Writing a reproducible paper with RStudio and Quarto

Paul C. Bauer Camille Landesvatter

First version:February 16, 2023This version: March 3, 2023

This paper provides a template to write a reproducible scientific paper with RStudio and Quarto. It falls into a series on other templates, namely on RMarkdown and pagedown that can be downloaded here (Bauer 2018) and here (Bauer and Landesvatter 2021). Below we outline some of the "tricks"/code (e.g., referencing tables, sections etc.) we had to figure out to produce this document. The underlying files which produce this document can be downloaded here. We are convinced that in the future there will be many improvements and developments with regards to RStudio and Quarto. We intend to update this file when we discover more convenient code. You can follow any updates on the github repository.

Why reproducible research (in R)?

Some arguments...

- **Access**: Research is normally funded by taxpayers (researchers are also taxpayers). Hence, it should be freely accessible to everyone without any barriers, e.g., without requiring commercial software. Importantly, researchers from developing countries are even more dependent on free access to knowledge (Kirsop and Chan 2005).
- **Reproducability**: Even if you have written a study and analyzed the data yourself you will forget what you did after a few months. A fully reproducible setup will help you to trace back your own steps. Obviously, the same is true for other researchers who may want to understand your work and built on it. It may sound like a joke but why not aim for a document that can be used to reproduce your findings in 500 years.
- **Errors**: Manual steps in data analysis (e.g., manually copy/pasting values into a table etc.) may introduce errors. R Markdown allows you to **automatize** such steps and/or avoid them.

- **Revisions**: Revising a paper takes much less time if you have all the code you need in one place, i.e., one `.qmd` file. For instance, if you decide to exclude a subset of your data you simply need to insert one line of your code at the beginning and everything is rebuilt/re-estimated automatically.

Why Quarto?

Quarto is an open-source, scientific and technical publishing system build on `pandoc`. Their creators from RStudio also call it „the next generation of R Markdown".
Like RMarkdown, Quarto uses `Knitr` to execute `R` code, and is therefore able to render most existing `.rmd` files without having to make any adjustments. As in `rmd` and other more recent developments like R's `pagedown`-package, one of the main goals of Quarto is to weave together narrative text and code and produce nicely formatted output documents.
Differences to RMarkdown are not only but especially due to two facts: first, Quarto is multilingual (R, Python, Julia, Observable JS, etc.) and second in addition to RMarkdown's workflow where the final conversion layer uses `pandoc` to create documents, Quarto uses pandoc with Lua filters (which include latex in pandoc metadata). This contributes to cross-language standardization of outputs. Put differently, it allows working regardless of language, since it allows shared syntax and shared format across languages and input formats.

Prerequisites

Quarto is a multi-language, next-generation version of R Markdown from RStudio. Multi-language means you can use R, Python, Julia and other languages to create quarto documents. In this template we'll focus on showing you how to use **RStudio with Quarto**.

To name prerequisites, we assume that you are familiar with using R and R Markdown. If you don't know what R Markdown is there are many great resources that you can use (e.g. watch this short video). An older template of ours [see Bauer (2018); https://osf.io/q395s] may provide a quick entry point to writing a reproducible manuscript with R Markdown and Latex. To further deepen your understanding of the different components underlying this template, go on and learn about R, Markdown, R Markdown and Quarto. For Quarto also consider reading their FAQ.

Like R Markdown, Quarto uses `Knitr` to execute R code (hence you should also be able to render existing Rmd files without modification), and can render documents of a dozens of formats (e.g., PDF, HTML). For a comprehensive overview watch this video which is a record of a talk introducing `quarto` given by Tom Mock from RStudio.

Then…

- …install R and Rstudio (important: most recent versions, e.g. RStudio v2022.07) (R Core Team 2017; RStudio Team 2015).
- …install the "rmarkdown"-package from CRAN using the code below (Allaire et al. 2017).

```
install.packages("rmarkdown")
```

- …install tinytex, a lightweight version of Tex Live (Allaire et al. 2017; Xie 2018b) in order to compile PDFs.

```
install.packages(c('tinytex', 'rmarkdown'))
tinytex::install_tinytex()
```

- …also install the packages below using the code below (Xie 2017, 2016, 2018a, 2015, 2014; Zhu 2017).

```
install.packages(c("knitr", "kableExtra",
                   "stargazer", "modelsummary", "knitr", "gt"))
```

- …download the 3 input files we created — `paper.qmd`, `references.bib` and `data.csv` — from this folder. Ignore the other files.
- …finally store all files from above together in one folder (and use this folder as your working directory later on).

Basics: Input files

All the input files you need to produce the present PDF file are:

- `paper.qmd` (the underlying R Markdown file).
- `references.bib` (the bibliography).
  - We use paperpile to manage references and export the `.bib` file into the folder that contains the `.qmd` file.
- `data.csv` (some raw data).

Download these files and save them into a folder. Close R/Rstudio and directly open `paper.qmd` with RStudio. Doing so assures that the working directory is set to the folder that contains `paper.qmd` and the other files.[1]

[1] You can always check your working directory in R with `getwd()`.

Basics: Output files and the YAML header

Quarto allows you to produce documents in various formats (e.g., HTML, PDF, MS Word). By default, once you run/render the `paper.qmd` file in Rstudio it creates an output file in HTML called `paper.html`. However in the YAML you can specify any alternative format (or formats) alongside different options.

Remember (cf. Section Prerequesites) that in order to compile PDFs you will need to install a recent distribution of LaTeX (e.g., TinyTex).

If you want your document to be rendered in multiple output formats (e.g., HTML and PDF like we did here), you need to specify both outputs in your YAML header. Take `paper.qmd` (the underlying quarto file of this pdf) and have a look at the YAML (line #14 - #16) to see how to specify different outputs.

Subsequently, you have two options on how to proceed for rendering:

- first, use the Render Button in R Studio. If you click it without using the drop down menu, by default it will create HTML since it is the first format listed in our YAML. If you use the drop down menu of the "Render"-button, you can specify the desired output (e.g., HTML or PDF) but only if you specified it in your YAML.
- second, if you are not using RStudio and/or you prefer to render from the R console, use the `quarto` package to render to all formats at the same time with the `quarto_render()` function:

```
quarto::quarto_render()
```

Beforehand don't forget to install the `quarto` package:

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

Here you can also again specify the name of the resulting document as well as the desired output format(s):

```
quarto::quarto_render(
  "paper.qmd",
  output_format = c("pdf", "html")
)

#quarto::quarto_render("paper.qmd") # defaults to html
```

Your output(s) will be saved in your working directory.

Referencing within your document

To see how referencing works simply see the different examples for figures, tables and sections below. For instance in Section you can find different ways of referencing tables. The code of the underlying `paper.qmd` will show you how I referenced Section right here namely with '`@sec-tables`' whereas the corresponding section title was assigned the corresponding label '`# Tables {#sec-tables}`'.

Software versioning

Software changes and gets updated, especially with an active developer community like that of R. Luckily you can always access old versions of R and old version of R packages in the archive. In the archive you need to choose a particular package, e.g dplyr and search for the right version, e.g., `dplyr_0.2.tar.gz`. Then insert the path in the following function: `install.packages("https://...../dplyr_0.2.tar.gz", repos=NULL, type="source")`. Ideally, however, results will be simply reproducible in the most current R and package versions.

I would recommend to use the command below and simply add it to the appendix as I did here in Appendix @ref(sec:rsessioninfo). This will make sure you always inform the reader about the package versions your relied on in your paper. For more advanced tools see packrat.

```
cat(paste("#", capture.output(sessionInfo()), "\n", collapse =""))
  # or use message() instead of cat()
```

Data

Import

Generally, code is evaluated by inserting regular code chunks.

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

Below we import an exemplary dataset (you can find `data.csv` in the folder with the other files).

```
data <- read.csv("data.csv")
head(data)
```

```
  Fertility Agriculture Examination Education Catholic Infant.Mortality
1      80.2        17.0          15        12     9.96             22.2
2      83.1        45.1           6         9    84.84             22.2
3      92.5        39.7           5         5    93.40             20.2
4      85.8        36.5          12         7    33.77             20.3
5      76.9        43.5          17        15     5.16             20.6
6      76.1        35.3           9         7    90.57             26.6
```

Putting your entire data into the .qmd file

Applying the function `dput()` to an object gives you the code needed to reproduce that object. So you could paste that code into your `.qmd` file if you don't want to have extra data files. This makes sense were data files are small.

```
structure(list(Fertility = c(80.2, 83.1, 92.5, 85.8, 76.9), Agriculture = c(17,
45.1, 39.7, 36.5, 43.5), Examination = c(15L, 6L, 5L, 12L, 17L
), Education = c(12L, 9L, 5L, 7L, 15L), Catholic = c(9.96, 84.84,
93.4, 33.77, 5.16), Infant.Mortality = c(22.2, 22.2, 20.2, 20.3,
20.6)), row.names = c(NA, 5L), class = "data.frame")
```

You can then insert the dput output in an rchunk in your `.qmd` as below.

```
data <- structure(list(Fertility = c(80.2, 83.1, 92.5, 85.8, 76.9), Agriculture = c(17,
45.1, 39.7, 36.5, 43.5), Examination = c(15L, 6L, 5L, 12L, 17L
), Education = c(12L, 9L, 5L, 7L, 15L), Catholic = c(9.96, 84.84,
93.4, 33.77, 5.16), Infant.Mortality = c(22.2, 22.2, 20.2, 20.3,
20.6)), class = "data.frame", row.names = c(NA, -5L))
```

Tables

When producing a scientific Quarto document, you most likely cannot skip creating good tables and cross-referencing these tables throughout your document. Examples that you may use are shown below.

Tables with kable() and kable_styling()

A great function is `kable()` (`knitr` package) in combination with `kableExtra`. **?@tbl-1** provides an example. To reference the table produced by the chunk you need to add `#| label: tbl-x` at the start of the chunk, i.e., `#| label: tbl-1` and would reference it by

adding "`@tbl-1`" in your text. See below for the full chunk code (the one that actually produces the table is hidden).

````
```{r}
#| label: tbl-1
#| tbl-cap: "Summary: Numeric variables"

library(knitr)
library(kableExtra)

kable(swiss[1:5,], row.names = TRUE,
      caption = 'Table with kable() and kablestyling()',
      booktabs = T, format = "html") %>%
        kable_styling(full_width = T,
                      latex_options = c("striped",
                                        "scale_down",
                                        "HOLD_position"),
                      font_size = 11)

```
````

Tables with modelsummary

The `modelsummary` package provides a variety of tables and plots to summarize statistical models and data in R. Modellsummary plots and tables are highly customizable and they can be saved to almost all formats, e.g., HTML, PDF and Markdown. This makes ist especially easy to embed them in dynamic documents. Please look at the package's extensive documentation where they also show examples for almost any plot or table you might be looking for. In this template we demonstrate an example for modelsummary's `datasummary` function. `Datasummary` creates frequency tables, crosstab tables, correlation tables, balance tables and many **more**.

Summarize numeric variables

?@tbl-2 shows a summary table for numeric variables.

```
library(modelsummary)
datasummary_skim(swiss,
                 type="numeric",
```

Table 1: **?(caption)**

Table 2: **?(caption)**

```
                  histogram=T,
                  title = "Summary: Numeric variables")
```

Summarize categorical variables

?@tbl-3 shows a summary table for categorical variables.

```
# Create categorical variables
swiss$Education_cat <- cut(swiss$Education,
                   breaks=c(-Inf, 6, 12, Inf),
                   labels=c("low","middle","high"))
swiss$Infant.Mortality_cat <- cut(swiss$Infant.Mortality,
                   breaks=c(-Inf, 18.15, 21.70, Inf),
                   labels=c("low","middle","high"))

library(flextable)
tab_cat <- datasummary_skim(swiss,
                            type="categorical",
                            output = 'flextable')

# additionally we want to change the font, fontsize and spacing
library("gdtools")

library(dplyr)
tab_cat <- tab_cat %>%
  font(fontname="Times New Roman", part="header") %>%
  font(fontname="Times New Roman", j=1:4) %>%
  fontsize(size=12, part="header") %>%
  fontsize(size=10, j=1:4) %>%
  line_spacing(space = 0.3, part = "all")

tab_cat
```

Regression table

?@tbl-4 shows the output for a regression table. Make sure you name all the models you estimate (even if its 50) and explicitly refer to model names (M1, M2 etc.) in the text.

Inline code & results

Reproduction reaches new heights when you work with inline code. For instance, you can automatize the display of certain coefficients within the text. An example is to include estimates, e.g., the coefficient of `dist` of the model we ran above. `#r# rinline("round(coef(model1)[2], 2)")` will insert the coefficient as follows: `#r# round(coef(model1)[2], 2)`. Or `#r# rinline("3 + 7")` will insert a 10 in the text.
Inline code/results that depend on earlier objects in your document will automatically be updated once you change those objects. For instance, imagine a reviewer asks you to omit certain observations from your sample. You can simply do so in the beginning of your code and push play subsequently.. at time you might have to set `cache = FALSE` at the beginning so that all the code chunks are rerun.
Researchers often avoid referring to results in-text etc. because you easily forget to change them when revising a manuscript. At the same it can make an article much more informative and easier to read, e.g., if you discuss a coefficient in the text you can directly show it in the section in which you discuss it. Inline code allows you to do just that. R Markdown allows you to that do so in a reproducible and automatized manner.

Graphs

R base graphs

Inserting figures can be slightly more complicated. Ideally, we would produce and insert them directly in the `.qmd` file. It's relatively simple to insert R base graphs as you can see in (**fig1?**).

But it turns out that it doesn't always work so well.

ggplot2 graphs

Same is true for ggplot2 as you can see in (**fig2?**).

Compiling the document

To view your paper, pagedown requires a web server (since it is based on paged.js)[2]. By compiling a document, R Studio will display your HTML page through a local web server, i.e., paged.js will work in RStudio Viewer.

There are several options, depending on your intention:

- click on the `Knit` button in R Studio which by default will provide a HTML document in the RStudio viewer pane (the HTML will be stored in your working directory)
- use pagedown's `chrome_print` function in the YAML (uncomment line #24 of this `Rmd` file) if additionally you want your HTML based web page to be printed to be PDF (the PDF will be stored in your working directory)
- to "live"-preview your pages do not click on the `Knit` button but use the **xaringan** (Xie 2021) RStudio add-in *Infinite Moon Reader*. You can simply call the function `xaringan::inf_mr()` (within your console). This will launch a local web server via the **servr** package (Xie 2021a) and display your pages in the RStudio viewer. Each time you save your document (*Ctrl+S*) xaringan updates your pages in the viewer.
- If you use the option `self_contained: false` (see line #21 of this `Rmd` file) (change to true for a self-contained document, but it'll be a litte slower for Pandoc to render), don't click on the `Knit` button in RStudio. Use instead the **xaringan** (Xie 2021) RStudio add-in *Infinite Moon Reader*.

Good practices for reproducibility

Every researcher has his own optimized setup. Currently we would recommend the following:

- Keep all files of your project (that matter for producing the PDF) in one folder without subfolders. You can zip and directly upload that folder to the Harvard dataverse.[3]
- Make sure that filenames have a logic to them.
  - Main file with text/code: "paper.qmd", "report.qmd"
  - Data files: "data_xxxxxx.*"
  - Image files: "fig_xxxxxx.*"
  - Tables files: "table_xxxx.*"
  - etc.

[2] open-source library to paginate content in the browser

[3] Another good folder setup would be to store all files needed as input files for the R Markdown manuscript in a subfolder called "input" and all output files that are produced apart from paper.html and paper.pdf in a subfolder called "output".

  - Ideally, your filenames will correspond to the names in the paper. For instance, Figure 1 in the paper may have a corresponding file called `fig_1_xxxxx.pdf`.

- Use the document outline in R studio (Ctrl + Shift + O) when you work with R Markdown.
- Name rchunks according to what they do or produce:
  - "`fig-...`" for chunks producing figures
  - "`table-...`" for chunks producing tables
  - "`model-...`" for chunks producing model estimates
  - "`import-...`" for chunks importing data
  - "`recoding-...`" for chunks in which data is recoded
- Use "really" informative variable names:
  - Q: What do you think does the variable *trstep* measure? It actually measures trust in the European parliament.
    - How could we call this variable instead? Yes, `trust.european.parliament` which is longer but will probably be understood by another researcher in 50 years.
  - If your setup is truly reproducible you will probably re-use the variable names that you generate as variable names in the tables you produce. Hence, there is an incentive to use good names.
- Use unique identifiers in the final R Markdown document paper.qmd that you upload:
  - Think of someone who wants to produce Figure 1/Model 1/Table 1 in your paper but doesn't find it in your code…
    - Name the chunks "fig-1", "fig-2" as the are named in the published paper.
    - Name the chunks that produce tables "table-1", "table-2" etc. as they are named in the published paper.
    - Name your statistical models in your R code "M1", "M2" as they are named in the published paper.

Additional tricks for publishing

- Make your script anonymous
  - Simply put a `<!-- ... -->` around any identifying information, e.g., author names, title footnote etc.
- Counting words
  - Use adobe acrobat (commerical software) to convert your file to a word file. Then open in word and delete all the parts that shouldn't go into the word count. The word count is displayed in the lower right.

  - Use an one of the online services to count your words (search for "pdf word count")
- Appendix: You can change the numbering format for the appendix in the rmd file
  - What is still not possible in this document is to automatically have separate reference sections for paper and appendix.
- Journals may require you to use their tex style: Sometimes you can simply use their template in your rmarkdown file. See here for a PLOS one example.

Citation styles

If your study needs to follow a particular citation style, you can set the corresponding style in the header of your `.qmd` document. To do so you have to download the corresponding `.csl` file.

In the present document we use the style of the American Sociological Association and set it in the preamble with `csl: american-sociological-association.csl`. However, you also need to download the respective `.csl` file from the following github page: https://github.com/citation-style-language/styles and copy it into your working directory for it to work.

The github directory contains a wide variety of citation style files depending on what discipline you work in.

References

Allaire, JJ, Jeffrey Horner, Vicent Marti, and Natacha Porte. 2017. *Markdown: 'Markdown' Rendering for r*. https://CRAN.R-project.org/package=markdown.

Bauer, Paul. 2018. "Writing a Reproducible Paper in R Markdown." *Open Science Framework Preprint*, 1–14.

Bauer, Paul, and Camille Landesvatter. 2021. "Writing a Reproducible Paper with R Markdown and Pagedown." *Open Science Framework Preprint*, 1–15.

Kirsop, Barbara, and Leslie Chan. 2005. "Transforming Access to Research Literature for Developing Countries." *Serials Review* 31 (4): 246–55.

R Core Team. 2017. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

RStudio Team. 2015. *RStudio: Integrated Development Environment for r*. Boston, MA: RStudio, Inc. http://www.rstudio.com/.

Xie, Yihui. 2014. "Knitr: A Comprehensive Tool for Reproducible Research in R." In *Implementing Reproducible Computational Research*, edited by Victoria Stodden, Friedrich Leisch, and Roger D. Peng. Chapman; Hall/CRC. http://www.crcpress.com/product/isbn/9781466561595.

———. 2015. *Dynamic Documents with R and Knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. https://yihui.name/knitr/.

———. 2016. *Bookdown: Authoring Books and Technical Documents with R Markdown*. Boca Raton, Florida: Chapman; Hall/CRC. https://github.com/rstudio/bookdown.

———. 2017. *Bookdown: Authoring Books and Technical Documents with r Markdown*. https://github.com/rstudio/bookdown.

———. 2018a. *Knitr: A General-Purpose Package for Dynamic Report Generation in r*. https://yihui.name/knitr/.

———. 2018b. *Tinytex: Helper Functions to Install and Maintain 'TeX Live', and Compile 'LaTeX' Documents*. https://CRAN.R-project.org/package=tinytex.

———. 2021. *Xaringan: Presentation Ninja*. https://CRAN.R-project.org/package=xaringan.

Zhu, Hao. 2017. *kableExtra: Construct Complex Table with 'Kable' and Pipe Syntax*. https://CRAN.R-project.org/package=kableExtra.

Online appendix

Attach R session info in appendix

Since R and R packages are constantly evolving you might want to add the R session info that contains information on the R version as well as the packages that are loaded.

```
R version 4.2.2 (2022-10-31)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Ventura 13.0.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

loaded via a namespace (and not attached):
 [1] compiler_4.2.2  magrittr_2.0.3  fastmap_1.1.0   cli_3.3.0
 [5] tools_4.2.2     htmltools_0.5.2 rstudioapi_0.13 yaml_2.3.5
 [9] stringi_1.7.6   rmarkdown_2.20  knitr_1.39      stringr_1.4.0
[13] xfun_0.37       digest_0.6.29   jsonlite_1.8.0  rlang_1.0.6
[17] evaluate_0.15
```

All the code in the paper

To simply attach all the code you used in the PDF file in the appendix see the R chunk in the underlying `.qmd` file:

```
knitr::opts_chunk$set(cache = FALSE)
# Use cache = TRUE if you want to speed up compilation
# A function to allow for showing some of the inline code
rinline <- function(code){
  html <- '<code  class="r">``` `r CODE` ```</code>'
  sub("CODE", code, html)
}
install.packages("rmarkdown")
install.packages(c('tinytex', 'rmarkdown'))
tinytex::install_tinytex()
install.packages(c("knitr", "kableExtra",
```

```
                  "stargazer", "modelsummary", "knitr", "gt"))
quarto::quarto_render()
install.packages("quarto")
quarto::quarto_render(
  "paper.qmd",
  output_format = c("pdf", "html")
)

#quarto::quarto_render("paper.qmd") # defaults to html
cat(paste("#", capture.output(sessionInfo()), "\n", collapse =""))
  # or use message() instead of cat()
x <- 1:10
x
data <- read.csv("data.csv")
head(data)
dput(data[1:5,]) # here we only take a subset
data <- structure(list(Fertility = c(80.2, 83.1, 92.5, 85.8, 76.9), Agriculture = c(17,
45.1, 39.7, 36.5, 43.5), Examination = c(15L, 6L, 5L, 12L, 17L
), Education = c(12L, 9L, 5L, 7L, 15L), Catholic = c(9.96, 84.84,
93.4, 33.77, 5.16), Infant.Mortality = c(22.2, 22.2, 20.2, 20.3,
20.6)), class = "data.frame", row.names = c(NA, -5L))
library(knitr)
library(kableExtra)

kable(swiss[1:5,], row.names = TRUE,
      caption = 'Table with kable() and kablestyling()',
      booktabs = T, format = "html") %>%
        kable_styling(full_width = T,
                      latex_options = c("striped",
                                        "scale_down",
                                        "HOLD_position"),
                      font_size = 11)


library(modelsummary)
datasummary_skim(swiss,
                 type="numeric",
                 histogram=T,
                 title = "Summary: Numeric variables")
# Create categorical variables
swiss$Education_cat <- cut(swiss$Education,
```

```
                    breaks=c(-Inf, 6, 12, Inf),
                    labels=c("low","middle","high"))
swiss$Infant.Mortality_cat <- cut(swiss$Infant.Mortality,
                    breaks=c(-Inf, 18.15, 21.70, Inf),
                    labels=c("low","middle","high"))

library(flextable)
tab_cat <- datasummary_skim(swiss,
                            type="categorical",
                            output = 'flextable')

# additionally we want to change the font, fontsize and spacing
library("gdtools")

library(dplyr)
tab_cat <- tab_cat %>%
  font(fontname="Times New Roman", part="header") %>%
  font(fontname="Times New Roman", j=1:4) %>%
  fontsize(size=12, part="header") %>%
  fontsize(size=10, j=1:4) %>%
  line_spacing(space = 0.3, part = "all")

tab_cat
library(modelsummary)
M1 <- lm(Fertility ~ Education + Agriculture, data = swiss)
M2 <- lm(Fertility ~ Education + Catholic, data = swiss)
M3 <- lm(Fertility ~ Education + Infant.Mortality + Agriculture, data = swiss)
models <- list("M1" = M1, "M2" =  M2, "M3" = M3)


library(gt)
# additionally we want to change the font, font size and spacing
modelsummary(models,
             output = 'gt',
             notes = "Notes: some notes...") %>%
    tab_spanner(label = 'Dependent variable: Fertility', columns = 2:4) %>%
  tab_options(
    table.font.size = 10,
    data_row.padding = px(1),
    table.border.top.color = "white",
    heading.border.bottom.color = "black",
```

```
    row_group.border.top.color = "black",
    row_group.border.bottom.color = "white",
    table.border.bottom.color = "white",
    column_labels.border.top.color = "black",
    column_labels.border.bottom.color = "black",
    table_body.border.bottom.color = "black",
    table_body.hlines.color = "white"
  )


plot(swiss$Catholic, swiss$Fertility)
library(ggplot2)
ggplot(swiss, aes(x=Catholic, y=Fertility, shape=Education_cat)) + geom_point() +
  labs(x="Agriculture", y = "Fertility",
       shape="Education") + theme_classic()
print(sessionInfo(), local = FALSE)
```