

# **Scientific documents with Quarto**

**COUSIN Workshops Series**

Felipe Ortega

María Jesús Algar

2024-12-20

# Table of contents

<b>Preface</b>	<b>1</b>
<b>I Quarto</b>	<b>2</b>
<b>1 Scientific documents</b>	<b>3</b>
1.1 Literate programming . . . . .	4
1.2 Reproducible research . . . . .	4
1.2.1 Reproducibility and replicability . . . . .	5
1.2.2 Replication levels . . . . .	7
1.2.3 Replicability tools . . . . .	8
1.3 Quarto for scientific publications . . . . .	10
1.4 Quarton installation . . . . .	11
<b>2 Types of documents</b>	<b>12</b>
2.1 Individual documents . . . . .	12
2.2 Books . . . . .	12
2.3 Articles and publications . . . . .	12
2.4 Presentations . . . . .	13
2.5 Websites . . . . .	13
2.6 Dashboards . . . . .	13
<b>3 Quarto workflow</b>	<b>14</b>
3.1 Document assembly line . . . . .	14
3.2 Producing HTML . . . . .	15
3.3 Producing PDF . . . . .	15
3.3.1 Customising PDF documents . . . . .	16
<b>4 Individual Documents</b>	<b>17</b>
4.1 Creating a document with RStudio . . . . .	17
4.2 Document structure . . . . .	17
4.2.1 The preamble . . . . .	20
4.2.2 List of options . . . . .	20
4.2.3 Basic Markdown syntax . . . . .	21
4.3 Creating documents ( <i>output</i> ) . . . . .	21
4.3.1 Preview . . . . .	21
4.3.2 Selecting output type . . . . .	21
4.3.3 Basic configuration options . . . . .	23
4.4 Executable code <i>chunks</i> . . . . .	23
4.5 Author toolkit . . . . .	26
4.5.1 Document sections . . . . .	26
4.5.2 Equations . . . . .	29
4.5.3 Tables . . . . .	30

4.5.4	Callouts . . . . .	32
4.5.5	Bibliographic references . . . . .	32
4.5.6	General document style . . . . .	33
<b>II</b>	<b>Quarto books</b>	<b>34</b>
<b>5</b>	<b>Books</b>	<b>35</b>
5.1	Creating a book project . . . . .	35
5.2	Configuration options . . . . .	36
5.3	Home page ( <i>preface</i> ) . . . . .	37
5.4	Writing tools . . . . .	37
5.4.1	Book structure . . . . .	38
5.5	Managing references . . . . .	39
5.6	Project preview . . . . .	39
5.7	Publication options . . . . .	40
5.8	Customisation and templates . . . . .	40
<b>6</b>	<b>Creting a field guide</b>	<b>41</b>
6.1	Templates . . . . .	41
6.2	Project management . . . . .	41
6.3	Publishing . . . . .	41
<b>III</b>	<b>Publications</b>	<b>42</b>
<b>7</b>	<b>Scientific publications</b>	<b>43</b>
7.1	The <code>keep-tex</code> option: <code>true</code> . . . . .	44
7.2	Figures and graphs for publication . . . . .	44
7.3	Facilitate citation of articles . . . . .	45
7.4	Example of using scientific article templates . . . . .	45
7.4.1	Elsevier Magazine Template . . . . .	45
<b>8</b>	<b>FAIR Principles</b>	<b>48</b>
8.1	Overview . . . . .	48
8.2	Publication of source code and technical documentation . . . . .	49
8.3	Dataset publication . . . . .	51
8.4	Reference management and open publication . . . . .	51
<b>9</b>	<b>Additional resources</b>	<b>52</b>
9.1	Quarto . . . . .	52
9.2	FAIR principles and open science . . . . .	52
	<b>References</b>	<b>53</b>
	<b>Appendices</b>	<b>54</b>
<b>A</b>	<b>Code reference</b>	<b>54</b>
A.1	Quarto statements . . . . .	54
A.2	R statements . . . . .	54

<b>B</b>	<b>Integrated Development Environments for Quarto</b>	<b>55</b>
B.1	R Studio . . . . .	55
B.2	Visual Studio . . . . .	55
B.3	Positron . . . . .	55
<b>C</b>	<b>Useful R packages</b>	<b>56</b>
C.1	Ecology . . . . .	56
C.2	Data visualisation . . . . .	56
C.3	Data processing . . . . .	56
C.3.1	Tidyverse . . . . .	56
C.3.2	Alternatives to the Tidyverse . . . . .	56
C.3.3	Pipelines . . . . .	56
C.4	Spatial data . . . . .	56
C.4.1	<code>sf</code> (Simple Features) . . . . .	56
C.4.2	<code>terra</code> . . . . .	56
C.5	Time series . . . . .	56
C.5.1	Tidyverts . . . . .	56
C.6	Data visualisation . . . . .	56
C.6.1	<code>ggplot2</code> . . . . .	56
C.7	Data analysis and Machine Learning . . . . .	56
C.7.1	Tidymodels . . . . .	56
C.7.2	<code>mlr3</code> . . . . .	56
<b>D</b>	<b>Producing PDF documents</b>	<b>57</b>
D.1	PDF documents with Quarto . . . . .	57
D.2	Quick LaTeX primer . . . . .	57
D.3	Available templates . . . . .	57
	<b>References</b>	<b>58</b>

# Preface

This workshop describes how to use Quarto, software for producing scientific documents and publications, in ecology and plant research.

Quarto is a powerful and versatile tool for researchers implementing **reproducible** workflows. The quest for open-access research, including the final product (manuscripts) and ancillary research materials like source code, datasets, figures, pipelines or setup files, has become a prominent concern among scholars and practitioners in many fields. Prestigious publications require authors to submit these materials alongside manuscript drafts to let other colleagues reproduce and validate the results, replicate studies in new cohorts or improve their interpretability.

Quarto combines formatted text and executable source code chunks into a single document. Code *chunks* can be written in different programming languages such as R, Python, Julia or Observable. As we will see, it is possible to combine different programming languages in the same document or collection of documents, increasing the flexibility of this tool.

This is a **practical guide**, presenting hands-on examples and code to produce your own Quarto documents quickly. In addition, key concepts and best practices are also presented to steer new Quarto apprentices in the right direction.

To learn more about Quarto visit the comprehensive guide. Quarto can produce standalone documents, books like this one, as well as complete websites.

**Part I**

**Quarto**

# 1 Scientific documents

In their daily work, students, academics and scientific specialists produce a large amount of documentation of all kinds: laboratory notes, lectures, memos, technical reports and, above all, scientific articles to publish their discoveries and advances in an area of knowledge. Normally, the creation of this type of scientific documents involves a large number of tasks involving different tools and possible points of failure.

Figure 1.1 shows a schematic overview of a classic workflow for creating scientific documents. The main element is often a word processor master file (Word, OpenOffice/LibreOffice, etc.), a web page, or a LaTeX file (if we are creating a PDF document) that holds all contents.

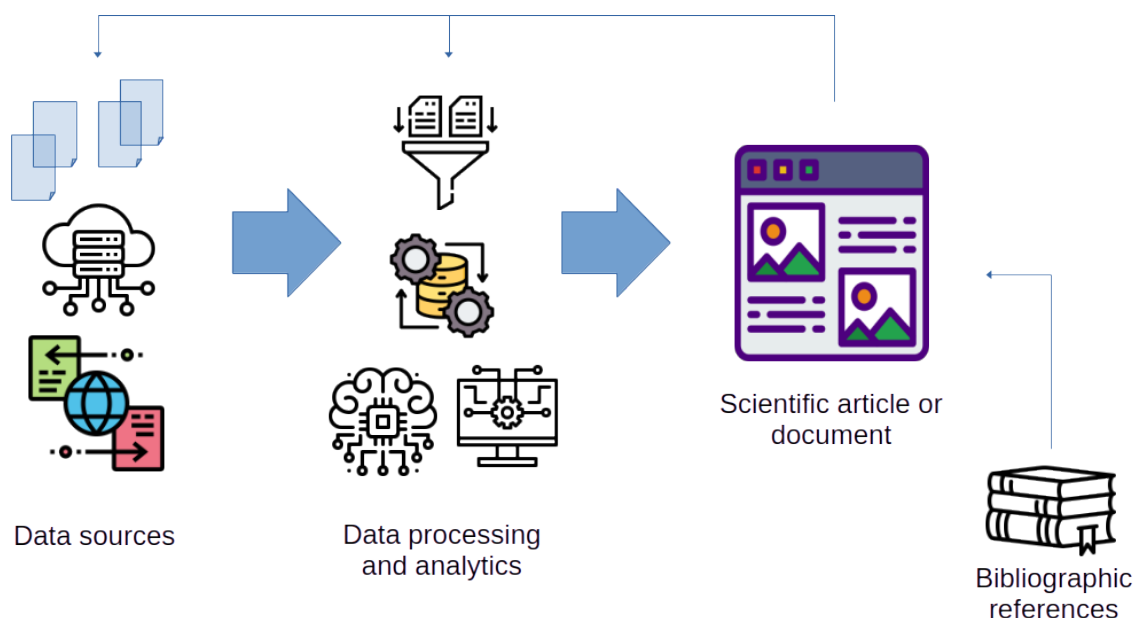


Figure 1.1: Creation process for scientific articles and documentation.

This master file is filled with content from a variety of sources, such as:

- figures and diagrams generated manually or through software code (such as data visualization charts);
- tables and summaries describing data sets and results;
- results and evaluation of the performance of models or algorithms; statistical or machine learning;
- mathematical formulas and equations;
- data tables and other useful information;
- bibliographic references (usually generated with the help of some bibliographic information management program).

Many of these elements force users to run external tools and programs, procedures, and other tasks over and over again to then incorporate the new results into the master file. We must admit that

this process, which is mostly manual, is not only tedious but also very prone to errors or oversights. “Wait! I forgot to update Figure 1.” “Are you sure these are the latest evaluation results for model  $M$ ?” “Have you checked that we have uploaded the latest version of the data file  $D$ ?” These are common questions that arise in the day-to-day work of scientific teams.

However, it would be great if it were not necessary to carry out all this manual and sometimes very frustrating process manually. Do we have any alternative to avoid it? Yes, we do. The answer to our needs is provided by a very powerful concept: **literate programming**.

## 1.1 Literate programming

The concept of literary programming was coined by Professor Donald E. Knuth (1984). Yes, you read that right, more than 40 years ago. This concept states that it should be possible to integrate, in a single scientific document, formatted text and results of the execution of software code to compose said document dynamically. So, why has it taken us so long to put this idea into practice? Knuth’s vision, although very ahead of its time, was correct, but the technology of the time did not allow it to be put into practice.

However, today we have all the essential elements to make it real. What’s more, we have a tool, Quarto, that lets us automate and manage the whole process of creating literary programming documents quickly and reliably.

## 1.2 Reproducible research

For many decades, the scientific method has been based on the publication of research papers describing the results of data analysis and experiments. In all cases, it is essential to be able to trust the conditions, the data collected, the method of analysis and execution of the experiments, as well as the various kinds of tools, including software, that the authors of the publication used to carry it out.

However, the numerous advances in recent years in the tools and methods of analysis make it much easier to check the results of these analyses. We might assume that this makes the work of scientists much easier, but in reality the opposite is true. Let us look at some examples:

- **Oncology** (Begley & Ellis, 2012): The Biotechnology Department of the firm Amgen (Thousand Oaks, CA, USA) was able to confirm only 6 of a total of 53 emblematic research articles published in this area. Bayer HealthCare (Germany) was able to validate only 25% of the studies analyzed.
- **Psychology** (Wicherts et al., 2006): 73% of the authors of a total of 249 articles published by the APA did not respond within a period of 6 months to the questions and requests formulated about the data they used in their research.
- **Economics and Finance** (Burman et al., 2010): A comparison of different software packages applied in the execution of various financial and statistical model analyses shows that each of these packages produces *very different* results using *the same statistical techniques* directly applied to *identical data* as those used in the original publication.



In fact, articles have even appeared suggesting that many of the results published in areas such as Medicine may not be entirely reliable (Ioannidis, 2005). As a result of all these recent findings, a great controversy has been generated throughout the scientific and research community, accompanied by a deep crisis of confidence.

Nevertheless, as a well-known comic strip about the academic world and research (see Figure 1.2) very well describes, the process of developing scientific publications is based primarily on the continuous review of methods and results (starting with the students themselves and their supervisors).

The Figure 1.3 shows a graph published in the prestigious journal Science Magazine (Brainard et al., 2018), which represents the data on the evolution of the number of research articles retracted or withdrawn for various reasons, between 1997 and 2014. In this graph, we can see how the improvement of tools and the greater availability of resources allow for the analysis and review of a greater volume of publications and analyses, which allows for the detection of a greater number of problematic cases.

### 1.2.1 Reproducibility and replicability

There is often talk of *reproducing* and *replicating* a data analysis or a scientific experiment (Leek & Peng, 2015). However, many evidences can be found showing that there are incompatible definitions of these two and other related terms (Barba, 2018). Be very careful, therefore, because depending on the scientific community or the field of knowledge in which we find ourselves, the meaning of these two terms may even be *entirely opposite* to their accepted definition in other areas<sup>1</sup>. Here we will stick to the definition accepted in a large number of areas, including statistics or scientific computing (see Barba, 2018, p. 33):

- **Reproducibility:** It is defined as the ability to recompute the results of an analysis, with the same data that were used in the original analysis, and knowing the details of the sequence (*workflow* or *pipeline*) of operations that make up said analysis. Certain premises must be able to be guaranteed:
  - If we use the same tools (e.g. R, a certain list of packages, the same versions of all packages and dependencies), as well as the same code (*R scripts*) on the same data, the results and conclusions must be consistent with those of the original analysis.
  - The authors of the original analysis must provide all the elements (data, code and procedure used) to allow the analysis to be reproducible (Barba, 2018).
- **Replicability:** It is defined as the ability to perform an experiment or analysis independent of the original, that addresses the same objective but on a set of data different from that used in the initial study. If the results are not consistent, it will be necessary to carry out more replications and harmonize the results and conclusions through appropriate techniques, such as **meta-analysis**.

---

<sup>1</sup>Among the most important examples of definitions that contradict those we give in this workshop are those adopted by the Federation of American Societies for Experimental Biology (FASEB), in immunology and microbiology, as well as those adopted by the Association for Computer Machinery (ACM) in computer science.

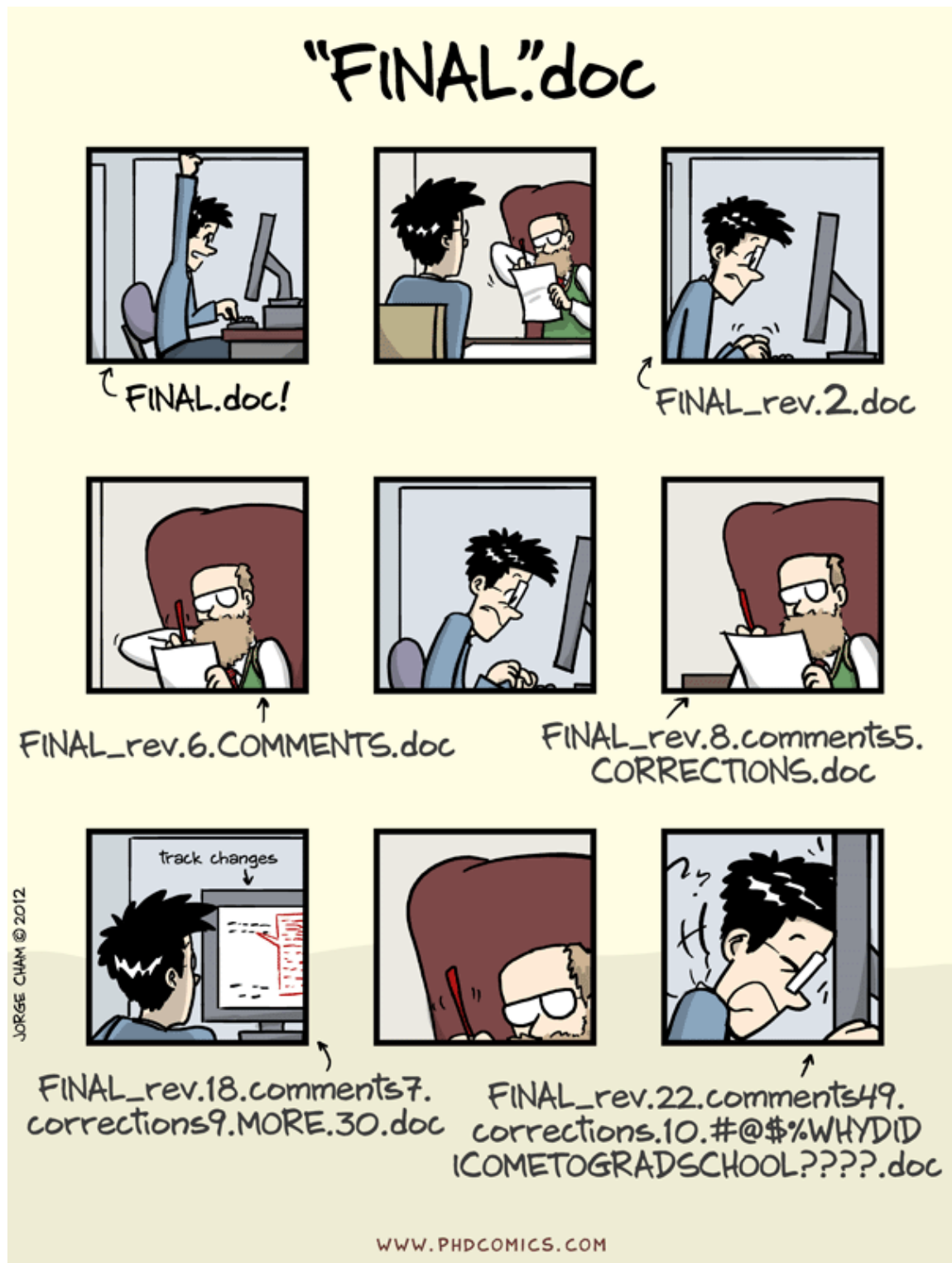


Figure 1.2: Comic strip representing the review model for scientific publications. Source: PhD comics.

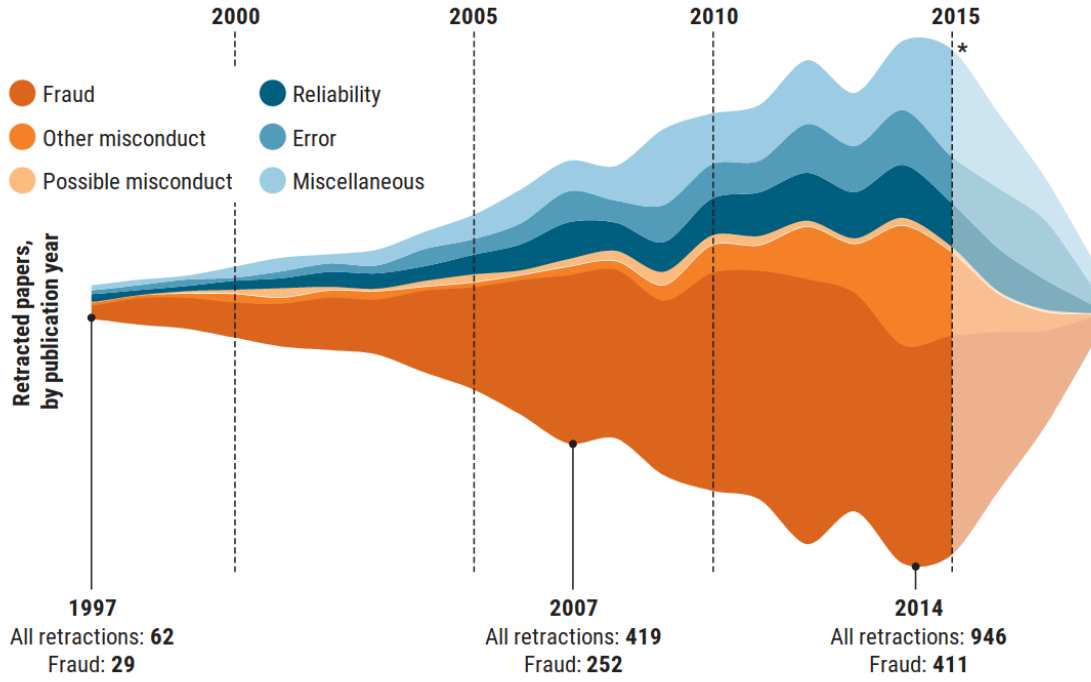


Figure 1.3: Evolution of the number of scientific publications retracted or withdrawn for various reasons, between 1997 and 2014. Source: Science Magazine (Brainard et al., 2018).

### 1.2.2 Replication levels

Depending on the elements published by the authors of the original study, as well as the level of detail with which the process for carrying out the study is described, the steps that have been followed and the tools used, we have different levels of replicability or reproducibility, represented in the Figure 1.4.

- *Not reproducible*: No data, code or any specific description of the implementation of the study or analysis is provided. Many scientific publications no longer accept publishing articles under these conditions.
- *Code or Data*: A good number of publishers request that the data sets used in the analysis or study of the publication be accessible through a URL, either because they are available in a public repository or because the authors of the article have published it. Likewise, many publications require that the software code to carry out the analysis is also publicly accessible, in an open source repository or in a freely accessible version control service project.
- *Code and data*: Ideally, both the code and the data should be publicly accessible for anyone who wants to examine them or use them to reproduce the results (validation) or replicate the analysis with other data or other cases.
- *Runtime environment and linked data*: A further step to facilitate the reproducibility of studies consists of publishing code and metadata files with more precise information about the programming language, the software packages used and any other dependencies necessary to carry out the same study or analysis. Another variant to facilitate reproducibility is to encapsulate the code and dependencies in a preconfigured virtual container, which can be downloaded and executed directly.

- *Gold standard*: The most advanced level would consist of documenting all the procedures performed during the study or analysis, including the coding of the tasks of obtaining, cleaning and preparing the data, as well as the generation of graphics to visualize the results or any other results derived from the study.

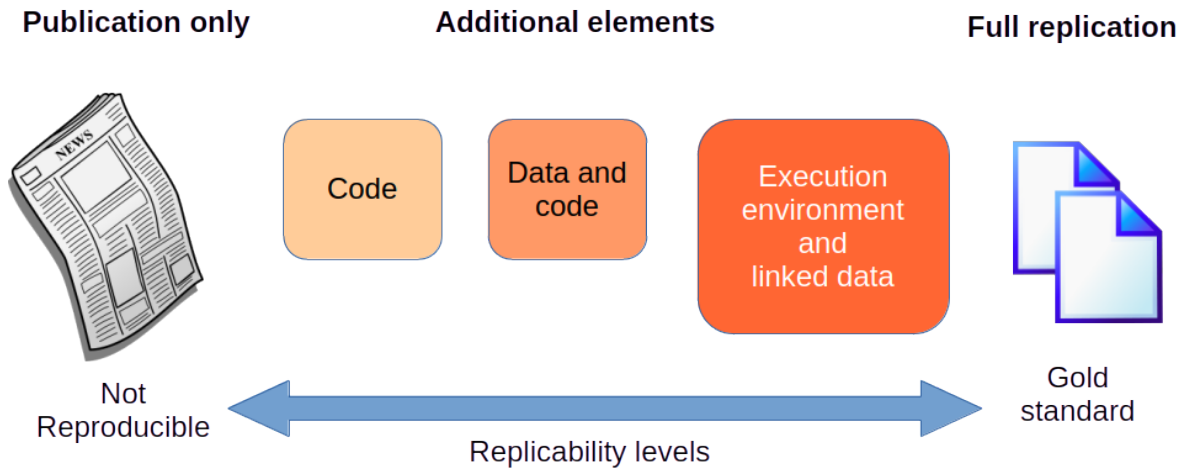


Figure 1.4: Replication levels spectrum in scientific publications. Source: Peng (2011).

### 1.2.3 Replicability tools

Certain technologies and tools that have become more sophisticated and refined in recent years are making it easier to replicate data processing and analysis.

- **Version Control Systems for software code (SCV)**: tools such as Git, Mercurial and web services such as GitHub or GitLab have popularized the creation and publication of projects that allow the management of the software code that has been created, controlling the changes and the released versions. Web services also integrate a good number of tools to support different facets of the software development process, such as the generation of documentation, manuals and examples, error reports and requests for improvements, continuous integration and continuous deployment (CI/CD), systematic testing of the generated code, etc. If you have not yet considered how using a source code version control tool can benefit you, take a look at Figure 1.5 where you will relive a situation that is unfortunately very common among researchers and scientists who develop software solutions.
- **Software virtualization and containers**: In a technological environment dominated by the contracting and deployment of computing infrastructure and services in cloud computing architectures, packaging and virtualization tools for software applications and services that can be installed and deployed in a short time have revolutionized the way software products are published and managed, including data processing and analysis products.
- **Data version control**: In a similar way to SCV for source code, software is appearing to apply the same principles to data files. In this way, we can control different versions of each data file, modifications made to them, etc. One of these tools is Data Version Control (DVC), which allows versioning of data and models. As a result, we can know at all times which version of the data and which list of *features* have been included in each model considered during the analysis, keeping the descriptive information about these three essential components that must always be cohesive.

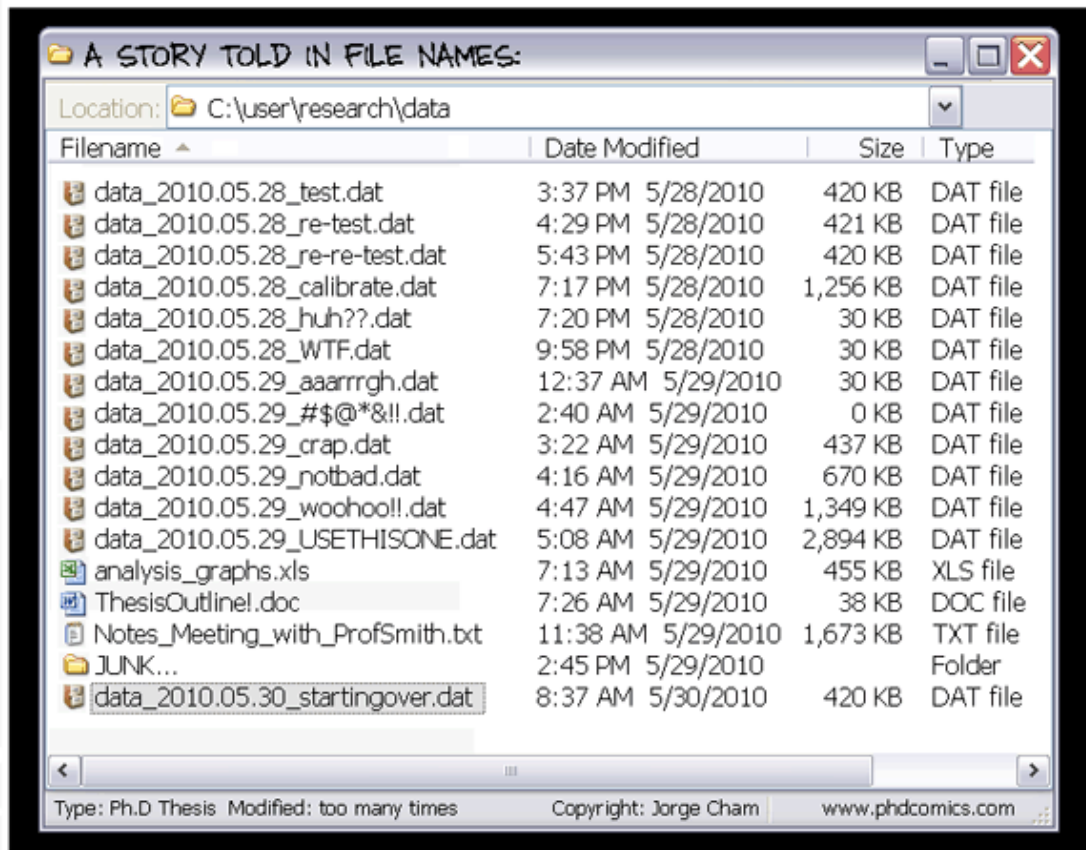


Figure 1.5: Software version control. Source: PhD Comics

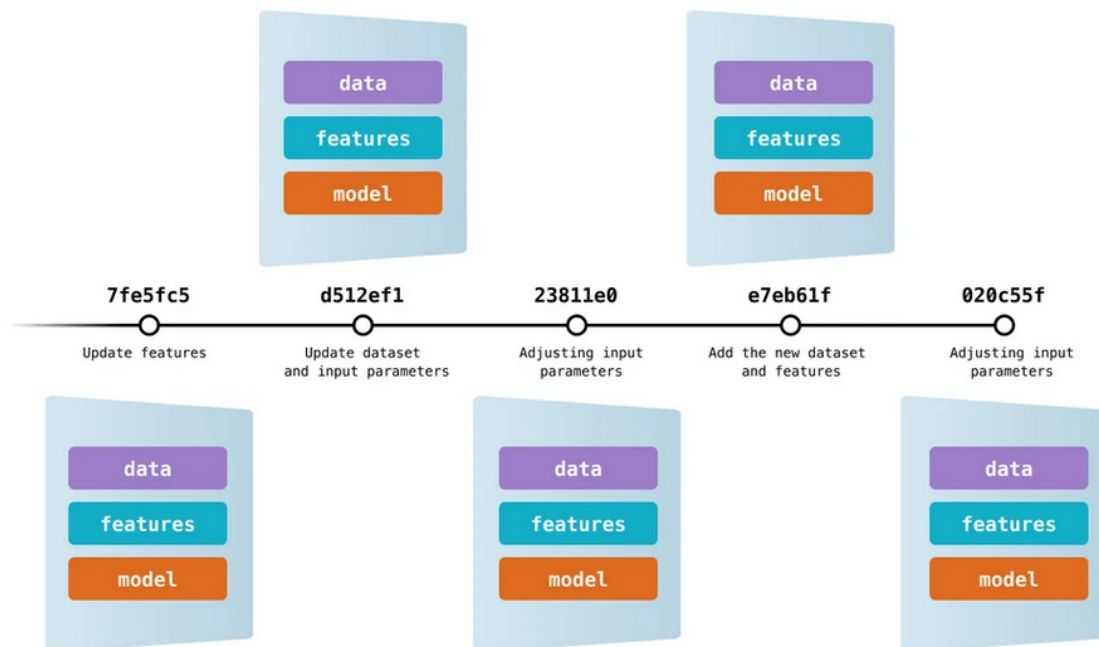


Figure 1.6: Data, code and model versioning example maintained by DVC. Source: DVC Documentation.

- Model and experiment management: Another type of machine learning project management tool is one that allows the organization, monitoring, comparison, and selection of the experiments and models we have carried out. One of the most recent notable examples is ML Flow, which provides support for model tuning, evaluation, and optimization, deployment of models in production environments, creation of a registry of pre-trained models, etc. Of course, it is possible to combine this type of tool with others such as DVC, creating a comprehensive management environment for our projects.
- Creation and management of data processing pipelines: the last essential element in any data processing and analysis project that must take care of scalability is a tool for creation and management of data processing and analysis flows or pipelines. The set of all the pipelines in our project make up the general workflow of the project. These tools are known as data or workflow orchestrators. In this category, we have both very powerful and feature-packed tools such as Apache Airflow or Prefect and simpler and more straightforward ones such as Luigi.

Of course, the R community has not remained oblivious to these new trends, in particular the R OpenSci initiative, within which we find many packages (published in the official CRAN repository) that cover various aspects of scientific work, including the management of *pipelines* and *workflows* through the `targets` package.

- User manual for the R package `targets`.

### 1.3 Quarto for scientific publications

Now that we know the fundamental concept on which Quarto works and its application to achieve a higher level of reproducibility and transparency in our scientific process, we are going to explain in more detail the process that Quarto follows to compose a document. The Figure 1.7 presents a diagram with the document creation process and the elements and tools that come into play to achieve it.

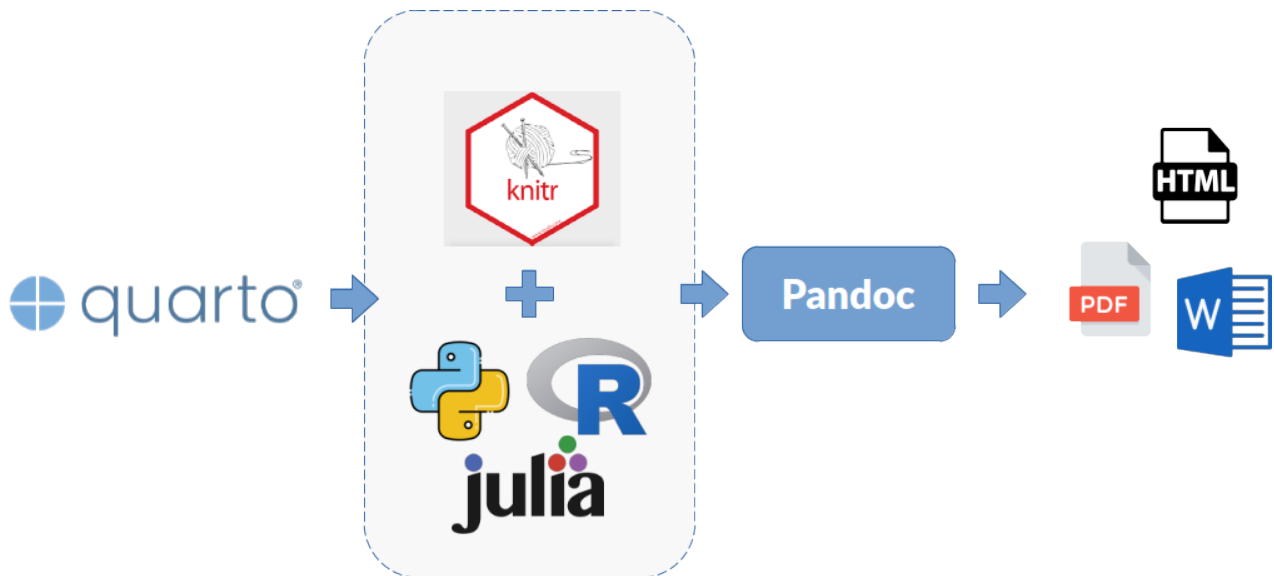


Figure 1.7: Content creation process with Quarto.

- **Quarto:** a software that allows you to create scientific documentation following the principles of literary programming.
- **Knitr and programming language:** the `knitr` package is responsible for the connection with an interpreter of a programming language (R, Python, Julia) that can be executed in a REPL environment, in order to be able to execute software code fragments integrated into the document and generate content in Markdown format as a result.
- **Markdown** (formatted content): a textual content markup language that allows easy formatting of the information in our documents created with Quarto.
- **Pandoc** (universal translator of document formats): this software receives the content already formatted using the Markdown standard, to convert it into the selected output type. There are several options available: HTML, PDF or Word, as well as slides, websites or interactive panels (*dashboards*).

## 1.4 Quartion installation

To install the latest version of Quarto software on your system, point your web browser to the page <https://quarto.org/docs/get-started/>. Here, download and install the file corresponding to your operating system.

At this time, the latest version of Quarto available is 1.5.57.

### Software requirements to generate PDF documents

By default, the output format of documents generated with Quarto is HTML. If we want to generate PDF documents, we need to have a LaTeX distribution installed. For more information, see Section 3.3.

## 2 Types of documents

In this chapter, we present the main types of documents and collections of scientific content that we can generate with Quarto.

### 2.1 Individual documents

The easiest way to work with Quarto is to create a document individual. Said document may use the sections or *chunks* of code to read input data or download it from some source, process them, analyze them and display the results. Graphics can be added, tables, equations, bibliographic references and many other elements.

The documents always have a standard structure:

- *Preamble*: in which configuration options are specified for the creation of the document with Quarto and its associated tools.
- *Body*: the section that houses the main content of the document, including sections of Markdown-formatted text and sections of code executable. The software code may be shown, if useful, or be hidden in the final result.
- *References*: References are included at the end of the document bibliographical, as is usual in scientific texts.

### 2.2 Books

The natural evolution of the previous case is to gather a collection of documents individuals in a single book. *Quarto books* allows you to create this type of documents, structured in parts, chapters and sections. The options of configuration will allow you to create an introductory cover for the site website that contains the chapters (one document per chapter) or the elements necessary to create a PDF book, similar to those published by a editorial.

### 2.3 Articles and publications

One of the key results in any scientific process is the production of articles and publications (technical reports, etc.) that collect the results and progress achieved scientists. In this case, Quarto can also help us, with the collaboration of other essential elements such as the R package `rticles`, which provides templates to generate articles according to the specifications of the main scientific publications and publishers in a multitude of fields of knowledge.



## 2.4 Presentations

It is also possible to generate presentations (usually in HTML format) with slides through Quarto. In this case, we would have the support of several packages and environments of creating web presentations at our disposal, such as `reveal.js` (HTML), Beamer (for LaTeX/PDF) or MS Office PPTX format.

We will not discuss this case in this workshop, but you can obtain more information in the online guide, available at <https://quarto.org/docs/presentations/>.

## 2.5 Websites

Another option that may be interesting is to create personal websites (for example, for show our CV and a selection of featured works, publications, etc.), blogs and even corporate websites (organization, research group) quickly using Quarto. There are numerous free and paid templates now available to create websites with a beautiful look. harmonized, although we will need to learn a little HTML and CSS to be able to customize further our website.

Here is an example of an environmental technology researcher website created with Quarto: <https://www.mm218.dev/>. More examples of different types of websites generated with Quarto: <https://drganghe.github.io/quarto-academic-site-examples.html>.

More information and tutorials for creating websites with Quarto can be found at <https://quarto.org/docs/websites/>.

## 2.6 Dashboards

Finally, it is possible to create custom dashboards for monitoring. of data, analysis of models and results or for examples and teaching applications using Quarto, as described in the guide <https://quarto.org/docs/dashboards/>.

In this case we can include among the tools Shiny, a package software for R (also available for Python) with which to create interactive applications based into data quickly and easily.

## 3 Quarto workflow

In this section we are going to explain some more details about the creation process of documents in Quarto, to better understand the components involved in this process and the configuration options we have available. The Figure 3.1 summarizes at a high level the phases of creating a document with Quarto.

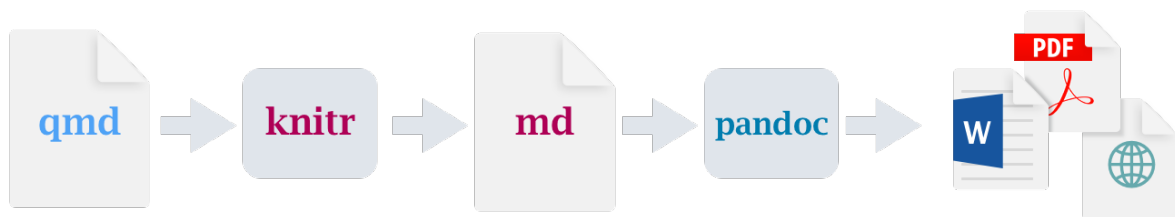


Figure 3.1: Content creation process with Quarto. Source: RStudio.

### 3.1 Document assembly line

We can consider the process of creating a document in Quarto as a chain assembly in which various software tools are applied sequentially to produce the final document in different output formats.

- Quarto: The first agent that intervenes in the interpretation of the file with extension `.qmd` is Quarto. The program must be previously installed on our computer so that the development environment that we decide to use (RStudio, Microsoft VS Code, etc.) can find it. Quarto is responsible for interpreting the content of the file and considering the different configuration options that we have inserted in the header of the document, as well as in executable code fragments, to delegate the construction tasks of the different parts of the document to other tools.

Quarto is also responsible for automatically inserting some authoring elements of documents (such as callouts, explained in Section 4.5.4), while the generation of other elements (numbering of figures, tables, bibliographic citations, etc.) It is delegated to other tools like Pandoc.

- Code execution engine (**engine**): One of the great advantages of document production in Quarto is the possibility of inserting fragments of executable code in our documents. Quarto can work with different engines (*engines* in Quarto terminology) that interpret the code and return the result of its execution to integrate it into the final document.
  - All R language code blocks use the `knitr` package as *engine* execution.
  - Executable code blocks in other languages supported in Quarto (Python, Julia, Observable) use the *kernel* available in the Jupyter tool for execution, except in the explicit case that blocks of code are combined in R and Python in the same document.

- In the particular case that the same document combines code blocks in R and Python, then the Knitr tool is used for the R code along with the R `reticulate` package to execute the blocks written in Python. This combination has the additional advantage that it is possible “pass” variables and results between the R and Python runtimes, so that we can use variables and data structures created in the R part within our Python code and vice versa.
- Markdown: The intermediate product of all the previous phases is a file in Markdown (`.md` extension), which already integrates the formatted textual content and many of the results and elements additional generated by the different tools that have come into play up to this point. If useful, this intermediate file can be stored for review or for use it for other purposes. This file is then sent to Pandoc for the last stage document creation.
- Pandoc: The Pandoc project offers a software tool to translate documents between different information representation formats. As you can quickly see On the main page of the project, the list of supported formats is really extensive. For purposes Practically speaking, Quarto uses Pandoc’s ability to receive as input a document in Markdown and generate output in three possible formats: HTML, DOC/DOCX or PDF. The Markdown file must include specific syntax to encode certain elements (cross-references, bibliographic citations, figures, tables, equations, etc.) that we will present in the following chapters, so that Pandoc can interpret these elements and represent them appropriately in each output format.

Finally, it should be noted that document viewing programs are needed to load the documents. Output documents, depending on format: web browser (HTML), MS Word (DOC/DOCX files), PDF viewer (PDF files).

## 3.2 Producing HTML

The default option for the output format of Quarto documents is to generate an HTML document, which can be viewed with most modern web browsers. This output format has several advantages:

- It is quite likely that the recipient of the document already has one or more browsers installed on your system to view the document, if we send or share it directly to you.
- It is easier to publish this type of documents on the Web, using one of the different platforms available for this purpose:
  - Quarto Pub, a document publishing service that allows are publicly accessible.
  - Individual documents (see Chapter 4) and books or collections of Documents (see Chapter 5) can be published easily and quickly on publishing sites. hosting software projects such as GitHub or GitLab, which also provide hosting services version control, bug/improvement reporting management, documentation, testing, etc.

## 3.3 Producing PDF

Unlike in HTML, when we generate PDF documents an additional compilation step is added of the document at the end of the entire assembly line, using LaTeX and the compilation engine XeLaTeX to generate PDF output. Therefore, if we select this output option it is **essential** have a **TeX/LaTeX distribution** previously installed on our system, to compile and generate the documents. If we do

not have any yet, you can install TinyTeX, a lightweight distribution of TeX Live which is much smaller in size (~100 MB vs. to more than 4 GB of full TeX Live).

#### 3.3.1 Customising PDF documents

Predefined LaTeX document templates can be used. By default, Quarto uses various templates from the LaTeX package collection koma-script.

Some of these templates can work relatively easily in Quarto, while Others require some adaptation, for which some knowledge about Latex. This is probably a more advanced topic for many users, so for now I don't We are going to discuss it in this introductory workshop.

However, as an example, we offer below a list of some examples that illustrate the enormous possibilities of this type of templates:

- Professor R.J. Hyndman has published Monash University Quarto document templates, that can be used as a starting point to customize them in our own projects.
- The repository Awesome Quarto Thesis collects a list of Quarto templates to generate TFG/TFM reports and doctoral theses for some universities. Also linked is a generic extension template for Quarto, designed to make it easier for other users to customize it according to the criteria marked by their own institution to generate these jobs.

## 4 Individual Documents

The easiest way to get started with Quarto is to create stand-alone documents. These are self-contained documents, incorporating formatted text and executable code in a single file.

To create a new document with Quarto, you can simply use the menu options in RStudio or MS Visual Code, or create a file with the extension `.qmd`.

### 4.1 Creating a document with RStudio

Before you start, make sure you have installed the Quarto software on your machine. It is a standalone software program, which needs to be installed for the rest of the process to work (see the Section 1.4 section).

If you already have a recent version of RStudio installed, you will need to install the following packages for the example:

```
install.packages("tidyverse")
install.packages("palmerpenguins")
install.packages("quarto")
```

Now, in RStudio we create a new project choosing the *Quarto project* option, as it appears in Figure 4.1.

We can name our project directory as `first-example` and click **Create Project**.

As a result, a new project should appear open on the screen, with the appearance shown in Figure 4.2.

Specifically, in the upper left panel we can see that, by default, the *Visual* editor has been opened, which allows creating Quarto documents in a more intuitive way. However, to start getting familiar with the structure of a quarto document from the beginning, we will switch to the *Source* editor to view the source code, by clicking on the button shown in the figure Figure 4.3.

### 4.2 Document structure

The following example code presents a basic structure of an individual Quarto document.

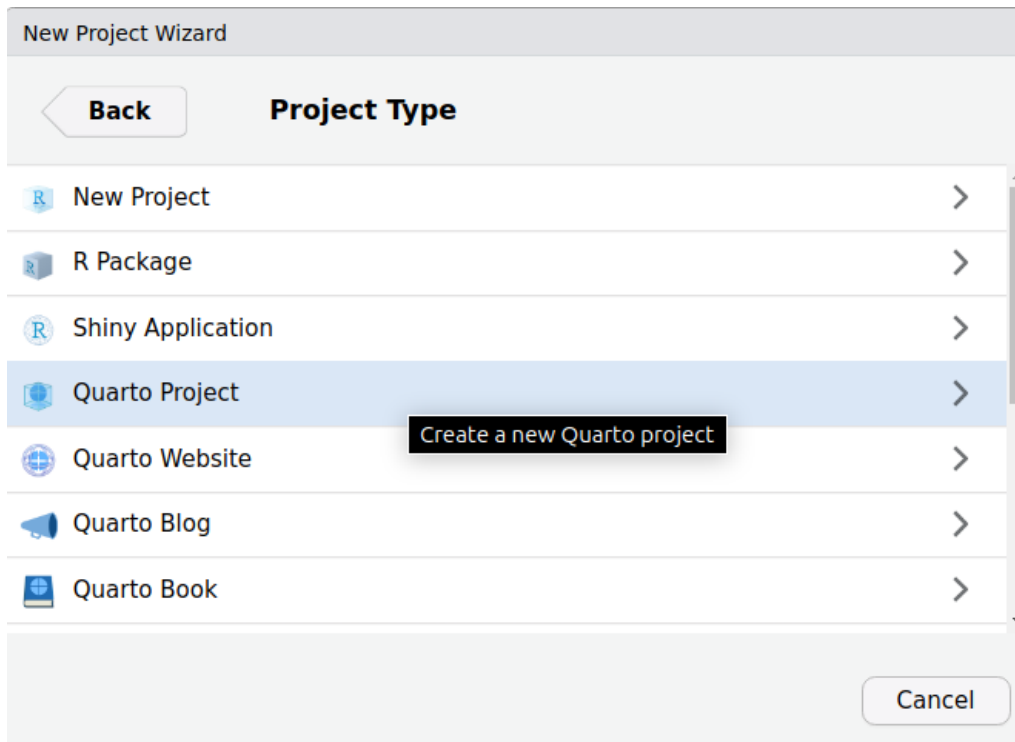


Figure 4.1: New Quarto project

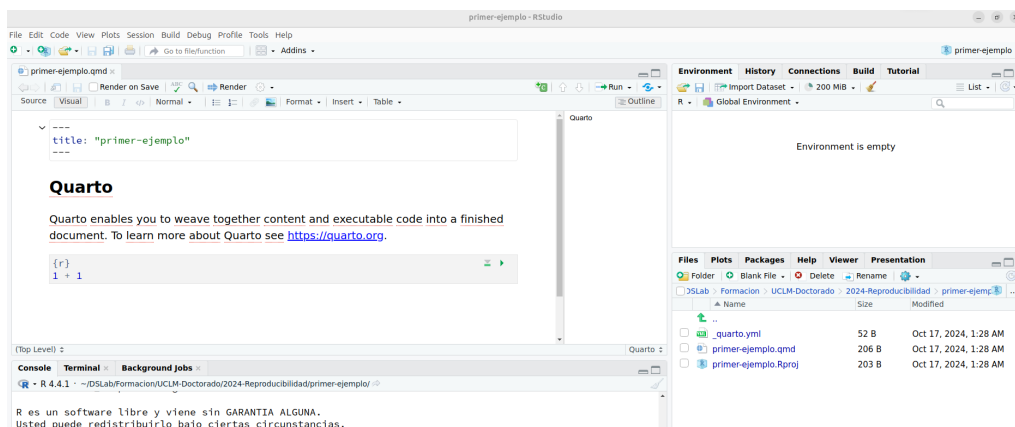


Figure 4.2: First example

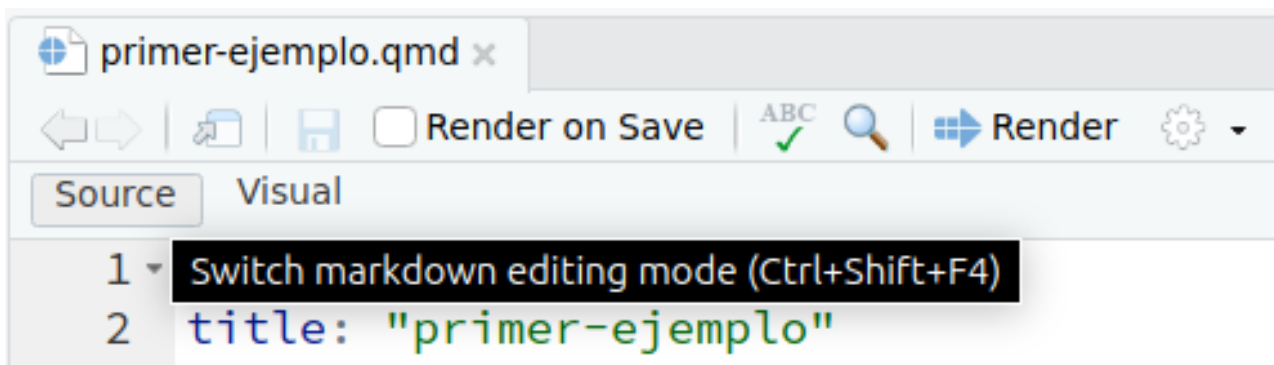


Figure 4.3: Source editor

```

---
title: "My first document"
author: John Doe
date: 2024-12-20
---

Here we have a line of formatted Markdown content.

```{r}
#| label: my-label

# This is a code chunk. You can place here executable
# code that will be run by the R interpreter. By default, the final
# document will automatically integrate both the input code and the
# output from its execution
a <- 1 + 1
```

*Additional* Markdown content after the code chunk.

```

The file content consists of two parts:

- **Preamble:** is delimited by two `---` tags. Within this area we can assign values to configuration options for layout and creating the document, such as title, author(s), date, etc.

We can also configure various options related to the output format of the documents.

- **Document body:** is composed of text paragraphs formatted using the Markdown markup syntax, which we will see later. In addition, executable code fragments or *chunks* can also be inserted into the text, which are marked up using a special syntax (as we see in the example above).

Each *chunk* of executable code is delimited as follows:

```

```{r}
# Código en R
```

```

#### 💡 Support for additional programming languages

Although in this workshop we focus on the R language, you should know that Quarto also supports other programming languages such as Python, Julia or Observable.

We can change the programming language of each code *chunk* by indicating its name at the beginning, for example:

```

```{python}
# Código en Python
```

```

Nevertheless, this code chunk in other languages requires additional configuration. For example, we must set up a Python interpreter to execute code chunks written in this language.

### 4.2.1 The preamble

A basic example of a preamble is as follows (although it would be sufficient to simply provide a title for the document):

```
---
title: "My first document"
author: John Doe
date: 2024-12-20
---
```

Of course, more options can be added, which we will explain next

### 4.2.2 List of options

There is an extensive list of configuration options that we can include in our documents.

- *HTML output options*: these allow us to configure various basic aspects of the document, such as the title and subtitle, date, author (or list of authors), summary or DOI; formatting options such as the subject or advanced styles for HTML content with CSS; numbering and table of contents, etc.
- Basic options for HTML with Quarto.
- Complete list of HTML options with Quarto.
- *PDF output options*: these offer the possibility of configuring multiple parameters for the creation of the document in this format, many of them similar to those for the HTML output. A particularly relevant option is to choose the LaTeX document format (`documentclass` option), which defines the general appearance of the layout to be used. By default, classes from the KOMA Script metapackage are used, such as `scrartcl` or `scrbook`. It is also important to indicate the `papersize` option, in our case to ensure that a standard format is used such as A4. The citation format is also relevant, being able to choose, for example, the BibLaTeX engine which is more powerful, with multilingual support and for native UTF-8 character encoding. Finally, it is also important to indicate the compilation engine. If you want full flexibility in document layout, it is highly recommended to use the XeLaTeX engine (option `pdf-engine: xelatex`), which is the *default* used by Quarto.
- Quarto PDF Basics.
- Complete list of PDF options available with Quarto.



### 4.2.3 Basic Markdown syntax

In the following link you can find a quick basic tutorial that shows the basic options of the Markdown syntax accepted in Quarto documents to format textual content.

- Basic guide to markdown syntax.

## 4.3 Creating documents (*output*)

By default, if we do not indicate anything, Quarto will generate a single output format of the document in HTML. However, it is possible to define more than one output format by including more configuration options. Of course, you can indicate different options to generate several output formats simultaneously, or to choose the output format that we want to produce based on our interests, selecting the format that we need when previewing or when generating the final document.

### 4.3.1 Preview

To preview the document we have to press the *Render* button in the tools menu of the RStudio interface, as shown in Figure 4.4.

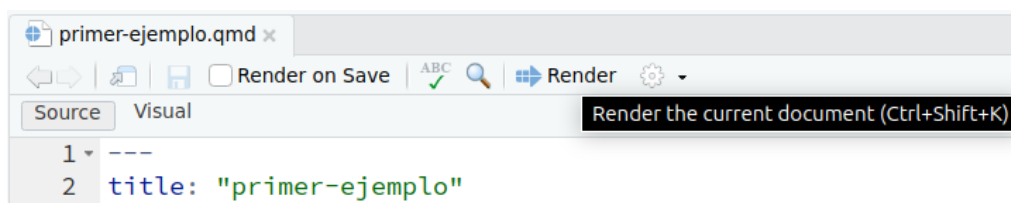
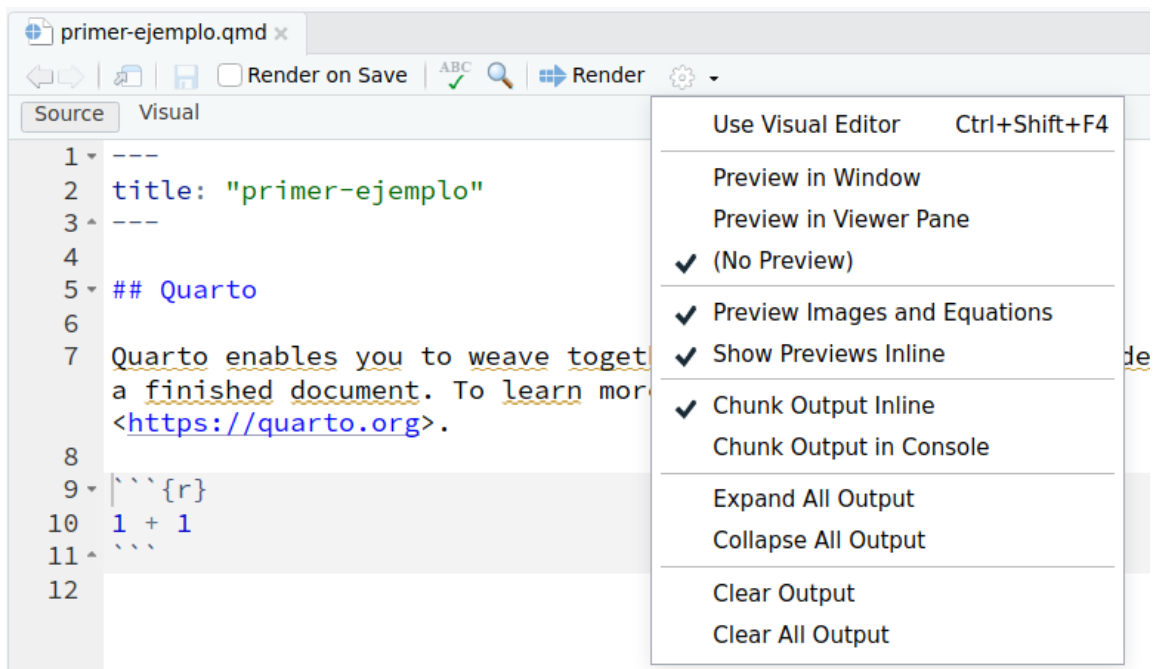
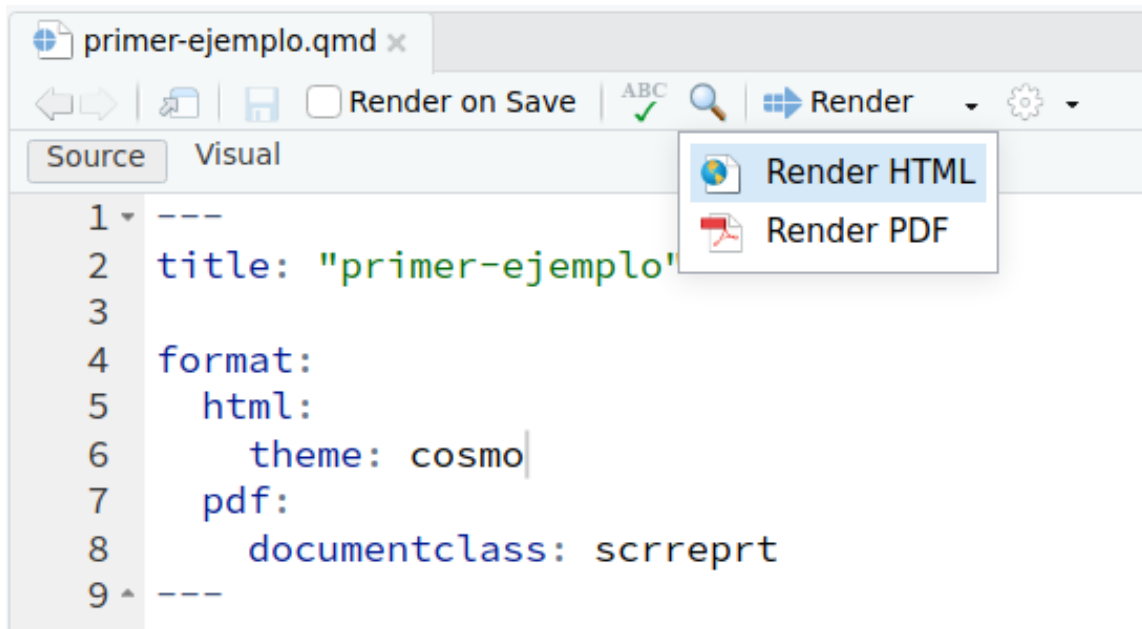


Figure 4.4: *Render* button to preview the generated document.

By default, our main web browser or a panel in the RStudio interface will open showing the HTML page with the document already generated. By clicking on the gear icon next to the *Render* button we can select, among other things, the type of preview we want to be launched after completing the creation of the document or completely disable said preview. Available options are shown in Figure 4.5

### 4.3.2 Selecting output type

When we have several output format options configured in our document, we can choose at preview time which of the formats is chosen to generate the document. In the Figure 4.6 you can see an example of a document that includes settings for two output formats (HTML and PDF) and the change in the *Render* button, in which a small black arrow now appears just to the right of the button icon to display the two available output options.

Figure 4.5: *Render* operation preview options.Figure 4.6: Selecting output formats with the *Render* operation.

### 4.3.3 Basic configuration options

Below is an example of some basic configuration options that are typically found in documents formatted as HTML output.

```

---
title: "My first document"
author:
  - "John Doe"
  - "Mary Jane"
date: 2024-12-20

lang: en
bibliography: references.bib

format:
  html:
    theme: cosmo
    toc: true
    number-sections: true
    html-math-method: katex
    css: styles.css
  pdf:
    documentclass: scrreprt
---

REST OF THE DOCUMENT

```

In this example, in addition to the author and the date, a list of two authors is indicated, the main language of the document (Spanish), the bibliography reference file (in `.bib` format) and, within the HTML options, the layout topic, the inclusion of a table of contents (located by default at the top right), section numbering, selection of the engine to render equations in the document and a custom styles file in CSS format to adjust some fine layout options.

One option worth highlighting is to force all resources (images, style information, etc.) to be integrated into the HTML file itself, to facilitate direct sharing or publication of the document without having to also provide the auxiliary files necessary to display it in the browser. This option is shown below:

```

format:
  html:
    embed-resources: true

```

## 4.4 Executable code *chunks*

The most distinctive feature of documents created with Quarto is the possibility of inserting executable code fragments, called *chunks*, into the document itself. This also includes the option for

said code to generate different results (numeric, graphic, tables, animations, etc.) that are integrated directly into the document. In this way, if we keep the code updated, the correct versions of said results will always be generated.

The executable code fragments have the following structure:

```
```{r}
#| label: id-fragmento

# Aquí va el código ejecutable
a = c(1, 2, 3, 4)
b = a^2
```
```

The triplet of characters ````` is called a *fence* and delimits the beginning and end of the code fragment. Immediately after the opening delimiter, the identifier of the programming language in which the code of that fragment is written is written in curly braces. This information is used to choose the appropriate syntax highlighting to display the code of that language and to select the interpreter that executes the code and produces the results.

In the following lines we can include one or several **configuration options** specific to that code fragment, using the syntax `#| option: value`. For example, in the previous fragment the option `#| label: fragment-id` creates a label (which must be unique) to identify that fragment of code within the document.

- List of options for code fragments.

Some frequently used options are:

- `eval: true | false | [...]`: Indicates whether the content of that snippet should be evaluated (executed). A list of positive or negative line numbers can be passed to explicitly select which lines of code are included (positive) or excluded (negative) from execution.
- `echo: true | false | fenced | [...]`: Indicates whether the source code of the snippet should be included in the document or not. The **fenced** option also includes the cell delimiter as part of the output. Finally, it also accepts a list of positive or negative line numbers to select which lines of code will or will not be displayed in the snippet.
- `output: true | false | asis`: To decide whether the result of the code execution is included in the document or not. The **asis** value forces the result to be treated as raw Markdown content.
- `warning: true | false`: Indicates whether warning messages should be included in the output.
- `error: true | false`: Indicates whether generated error messages are included in the output.
- `message: true | false`: Indicates whether generated information messages are included in the output.

When fragments generate figures, these are inserted into the document itself. Let's look at an example:

```

```{r}
#| label: fig-example-cars
#| fig-cap: "Gráfico de correlación lineal positiva entre el kilometraje en ciudad
  ↪ y en carretera de diferentes modelos de coches."

library(ggplot2)
#| label: scatterplot
#| echo: true

ggplot(mpg, aes(x = hwy, y = cty, color = cyl)) +
  geom_point(alpha = 0.5, size = 2) +
  scale_color_viridis_c() +
  theme_minimal()
```

```

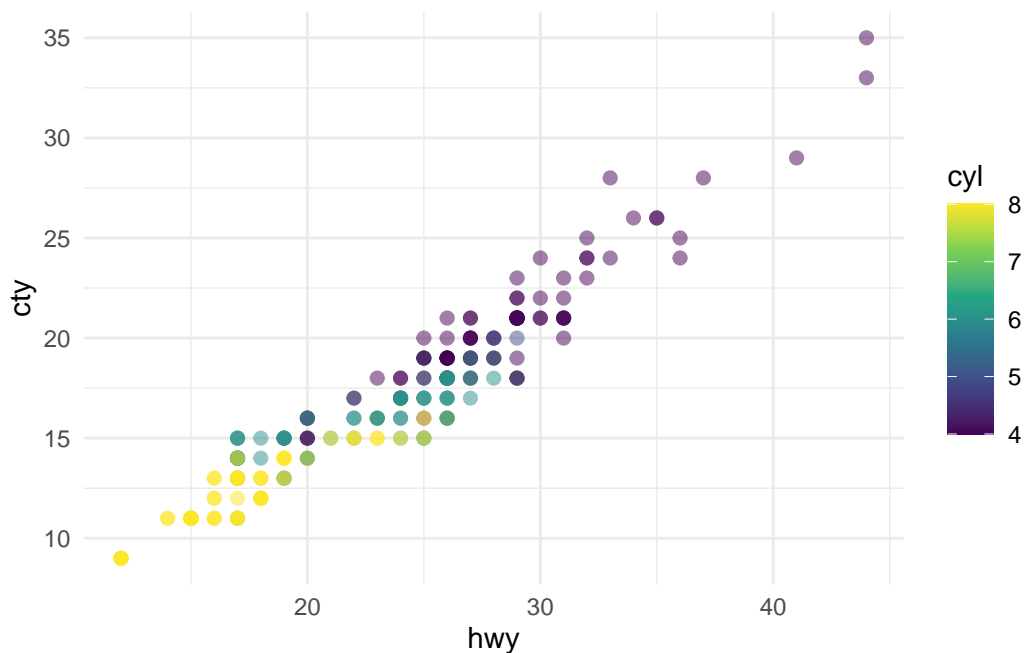


Figure 4.7: Gráfico de correlación lineal positiva entre el kilometraje en ciudad y en carretera de diferentes modelos de coches.

#### 💡 Automated figure numbering

It is important that *the fragment identifier* we choose for the code that generates one or more figures *begins with the prefix fig-*.

In this way, we ensure that Quarto automatically assigns a number to the generated figure and that we can create cross-references (internal links) to that figure in our document.

As we will see later, other output types such as tables also need to be assigned a specific pattern in their fragment identifier so that they are automatically numbered and can be referenced within the document.

Figure management in Quarto is quite sophisticated, to the point that you can easily organize several subfigures with their respective individual descriptions, as shown in the following example using some additional options.

```
```{r}
#| label: fig-mpg-subplot
#| fig-cap: "City and road mileage of 38 popular car models."
#|
#| fig-subcap:
#|   - "Color by num. of cylinders."
#|   - "Color by motor displacement."
#| layout-ncol: 1

ggplot(mpg, aes(x = hwy, y = cty, color = cyl)) +
  geom_point(alpha = 0.5, size = 2) +
  scale_color_viridis_c() +
  theme_minimal()

ggplot(mpg, aes(x = hwy, y = cty, color = displ)) +
  geom_point(alpha = 0.5, size = 2) +
  scale_color_viridis_c(option = "E") +
  theme_minimal()
```
```

Some common options for chunks that generate figures are:

- **fig-width**: Width of the figure.
- **fig-height**: Height of the figure.
- **fig-cap**: String in quotes to be inserted as a caption.
- **fig-alt**: Alternative text message that fills the **alt** attribute of the HTML image (for example, to improve the accessibility of the content).
- **fig-dpi**: Resolution setting of the figure (in dots per inch).

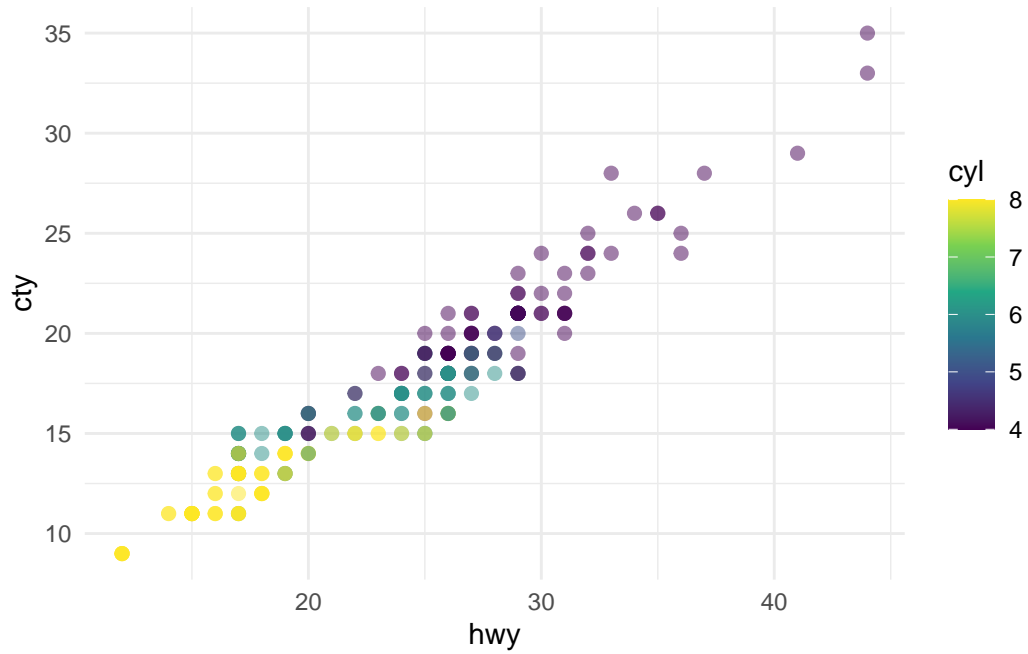
The tutorial on executable code snippets in the official documentation presents more information and examples on how to use this powerful Quarto feature.

## 4.5 Author toolkit

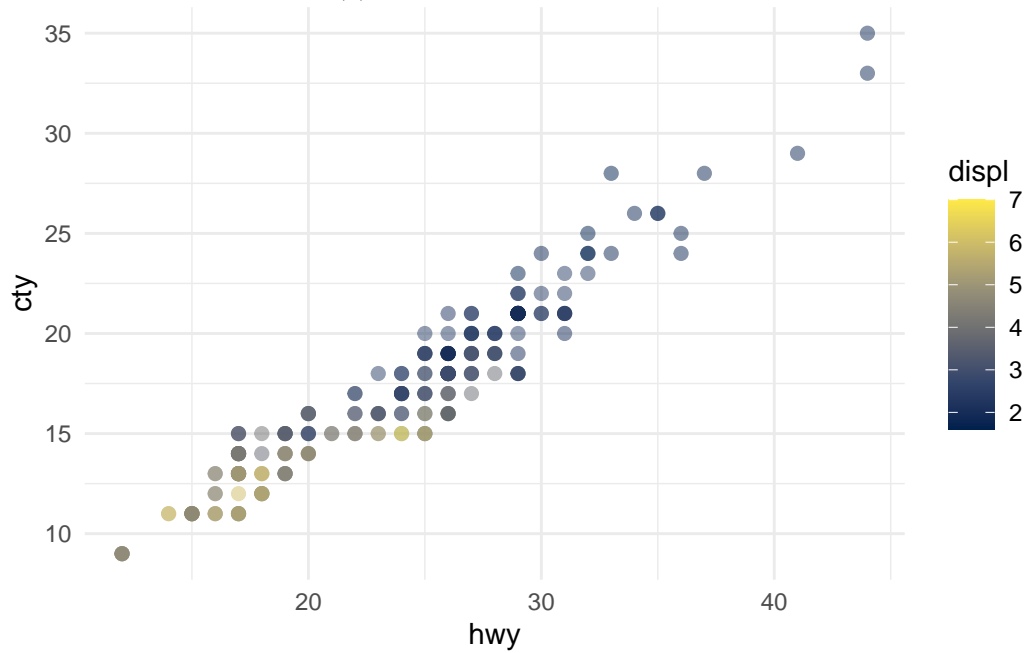
In addition to the ability to integrate executable code and its results into our scientific papers, Quarto includes a number of resources and tools to provide a complete and efficient authoring experience.

### 4.5.1 Document sections

As we saw in the Section 4.3.3 example, there are two HTML document configuration options that allow us to number sections and incorporate an automatically generated table of contents at the top right of our document.



(a) Color by num. of cylinders.



(b) Color by motor displacement.

Figure 4.8: City and road mileage of 38 popular car models.

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

An important feature for creating scientific documentation is being able to include **cross-references**, that is, links that take us to other sections of the document. In Quarto this is achieved by following a simple procedure in two steps:

1. We add a *unique tag* to identify the section with the syntax:

```
## Section header {#sec-tag}
```

2. We reference the tag we created for that section in another part of the text, so that Quarto automatically creates the link (cross-reference) to that section:

```
In the text we add a reference to the @sec-tag.
```

An example of this type of automatically created cross-reference can be seen at the beginning of this section. On the other hand, if we want a section of the document to be excluded from the numbering scheme of the rest of the sections, we use the special tag in the title of that section:

```
## Unnumbered section {.unnumbered}
```

There are several additional options that control the way and style in which sections are created and numbered. Some of them are:

- **anchor-sections**: Causes an anchor link to be displayed (to link directly to that section in another document) when the mouse hovers over a section title.
- **toc-depth**: Specifies how many levels deep the section numbering appears in the table of contents. By default, 3 levels are displayed.
- **toc-location**: `body | left | right | left-body | right-body`: Controls the location where the table of contents appears in the document.
- **toc-title**: String with the title of the table of contents.
- **toc-expand**: Indicates whether all sections in the table of contents should be expanded or collapsed so the user can click on the ones they want to expand.
- **number-depth**: Determines the maximum depth to which sections in the document are numbered (note, this should be in line with the value assigned to the **toc-depth** option).
- **number-offset**: Allows you to set the number at which sections are numbered. If we want the document to start numbering the highest level section as “4” then we use **number-offset**: 3. If we want the document to start at a level 2 section numbered “1.5” we must specify **number-offset**: [1,4]. Setting a value for this option means that **number-sections**: true is automatically set.



### 4.5.2 Equations

Another essential aspect of scientific documents is the appearance of mathematical symbols, formulas, and equations. There are several HTML libraries that allow properly formatted equations to be displayed on the screen. For its part, LaTeX, due to its origins, has always included powerful and versatile tools to handle this type of content, so support is guaranteed for PDF documents.

In general, the syntax used to write equations is very similar to that used in LaTeX.

- Tutorial on mathematical expressions in LaTeX.
- Summary of mathematical syntax in LaTeX.

There are two ways to display equations in our content, also following a similar philosophy to that of LaTeX documents:

- Equations in line with the text: to display the equation within a line or paragraph, at the same height as the rest of the text.
- Equations in *display* mode: the equation is displayed in a separate space, between two paragraphs of text and with a certain margin of space at the top and bottom.

Example of an inline equation: `$F = m \cdot a$`

Result: example of an inline equation:  $F = m \cdot a$ .

Example of an equation in display mode:

`$$E = mc^2$$`

Which produces the following result (see below how to add the numbering):

$$E = mc^2 \tag{4.1}$$

If we also want to number our equations, we must remember to use the unique identifier tag pattern `eq-tag` to identify it and then be able to insert internal references to said equation in the text.

`$$ E = mc^2 $$ {#eq-energy}`

As a result, we can insert a cross-link to Equation 4.1.

### 4.5.3 Tables

Tables are another relevant content that we can format in different ways in the documents generated with Quarto.

- Introduction to creating tables in Quarto.

In this case, the visual editor can greatly simplify this task for us. It is advisable to try it to see the difference, since it is a very intuitive tool. However, following the same line as the rest of the workshop, here we will describe the details to create this content directly in the Markdown code of the file.

The most direct way to create a table in Markdown is to compose a **pipe** table, so called because its syntax is based on the `|` operator on the command line. Let's see an example.

```
Default	Left	Right	Center
1	2	3	4
22	23	24	25
4	3	2	1
```

The result of including the previous code in our document is:

| Default | Left | Right | Center |
|---------|------|-------|--------|
| 1       | 2    | 3     | 4      |
| 22      | 23   | 24    | 25     |
| 4       | 3    | 2     | 1      |

We can see how the key to controlling the horizontal alignment of the table content is to properly place the `:` symbol on the line just below the title line, which separates it from the body of the table. If we do not want to include a title, it is mandatory to include the first line, but we can leave the cells blank.

Below the table we can insert the expression `: Table Caption` to include a descriptive message. It is also possible to directly use some style elements included in the classes of Bootstrap, the web style framework that Quarto uses to compose pages (we have seen before how to use the document option `theme: cosmo` to use the Cosmo theme of Bootstrap). There are different effects, and one of the most frequent is to color the background of the rows in gray alternately as well as highlight the row on which the mouse arrow is placed. These two effects are `.striped` and `.hover`, respectively.

```
Default	Left	Right	Center
1	2	3	4
22	23	24	25
4	3	2	1

: This is the table caption. {.striped .hover}
```

Table 4.2: This is the table caption.

| Default | Left | Right | Center |
|---------|------|-------|--------|
| 1       | 2    | 3     | 4      |
| 22      | 23   | 24    | 25     |
| 4       | 3    | 2     | 1      |

Finally, similar to what we do to internally reference equations and figures in our document, we can also label tables using the pattern `#tbl-label` to reference it as `@tbl-label` which is formatted like this: Table 4.3.

| Default | Left | Right | Center |
|---------|------|-------|--------|
| 1       | 2    | 3     | 4      |
| 22      | 23   | 24    | 25     |
| 4       | 3    | 2     | 1      |

: This is the table caption. `{#tbl-label .striped .hover}`

Table 4.3: This is the table caption.

| Default | Left | Right | Center |
|---------|------|-------|--------|
| 1       | 2    | 3     | 4      |
| 22      | 23   | 24    | 25     |
| 4       | 3    | 2     | 1      |

The same tag pattern should be used in the code chunk identification option `#| label: tbl-label` if we later want to reference the table generated by that code chunk with a cross-reference.

More details can be found on creating subtables, changing caption location, as well as creating grid tables that use a different syntax and allow arbitrary block elements to be included in each cell (multiple paragraphs, code blocks, unnumbered or numbered lists, etc.).

| Fruit   | Price  | Benefits           |
|---------|--------|--------------------|
| Bananas | \$1.34 | - wrapping         |
|         |        | - brilliant colour |
| Oranges | \$2.10 | - rich in vitam. C |
|         |        | - tasty            |

: Sample grid table.

Table 4.4: Sample *grid* table.

| Fruit   | Price  | Benefits  |
|---------|--------|---|
| Bananas | \$1.34 | • wrapping  |
| Oranges | \$2.10 | • brilliant colour<br>• rich in Vitam. C<br>• tasty |

#### 4.5.4 Callouts

It is possible to include callout blocks, to highlight practical notes, warnings, or tips of special interest. In addition, a title is often given to the callout to make it even more informative.

```

::: {.callout-note}
## Callout title

There are five different types of callouts:
`note`, `tip`, `warning`, `caution`, and `important`.
:::

```

**i** Callout title

There are five different types of callouts: **note**, **tip**, **warning**, **caution**, and **important**.

- Introduction to using callouts in Quarto.

#### 4.5.5 Bibliographic references

The management of bibliographic references in Quarto is done by encoding the information in BibTeX format. This allows you to use any of the bibliographic citation formats supported by this package, or to include a CLS file that defines a standard format (APA, Chicago, IEEE, etc.).

For example, the document options

```

---
title: "My Document"
bibliography: references.bib
csl: nature.csl
---

```

indicate a **references.bib** file where we can store the information about bibliographic references (which we can obtain from Google Scholar, Zotero or other tools and services on the Internet), as well as a citation style file **nature.csl** (style defined by the Nature publisher).

- CLS repository with citation styles.
- Zotero repository with citation styles.

Depending on the style and format of the citation, we can use one or another syntax to indicate the author and the year in parentheses, the author outside the parentheses, page numbers, chapters, etc.

- Quarto citation syntax reference table.

Finally, the ordered list of bibliographic references (according to the citation style criteria we have selected) must appear at the end of the document. To achieve this in an HTML document, we must include a special code, which is normally placed in a separate, unnumbered section, as shown in Figure 4.9.

```
1      # References {.unnumbered}
2
3      ::: {#refs}
4      :::
```

Figure 4.9: Syntax to display bibliographical references at the end of the document.

When the generated output is in PDF format and the BibLaTeX or natbib reference management engines are used, then the list of references always appears at the end of the document and the previous tag is ignored. Finally, in the rare case that we do not want to include any bibliographical references in our document, we can include in the header metadata the option `suppress-bibliography: true`.

### 4.5.6 General document style

So far, the example document we have shown as well as these notes always use a style format or `theme` from the Bootstrap web development environment, called `cosmo`. However, there is a wide list of alternative themes to modify the general style of our document (color scheme, typography and font size, organization of content, appearance of links, etc.). The Quarto project is responsible for regularly combining the most popular style themes so that they are available as an option for the document.

In this theme directory on GitHub you can check an updated list of the possible values that we can assign to the `theme` option in the header of the document. It is useful to experiment with various options to find the one that best suits the type of document generated, its content and the audience it is intended for.

An online demo of many of the available themes can be accessed on the website <https://bootswatch.com/>.

**Part II**

**Quarto books**

## 5 Books

Now that we know all the basic elements for creating individual literary programming documents with Quarto, the natural evolution is to ask ourselves if we can manage ordered collections of documents in a single project to, for example, create a book, a laboratory notebook, or an experiment log.

Book-type projects in Quarto are the answer to these needs, allowing us to group and organize several individual documents in a single website or a single volume (PDF) for publication.

### 5.1 Creating a book project

The first step is to create a book project, using the IDE of our choice, for example, RStudio. The Figure 5.1 shows the RStudio interface for creating a new book project with Quarto. As usual, we select a name for the directory that stores the project as shown in the Figure 5.2.

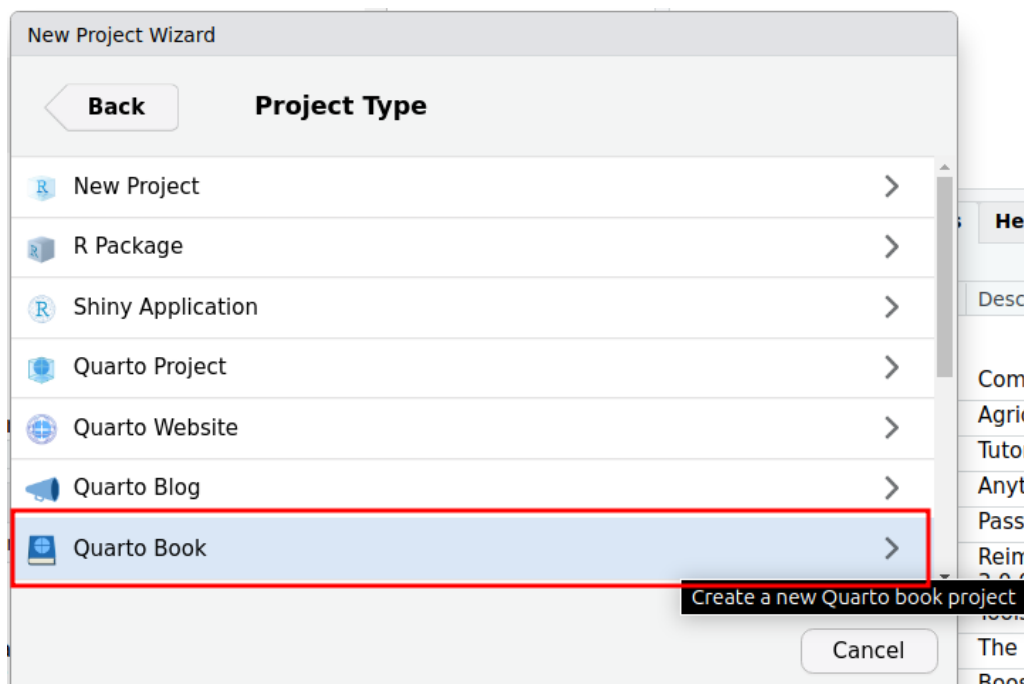


Figure 5.1: RStudio interface for creating a new Quarto project of type *book*.

- Home page for the Quarto tutorial on books.

Figure 5.2: Dialog to indicate the name of the directory that stores the *book* project and some basic configuration options, such as the execution engine for code blocks embedded in documents (highlighted in red).

## 5.2 Configuration options

Unlike individual documents, in projects such as *book* type projects that group several documents (*.qmd* files) into a single collection, we can specify global configuration options for the entire project in a separate *\_quarto.yml* file, which must be present in the root directory of our project.

By default, the configuration options presented by this file in a newly created project of this type are the following.

```
project:
  type: book

book:
  title: "First book"
  author: "Norah Jones"
  date: "20/12/2024"
  chapters:
    - index.qmd
    - intro.qmd
    - summary.qmd
    - references.qmd

bibliography: references.bib
```



```
format:
  html:
    theme: cosmo
  pdf:
    documentclass: scrreprt

editor: visual
```

The new elements in this list of options are:

- **project -> type: book:** Allows you to indicate the type of project you want to set up so that Quarto adjusts its behavior and can group a collection of documents. It activates the interpretation of options related to this type of project.
- **book:** New category of options that indicates the specific configuration applicable to all documents in this project, as well as global options. For instance, the title, author, and date will be displayed on a special cover page.
- **chapters:** New subcategory of options that allows you to insert a list of file names that contain the different sections or chapters of your book.

We must keep in mind that the sections or chapters will be processed in exactly the same order in which they appear in this list, so it is important to pay attention to this order.

### 5.3 Home page (*preface*)

In addition to the new `_quarto.yml` file, another file called `index.qmd` is generated which contains the material that will be presented as the cover of the book or collection of documents.

A general file is also created in other types of projects, for example for the home page of a website generated with Quarto. It is in this cover page that the general configuration information is contained (title, authors, date, etc.).

The content of this file is the same as in any other document, following the same Quarto syntax rules for `.qmd` files that we have already seen.

It should also be noted that the title of this cover page is not usually numbered and, therefore, it is common for it to be configured as:

```
# Preface {.unnumbered}
```

### 5.4 Writing tools

All the writing tools we have already seen in the Chapter 4 can be used in the case of chapters in a book. It is important to note that links to sections also work from documents in other sections or chapters of the book, even if they are in a different file.

### 5.4.1 Book structure

In addition to standard chapters, we can also organize the book's content into parts (which group together related chapters), as well as appendices, presented after the main content of the book to provide additional material.

Let's look at an example of a book configuration that includes several parts and that we can integrate into the rest of the project configuration, within the `_quarto.yml` file.

```
chapters:
  - index.qmd
  - preface.qmd
  - part: dice.qmd
    chapters:
      - basics.qmd
      - packages.qmd
  - part: cards.qmd
    chapters:
      - objects.qmd
      - notation.qmd
      - modifying.qmd
      - environments.qmd
```

It is important to note that in this case the `part` option can accept either a file with a `.qmd` extension (as in the example), or a quoted string that simply indicates a title for the part.

Now we present an example for appendices.

```
book:
  title: "mybook"
  author: "Jane Doe"
  date: "5/9/2021"
  chapters:
    - index.qmd
    - intro.qmd
    - summary.qmd
    - references.qmd
  appendices:
    - tools.qmd
    - resources.qmd
```

It is important to note that these configuration options generate the appropriate output (parts and appendices) both in the case of HTML websites and when generating a PDF document, following in the second case the standard LaTeX syntax to indicate the structure of the document.

- Example of a well-known book by H. Wickham created in Quarto and publicly accessible, which is organized in parts that group chapters.
- Additional configuration options for the structure of a book in Quarto.

## 5.5 Managing references

As mentioned above, internal references (to figures, tables, equations, and other sections of the document, among other items) work exactly the same as in individual documents (see Section 4.5.1), with the added advantage that in a book that combines different chapters, the numbering of all items is updated to reflect the chapter number as a prefix to the item number (e.g., “Figure 1.2” for the second figure in chapter 1).

It is important to note that for automatic item numbering to work properly, it is important that items begin with the appropriate prefix (`#sec-` for sections, `#fig-` for figures, etc.).

## 5.6 Project preview

To preview on our local machine the project we have created using RStudio, we must select the *Build* tab in the upper right panel and press the *Render Book* button to generate all the output formats that are configured in the `_quarto.yml` file, as shown in Figure 5.3. There is also the option to select only one of these output formats by carefully clicking on the small arrow next to the “Render Book” button, to display a list of output format options and select one of them, as shown in Figure 5.4

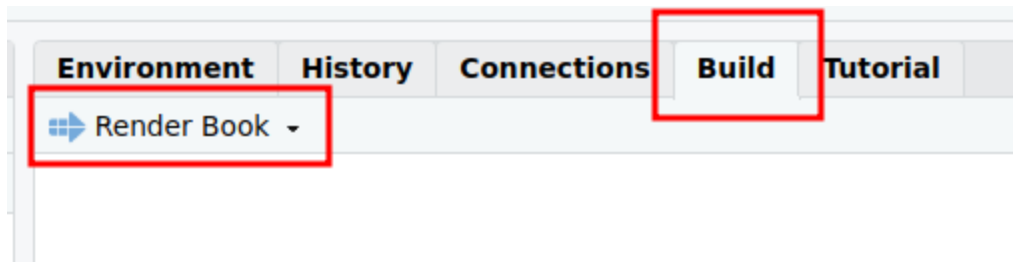


Figure 5.3: Button to launch the book preview process in RStudio.

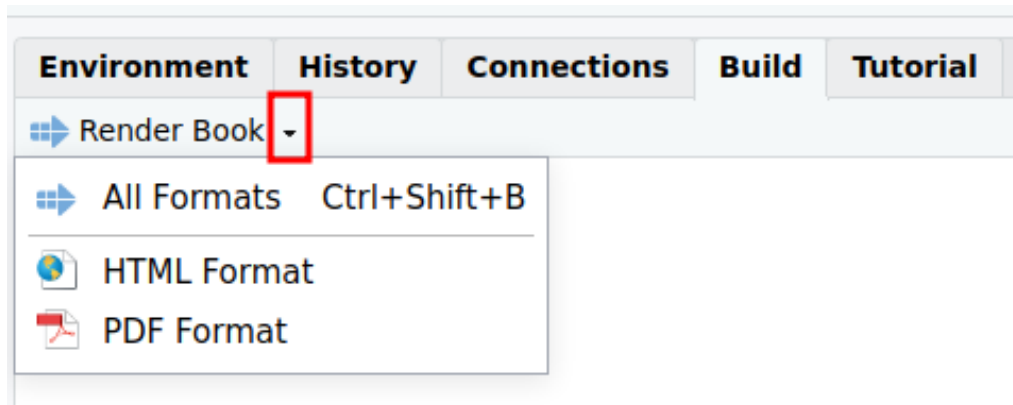


Figure 5.4: List of configuration options that appear when clicking the arrow next to the *Render Book* button

## 5.7 Publication options

There are several [publishing options] to make our book or collection of documents available to other users, including:

- Quarto Pub.
- GitHub Pages, very convenient if we want an integrated solution for version management of our project's source code.
- Netlify, a professional web publishing platform that allows more configuration and setup options.

It is also possible to use other services to publish documents, books, and websites with Quarto, including GitLab (an alternative to GitHub), although they will not be as automated and integrated with the tools offered by Quarto as the previous options.

## 5.8 Customisation and templates

As we have explained for individual documents, analogous configuration options can be used to customize the style and design theme that we can apply to our book or collection of documents, both in the HTML website version and in the PDF output version.

- Guide to customizing the style of books in Quarto.
- Example gallery of some styles to customize the appearance of Quarto documents.

## **6 Creating a field guide**

### **6.1 Templates**

### **6.2 Project management**

### **6.3 Publishing**

# **Part III**

## **Publications**

## 7 Scientific publications

In this chapter we see another important characteristic of Quarto, such as the ability to create document formats by extending the formats basic output file like `html`, `pdf` or `docx`. One of the uses main aspects of this capacity of Quarto is to produce documents that meet the requirements requested by the publishers scientific for publication of research articles in journals that they edit.

It is worth noting that Quarto, as a minimum, always attempts to produce output in HTML and PDF formats, so templates should contemplate these two scenarios. Additionally, the use of *span* and *div* environments in different sections lets us apply the requested design and style tools for each editorial template for both HTML (with CSS) and PDF (with environments and LaTeX macros). Another interesting aspect is that these templates aim to unify the coding of information about authors and their affiliations, so that it is written only once and it can be reused in different output formats. Finally, Quarto can also manage styling requirements of bibliographic citations imposed by scientific publishers and magazines in their publications.

- List of editorial formats supported in Quarto.
- Alternative list of editorial formats supported in Quarto.

Sometimes it will be necessary to add additional code to make fine adjustments to the document templates, so that we can adhere to the requirements imposed by the publisher for that publication. The following tutorial offers information on the different points at which we can insert additional code to perform these fine adjustments.

- Templates adjustment.

Finally, in case there is no template for the format we need, it is always possible to create our own article format.

- Creating our own article template.

The initial step is to create a new project directory and run within it the command to download the standard template (not customized yet) for creating post formats scientists in Quarto. Suppose the project directory is called `jourA`. Be careful, because in this example the symbol `$` should not be entered in the terminal; we are just using it to differentiate the user input from the response it returns the terminal when executing the command.

Several files are created, including:

- `_extensions/jourA/_extension.yml`, which defines the output formats available in this article template. For example, if we define an output format `html` and another `pdf` in this template, these will be available in the Quarto document as `jourA-html` and `jourA-pdf` when a Quarto document uses our template.
- `template.pdf`, which is the example document that is generated to demonstrate to the user the structure and options available and start working on it.

---

**Listing 7.1** Terminal

---

```
$ cd jourA
$ quarto create extension journal
? Extension Name > jourA

Creating extension at /home/jfelipe/quarto/dev/jourA:
- Created README.md
- Created template.qmd
- Created _extensions/jourA/jourA.lua
- Created _extensions/jourA/styles.css
- Created _extensions/jourA/_extension.yml
- Created _extensions/jourA/header.tex
- Created bibliography.bib
```

---

## 7.1 The `keep-tex` option: `true`

If you review the examples offered in the publication template creation guide, you will see as in the output format options of the `template.qmd` file, within the PDF output the `keep-tex: true` option is usually included. This option forces the file not to be deleted LaTeX (with `.tex` extension) that is created as a previous step to compile the final document in PDF. The reason is to allow the user to modify the file if necessary. LaTeX directly and compile it manually.

However, remember that if you press the *Render* button again for the PDF format in RStudio, or run `quarto render --to pdf` on the command line, said file with `.tex` extension **it is overwritten** and we would lose the changes we have made. Consequently, it is better to copy it to another location before making manual adjustments or create a new one branch (if we use version control) to adjust the file in it without the risk to overwrite the changes made.

## 7.2 Figures and graphs for publication

An important advantage of using Quarto to create our articles is being able to integrate the results of the execution of our code (graphs, tables, results of evaluation of models and algorithms, etc.) directly in our scientific documents. This greatly mitigates the drawbacks already mentioned at the beginning of the workshop to keep all elements updated, allowing us to ensure that we are using the correct version of the code on the appropriate data.

- **Publication Perfect**: a tutorial with open online materials on how to improve the elements of our publications and articles, created by the Harvard Chan Bioinformatics Core group.

Let's remember some of the packages in R that allow us to create graphs and tables now prepared for publication:

- **Hmisc**: includes many functions for description of data and creation of graphs and tables summarizing data and models, ready to publish.



- Examples reproducible with `Hmisc`. It is indicated that the examples are for R Markdown, but they actually work also for Quarto (since that the same `knitr` engine is used by default to process and execute the blocks of code.)
- `summarytools`: offers many features and tools to present summaries of data tables and data frames, both in table as in graphs for EDA.
- Examples of use of `summarytools`.
- `ggpubr`: package to help customize graphics created with `ggplot2`, so that they are ready for inclusion in publications scientific.
- Collection of tutorials and examples for creating graphs with `ggpubr`.
- List and examples of packages to create tables in publications.

## 7.3 Facilitate citation of articles

In a context of the academic and research world in which the volume of works and published articles has grown exponentially in digital media, it is very important to facilitate the work of other researchers as much as possible when they want to cite our publications. On the other hand, there are many reference services of citations and databases that help researchers find materials that they should review.

We can include metadata in our document to facilitate full automation or partial of this process of indexing publications and citations of our work.


The following tutorial shows several examples of metadata fields to facilitate the citations that we can include in the header of the document, including information about the journal or publication that contains the article (such as the DOI), as well as specific formats compatible with scientific bibliography indexing engines such as Google Scholar.

- Guide for creating citable articles in Quarto.

By default, when we include this information in the header of the Quarto document it must create an appendix with the citation information in plain text and in BibTeX.

## 7.4 Example of using scientific article templates

### 7.4.1 Elsevier Magazine Template

 Prerequisite: have LaTeX installed

This section shows how to generate a *draft* of an article to send to a scientific journal from the Elsevier publishing house. For this process to work, remember You must previously have a LaTeX distribution installed on your computer (such as TeX Live) or install the minimal TinyTeX distribution in RStudio.

In this case, the steps to follow are quite simple:

1. We create a new directory to save our project, for example, a folder with the name `example-elsevier`.
2. In the terminal (for example in RStudio, tab in the bottom left panel), We enter the newly created folder and execute a command to download the template and start using it:

---

**Listing 7.2** Terminal
 

---

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

---

3. We create a new project in RStudio on the already existing directory in which we have downloaded the template files.
4. We open the file `example-elsevier.qmd` and press the *Render* button to generate the the output in PDF.

If we had previously created our project with a generic Quarto document, it will not we have to start from scratch. Just open a terminal inside the project directory and execute:

---

**Listing 7.3** Terminal
 

---

```
quarto add quarto-journals/elsevier
```

---

Finally, if we want to generate the *draft* article in PDF format from the command line, we execute:

---

**Listing 7.4** Terminal
 

---

```
quarto render article.qmd --to elsevier-pdf
```

---

Take a good look at the output format option specified in the `.qmd` file to generate the *draft* article:

---

**Listing 7.5** example-elsevier.qmd

---

```
format:
  elsevier-pdf:
    keep-tex: true
```

---

## 8 FAIR Principles

### 8.1 Overview

For years, a movement has developed within the scientific community unstoppable to promote access to all information related to jobs, experiments and scientific publications, so that their validation is facilitated and reproduction/replication by other interested researchers or experts.

This movement has especially concentrated, in its initial phase, on guaranteeing at least access to the raw materials necessary to develop many of these projects: the data. The fundamental principles that must be met so that data can be indexed and reused as much as possible, they are called **FAIR Principles** and are included, among other sources, in the European Commission guidance documents for researchers participating in projects funded by said organization. These FAIR principles were initially established by Wilkinson et al. (2016) and they are:

- **Findable:** Data and metadata (data that describes the data, such as its format, content, meaning, link with other data, etc.) must receive a globally unique and persistent identifier that allows it to be located directly. The most used standard today is the DOI system (ISO 26324). The metadata must include clearly and explicitly identify the data they describe, and both data and metadata must be registered or indexed in sources that allow their search and retrieval.
- **Accessible:** Data and metadata must be obtainable through a standard and open communication protocol. Metadata must continue to remain accessible even when the data is no longer available.
- **Interoperable:** Data and metadata must use standard and open knowledge representation formats, vocabularies that follow FAIR principles and must include references to other data and metadata with which they are related.
- **Reusable:** The data and metadata are described in a rich and precise manner, with multiple relevant attributes that facilitate their use by other users.

It is important to emphasize the importance of publishing openly and following good reviewable and reproducible/replicable research practices. For example, The National Commission for the Evaluation of Research Activity (CNEAI) has published in 2023-2024 new criteria for evaluation of publications valid for be able to be evaluated in the granting of a six-year period of research. Likewise, in all the recent calls financed by the different ministries of the Government of Spain It is required that research results, especially all publications, *datasets*, software and procedures are publicly accessible and comply with FAIR principles.

However, there is still some way to go. A recent study (Kumar et al., 2024) analyzed the degree of compliance with the FAIR principles (the so-called FAIRness) of the results research published by recently funded multi-stakeholder projects within the European H2020 framework program and related to the agri-food sector. As main conclusions, less than 10% of the projects analyzed managed

to comply with the FAIR principles, although these principles were fulfilled to a greater extent in the articles of research published in journals and conferences, as well as in books.

In general, another conclusion is that the European agri-food and rural sector is becoming increasingly dependent on data and that the application of the principles FAIR contributes to improving decision-making and better exploiting innovation results derived from these projects. However, it is also noted that the research community for the development of the agri-food and rural sector still has limited experience in the application of these principles.

## 8.2 Publication of source code and technical documentation

- Platforms like GitHub and GitLab make project management and publishing much easier software within research initiatives and projects, as well as publication and maintenance of digital technical documentation centers on tools, procedures and good practices related to the activity of said initiatives.
- Examples of the SoilWise project, funded within the HE program of the European Community.
  - SoilWise digital co-creation space on GitHub. It has been achieved creating an organization (free of charge), so that repositories can be created on this platform, grouped under the umbrella of the project, for different purposes: lists of software of interest, technical documentation, user manuals, architecture of the proposed platform, etc.
  - Documentation Center. Here we use the MkDocs solution, which works with the Python language (Quarto has great advantages in this aspect).
  - Repository with data collection tools, documented in the page describing the data collection process in the project infrastructure.
- Open repositories of data and research material such as Zenodo and Figshare can help publicize and cite research material. For example, Zendo can issue a persistent identifier (DOI) for a software repository on GitHub pointing to a particular version of the software that has been released and tagged from that project on GitHub. This allows it to be included in a scientific publication to know with certainty what exact version of the code was software has been used to perform the work reported in that publication.

As an example, Figure 8.1 shows the Zenodo page corresponding to the PyMPDATA software, which points to the original software repository hosted on GitHub, shown in turn in the Figure 8.2. we can see metadata cross-references that maintain connection consistency in both directions.

- The project page in Zenodo points to a specific version of the GitHub repository, and maintains a list of all the previously referenced versions of the same project. Each new version receives a different DOI, to differentiate them univocally.
- The repository description page on GitHub displays, among other tags, the DOI for that version in particular, the DOI to the scientific article published in JOSS explaining this software, as well as other labels for attribution of project financing sources.

zenodo Search records... Communities My dashboard Log in Sign up

open-atmos

Published October 24, 2024 | Version v1.2.0

## PyMPDATA

Arabas, Sylwester<sup>1</sup>; Banaśkiewicz, Jakub<sup>2</sup>; Bartman, Piotr<sup>2</sup>; Derlatka, Kacper<sup>2</sup>; Drenda, Szymon<sup>2</sup>; Manna, Maciej<sup>2</sup>; Olesik, Michael<sup>2</sup>; Magnuszewski, Paweł<sup>3</sup>; Manna, Maciej<sup>2</sup>; Rozwoda, Paweł<sup>2</sup>; Sadowski, Michał<sup>2</sup>

Numba-accelerated Pythonic implementation of MPDATA with examples in Python, Julia and Matlab

Files

- open-atmos/PyMPDATA-v1.2.0.zip
- open-atmos/PyMPDATA-v1.2.0.zip
- open-atmos-PyMPDATA-8ff7ce
  - .appveyor.yml
  - binder

935 Bytes

1K VIEWS 124 DOWNLOADS Show more details

Versions

| Version        | DOI                     | Date         |
|----------------|-------------------------|--------------|
| Version v1.2.0 | 10.5281/zenodo.13988934 | Oct 24, 2024 |
| Version v1.1.6 | 10.5281/zenodo.13823030 | Sep 21, 2024 |
| Version v1.1.5 | 10.5281/zenodo.13625733 | Sep 1, 2024  |
| Version v1.1.4 | 10.5281/zenodo.13623679 | Aug 31, 2024 |
| Version v1.1.3 |                         | Aug 28, 2024 |

Figure 8.1: PyMPDATA project page (v1.2.0) at Zenodo

README License

PyMPDATA

Python 3 LLVM Numba Linux macOS Windows Jupyter Maintained? yes Open Hub 7.38K Lines

JOSS 10.21105/joss.03896 DOI 10.5281/zenodo.13988934

EU Funding by FNP PL Funding by NCN License GPL v3

tests+pypi passing build passing codecov 92%

pypi package 1.1.6 docs pdoc.dev

PyMPDATA is a high-performance Numba-accelerated Pythonic implementation of the MPDATA algorithm of Smolarkiewicz et al. used in geophysical fluid dynamics and beyond for numerically solving generalised convection-diffusion PDEs in 1D, 2D and 3D structured meshes with coordinate transformations.

Contributors 14

Deployments 118

github-pages 2 hours ago

+ 117 deployments

Languages

Python 96.2% TeX 3.7% Shell 0.1%

Figure 8.2: PyMPDATA (v1.2.0) project page on GitHub

### 8.3 Dataset publication

- Zenodo is, probably along with Figshare, one of the most well-known and widely used open data repositories in research, especially in Europe, being a project integrated into OpenAIRE, a non-profit organization created by the European Union to promote open science.
- Scientific publications in open access (*open access*).
- Important to link the DOI of a dataset in an article, so that they can be traced (in reverse) the DOIs of the publications that use said *dataset*.

### 8.4 Reference management and open publication

In addition to the assignment of a DOI to articles, data sets, software and other elements of research work, there are some additional tools that facilitate the identification of the works and the attribution of their authorship (many of them with support explicit in Quarto)

- The ORCID is a free, unique and persistent identifier for individuals who engage in research, innovation and academic activities. Allows you to identify quickly the identity of an author in a publication or scientific work or the person in charge to publish and maintain a resource (*dataset*, software repository, etc.). Furthermore, the ORCID allows you to generate a list of all the contributions made by the identified individual.
- There is an extensive list of *preprints* files that allow the publication of preliminary works that have not yet gone through a review process by pairs for publication in a magazine. More and more publishers accept (and even encourage) publication of these documents to quickly record research progress, due to the high times required by the review and publication process in many prestigious journals.
- PLOS is a non-profit Open Access publisher, which edits and publishes several high-impact, wide-spread digital magazines in many areas including, transformation and sustainability.

A growing number of publishers are also adopting open publishing principles (Open Access), although usually impacting a significant cost on the authors or the institutions to which they are affiliates to cover publication costs.

- Guide to creating citable articles with Quarto.

## 9 Additional resources

### 9.1 Quarto

- Getting started: <https://quarto.org/docs/get-started/>.
- Complete guide (online): <https://quarto.org/docs/guide/>.
- Reference of functions and options (online): <https://quarto.org/docs/reference/>.
- Gallery of example projects: <https://quarto.org/docs/gallery/>.

### 9.2 FAIR principles and open science

- Article on FAIR principles: <https://www.nature.com/articles/sdata201618>.
- Mandate on scientific data management in the Horizon Europe (EC) Programme: <https://www.openaire.eu/how-to-comply-with-horizon-europe-mandate-for-rdm>.
  - All HE projects must publish at the beginning of the work plan a Data Management Plan (DMP), explaining in great detail how they will be obtained, processed, analyze and manage the data used and generated in the project activities.
    - \* DMP TOOL: Online tool to create DMPs following a structured procedure.
  - “*How to make your data FAIR*”: <https://www.openaire.eu/how-to-make-your-data-fair>.
  - Data Management Plan (University) of Cambridge.
- ROpenSci Community: <https://ropensci.org/es/>.
- ROpenSpain Community: <https://ropenspains.es/>.



# References

- Barba, L. A. (2018). Terminologies for reproducible research. *arXiv Preprint arXiv:1802.03311*.
- Begley, C., & Ellis, L. (2012). *Drug development: Raise standards for preclinical cancer research. Nature.[Online]. 483 (7391)*.
- Brainard, J., You, J., et al. (2018). What a massive database of retracted papers reveals about science publishing’s “death penalty.” *Science*, 25(1), 1–5.
- Burman, L. E., Reed, W. R., & Alm, J. (2010). A call for replication studies. *Public Finance Review*, 38(6), 787–793.
- Ioannidis, J. P. (2005). Why most published research findings are false. *PLoS Medicine*, 2(8), e124.
- Knuth, D. E. (1984). Literate programming. *Comput. J.*, 27(2), 97–111. <https://doi.org/10.1093/comjnl/27.2.97>
- Kumar, P., Hendriks, T., Panoutsopoulos, H., & Brewster, C. (2024). Investigating FAIR data principles compliance in horizon 2020 funded agri-food and rural development multi-actor projects. *Agricultural Systems*, 214, 103822. <https://doi.org/10.1016/j.agsy.2023.103822>
- Leek, J. T., & Peng, R. D. (2015). Reproducible research can still be wrong: Adopting a prevention approach. *Proceedings of the National Academy of Sciences*, 112(6), 1645–1646.
- Peng, R. D. (2011). Reproducible research in computational science. *Science*, 334(6060), 1226–1227.
- Wicherts, J. M., Borsboom, D., Kats, J., & Molenaar, D. (2006). The poor availability of psychological research data for reanalysis. *American Psychologist*, 61(7), 726.
- Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., Silva Santos, L. B. da, Bourne, P. E., et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(1), 1–9. <https://doi.org/10.1038/sdata.2016.18>

# **A Code reference**

## **A.1 Quarto statements**

## **A.2 R statements**

## **B Integrated Development Environments for Quarto**

### **B.1 R Studio**

### **B.2 Visual Studio**

### **B.3 Positron**

## **C Useful R packages**

### **C.1 Ecology**

### **C.2 Data visualisation**

### **C.3 Data processing**

#### **C.3.1 Tidyverse**

#### **C.3.2 Alternatives to the Tidyverse**

#### **C.3.3 Pipelines**

### **C.4 Spatial data**

#### **C.4.1 sf (Simple Features)**

#### **C.4.2 terra**

### **C.5 Time series**

#### **C.5.1 Tidyverts**

### **C.6 Data visualisation**

#### **C.6.1 ggplot2**

### **C.7 Data analysis and Machine Learning**

#### **C.7.1 Tidymodels**

#### **C.7.2 mlr3**

## **D Producing PDF documents**

**D.1 PDF documents with Quarto**

**D.2 Quick LaTeX primer**

**D.3 Available templates**

# References

- Barba, L. A. (2018). Terminologies for reproducible research. *arXiv Preprint arXiv:1802.03311*.
- Begley, C., & Ellis, L. (2012). *Drug development: Raise standards for preclinical cancer research. Nature.[Online]. 483 (7391)*.
- Brainard, J., You, J., et al. (2018). What a massive database of retracted papers reveals about science publishing’s “death penalty.” *Science*, 25(1), 1–5.
- Burman, L. E., Reed, W. R., & Alm, J. (2010). A call for replication studies. *Public Finance Review*, 38(6), 787–793.
- Ioannidis, J. P. (2005). Why most published research findings are false. *PLoS Medicine*, 2(8), e124.
- Knuth, D. E. (1984). Literate programming. *Comput. J.*, 27(2), 97–111. <https://doi.org/10.1093/comjnl/27.2.97>
- Kumar, P., Hendriks, T., Panoutsopoulos, H., & Brewster, C. (2024). Investigating FAIR data principles compliance in horizon 2020 funded agri-food and rural development multi-actor projects. *Agricultural Systems*, 214, 103822. <https://doi.org/https://doi.org/10.1016/j.agry.2023.103822>
- Leek, J. T., & Peng, R. D. (2015). Reproducible research can still be wrong: Adopting a prevention approach. *Proceedings of the National Academy of Sciences*, 112(6), 1645–1646.
- Peng, R. D. (2011). Reproducible research in computational science. *Science*, 334(6060), 1226–1227.
- Wicherts, J. M., Borsboom, D., Kats, J., & Molenaar, D. (2006). The poor availability of psychological research data for reanalysis. *American Psychologist*, 61(7), 726.
- Wilkinson, M. D., Dumontier, M., Aalbersberg, Ij. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., Silva Santos, L. B. da, Bourne, P. E., et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(1), 1–9. <https://doi.org/https://doi.org/10.1038/sdata.2016.18>