Quarto

Last modified: 2026-11-03 08:35:08 (PST)

Introduction

Quarto is an open-source scientific and technical publishing system that allows you to create documents, books, websites, presentations, and more. Quarto provides a unified authoring framework for data science, combining your code, its results, and your prose. Quarto documents are fully reproducible and support dozens of output formats, like PDFs, Word files, presentations, and more.

Quarto files are designed to be used in three ways:

1. **For communicating to decision-makers,** who want to focus on the conclusions, not the code behind the analysis.
2. **For collaborating with other data scientists** (including future you!), who are interested in both your conclusions, and how you reached them (i.e., the code).
3. **As an environment in which to do data science,** as a modern-day lab notebook where you can capture not only what you did, but also what you were thinking.

Key Features

Multi-format Output: Quarto documents can be rendered into HTML, PDF, MS Word, ePub, PowerPoint, Revealjs presentations, dashboards, websites, and books from a single source file. This allows authors to maintain one document but publish it in multiple formats without rewriting content.

Rich Markdown Authoring: Content is created in markdown, with support for figures, tables, equations (LaTeX), citations, cross-references, and advanced layout features like tabs, callouts, and panels.

Embedded Executable Code: Integrate code chunks (R, Python, Julia, Observable JS) that can be executed and the results rendered directly in the document. This allows for dynamic results, data analysis, plots, and reproducible research workflows.

Interactivity: Add interactive components such as widgets, tab sets, and collapsible sections for richer communication with readers.

Customization: Extensive theming and styling options, including custom CSS and advanced layout controls for polished, publication-quality output.

Project Management: Organize large projects and integrate with version control tools like Git. Use Quarto projects to group related documents, manage dependencies, and orchestrate rendering.

Why Quarto?

Quarto is the successor to R Markdown. Quarto runs Python and Julia as well as R, and it replaces the add-on packages that R Markdown needed for books, websites, and slides (Wickham et al. 2023, ch. 28).

Getting Started

To get started with Quarto:

- **Installation:** Quarto CLI is included with RStudio, so if you have a recent version of RStudio, you already have Quarto. Otherwise, visit https://quarto.org/docs/get-started/
- **Documentation:** The official Quarto documentation is available at https://quarto.org/docs/guide/
- **R4DS Chapter:** For an excellent introduction to using Quarto with R, see the Quarto chapter in R for Data Science (Wickham et al. 2023)

Quarto Basics

A Quarto document is a plain text file with the extension `.qmd`. It contains three important types of content:

1. An (optional) **YAML header** surrounded by `---`s
2. **Chunks** of code surrounded by ```` ``` ````
3. Text mixed with simple text formatting like `# heading` and `_italics_`

Creating a New Quarto Document

In RStudio, create a new Quarto document using **File > New File > Quarto Document...** in the menu bar. RStudio will launch a wizard that you can use to pre-populate your file with useful content that reminds you how the key features of Quarto work.

Visual vs. Source Editor

RStudio provides two ways to edit Quarto documents:

Visual Editor: The Visual editor provides a WYSIWYM interface for authoring Quarto documents. If you're new to computational documents but have experience using tools like Google Docs or MS Word, the visual editor is the easiest way to get started. In the visual editor you can use the buttons on the menu bar to insert images, tables, cross-references, etc., or you can use the catch-all + / or Ctrl + / shortcut to insert just about anything.

Source Editor: The source editor allows you to edit the raw markdown and code. While the visual editor will feel familiar to those with experience in word processors, the source editor will feel familiar to those with experience writing R scripts or R Markdown documents. The source editor can also be useful for debugging any Quarto syntax errors since it's often easier to catch these in plain text.

You can switch between the visual and source editors at any time using the toggle in the top-left of the editor pane.

Rendering Documents

To produce a complete report containing all text, code, and results:

- Click **"Render"** in RStudio, or
- Press **Cmd/Ctrl + Shift + K**, or
- Use `quarto::quarto_render("document.qmd")` in R, or
- Use `quarto render document.qmd` in the terminal

When you render the document, Quarto sends the `.qmd` file to **knitr**, which executes all of the code chunks and creates a new markdown (`.md`) document which includes the code and its output. The markdown file generated by knitr is then processed by **pandoc**, which is responsible for creating the finished file in your chosen format (HTML, PDF, Word, etc.).

Code Chunks

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

1. The keyboard shortcut **Cmd + Option + I** / **Ctrl + Alt + I**
2. The **"Insert"** button icon in the editor toolbar
3. By manually typing the chunk delimiters ```` ```{r} ```` and ```` ``` ````

Chunks can be given an optional label and various chunk options:

````
```{r}
#| label: simple-addition
#| echo: false
1 + 1
```
````

Common chunk options include:

- `#| label:` - give the chunk a name
- `#| echo: false` - hide the code but show the output
- `#| code-fold: true` - allow readers to toggle code visibility (useful when output is more important to the narrative than the code)
- `#| eval: false` - show the code but don't run it
- `#| include: false` - run the code but hide both code and output
- `#| warning: false` - hide warnings
- `#| message: false` - hide messages

Use `code-fold: true` for chunks where the output is important to the narrative and not the code used to produce it. This allows interested readers to expand and view the code while keeping the document focused on results.

Format-Specific Settings

When rendering to multiple output formats (HTML, PDF, DOCX, EPUB), you may want different chunk options or behavior for different formats. Use `knitr::pandoc_to()` with `if()` statements to detect the output format and set format-specific settings.

Example: Different figure sizes for different formats

```{r}
#| label: example-plot
#| fig-width: !expr if (knitr::pandoc_to("html")) 8 else 6
#| fig-height: !expr if (knitr::pandoc_to("html")) 6 else 4

plot(1:10)
```

Example: Conditional code execution based on format

```{r}
if (knitr::pandoc_to("docx")) {
  # DOCX-specific code
  knitr::kable(data, format = "simple")
} else if (knitr::pandoc_to("html")) {
  # HTML-specific code
  knitr::kable(data, format = "html")
} else {
  # PDF or other formats
  knitr::kable(data, format = "latex")
}
```

Common format detection patterns:

- `knitr::pandoc_to("html")` - returns `TRUE` for HTML output
- `knitr::pandoc_to("latex")` - returns `TRUE` for PDF output
- `knitr::pandoc_to("docx")` - returns `TRUE` for Word output
- `knitr::pandoc_to("epub")` - returns `TRUE` for EPUB output

This technique is particularly useful when you need to:

- Adjust figure dimensions for different page sizes
- Use different table formatting for different outputs
- Include or exclude content based on output format
- Set format-specific styling or options

Text Formatting

Quarto uses Pandoc's markdown for text formatting:

- `*italic*` or `_italic_` produces *italic* text

- `**bold**` or `__bold__` produces **bold** text
- `` `code` `` produces `code` formatting
- `# Heading 1`, `## Heading 2`, `### Heading 3` for headings
- Bullet lists start with `-` or `*`
- Numbered lists start with `1.`, `2.`, etc.
- `[link text](url)` creates hyperlinks
- `![alt text](image.png)` inserts images

Important: Always include a blank line before bullet lists and numbered lists in markdown and Quarto documents.

For more details on using Quarto for writing and analysis, see the Quarto chapter in R for Data Science (Wickham et al. 2023).

Building Quarto Books

Quarto books let you author entire books (or course notes, manuals, dissertations, etc.) in markdown. Quarto books are ideal for documentation, tutorials, lab manuals, and other long-form content.

Creating a Quarto Book

Starting from a template (recommended):

Using a template is the fastest way to get started with a Quarto book, as it provides pre-configured settings, example content, and GitHub Actions workflows for automated deployment:

1. **Quarto Book Template (`qbt`)** - our recommended template, with pre-configured settings and CI:
    - Repository: https://github.com/Morrison-Lab/qbt
    - Click "Use this template" → "Create a new repository" on GitHub
    - Clone your new repository and start editing
    - Includes GitHub Actions for automatic deployment to GitHub Pages
2. **Coatless Tutorials Quarto Book Template** - Another template with helpful examples:
    - Repository: https://github.com/coatless-tutorials/quarto-book-template
    - Includes examples of common Quarto book features
3. **DataLab Quarto Template** - Template from the UC Davis DataLab and Davis R Users Group:
    - Repository: https://github.com/d-rug/datalab_template_quarto

- Provides a starting point for DataLab workshop materials and tutorials

While these templates jumpstart your project with up-to-date configuration and workflow files, you should still come up to speed on what all the config files do (particularly `_quarto.yml` and any GitHub Actions workflows) so you can modify and debug them as needed. The templates serve as central locations for the most current versions of these files and best practices.

Starting from scratch:

If you prefer to start from scratch, you can create a new Quarto book project using the Quarto CLI:

```
# Create a new Quarto book project
quarto create project book mybook
cd mybook
```

This will create a basic book structure with:

- `_quarto.yml` - configuration file for your book
- `index.qmd` - the home page / preface
- Sample chapter files
- `references.qmd` - bibliography/references page

Building and Previewing

Once you have a Quarto book project, you can build and preview it:

```
# Render the entire book
quarto render

# Preview with live reload (recommended during development)
quarto preview
```

The `quarto preview` command starts a local web server and automatically refreshes the preview whenever you save changes to your files.

Book Structure

A typical Quarto book is organized as follows:

`_quarto.yml`: The main configuration file that defines:

- Book metadata (title, author, date)

- Chapter order
- Output formats (HTML, PDF, ePub, etc.)
- Styling and theme
- Navigation options

Chapter files: Individual `.qmd` files for each chapter. These are listed in the `chapters` section of `_quarto.yml`.

Parts: You can organize chapters into parts for better structure:

```yaml
book:
  chapters:
    - index.qmd
    - part: "Getting Started"
      chapters:
        - intro.qmd
        - basics.qmd
    - part: "Advanced Topics"
      chapters:
        - advanced.qmd
```

Book Features

Quarto books support many advanced features:

Cross-references: Reference figures, tables, equations, and sections throughout your book:

```
See @fig-plot for details.
As shown in @tbl-results.
Refer to @sec-introduction.
```

Citations: Include a bibliography and cite sources:

```
According to @smith2020, the method works well.
```

Search: Automatic full-text search in HTML output.

Downloads: Offer PDF, ePub, and Word versions alongside HTML.

Navigation: Automatic table of contents, previous/next chapter buttons, and breadcrumbs.

Customization: Custom themes, CSS, and templates for professional appearance.

Example: The UCD-SERG Lab Manual

The UCD-SERG lab manual is a Quarto book. You can view its source code at https://github.com/UCD-SERG/lab-manual to see how it structures chapters, used includes for modular content, and configured various output formats.

Resources

- Quarto Books Guide - comprehensive documentation for Quarto books
- Quarto Publishing Guide - how to publish your book online
- R4DS Quarto chapter (Wickham et al. 2023) - excellent introduction to Quarto

Dashboards

Quarto dashboards turn a `.qmd` (or Jupyter notebook) into a data dashboard by setting `format: dashboard` in the document YAML.

Layout is driven by ordinary markdown headings, which define pages, rows, columns, and tabsets; each computational cell becomes a "card" holding a plot, table, or text. Value boxes highlight key metrics, and sidebars and toolbars hold interactive inputs.

Dashboards work with R, Python, Julia, and Observable JavaScript, and display both interactive widgets (`{plotly}`, `{leaflet}`, `{htmlwidgets}`) and static graphics (`{ggplot2}`, `Matplotlib`).

By default a dashboard renders to static HTML, so it can be published anywhere a web page can (including GitHub Pages, like other Quarto output) with no server behind it. When server-side interactivity is genuinely needed, a Shiny backend can drive the dashboard, at the cost of needing a Shiny server to host it. For most research reporting, start with the static form and add Shiny only if filtering or recomputation on the reader's side becomes necessary.

See the dashboard examples gallery for layouts to copy from.

Quarto Profiles

Quarto profiles allow you to customize rendering behavior for different purposes. A profile is a named set of configuration options that can be activated when rendering. This is particularly useful when you want to render the same source files in different formats or for different audiences.

What are Profiles?

Profiles let you maintain multiple `_quarto.yml` configuration files in the same project. For example, you might have:

- `_quarto.yml` - default book configuration
- `_quarto-revealjs.yml` - configuration for rendering chapters as slide decks
- `_quarto-print.yml` - configuration optimized for PDF printing

Example: Rendering Chapters as Slides

A excellent example of using Quarto profiles comes from the Regression Models for Epidemiology course materials by D. Morrison.

The project includes a `_quarto-revealjs.yml` profile that allows each chapter to be compiled as a RevealJS slide deck, in addition to being part of the book.

To render a single chapter as slides:

```
quarto render chapter-name.qmd --profile=revealjs
```

To render all chapters listed in the profile as slides:

```
quarto render --profile=revealjs
```

Creating a Profile

To create a profile:

1. Create a new YAML file named `_quarto-{profile-name}.yml`
2. Include only the configuration options that differ from your default `_quarto.yml`
3. Activate the profile when rendering using `--profile={profile-name}`

Example profile structure:

`_quarto-slides.yml`:

```yaml
project:
  type: default
  output-dir: slides

format:
  revealjs:
```

```
    theme: serif
    slide-number: true
    preview-links: auto
```

Common Use Cases

Multiple output formats: Maintain separate configurations for web, print, and presentation versions of your content.

Different audiences: Create versions with or without solutions, technical details, or instructor notes.

Development vs. production: Use a development profile with faster rendering options during writing, and a production profile with full features for final output.

Course materials: Render the same content as both a reference book and lecture slides, as demonstrated in the RME course.

Resources

- Quarto Profiles Documentation
- RME Example - see the source at https://github.com/d-morrison/rme for a working example

Advanced Features

Cross-References

Quarto provides a powerful cross-reference system for figures, tables, equations, and sections. Cross-references automatically number your content and create clickable links in HTML and PDF output.

Required label prefixes:

- Figures: `#fig-` (e.g., `#fig-workflow-diagram`)
- Tables: `#tbl-` (e.g., `#tbl-summary-stats`)
- Equations: `#eq-` (e.g., `#eq-regression-model`)
- Sections: `#sec-` (e.g., `#sec-introduction`)
- Theorems: `#thm-`, Lemmas: `#lem-`, Corollaries: `#cor-`
- Propositions: `#prp-`, Examples: `#exm-`, Exercises: `#exr-`

For figures (static images):

```
![Caption text](path/to/image.png){#fig-label}
```

For code-generated figures:

````
```{r}
#| label: fig-plot-name
#| fig-cap: "Caption text describing the plot"

# R code to generate plot
ggplot(data, aes(x, y)) + geom_point()
```
````

For tables (markdown tables):

```
| Column 1 | Column 2 |
|----------|----------|
| Data     | Data     |

: Caption text {#tbl-label}
```

For code-generated tables:

````
```{r}
#| label: tbl-table-name
#| tbl-cap: "Caption text"

# R code to generate table
knitr::kable(data)
```
````

Referencing in text:

- Figures: `@fig-label` produces "Figure X"
- Tables: `@tbl-label` produces "Table X"
- Equations: `@eq-label` produces "Equation X"
- Sections: `@sec-label` produces "Section X"

Benefits:

- Automatic numbering of figures, tables, and equations

- Automatic updates when content is reordered
- Clickable cross-references in HTML and PDF output
- Consistent formatting across all output formats
- Better accessibility for screen readers

For complete details, see the Quarto Cross-References documentation.

Using Includes for Modular Content

Quarto's include feature allows you to decompose large documents into smaller, more manageable files. This is particularly useful for books and long documents.

Basic syntax:

```
{{< include path/to/file.qmd >}}
```

Benefits:

1. **Better Git History:** When sections are reordered, only the main chapter file changes (moving include statements), making it immediately clear that content was reorganized rather than edited.
2. **Easier Code Review:** Reviewers can see exactly what changed—either the organization (main file) or the content (include file).
3. **Modular Maintenance:** Each section lives in its own file, making it easier to find and edit specific content, reuse sections across chapters, and work on different sections simultaneously without merge conflicts.
4. **Clear Structure:** The main chapter file becomes a table of contents showing the organization at a glance.

Recommended pattern:

Main chapter file (e.g., `05-coding-practices.qmd`):

```
# Coding Practices

## Section Heading

{{< include coding-practices/section-name.qmd >}}

## Another Section

{{< include coding-practices/another-section.qmd >}}
```

Include files (e.g., `coding-practices/section-name.qmd`):

- Stored in a subdirectory matching the chapter name
- Contains only the content for that section (no heading)
- The heading stays in the main chapter file
- Named descriptively using kebab-case

Note: The heading must be in the main file, followed by a blank line, then the include statement. This keeps the document structure clear in the main file.

For more details, see the Quarto Includes documentation.

Including External Code Files

When embedding raw source code from external files without executing them, the `include-code-files` extension provides a filter to include code directly into fenced code blocks with syntax highlighting and line or snippet selection.

Installation:

```
quarto add quarto-ext/include-code-files
```

Add the filter to your document or `_quarto.yml`:

```
filters:
  - include-code-files
```

Usage:

````
```{.r include="path/to/script.R"}
```
````

Key capabilities:

- Embed external scripts directly without copy-pasting
- Slice specific line ranges (e.g., `start-line=10 end-line=25`)
- Include labeled snippets (e.g., `snippet="main"`)
- Keep documentation in sync with working standalone script files

Mermaid Diagrams

Mermaid diagrams are a powerful tool for creating flowcharts, sequence diagrams, state diagrams, and other visualizations directly in your Quarto documents using text-based syntax. While mermaid diagrams work well in HTML output, they have significant limitations when rendering to DOCX (Word) and PDF formats as of early 2026.

Creating Mermaid Diagrams

Mermaid diagrams are created using fenced code blocks with the `mermaid` language tag:

````
```{mermaid}
flowchart LR
  A[Start] --> B{Decision}
  B -->|Yes| C[Result 1]
  B -->|No| D[Result 2]
```
````

This creates a simple flowchart that renders beautifully in HTML output.

Known Issues with DOCX and PDF Output

As of early 2026, rendering mermaid diagrams to DOCX or PDF formats in Quarto is unreliable and problematic. These issues are well-documented in the Quarto community and stem from the underlying tooling used to convert diagrams to images.

Rendering Failures and Hangs

When rendering documents with mermaid diagrams to DOCX format, you may experience:

- **Indefinite hangs:** Quarto may hang indefinitely during rendering, leaving orphaned Chrome/Edge browser processes running (Gewerd-Strauss 2023)
- **Complete rendering failures:** The rendering process may fail entirely with errors
- **Missing diagrams:** The document may render successfully but with missing diagrams ("Quarto Cannot Render Mermaid or Dot Images to Docx and Pdf Formats" 2023)

These issues occur because Quarto relies on headless browser engines (Chrome or Edge via Puppeteer) to convert mermaid diagrams to images for inclusion in DOCX and PDF outputs. The browser integration can fail or hang, particularly when running in automated environments or when multiple diagrams are present.

Image Quality Issues

Even when rendering succeeds, mermaid diagrams in DOCX and PDF output often suffer from:

- **Small or blurry images:** Diagrams appear rasterized at low resolution
- **Incorrect sizing:** Diagrams may not respect sizing options specified in the document

Recommended Workarounds

Given these limitations, we recommend the following approaches when you need DOCX or PDF output:

1. Manual Export and Embedding (Most Reliable)

The most robust workflow is to:

1. First render your mermaid diagrams to HTML
2. Export them as PNG or SVG images manually using:
    - Browser developer tools (save rendered diagram)
    - Online mermaid editors (e.g., https://mermaid.live/)
    - Mermaid CLI tools
3. Include the exported images in your Quarto document using standard markdown syntax

Example:

```
![Workflow diagram description](assets/images/workflow-diagram.png){#fig-workflow}
```

This approach guarantees compatibility with all output formats and gives you full control over image quality.

2. Conditional Content for Different Formats

Use Quarto's conditional content features to show different content based on output format:

````
::: {.content-visible when-format="html"}
```{mermaid}
flowchart LR
  A[Start] --> B[End]
```
:::
````

```
::: {.content-visible unless-format="html"}
![Workflow diagram](assets/images/workflow-diagram.png)
:::
```

This shows the live mermaid diagram in HTML output and a static image in DOCX/PDF outputs.

3. Ensure Chromium is Installed (Partial Solution)

If you want to attempt rendering mermaid diagrams to DOCX, ensure you have Chrome or Edge installed. You can install Chromium via Quarto:

```
quarto tools install chromium
```

However, this does not guarantee successful rendering and may still result in hangs or failures.

4. Prefer HTML Output When Possible

When creating documentation that includes mermaid diagrams, prefer HTML as your primary output format. HTML rendering of mermaid diagrams is fast, reliable, and produces high-quality interactive diagrams.

If DOCX output is required for collaboration or submission, use one of the workarounds above.

Real-World Example

In UCD-SERG/rpt PR #70, the maintainers removed mermaid diagrams from package vignettes specifically because they were incompatible with DOCX output. This demonstrates the practical impact of these limitations in production workflows.

Future Outlook

The Quarto development team is aware of these issues and working to improve diagram rendering support. Monitor the following resources for updates:

- Quarto CLI Issue #3809 - Tracking mermaid/dot rendering to DOCX/PDF
- Quarto Discussion #6085 - Mermaid rendering hangs with DOCX
- Quarto Diagrams Documentation - Official documentation on diagram rendering

Check for improvements in newer Quarto versions, but as of early 2026, manual export and embedding remains the most reliable approach for DOCX and PDF outputs.

Summary

Key Takeaways:

- Mermaid diagrams work well in HTML output but are problematic for DOCX and PDF
- Rendering failures, hangs, and quality issues are common
- For reliable DOCX/PDF output, export diagrams manually as images
- Use conditional content to show different formats based on output type
- Prefer HTML output when mermaid diagrams are important to your document

Troubleshooting

When encountering issues with Quarto, the official Quarto Troubleshooting Guide provides comprehensive diagnostic tools and strategies. This section summarizes key troubleshooting approaches.

Checking Your Setup

Use `quarto check` to verify your installation and dependencies:

```
quarto check
```

This command checks for:

- Quarto version and binary dependencies (Pandoc, Dart Sass, Deno, Typst)
- LaTeX installation (TinyTeX or system TeX)
- Python, Jupyter, and available kernels
- R, knitr, and rmarkdown versions
- Library paths and installed packages

See the `quarto check` output and the official guide for the complete, authoritative list of checks.

Diagnostic Tools

Enable verbose logging to see detailed rendering information.

On Unix-like systems (macOS, Linux):

```
export QUARTO_LOG_LEVEL=DEBUG
quarto render document.qmd
```

On Windows PowerShell:

```
$env:QUARTO_LOG_LEVEL="DEBUG"
quarto render document.qmd
```

Or pass at command line (all platforms):

```
quarto render document.qmd --log-level debug
```

Get stack traces for better error messages.

On Unix-like systems (macOS, Linux):

```
export QUARTO_PRINT_STACK="true"
quarto render document.qmd
```

On Windows PowerShell:

```
$env:QUARTO_PRINT_STACK="true"
quarto render document.qmd
```

Inspect log files for diagnostic information. Log file locations:

- **Windows:** `%LOCALAPPDATA%\quarto\logs`
- **macOS:** `~/Library/Application Support/quarto/logs`
- **Linux:** `~/.local/share/quarto/logs` (or `${XDG_DATA_HOME}/quarto/logs` if `${XDG_DATA_HOME}` is set)

Common Issues

Out-of-memory errors: When building large projects, increase memory allocation (example sets 8GB limit).

On Unix-like systems (macOS, Linux):

```
export QUARTO_DENO_V8_OPTIONS="--max-old-space-size=8192"
quarto render
```

On Windows PowerShell:

```
$env:QUARTO_DENO_V8_OPTIONS="--max-old-space-size=8192"
quarto render
```

PDF rendering issues: If you encounter PDF rendering problems, ensure you have an up-to-date LaTeX installation, or install Quarto's TinyTeX:

```
quarto install tinytex
```

Environment conflicts: If Quarto finds the wrong R or Python installation, use `quarto check` output to verify which installation Quarto is using. For R, you can render a diagnostic document with `sessionInfo()` to compare Quarto's environment with your interactive R session.

Advanced Troubleshooting

For more specialized issues, consult the official Quarto Troubleshooting Guide, which covers:

- Jupyter engine debugging and kernel issues
- Lua filter debugging and tracing
- Julia-specific troubleshooting
- Detailed environment and dependency diagnostics
- Installation problems (including macOS installer logs)

When seeking help on the Quarto issue tracker, include diagnostic information from `quarto check` and relevant log files.

Additional Resources

Official Documentation

- Quarto Official Guide - comprehensive official documentation
- Quarto Books Guide - documentation specific to creating books
- Quarto Publishing Guide - how to publish your Quarto content online
- Quarto Getting Started - installation and basic usage

Learning Resources

- R for Data Science - Quarto Chapter (Wickham et al. 2023) - excellent introduction to using Quarto with R
- litedown: R Markdown Reimagined - a book by Yihui Xie on `{litedown}`, a lightweight R Markdown implementation aimed at simplicity and speed
- Regression Models for Epidemiology - example of a Quarto book with profiles for rendering chapters as slides
- UCD-SERG Lab Manual Source - a Quarto book whose source provides examples of:
  - Book structure and organization
  - Using includes for modular content
  - Configuring multiple output formats (HTML, PDF, ePub, Word)
  - Cross-references for figures, tables, and sections

Templates

- Quarto Book Template (`qbt`) - our recommended book template
- Quarto Website Template (`qwt`) - our recommended website template, with slide, PDF, and Word output for each page
- Coatless Tutorials Quarto Book Template - another frequently-used template with helpful examples
- DataLab Quarto Template - template from the UC Davis DataLab and Davis R Users Group

Extensions and Tools

- `include-code-files` - Quarto filter extension to include external code files directly inside fenced code blocks with syntax highlighting and line slicing

Development Roadmap and Known Issues

The Quarto development team uses GitHub "epics" to track collections of related bugs and planned improvements. These are useful for staying informed about known limitations and upcoming features:

- All open Quarto epics - index of all open epic issues in the Quarto CLI repository
- reveal.js bugs & improvements epic - example epic tracking reveal.js-related issues

Related chapters

For additional context about using Quarto:

- R Coding Practices - R coding practices that apply to code in Quarto documents
- R Code Style - Code style guidelines including formatting for Quarto documents
- Git and GitHub - Version control for Quarto projects

Gewerd-Strauss. 2023. "Mermaid Rendering to Docx Halts Indefinitely When Run Through Command Line." Quarto Development Team. https://github.com/orgs/quarto-dev/discussions/6085.

"Quarto Cannot Render Mermaid or Dot Images to Docx and Pdf Formats." 2023. Quarto Development Team. https://github.com/quarto-dev/quarto-cli/issues/3809.

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