

RUM Quarto Write-up

The goal of this guide is to help you produce a fully-reproducible conference or journal article using [Quarto](#) and [Docker](#). We will begin by using a quarto journal article template to create a formatted .pdf for submission. Then we will use Docker to build a container that captures the computational environment that the analysis, data wrangling etc. took place in. This guide assumes you have installed R, Rstudio (version v2022.07.1 and later is bundled with Quarto), Docker, and a LaTeX engine. If you do not have a local engine, you can install [TinyTex](#) by using the following commands:

```
install.packages('tinytex')
tinytex::install_tinytex()
```

Part 1: Quarto Article Templates

[Quarto](#) is the spiritual successor to [Rmarkdown](#). We can mix markdown-flavoured text with code chunks in [R](#), [Python](#), [Julia](#), and [Observable JavaScript](#). This guide will only deal with code chunks in R, as I have little to no experience with the other languages.

First, let's open Rstudio. Working with Rprojects makes keeping track of directories much simpler, so let's first create one. Call the directory something like "RUM_test", and choose the default, blank R project when prompted. You should now be in the project working directory, which will only have "RUM_test.Rproj" in it.

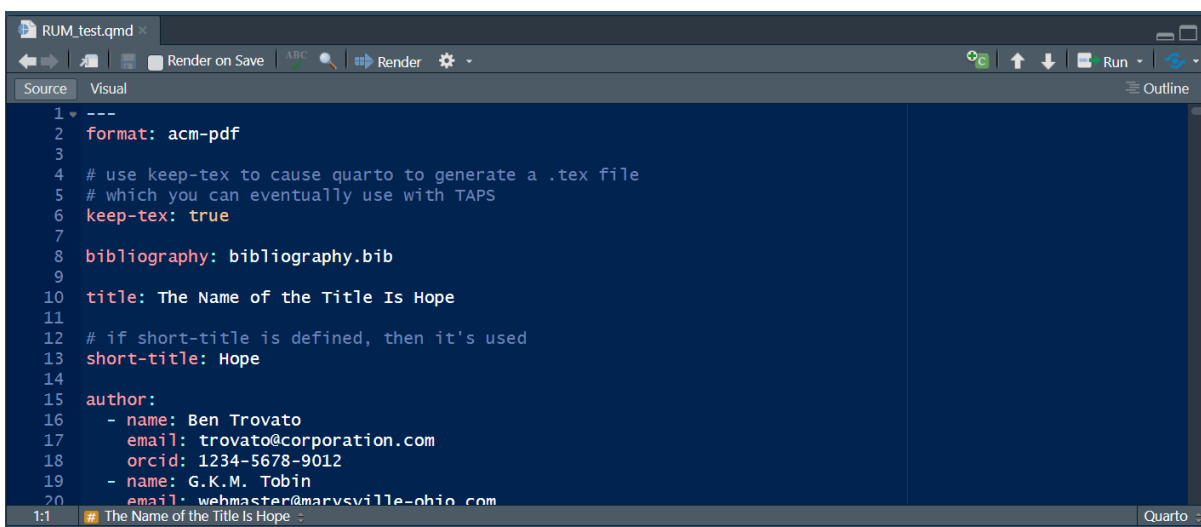
Let's say we wanted to write a paper for the ACM CHI Conference. This conference just uses the ACM format. Go into the terminal tab in R and enter the following command:

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

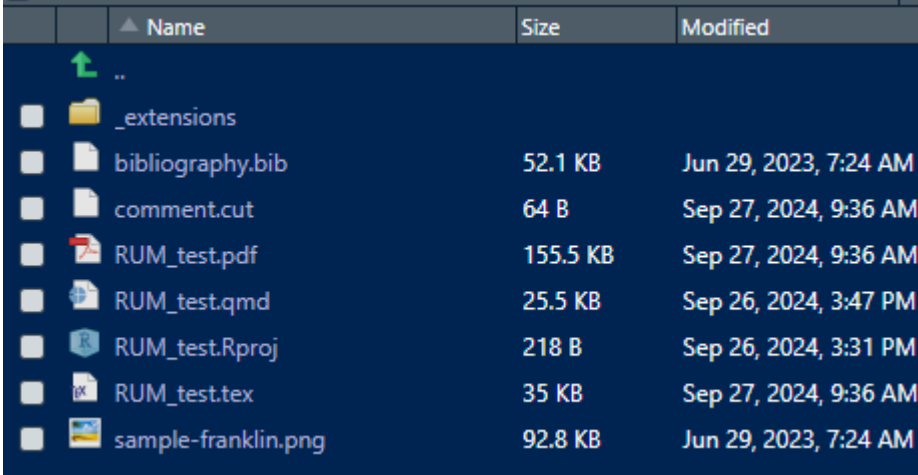
Choose "Yes" for trusting the authors of the template, "No" for creating a new subdirectory (as we're already in the project directory, and "Yes" for downloading any additional changes. At the time of writing, quarto is (sometimes) not populating the _extensions folder correctly, so run the following command to fix this:

```
quarto add quarto-journals/acm
```

Quickly check that the _extensions folder has been populated. Once that is done, we're good to go! Go ahead and open RUM_test.qmd from the files tab. It should look like this:



RUM_test.qmd is a **quarto markdown** file. It is similar to a Rmarkdown file, if you're familiar with that. This is where we will write our paper, run our analyses, create our figures, and so on. Press render at the top or hit *ctrl-shift+k* to render the document. If this is your first time rendering with a TeX engine, this may take quite a while, so just be patient. Once this process has finished, you should have a file tab that looks like this:



| | Name | Size | Modified |
|--|---------------------|----------|-----------------------|
| | .. | | |
| | _extensions | | |
| | bibliography.bib | 52.1 KB | Jun 29, 2023, 7:24 AM |
| | comment.cut | 64 B | Sep 27, 2024, 9:36 AM |
| | RUM_test.pdf | 155.5 KB | Sep 27, 2024, 9:36 AM |
| | RUM_test.qmd | 25.5 KB | Sep 26, 2024, 3:47 PM |
| | RUM_test.Rproj | 218 B | Sep 26, 2024, 3:31 PM |
| | RUM_test.tex | 35 KB | Sep 27, 2024, 9:36 AM |
| | sample-franklin.png | 92.8 KB | Jun 29, 2023, 7:24 AM |

Open the pdf and you will see an ACM-formatted article template. Importantly, because keep-tex is set to true in line 6, we also have the intermediate LaTeX file. The full list of quarto-supported journal articles can be found [here](#). There will undoubtedly be more added in the future, and if you need to write with a template that does not exist, go ahead and [create your own](#) (and contribute it to the community!).

Part 2: Starting to Write Your Paper

.qmd files use markdown syntax. There is a good basic guide to this [here](#). In this section, we will focus on how to use code chunks in your .qmd to produce figures and tables, on modifying the [YAML \(front matter\)](#), and on working with citations and bibliographies.

Code Chunks

With your cursor anywhere in the .qmd (after the YAML), hit *ctrl-alt+i* to insert a code chunk, which should look like this:

```
```{r}
```

The part in the curly brackets tells quarto which language to use to interpret the code chunk. There are options for code chunks, which are detailed [here](#), but to keep things simple, let's set the defaults for all code chunks in our YAML front matter. Copy and paste the following into your YAML, somewhere near the top (NB indentation matters in the YAML):

```
knitr:
 opts_chunk:
 cache_comments: false
 crop: true

execute:
 echo: false
 warning: false
 message: false
 include: false
```

These defaults are what we want for most code chunks - if we manually set an option in a code chunk later on, this will override what we've put in the YAML. These default behaviours mean we won't receive echos (replications of the code in the chunk), warnings, or messages, and that the results of the code chunk won't be included in the knitted document; this is fine, as most of the chunks will probably be data wrangling/handling in the background.

The first thing you should add after the three dashes at the end of your YAML is a chunk labelled "setup". This should at least include all your required libraries, but I also like to use it to fix function conflicts with the [conflicted](#) package.

## Tables & Figures

The first thing you should do with every code chunk is give it a label. This label should always be meaningful, but for figures and tables, it must start like this `fig- /tbl-`. For figures, we also want to make sure that quarto includes the resulting image and that we provide a figure caption. Optionally we can also specify a LaTeX figure environment, the aspect ratio of the image, and the width of the page that the figure will take up (the outwidth). Below is an example of a code chunk for a figure that utilises all of these options:

```
```{r}
#| label: fig-example
#| include: true
#| fig-env: "figure*"
#| fig-asp: 1
#| out-width: "100%"
#| fig-cap: "A figure caption."

ggplot(aes(x, y), data = df)
```
```

Tables work in much the same way. Most journal formats expect tables to be formatted to publication-ready quality using the [booktabs](#) package. To achieve this via quarto, we can use the `kableExtra` package - make sure you have `library(kableExtra)` somewhere in your setup chunk. The usage for this is quite simple. You give `kableExtra` a dataframe, and it turns in into an appropriately-formatted table. You can see an example of this below:

```
```\{r\}  
#| label: tbl-example  
#| include: true  
#| tbl-cap: "A table caption."  
  
kbl(table_df, booktabs = TRUE, digits = c(0,2,3), escape = FALSE)  
```\}
```

Here, we tell kableExtra to use the table\_df dataframe with booktabs formatting. We then specify how many digits should be used for the numbers in each column. The last argument is something we need to get LaTeX formatted text through, but don't worry about that for now.

## Citations & Bibliography

Working with citations is much the same as it is in LaTeX. Once you have your bibliography in a .bib file, copy it into your project directory and make sure your YAML is pointing to it correctly. There is a full quarto citation guide [here](#), but in short, citations go inside square brackets and are preceded by an @ symbol, such as: *There is evidence that visualisation is persuasive [ @strain\_2023 ] in certain contexts.*

## Next Steps

The information presented here should be enough to get you started on writing a paper using quarto. I recommend starting from the template and modifying it from there. I have several examples of full journal and conference papers written entirely using quarto hosted on my [GitHub](#). Downloading one of these and playing around with it should give you an appreciation of how much you can do with quarto. Sometimes, some of the formatting you specify in quarto doesn't quite translate into LaTeX - in this case, it is best to just manually edit the .tex file and render it using a local TeX engine.

## Part 2: Containerising Your Paper With Docker

The packages we use to wrangle our data and perform our analyses change over time. Sometimes these changes are so great that the functions we previously used no longer work, or work in ways that alter the statistical results and subsequent conclusions. To increase the longevity of the work we do, it is therefore important to make sure that future researchers can replicate our results precisely.

One way to do this is to capture an image of the packages and software you used, at the time you used them, and to freeze this in time. This means that future researchers can run your code in exactly the same way you did. I do this using [Docker](#). This tool has many uses, but for our purpose, is deceptively simple.

### 1. Install [Docker Desktop](#)

### 2. Run Docker Desktop

- As this requires virtualisation, running Docker containers requires you to have admin access on the computer you are using.

### 3. Create a Dockerfile in the project directory

- This is a text file that tells Docker how to build your image. It will tell Docker to download an instance of Rstudio server, add your local data and folders to the image, and install all the packages your paper requires to run. For now, copy and paste the following into a blank text

file, rename it “Dockerfile”, then remove the file extension. Make sure that the .qmd that the Dockerfile is adding is the same one that’s in your directory.

```
Add the rocker/verse Docker image for R 4.4.1

FROM rocker/verse:4.4.1

Add our files to container

ADD RUM_test.qmd /home/rstudio/
ADD _extensions/ /home/rstudio/_extensions/
ADD bibliography.bib /home/rstudio/

Add appropriate versions of required R packages to container

RUN R -e "install.packages('devtools')"

RUN R -e "require(devtools)"

tidyverse is included in rockerverse image

RUN R -e "devtools::install_version('MASS', version = '7.3-61', dependencies = T)"
```

#### 4. Run commands to build the image

- Using command prompt/terminal, navigate to your project folder (the one with the Dockerfile in it), and run the following commands:

```
docker build -t RUM_test .
```

(OR ON ARM:

```
Docker buildx build --platform linux/amd64 -t RUM_test .)
```

```
docker run --m -p 8787:8787 -e PASSWORD=password RUM_test
```

- This can take a while, so be patient here. Once this is all done, go into your web browser and type “localhost:8787” into the address bar. The username is “rstudio”, and the password is “password”.

#### 5. Render your paper in your Docker image

NB: make sure that the package versions you specify in the Dockerfile are the same as the ones you used initially in your local instance of R. You can verify these by loading all packages, then calling `sessionInfo()` in R. Also make sure that you have included everything you need to build the paper. My [GitHub](#) contains full papers written with quarto and containerised with Docker, such as my most [recent paper](#). Clone this locally for a full working example of everything detailed in this document.