Paper PP06

Getting Started with Quarto

Kevin Putschko, Experis, USA

Abstract

Clinical reports have their own defined reporting styles, and that is well documented. But what about using Quarto Books or Quarto Documents to easily maintain metadata regarding clinical studies? This could include querying your data for quality control, keeping up-to-date listings of all the variables and variable summaries, or including interactive tables and plots to help researchers explore data for new insights. Quarto is the next generation of dynamic and repeatable reporting with R, blending code, visualizations, tables, and documentation. With the ability to create and reuse custom templates, maintaining the look and feel of clinical tables, figures, and listings has never been easier. In this poster, we will explore the transformative capabilities of Quarto with a brief introduction to the creation and use of custom templates.

Introduction

Quarto is the next generation of R Markdown, and is language independent. It works within the R framework, but also with Python, Julia, and Observable. It can work also work with Jupyter Notebooks. Quarto makes it easy to create reports that are easily updated to reflect changes in source files, data, or analyses.

There are default templates that Quarto will use for your chosen output format, and are good enough for many use cases. However, when you want to customize the output to conform to your personal standards, or your organization's standards, then you will have to either modify these default templates to match your ideal output, or you will have to provide an entirely new template for Quarto to use.

A Basic Example

A basic LaTeX custom template `template.tex` might look like this, where the *body* will be occupied by the contents and results of the Quarto document.

```
\documentclass{scrartcl}
    \begin{document}
        $body$
    \end{document}
```

And in your Quarto document you can use this as your preferred template for PDF documents by setting the `template` parameter in the YAML header of the Quarto document.

```
format:
    pdf:
        template: template.tex
```

Modify or Replace

We can either create an entirely new template, or we can use partial templates to fit in with existing templates. Partial templates are simply references to other template files from within a template file. We may want to completely replace a template when we want to have full control over every aspect of the final document. This requires deeper knowledge of the Quarto and Pandoc requirements stored in the default templates.

So, using partial templates will be an easier option in many cases when working within Quarto. To do this, one could examine the default templates and find the appropriate partials to replace, or one could include the Quarto and Pandoc requirements in their own partial template and reference it in the new customized template.

See the Quarto Github page for the existing Quarto templates, with source files that you can further explore.

Partial Templates

Partial templates will require that you override existing templates in the overall default template. In order to to that, you'll need to store your template modifications in files that are named specifically to target a certain existing template partial.

You can see the existing partial templates here and you can see a description of what each of these targets here.

We can use the template partial in the Quarto YAML header with the `template-partial` parameter

```
format:
    pdf:
        template-partials:
            - title.tex
```

In this example, we will only modify the document title information by using the `title.tex` partial template. If we wanted to adjust the display aspects of the final document, like font size and page size, then we would store those in a `doc-class.tex` file.

Note, we've been focusing on LaTeX thus far, but for HTML templates the configuration is similar. You can view available HTML partial templates here with descriptions of these partials here.

Replacing Default Templates

There are many starter templates available in Quarto that offer some pre-defined customizations that differ from the default template, where some might conform to a variety of journal standards. It could be useful to explore these options as a starting point for developing full replacement templates that conform to your organization's standards.

Working with custom templates requires knowledge beyond R and Quarto, it'll require knowledge of the structure of the style you're trying to replicate. So working with custom LaTeX templates will require thorough knowledge of how LaTeX works, and creating custom HTML templates will require deep knowledge of HTML.

You can find a list of Quarto starter templates here.

LaTeX Templates

The `rticles` package offers dozens of LaTeX templates for PDF documents. You can either start here by using one of these custom templates, or using these as inspiration for your own custom templates. This package makes it easy to use these templates because you are able to select the template you want to use when creating a new Quarto document from the RStudio menu. Learn more about `rticles` at the documentation website.

Recommended Reading

For a step by step guide on creating a custom LaTeX template for PDF documents, see this blog post by Nicola Rennie, or this blog post by Christopher Kenny.

For more information on customizing PDF reports with templates that accept dynamic parameters, see this blog post by Meghan Hall.

DOCX Templates

Build a reference file with all the custom formatting before reading it in as a Quarto template. You can use Quarto at the command line to create a Pandoc default Word document, allowing you to modify the default settings without starting from an empty document. Once you are satisfied with the changes to the reference document, you read this document in with the `reference-doc` parameter in the YAML header to ensure your final document conforms to this standardized style.

```
format:
    docx:
        reference-doc: reference.docx
```

See the Quarto documentation for more detailed information on how this process works. And to see a sample workflow of a Word document template, see the Github page for the standard reporting template used by the government of France.

Lua Filters

Lua filters are another type of Quarto extension, in addition to custom templates which modify the markdown rendering operations. The process of transforming text from Quarto markdown and R markdown into its final output format requires filters to read and write from input to output. Most of the operations going on in the background when you use the `Render` command in Quarto are built on these filters. Typically filters are written using the Lua coding language because it has no external dependencies and is highly performant, and it can easily make use of the existing Pandoc and Quarto Lua functions.

The process of creating Lua filters requires knowledge of the Lua language, but you can find the steps required to create these filters in the Quarto documentation pages.

Once you've created your own filters, you can load them for use in your Quarto documents with the following YAML header:

```
filters:
    - spellcheck.lua
    - new_filter_1.lua
    - old_filter_2.lua
```

See the Quarto documentation and the Pandoc documentation for more information on creating and using Lua filters.

Demonstration

Use Case

For a rough demonstration of how this process can work, I've created a function that scans all the RDS R data sets in a directory and provides a summary table of each. This summary table reports for each variable:

- Order
- Name
- Type
- Label
- Listing of distinct values for categorical variables, or mean, standard deviation, and range for numeric variables

You can see the definition of this function for yourself at the Github page for this project.

Template Modifications

This function is used inside of a Quarto document where I'm using LaTeX partial templates to help me customize the final report. Specifically, I'm using the following YAML header information to ensure the final report is in landscape orientation, with 75pt margins on the left and right side.

```
format:
  pdf:
    classoption: [landscape]
    margin-left:   "75pt"
    margin-right:  "75pt"
    include-in-header:   latex/header.tex
    include-before-body: latex/before_body.tex
```

Then, in the `include-in-header` parameter, I'm linking to the `header.tex` file that contains the following code. These commands load the `fancyhdr` and `lastpage` LaTeX packages.

```
\usepackage{fancyhdr}
\usepackage{lastpage}
```

Then with `include-before-body: latex/before_body.tex`, I'm including only a `\newpage` command to ensure that my title page, i.e. the portion before the body of the document, is on a separate page from the table of contents.

```
\newpage
```

Within the Quarto document itself, I'm using the following LaTeX commands to further customize my report. These commands set the header and footer height, define the LaTeX template as using the `fancy` style, and then insert text into the header and footer at the left, center, and right locations, respectively.

```
\setlength{\headheight}{50pt}
\setlength{\footheight}{50pt}
\pagestyle{fancy}

\lhead{\small{Data Directory: '{`r display_directory`}'}}
\chead{}
\rhead{PHUSE 2024}

\lfoot{}
```

```
\cfoot{}
\rfoot{\small{Page \thepage\ of\ \pageref{LastPage}}}
```

Dynamic Coding

In the following chunk of code, I want to dynamically create sections of the report based on the data present in the file directory. To accomplish this, I create a default R code chunk where I use the `results: asis` options to specify that this output should be read as markdown code instead of R code.

Then within the loop, I can use the `str_glue` function from `tidyverse` to help me create dynamic footers.

```
table_footer <-
    str_glue(
      "\\lfoot{{
        \\small{{
          Table {i}: '{n}' has {r} rows and {c} columns
        }}
      }}",
      n = loop_input$data_name |> escape_latex(),
      r = loop_input$data_nrow |> number(big.mark = ","),
      c = loop_input$data_ncol)
```

You'll notice here that I'm using R code to help me insert R results into LaTeX code. This can then be converted to raw Quarto markdown code using the `cat` R function.

```
cat("\\newpage")
cat(table_footer)
cat("\n")
paste("# Dataset:", loop_input$data_name) |> cat()
cat("\n")

loop_output <-
    fn_latex_table(
        loop_input$data_summary,
        loop_input$data_name,
        loop_input$data_nrow,
        loop_input$data_ncol)
```

```
print(loop_output)
```

These `cat` functions will be converted to raw LaTeX code where we first create a new page for each iteration of the loop, followed by placement of the table footer for data being summarized in that iteration. Then `\n` is used to create a new line in the code. I use `paste` here to create the Quarto markdown header for the data set being used in the iteration, followed by another `\n` for an empty line in the code.

Lastly, the actual summary table (created with the `fn_latex_table` function that you can see here) is printed with the `print` command. It is important in this case to use `print` to ensure the LaTeX table is explicitly printed in its LaTeX format, and not to the console as R output.

Results

All of that leads to the creation of our final data dictionary in PDF format, with unique page footers, table headers, and table content for each data set in the file directory.

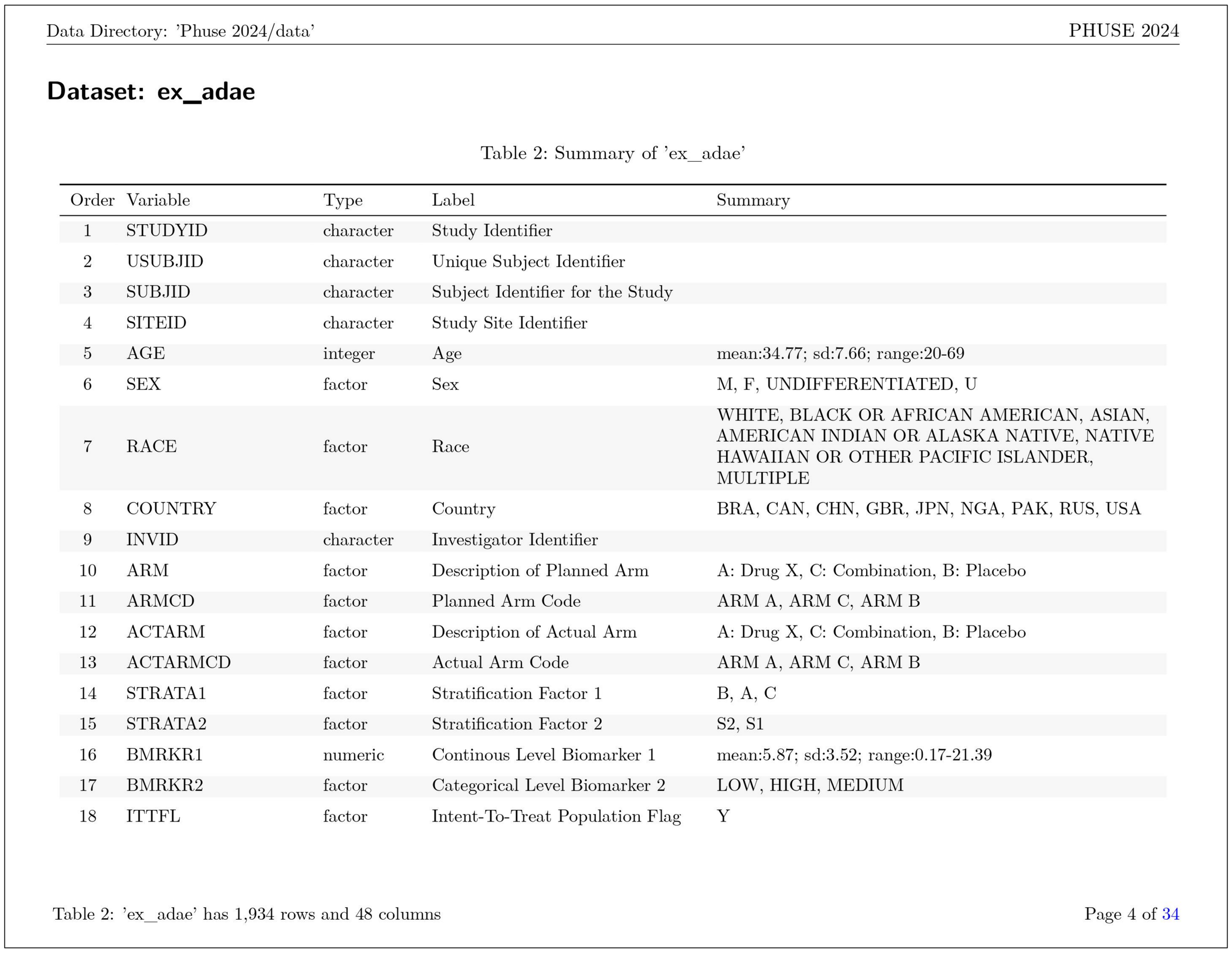
You can see the full PDF report here.

Conclusion

Quarto is fairly straightforward to begin using, but it is powerful enough to help you accomplish exactly what you are trying to do. With Quarto templates and filters, you can modify the output format to match the specific needs of you or your organization. One of the greatest things about Quarto is how well documented its features are, and how active its community is in providing blog posts or answering questions about Quarto on sites like StackOverflow.

Acknowledgements

Thank you to Experis for sponsoring this entry to Phuse 2024, and thank you to my friend Jen Nguyen for the inspiration to explore Quarto templates in the first place.



Data Directory: 'Phuse 2024/data' PHUSE 2024

Dataset: ex_adae

Table 2: Summary of 'ex_adae'

Order	Variable	Type	Label	Summary
1	STUDYID	character	Study Identifier	
2	USUBJID	character	Unique Subject Identifier	
3	SUBJID	character	Subject Identifier for the Study	
4	SITEID	character	Study Site Identifier	
5	AGE	integer	Age	mean:34.77; sd:7.66; range:20-69
6	SEX	factor	Sex	M, F, UNDIFFERENTIATED, U
7	RACE	factor	Race	WHITE, BLACK OR AFRICAN AMERICAN, ASIAN, AMERICAN INDIAN OR ALASKA NATIVE, NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER, MULTIPLE
8	COUNTRY	factor	Country	BRA, CAN, CHN, GBR, JPN, NGA, PAK, RUS, USA
9	INVID	character	Investigator Identifier	
10	ARM	factor	Description of Planned Arm	A: Drug X, C: Combination, B: Placebo
11	ARMCD	factor	Planned Arm Code	ARM A, ARM C, ARM B
12	ACTARM	factor	Description of Actual Arm	A: Drug X, C: Combination, B: Placebo
13	ACTARMCD	factor	Actual Arm Code	ARM A, ARM C, ARM B
14	STRATA1	factor	Stratification Factor 1	B, A, C
15	STRATA2	factor	Stratification Factor 2	S2, S1
16	BMRKR1	numeric	Continous Level Biomarker 1	mean:5.87; sd:3.52; range:0.17-21.39
17	BMRKR2	factor	Categorical Level Biomarker 2	LOW, HIGH, MEDIUM
18	ITTFL	factor	Intent-To-Treat Population Flag	Y

Table 2: 'ex_adae' has 1,934 rows and 48 columns Page 4 of 34

Figure 1: A Dynamic Summary Table

Recommended Reading

- The Pandoc Website
- The Quarto Website
- Quarto Templates
- "Awesome Quarto" Tutorials
- Posit's Guide to Quarto Workflow

Contact Information

Your comments and questions are valued and encouraged. Contact the author at:

Kevin Putschko, Experis Inc.

kputschko@gmail.com

You can find the most up to date version of this paper at

https://github.com/kputschko/phuse2024-quarto