

  The value yes is a string.
  The error happened in location cache.
  Quarto uses YAML 1.2, which interprets booleans strictly.
  Try using true instead.

ERROR: Validation of YAML cell metadata failed.
ERROR: Render failed due to invalid YAML.
```

How to solve

There's a bit of text here, but the key piece is:

```
(line 8, columns 11--14) Field "cache" has value yes, which must instead be `true` or `fal
```

We have specified `#| cache: yes` instead of `#| cache: true`. A good thing to internalise here is that Quarto always uses lowercase `true` or `false` and never `yes` or `no`.

17.3.10 "The Legend of Link I": Your images in `![]()` don't work.

I often forget that it is `![](path/to/image)`, and not `![]("path/to/image")`. There are no quote marks!

Legend of link

`![]("images/australia.png")`

`![](images/australia.png)`

17.3.11   LaTeX errors

There is no panacea for LaTeX errors, but if you aren't familiar with "what that error message" might look like, here are some details.

What it might look like

The error message

How to solve

17.3.12   I want to include inline R code verbatim to show an example

... Like for a book on using Quarto or something.

You can use the chunk option `echo: fenced`. See the Quarto documentation on fenced echo for more details.

> **Tip**
>
> Back before we had this option we used to have to do things like this:
> blog post by T. Hovorka from R Views
> It boils down to this:
> `r "\u0060r expression\u0060"` .
> Thankfully the folks at Quarto have made this much easier!

17.3.13 My Figure or Table isn't being cited

What it might look like

You create a figure,

The error message

There isn't one - you just get `@fig-chunk-name` printed.

How to solve

You need to make sure that you actually print the table or plot. If you create the plot and save it, but do not print it in the document, then you will not be able to reference the plot or table.

17.3.14 `::: {.cell}` appears in my quarto document

Generally look for a missing `:::` partner - however sometimes I do find this problem a bit sticky!

18

Different Outputs and Extensions

There are many different outputs for Quarto - as a start, THIS VERY BOOK IS WRITTEN IN QUARTO! How cool is that? I think it's pretty cool.

What's especially great is that the extra barriers to moving from one format to another are relatively low. Generally speaking, the things that you will change in your overall setup are:

- Adding a `_quarto.yml` file
  - Making some changes inside this file to tell it whether it is a book/manuscript/slideshow

There are some small differences in how files are setup, but by and large the work that you need to do will be on the writing of the content, and your code. Which is good! That's where we want out focus to be.

18.1 Alternative output formats

- Write a book
- Build a website
- Create a simple dashboard
- Use shiny with Quarto
- Generate multiple reports with parameterised reports - blog post by Mike Mahoney, blog post by Mandy Norrbo
- typst (like a next-generation LaTeX - still new.)

18.1.1 Slideshows / Presentations

- HTML: revealjs
- PDF: beamer
- powerpoint

18.1.2 Quarto Manuscripts

Quarto manuscripts are a relatively new feature in quarto. They essentially

help you share a bundled folder with the journal document, the rendered code, and other bits and pieces. In a future version of this book I will discuss using these in your writing.

18.1.3   Quarto Extensions

Quarto has an official extensions API, you can see all their extensions on their extensions page. Perhaps the most relevant is the journal extensions page, which I discuss below. For more information on creating extensions, see their creating extensions page.

18.1.3.1   For Journals

Quarto has provided a substantial list of supported journal formats in the journal listings Quarto extensions page. This is similar to rticles in rmarkdown.

18.1.3.2   Other extensions

Other formats and extensions for PDF, for example, the `hikmah.pdf` extension by Andrew Heiss.

These can be found at the Quarto Custom Formats Page.

19

Next Steps

So now you've got a handle on Quarto, what are some of the other things to think about learning? Here are some of my recommendations.

19.1   Learn how to use git and github

git is a version control system. Not sure what a version control system is? No worries, let me explain. If you've ever named a document something like:

```
Final
Final 2
Really final
```

[Relevant PhD comics link](#)

Or even if you have something like:

- `2018-10-10-document.qmd`
- `2018-10-11-document.qmd`

These are ways of managing which version you have.

To learn git and github, I'd highly recommend [Happy Git with R](#) by [Jenny Bryan](#), the [STAT 545 TAs](#), and [Jim Hester](#)

19.2   Learn how to make reproducible examples

(The following is an excerpt from my blog post, ["How to get good at R"](#))

When you run into a problem, or an error, if you can't work out the answer after some tinkering about, it can be worthwhile spending some time to construct a small example of the code that breaks. This takes a bit of time, and could be its own little blog post. It takes practice. But in the process of reducing the problem down to its core components, I often can solve the problem

myself. It's kind of like that experience of when you talk to someone to try and describe a problem that you are working on, and in talking about it, you arrive at a solution.

There is a great R package that helps you create these reproducible examples, called reprex, by Jenny Bryan. I've written about the reprex package here

For the purposes of illustration, let's briefly tear down a small example using the somewhat large dataset of `diamonds`

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----------------------- tidyverse 2.0.0 --
v dplyr      1.1.4      v readr      2.1.5
v forcats    1.0.0      v stringr    1.5.1
v ggplot2    3.5.1      v tibble     3.2.1
v lubridate 1.9.4       v tidyr      1.3.1
v purrr      1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to be
```

```r
diamonds
```

```
# A tibble: 53,940 x 10
   carat cut       color clarity depth table price     x     y     z
   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
 7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
 8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
 9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
# i 53,930 more rows
```

Let's say we had a few steps involved in the data summary of diamonds data:

```r
diamonds %>%
  mutate(
    price_per_carat = price / carat
  ) %>%
  group_by(
    cut
    ) %>%
```

```
  summarise(
    price_mean = mean(price_per_carat),
    price_sd = sd(price_per_carat),
    mean_color = mean(color)
  )
```

```
Warning: There were 5 warnings in `summarise()`.
The first warning was:
i In argument: `mean_color = mean(color)`.
i In group 1: `cut = Fair`.
Caused by warning in `mean.default()`:
! argument is not numeric or logical: returning NA
i Run `dplyr::last_dplyr_warnings()` to see the 4 remaining warnings.

# A tibble: 5 x 4
  cut       price_mean price_sd mean_color
  <ord>          <dbl>    <dbl>      <dbl>
1 Fair           3767.    1540.         NA
2 Good           3860.    1830.         NA
3 Very Good      4014.    2037.         NA
4 Premium        4223.    2035.         NA
5 Ideal          3920.    2043.         NA
```

We get a clue that the error is in the line `mean_color`, so let's just try and do that line:

```
diamonds %>%
  mutate(
    mean_color = mean(color)
  )
```

```
Warning: There was 1 warning in `mutate()`.
i In argument: `mean_color = mean(color)`.
Caused by warning in `mean.default()`:
! argument is not numeric or logical: returning NA

# A tibble: 53,940 x 11
   carat cut       color clarity depth table price     x     y     z mean_color
   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>      <dbl>
 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43         NA
 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31         NA
 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31         NA
 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63         NA
 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75         NA
 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48         NA
 7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47         NA
 8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53         NA
```

```
 9  0.22 Fair      E     VS2      65.1   61   337  3.87  3.78  2.49         NA
10  0.23 Very Good H     VS1      59.4   61   338  4     4.05  2.39         NA
# i 53,930 more rows
```

We still get that error, so what if we just do

```
mean(diamonds$color)
```

```
Warning in mean.default(diamonds$color): argument is not numeric or logical:
returning NA
```

```
[1] NA
```

OK same error. What is in color?

```
head(diamonds$color)
```

```
[1] E E E I J J
Levels: D < E < F < G < H < I < J
```

Does it really make sense to take the mean of some letters? Ah, of course not!

20

References

Quarto website

The Posit cheatsheet

R For Data Science section on Quarto

Extending word templates: https://quarto.org/docs/output-formats/ms-word-templates.html

happy git with r

21

Acknowledgements

This books was first written to be a guide for a course run by the Statistics Society Australia (SSA), and Melbourne Integrative Genomics (MIG) on November 12, 2018. Initially written as "Rmarkdown for Scientists", "Quarto for Scientists" takes the same format and makes it about Quarto.

I'd like to first thank Miles McBain, for his working book, "Git For Scientists". This book inspired the structure and workflow of this existing book.

I'd also like to thank Karthik Ram, Yoav Ram, Martin Fenner, Puneet Kishor, and Jonathan Dugan, involved with the Scholarly Markdown site. This has helped inform some of the structure of this book. I'd also like to thank Patrick Robotham for his helpful discussions when first creating this book.

There have been various wonderful contributions from the community to fix typos in this book, I would like to thank Allison Presmanes Hill PR1, PR2, as well as the many offline helpful conversations about serving this book online and other matters. I'd also like to thank Murray Cadzow PR, and Federico Marini PR, and Raymond B Huey for their thoughtful contributions.

A

Visual mode

This section will be best demonstrated live in the course.

But needless to say, some things worth checking out:

- citing papers in a linked .bib file
- insert anything shortcut
- insert table

Warn against potential issues that may arise due to text changes from swapping to visual model

There is also a VS Code extension for visual mode, which this book does not (currently) focus on.

B

Using Zotero with Quarto

https://quarto.org/docs/visual-editor/technical.html#citations-from-zotero

C

Templates

C.1 Controlling the outputs

Depending on the output type, HTML, PDF, or word, you can actually control how the document looks

C.1.1 Options for HTML

Some common options for HTML include:

- Adding tab sets
- floating table of contents

C.1.2 Options for PDF

- Adding page breaks
- injecting LaTeX into your Quarto document

C.1.3 Options for Word

- templates using .doct files

C.2 How do I set options specific to each output

Sometimes you might want to have specific output changes to

D

FAQ

(A place for questions)

D.1 How can I include a screenshot of an interactive graphic in PDF or Word?

You might wish to include a screenshot of an interactive graphic you had in your HTML document. To do this, you can use the webshot2 package

E

HTML document extensions

Some common options for HTML include:

- Adding tab sets
- floating table of contents

E.0.1 Adding Tab sets

A tab set looks like the following:

```
## Example tab set {.tabset}

### Tab 1

> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque sed eleifend velit. Sed

### Tab 2

```{r}
#| label: tab-plot
plot(iris$Sepal.Length, iris$Sepal.Width)
```
```

To add a tab set, you include `{.tabset}` after your heading

E.0.2 Floating table of contents

A floating table of contents can be added with the following lines in the YAML header:

```
---
title: "Your title"
author: "Your name"
output:
  html_document:
    toc: true
    toc_float: true
---
```

i Your Turn

1. Add tab sets to your document
2. Add a floating table of contents

Using manuscripts in your writing