Example APA7 Manuscript Made with Quarto and apaquarto

Natalie Dowling

MA Program in the Social Sciences, University of Chicago

Author Note

This example is based on the apaquarto template, created by W. Joel Schneider as part of the apaquarto extension for the Quarto publishing system. The author is grateful Dr. Marisa Casillas for creating the original version of this course. Author roles were classified using the Contributor Role Taxonomy (CRediT; https://credit.niso.org/) as follows: Natalie Dowling: conceptualization, writing

Correspondence concerning this article should be addressed to Natalie Dowling, MA Program in the Social Sciences, University of Chicago, 1155 E 60th St., Chicago, IL 60637, USA, Email: ndowling@uchicago.edu

Abstract

This document is a template demonstrating the apaquarto format. It includes examples of how to create figures and tables, as well as how to reference them in the text. The document is written in Quarto, a system for creating documents with R Markdown. The apaquarto extension provides a template for creating APA7-formatted manuscripts.

Keywords: R programming, ggplot2, data communication

Example APA7 Manuscript Made with Quarto and apaquarto

Introduction

This document is a template demonstrating the apaquarto format, to be used in the course From Data to Manuscript in R (Dowling, 2025). It includes examples of how to create figures and tables, as well as how to reference them in the text. The document is written in Quarto, a system for creating documents with R Markdown. The apaquarto extension provides a template for creating APA7-formatted manuscripts.

When rendered into html, pdf, or Word, this example produces an APA styled manuscript. Although the contents of the manuscript are not what you would expect in a psychology journal article, the formatting should demonstrate both the capabilities of the apaquarto extension and the basic template an actual manuscript would follow.

You can learn more about APA style in the APA Style Manual. Details about creating documents using Schneider (2025) in the documentation.

The example demonstrates the mechanics of markdown manuscripts using Quarto and the apaquarto extension, specifically. While many of the topics covered are the same in other markdown systems, like the older R Markdown or the `papaja` package, the specific syntax and options may differ.

Your introduction should be a clear and concise explanation of the research question, the background of the problem, and the purpose of the study. It should include both a Literature Review and Present Study section.

Literature Review

The literature review includes citations to relevant research and other sources. Citing sources requires formatting according to some standardized style, like APA or Chicago Manual of Style. In this document, build with apaquarto, you should not be surprised to learn that we use APA style.

Information about your references are stored in a consolidated .bib file, typically located in the same directory as your manuscript and conventionally named `bibliography.bib`. The bib file contains all the information needed to cite a source, including the author, title, publication date, and other relevant information. The information for any single source is attached to a unique key, which you can then use to cite the source in your text. You can reference a source from your bibliography the same way you'd reference a figure, table, or document section: with the `@` symbol followed by the key.

For example, many of the readings for this class come from Hadley Wickham's book *R for Data Science*. To cite this book, you would use `@wickhametal_2023_data` (the key assigned to it in the bibliography file). This will render as (Wickham, 2023) in the text and as a full citation in the references section at the end of the document.

There are a few different formats you'll need to know for citing sources in your text. Most commonly, you'll want to cite the author(s) and the year of publication within a parenthetical citations (e.g., "Wickham, 2023"). Also common is the author name(s) in the text and then only the year in parentheses (e.g., "Wickham (2023)").

There's simple syntax to do each of these with bibtex keys, but you can do more complex things, too. You can (among other things) cite multiple sources at once and have them automatically be alphabetized and formatted, specify chapters and page number, and add in-parenthetical information like "see also" or "for more information." Let's see some in action:

Dowling (2025) claims that Quarto is awesome. In fact, in Dowling's UChicago course on the topic (2025), she never seems to stop saying that RMarkdown is awesome. She *really* likes RMarkdown (Dowling, 2025). Of course, it is well established that RMarkdown and Quarto are, indeed, awesome (Dowling, 2025; Schneider, 2025; Xie, 2018). Nearly everyone agrees on this very obvious point (Dowling, 2025; e.g., Xie, 2018).

You can include notes, like the "e.g.," in the examples above, by simply adding it at the beginning of the list of keys. The added text will get "attached to" the first citation, so it's important that you put it before whichever citation will actually show up first.

For the most part, BibTeX will also handle alphabetical ordering of multiple citations within the same parenthetical citation (Dowling, 2025; Wickham, Cetinkaya-Rundel, et al., 2023).

That means that in the example in the previous paragraph, the text is attached to the "Xie" citation and so will render between "Dowling" and "Xie" once they are ordered.

Oddly, if you put the extra text after the semicolon, it seems to just stop trying to order the citations and will render them in the order you put them in the text.

If you use citr to add references, it will do the sorting before it adds the keys to your document, so you don't have to worry about any of that

In BibTeX, same-year, same-author publications are automatically disambiguated (Heritage, 2012b, 2012a).

You can also include page numbers or chapters in your citations. For example, you can cite a specific chapter in a book like this: (Wickham, 2016a, ch. 1). This will render as (Wickham, 2016, ch. 1) in the text and as a full citation in the references section at the end of the document. We learned all about ggplot (see Wickham, 2016a, ch. 1). The best introduction to ggplot is with Wickham (2016a, ch. 1).

You know what else is awesome besides RMarkdown? Interactive gestures, which allow interlocutors to expediently facilitate conversation nonverbally (Bavelas et al., 1992; Dowling, 2022).[1]

Brackets and hyphens allow you to specify which part of the citation to include in parentheses (or at all). Briefly, brackets say the whole thing (whatever that may be) should be in parentheses and hyphens say the author name should be excluded.

- Brackets: `[@dowling_2025_data]` (Dowling, 2025)
- No brackets: `@dowling_2025_data` Dowling (2025)
- Brackets and hyphen: `[-@dowling_2025_data]` (2025)

Present Study

The present "study" is a demonstration of the capabilities of the apaquarto extension for Quarto. It's not a study, but whatever. This is the present study section.

[1] Sure this isn't relevant to anything else, but it's a great excuse to demo `citr` and talk about gesture at the same time!

Methods

In this section, we will go over how to create, render, and reference tables in apaquarto documents.

At some point in your methods section, you'll want to say that you used R, and you may want to call out specific packages used for statistical analyses or plotting. You can do that automatically with the `papaja::r_refs()` and `papaja::cite_r()` functions.

This process has three steps:

1. Modify the YAML bibliography option to be a bracketed list of both the .bib file you're using for your regular references and a separate .bib file for your R references. For example, `bibliography: [bibliography.bib, r-references.bib]`.
2. In a setup chunk at the beginning of your document, call `papaja::r_refs()` to generate the references for the packages you used in the specified .bib file. Run this *after* you load the packages you want to cite. Useful arguments:

    1. (Required) `file`: The name of the file you specified in the YAML bibliography option. The function will create the file if it doesn't exist and overwrite it if it does.
    2. (Optional) `append = TRUE`: Set to `FALSE` to overwrite the file rather than add on missing references to the end. Useful if you are trying out different packages and know you won't end up using them all.
    3. (Optional) `tweak = TRUE`: This function is a wrapper for `create_bib()`, and `tweak = TRUE` fixes some known issues with `create_bib()`. If it's not working as expected, you can try setting `tweak = FALSE` and see if that helps.

3. In the text, call `papaja::cite_r()` to insert the references where you want them. Useful arguments (all are optional):

    1. (Treat this like it's required) `file = NULL`: To use this like we want, set this to name of the file you specified in the YAML bibliography option. This is the same file you

used in `r_refs()`. If you don't specify it, the default is `NULL`, which will cite `R` itself but no packages.

2. `footnote = FALSE`: By default, prints where you put it. If you set this to `TRUE`, it will print the references as a footnote. This is useful if you have a lot of packages and don't want to clutter up your text.
3. `pkgs = NULL`: If you only want to cite specific packages, you can list them here. This is useful if you have a lot of packages and only want to cite a few.
4. `omit = FALSE`: Use together with `pkgs`. Set this to `TRUE` to *omit* the packages listed in `pkgs` from the citation.

The `r_refs()` function will generate the references for the packages you used in the specified .bib file. The `cite_r()` function will insert the references where you put it in the text. The YAML option will tell Quarto to combine the references from this file with the references from your regular bibliography file.

We used R (Version 4.3.3; R Core Team, 2024) and the R-packages *broom* (Version 1.0.7; Robinson et al., 2024), *conflicted* (Version 1.2.0; Wickham, 2023a), *dplyr* (Version 1.1.4; Wickham, François, et al., 2023), *flextable* (Version 0.9.7; Gohel & Skintzos, 2024), *forcats* (Version 1.0.0; Wickham, 2023b), *ftExtra* (Version 0.6.4; Yasumoto, 2024), *ggplot2* (Version 3.5.1; Wickham, 2016b), *knitr* (Version 1.48; Xie, 2015), *lubridate* (Version 1.9.3; Grolemund & Wickham, 2011), *papaja* (Version 0.1.3; Aust & Barth, 2024), *purrr* (Version 1.0.2; Wickham & Henry, 2025), *readr* (Version 2.1.5; Wickham, Hester, et al., 2024), *shiny* (Version 1.9.1; Chang et al., 2024), *stringr* (Version 1.5.1; Wickham, 2023c), *tibble* (Version 3.2.1; Müller & Wickham, 2023), *tidyr* (Version 1.3.1; Wickham, Vaughan, et al., 2024), *tidyverse* (Version 2.0.0; Wickham et al., 2019) and *tinylabels* (Version 0.2.4; Barth, 2023) to prepare this document.

There are other ways to do this, and considering papaja doesn't get regular updates it may be best to find a more reliable alternative. You can read about one other way to do so here and do some googling for other options. At least for now the papaja functions are working and (IMO) easier to use.

Tables

In this section, we will go over how to create, render, and reference tables in apaquarto documents.

For tables produced by executable code cells, include a label with a `tbl-` prefix to make them cross-referenceable. For example:

Table 1

Iris Data (First 6 Rows)

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.10	3.50	1.40	0.20	setosa
4.90	3.00	1.40	0.20	setosa
4.70	3.20	1.30	0.20	setosa
4.60	3.10	1.50	0.20	setosa
5.00	3.60	1.40	0.20	setosa
5.40	3.90	1.70	0.40	setosa

Note. The iris dataset is a classic dataset in R.

In the chunk above, I produce Table 1, which is a table of the first six rows of the `iris` dataset. It renders from the chunk (it is not saved to an object) at the location of the chunk in the manuscript. I can reference this table in the text as `@tbl-iris`, which will render as "Table 1" in the rendered document.

Table 1, when rendered, has three components:

1. The table itself
2. The table *label*, which is the word "Table" and the generated number in the order it was rendered

3. The table caption, which is the title of the table

Take note that the `tbl-cap` option that assigned the "caption" is actually assigning with APA7 would refer to as the "title" of the table.

There are several important things to note about the simple figure chunk above:

1. The chunk options are in the "comment style" format. They are preceded by `#|` and are within the chunk, not the chunk header (the "'`{r}` part).
2. The first chunk option, label, is in the line *immediately following the chunk header*. This is important for the table to be recognized as a table. If there is anything above the label, including comments and whitespace, the table will not render as a table and the document may not render at all without error
3. The label begins with "tbl-", which tells apaquarto that this is a table. If the chunk has any other name, it will still render images in the chunk (including generated plots), but they will not be recognized as tables.
4. The caption in the `tbl-cap` option. As discussed above, this "caption" is the title of the table and should be in title case, with no period.
5. The `apa-note` *apaquarto specific* option is a note that appears below a table or figure (the chunk option can be used in with either). This is the explanatory text that you might think of as a "caption" in other contexts. When rendered, "*Note.*" will appear before the note text; do not add "Note" to the text of the note.

While there are many approaches to creating tables, the apaquarto extension works well with the flextable package. As you can see in Table 1, `theme_apa()` function from the flextable package will format tables to APA style. It's not always perfect, but it's very close. Given that this is a very simple process, it's the best "bang for your buck," so to speak, in terms of effort and output. The process is simply:

1. Create a table using any method you like. The flextable package works well with tibbles, but you can use any method you like.

2. Pipe the table/tibble into `flextable()`.
3. Pipe the flextable into `flextable::theme_apa()`. The `flextable::` prefix is not required, but is recommended to avoid confusion with the `theme_apa()` function for ggplots from the papaja package, which is not compatible with the apaquarto extension.

Table 2

Mtcars Data Summary Data

Cylinders	MPG (mean)	MPG (sd)	N
4.00	26.66	4.51	11
6.00	19.74	1.45	7
8.00	15.10	2.56	14

Note. The mtcars dataset is a classic dataset in R.

For relatively simple analysis output, you can use the `broom` package to create tibbles from model output. For example, here I run a linear regression and then use `broom::tidy()` to create a tibble of the model output.

Table 3

Mtcars Data Linear Model

term	estimate	std.error	statistic	p.value
(Intercept)	37.23	1.60	23.28	0.00
wt	-3.88	0.63	-6.13	0.00
hp	-0.03	0.01	-3.52	0.00

Note. The broom package creates tidy regression output.

More complex analysis output may require more complex solutions. The `stargazer` package is designed to create tables from model output, but those tables need tweaking to be APA style.

The `apaTables` package is designed for precisely this purpose (rendering APA-publication-ready output for common psych analyses), but I have found it to be unreliable and occasionally even produce fatal R errors that crash the whole R session in addition to the render.

Results

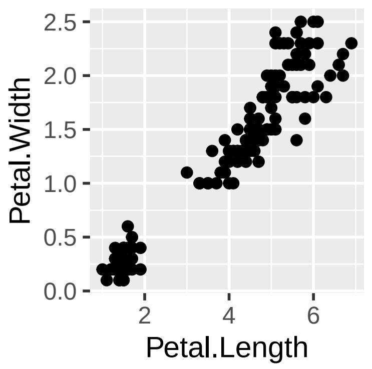
Figures can be created in R chunks and rendered as figures in the text. They can be referenced in the text and will be numbered in the order they are rendered.

Figure Chunk Options

Figure chunks should use the quarto-preferred "comment style" chunk option settings. Minimally, they should include a label and a caption. The label should begin with "fig-" to be recognized as a figure. The caption should be a string in title case. The image (file or object) should be rendered in the chunk.

Figure 1

The "Caption" (aka Title) of a Rendered Plot



In the chunk above, I produce Figure 1 a scatterplot of the `iris` dataset. It renders from the chunk (it is not saved to an object) at the location of the chunk in the manuscript. I can reference this plot in the text as `@fig-iris-rendered-plot`, which will render as "Figure 1" in the rendered document.

Figure 1, when rendered, has three components:

1. The plot itself
2. The figure *label*, which is the word “Figure” and the generated number in the order it was rendered
3. The figure caption, which is the title of the plot

Take note that the `fig-cap` option that assigned the “caption” is actually assigning with APA7 would refer to as the “title” of the figure. The thing you probably think of when you hear caption – the explanatory text below the figure – is called a “note.” There is an option to include a note, but it is not required or included in this minimal example.

There are several important things to note about the simple figure chunk above:

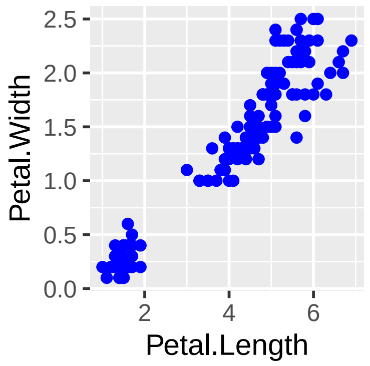
1. The chunk options are in the “comment style” format. They are preceded by `#|` and are within the chunk, not the chunk header (the “‘`{r}` part).
2. The first chunk option, label, is in the line *immediately following the chunk header*. This is important for the figure to be recognized as a figure. If there is anything above the label, including comments and whitespace, the figure will not render as a figure and the document may not render at all without error
3. The label begins with “fig-”, which tells apaquarto that this is a figure. If the chunk has any other name, it will still render images in the chunk (including generated plots), but they will not be recognized as figures.
4. The caption in the `fig-cap` option. As discussed above, this “caption” is the title of the figure and should be in title case, with no period.

Figure 1 is rendered by actually creating a ggplot in the chunk. However, you can also create a ggplot object elsewhere and render it as a figure. This is useful if you want to use the same plot in multiple places in the document, or if you want to create a very complicated plot in a sourced script.

Creating a chunk that assigns a plot to an object does not render the plot, since R code that assigns an object does not return anything. It only creates the object `iris_plot`. To render the plot, you need to call the object in a chunk with the figure options.

Figure 2

The Caption of a Rendered Plot Object



I can render the plot in a separate chunk by calling the plot object `iris_plot` in the chunk. This will render the plot as a figure in the text. I can reference this plot as Figure 2.

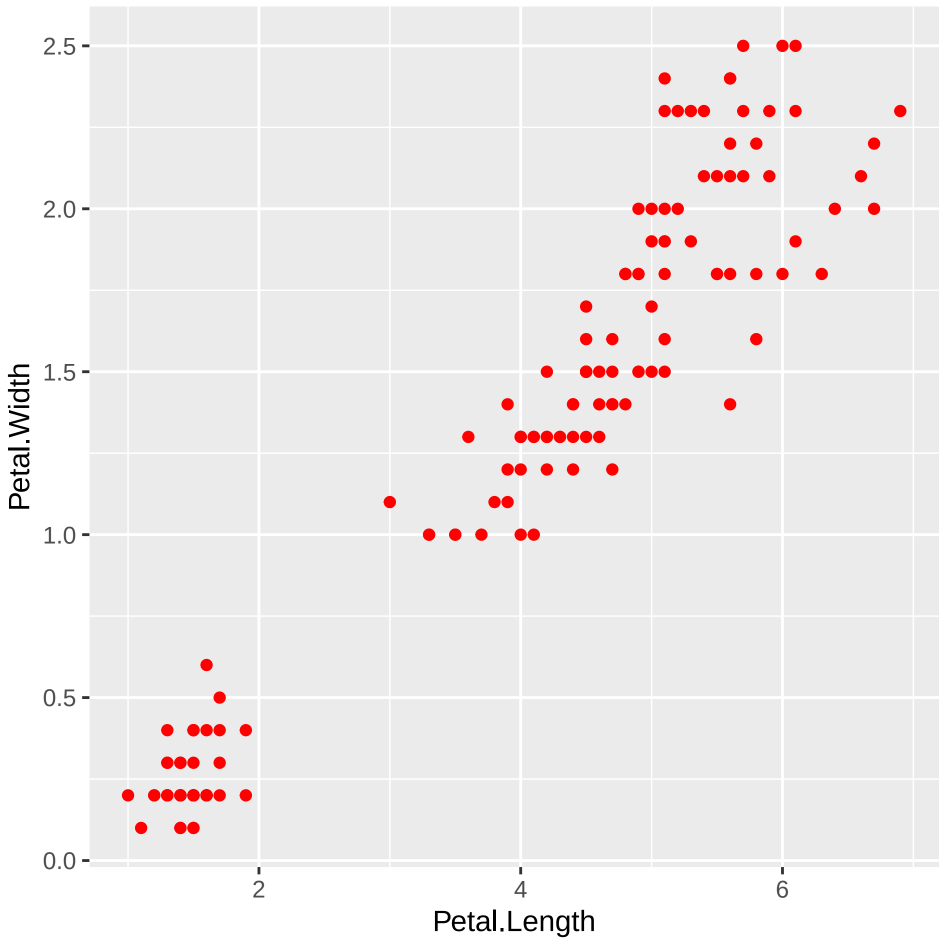
Quarto, and the apaquarto extension, can accept many additional chunk options. Like the label and caption options, these are set in the comment style within the chunk. Here are some additional options that can be used in figure chunks:

1. `apa-note`: A note that appears below the figure caption. This is the explanatory text that you might think of as a "caption" in other contexts. Unlike the options that follow, this is a feature of apaquarto specifically, not markdown or Quarto.
2. `fig-scap`: A short caption that appears in the list of figures. This is useful for long figure captions that are unwieldy in a list of figures.
3. `fig-alt`: Alt text for accessibility. This will appear if you render as HTML and is useful for screen readers.
4. `fig-align`: The alignment of the figure. This can be "left", "right", or "center" (the default).

5. `fig-width` and `fig-height`: The width and height of the figure in inches. This is useful for controlling the size of the figure in the rendered document. The default is 7 inches wide and 5 inches tall.

Figure 3

A Plot With More Chunk Options



Note. A note appearing below the figure.

You can see all the effects of these chunk options in Figure 3. The figure is aligned to the right, has a width of 5 inches and a height of 6 inches, and has a note below the caption. The short caption is what will appear in the list of figures, and the alt text will appear when a reader mouses over the figure in an HTML document (it does not do anything in pdf or Word documents).

Referencing figures. As seen in the text above, figures can be referenced in text using the `@` symbol followed by the figure label. This will render as "Figure X" in the rendered document, where X is the order in which the figure was rendered. The figure label should be unique and begin with "fig-" to be recognized as a figure.

Plots are the only thing that count as figures. In APA documents, all images are typically treated as figures. You can include images as figures in your document by rendering them in a chunk using the `include_graphics()` function from the knitr package. Like with any other figure, the chunk required a label beginning with "fig-" and a caption and may take additional chunk options.

For images, it's usually better to use `out-width` or `out-height` rather than `fig-width` and `fig-height`. The image files already have an inherent size (unlike rendered plots), which can create problems when you try to give them new absolute dimensions (with the `fig-` options). The `out-` options let you use a relative sizing as a percentage, which is usually more reliable.

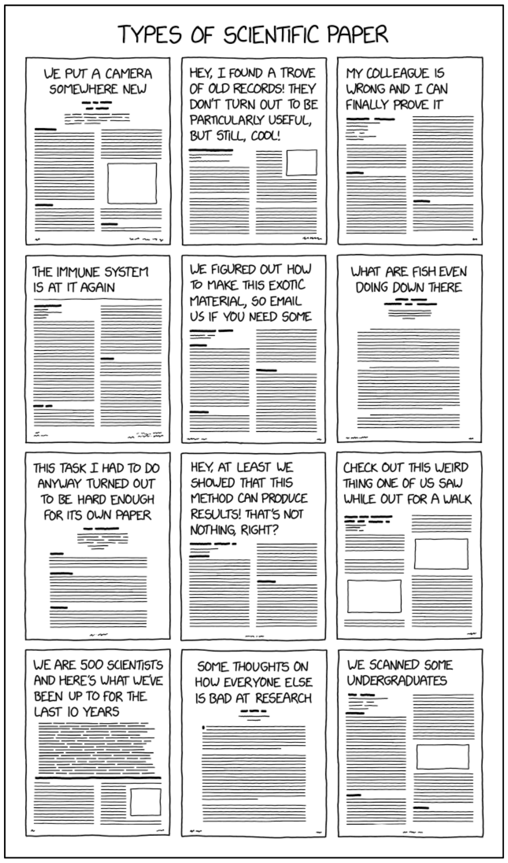
Why not take the opportunity to look at a few more xkcd comics?

When you knit this document to a pdf, knitr/Quarto will try to pick the best location to include it. If your image is very large, or if there is very little text between images, the location it chooses may not be precisely where you put the chunk. You can force the image to render exactly where the chunk is with the option `fig-pos: "H"`. Either way–the default or the forced hold–can produce unexpected consequences, so keep an eye out for issues and try out alternatives as needed.

Back to xkcd. Here's a good one about types of eclipses:

Figure 4

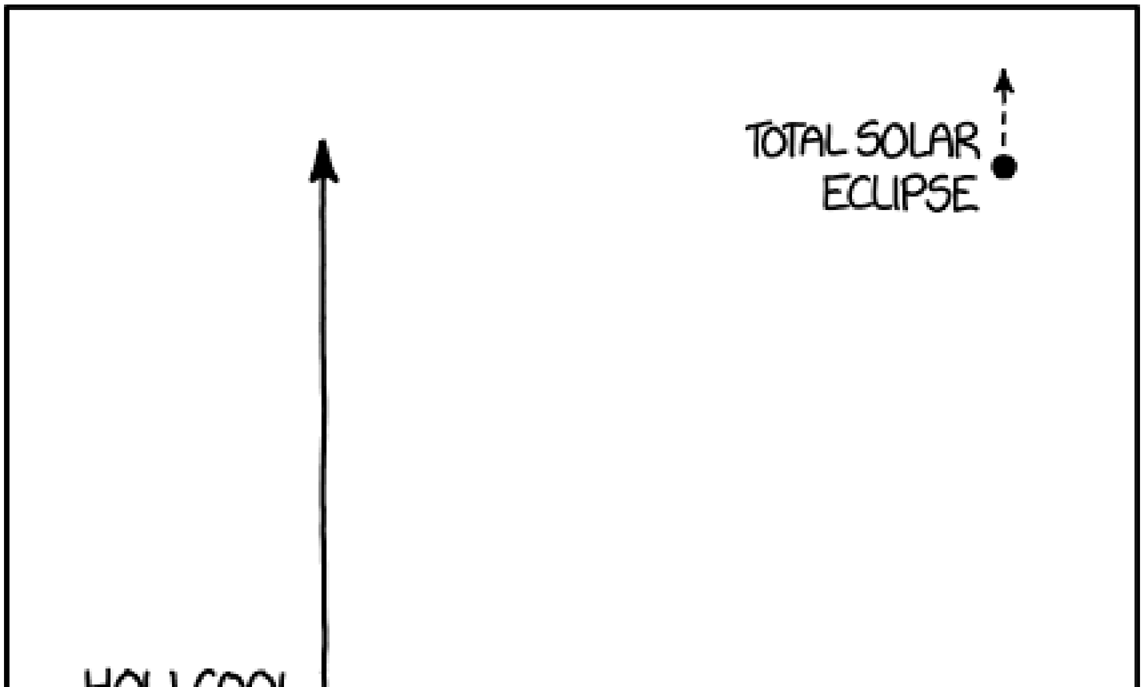
Types of Scientific Papers


TYPES OF SCIENTIFIC PAPER
WE PUT A CAMERA SOMEWHERE NEW
HEY, I FOUND A TROVE OF OLD RECORDS! THEY DON'T TURN OUT TO BE PARTICULARLY USEFUL, BUT STILL, COOL!
MY COLLEAGUE IS WRONG AND I CAN FINALLY PROVE IT
THE IMMUNE SYSTEM IS AT IT AGAIN
WE FIGURED OUT HOW TO MAKE THIS EXOTIC MATERIAL, SO EMAIL US IF YOU NEED SOME
WHAT ARE FISH EVEN DOING DOWN THERE
THIS TASK I HAD TO DO ANYWAY TURNED OUT TO BE HARD ENOUGH FOR ITS OWN PAPER
HEY, AT LEAST WE SHOWED THAT THIS METHOD CAN PRODUCE RESULTS! THAT'S NOT NOTHING, RIGHT?
CHECK OUT THIS WEIRD THING ONE OF US SAW WHILE OUT FOR A WALK
WE ARE 500 SCIENTISTS AND HERE'S WHAT WE'VE BEEN UP TO FOR THE LAST 10 YEARS
SOME THOUGHTS ON HOW EVERYONE ELSE IS BAD AT RESEARCH
WE SCANNED SOME UNDERGRADUATES

Note. From xkcd (#2456) by Randall Monroe

Figure 5

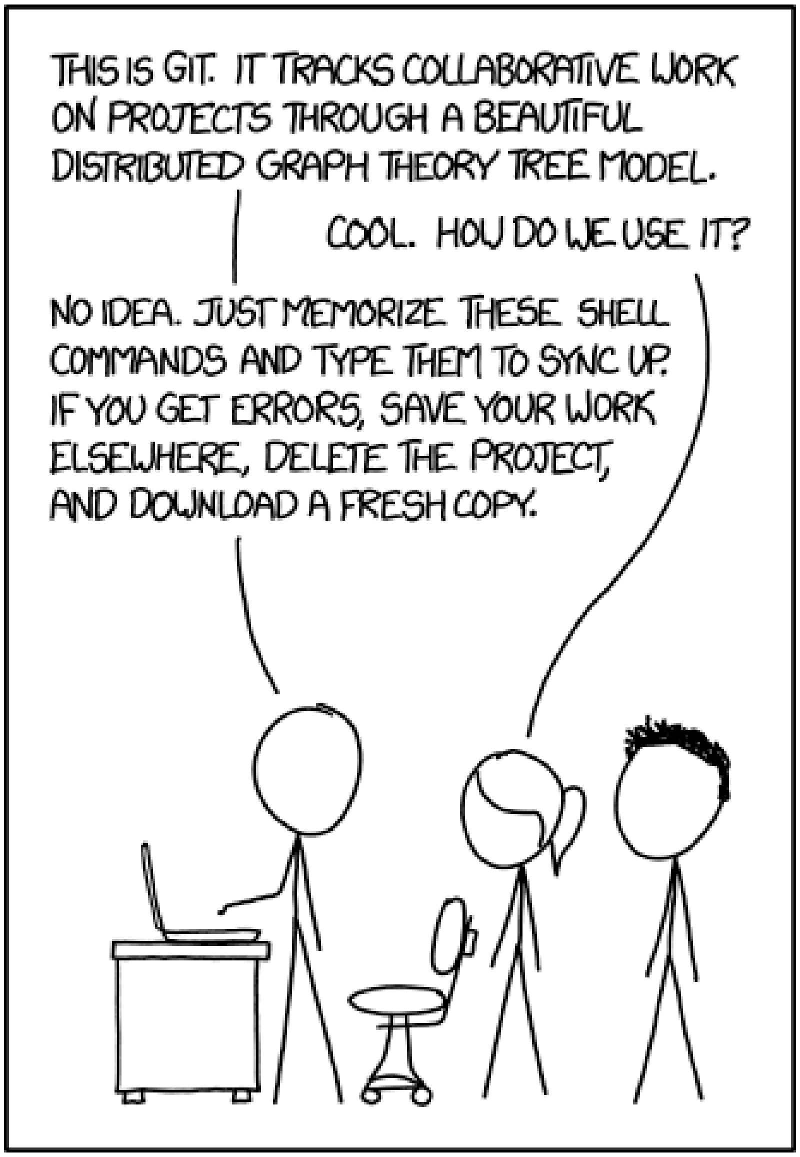
Types of Eclipses



And of course there’s this classic about literally everyone’s experience using git:

Figure 6

The Tragedy of git


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Note. From xkcd (#1597) by Randall Monroe

Love a good xkcd. They’re all great, but right now Figure 6 really speaks to me. (You too, maybe?) Figure 5 is good and it’s got a plot, but it’s not as relevant to our class as Figure 4 or Figure 6.

I have a point. Notice that the numbers assigned to each figure are based on the order in which they are rendered, not the order in which they are referenced.

Analyses & Inline Code

In this section, we discuss running and referencing statistical analyses.

A critical component of using our D2MR workflow for reproducible research is the ability to seamlessly integrate references to objects and object elements into the narrative text of your publication. When you run a model or summary and then identify coefficients, summary stats, or other important values in the output, you *could* type out those values as literal, static text. This is not only time-consuming, it problematically opens the door for significant human error each time the values must be manually updated.

Inline R code is an alternative that allows you to replace this static text with dynamic code. Rather than changing the β, p-value, mean, etc. every time any tiny thing in your data or code processes change, the values are calculated during the knitting process. You add the code once, then (ideally) never need to modify it so long as the structure of your data and modeling stays the same.

Fundamentally, think about inline R code as a teeny-tiny code chunk that takes the "1-chunk-1-thing" rule to the extreme, so much so that the output of the whole thing is the same kind of thing you'd just type into a Word document otherwise. Everything within the code will run exactly like it's run in the console, meaning you can always quickly check how it will render when knit by just copying and pasting that code into your console in RStudio.

The general format for inline R is a backtick[2], then a space, then as much code as you like (remember, literally what you'd see if you copied and pasted it into the console), then a closing backtick:

```
`r yourCodeHere()`
```

[2] If you want to make something appear as "verbatim" text that looks like code the way it shows up in this notebook, surround it in backticks without the "r" in front: `` `verbatim text` ``. If you want to include backticks within the literal code, smash your head against your desk until you give up.

Referencing values from lists and tibbles. In the following chunk, I create a vector, a list, and a tibble. These are basic R objects you'll need to extract references from often.

To reference a value from a vector, list, or tibble, you need to know the index of the value you want to reference. In R, indexing starts at 1[3], so the first element of a vector, list, or tibble is at index 1. You can reference the value at a specific index by using the object name followed by square brackets containing the index. For example, to reference the second element of a vector `ex_vector`, you would write `ex_vector[2]`: 30.

If you've got a list that is really a list and not a vector that you're casually calling a "list," it's a little weirder. List objects look like a series of values, but they are actually a series of other lists. If you referenced the second element of a list object `ex_list` with `ex_list[2]`, you would get a list that contains the second element: 30.[4] To get the actual value, you need to reference the index of the value within the list. For example, to reference the second element of a list object `ex_list`, you would write `ex_list[[2]]`: 30.

You can reference values from a tibble in the same way you would reference values from a vector, but you have to index both the row and column. For example, to reference the value in the second row and third column of a tibble `ex_tibble`, you would write `ex_tibble[2, 3]`: 75.

You can also use brackets to index by names. In a tibble, that usually means column/variable names, but you could use row names if your df has them. For example, to reference the value in the second row and the column named "mass" in a tibble `ex_tibble`, you would write `ex_tibble[2, "mass"]` to get the same as the above `ex_tibble[2,3]`: 75.

In your code, you'll often need to reference whole columns or rows, which you can do by omitting a value on either side of the comma. So to get the second row of `ex_tibble`, you would

[3] This is probably the biggest reason I will never be a python person. I'm team counting-starts-at-1 all the way. It's the *1st* thing! There's a 1 right in the name!

[4] Quarto is actually smart enough to know you're trying to reference the value and not the list, so if you use this in your markdown it will knit to the value. But if you're actually using it for any kind of programming (not just printing directly), you'll need to use the correct data type.

write `ex_tibble[2, ]`: C-3PO, 167, 75, list(c(“A New Hope”, “The Empire Strikes Back”, “Return of the Jedi”, “The Phantom Menace”, “Attack of the Clones”, “Revenge of the Sith”)).

You *can* call a tibble *row* in markdown and render it as text.

To get the column named “mass” from `ex_tibble`, you would write `ex_tibble[, "mass"]`. If you don’t include the comma, it will assume you want the variable. So `ex_tibble["mass"]` will return the column named “mass”. More commonly, if you want to call a column or row, you can use the `$` operator. For example, to reference the column named “mass” from `ex_tibble`, you would write `ex_tibble$mass`.

You *cannot* call a tibble *column* in markdown and render it as text. (What would that even look like?)

Tips and Tricks

The **`apa_p()` function** from the `papaja` package can be used to format p-values in APA style. It handles rounding, and prevents the p-value from being displayed as `0.000` when it is very small.

Use the **`broom` package** to extract values from model objects. The `tidy()` function extracts coefficients and other values from a model object, and the `glance()` function extracts model-level statistics.

Store common text as objects in your R script. This can be useful for text that is repeated throughout the document, especially if it’s text that you may want to change later. For example, in the Language Development Project (a dataset I frequently work with) there are two cohorts of children. The first is a group of children who are “typically developing,” and the second is a group of children with perinatal brain lesions. The latter group is referred to by a handful of different names depending on the context of the publication, e.g., “atypically developing”, “perinatal brain lesion”, “brain lesion”, “brain injury”, etc. I can store the variable `group2 <- "perinatal brain lesion"` and use it throughout the document. If I need to change the name of the group, I only need to reassign `group2 <- "brain injury"` in one place.

Write your own functions for commonly reported results. For example, you can write a function that takes a model object as an argument and returns a string with the results of the model. This can be useful for reporting results in APA style.

As an example, start by building a simple linear regression model with the `mtcars` dataset. Does vehicle weight predict fuel efficiency?

```r
# Create a simple linear regression model
model <- lm(mpg ~ wt, data = mtcars)
```

We can use the `tidy()` function and the `apa_p()` function to extract and format the coefficients and p-values from the model object and print into a string.

```r
# report_regression takes a lm model object as an argument
# and returns a string of format:
# "$F$(df, df.residual) = F-value, $p$ = p-value, $R^2$ = r-squared"
report_regression <- function(model) {

  # Extract coefficients and p-values
  m_glance <- glance(model)

  # Format the p-value
  p_value <- apa_p(m_glance$p.value, add_equals = TRUE)

  # Create a string with the results
  report <- paste0("$F$(",
                   m_glance$df,
                   ", ", m_glance$df.residual,
                   ") = ", round(m_glance$statistic, 2),
                   ", $p$ = ", p_value, ", $R^2$ = ", round(m_glance$r.squared,2))

  return(report)


}



# report_beta takes a lm model object and a variable name (string) as arguments
# and returns a string of format:
# "beta = beta-value, se = se-value, p = p-value"
report_beta <- function(model, var) {

  # Extract coefficients and p-values
  var_tidy <- tidy(model) %>% filter(term == var)

  # Format the p-value
  p_value <- apa_p(var_tidy$p.value, add_equals = TRUE)

  # Create a string with the results
  report <- paste0("$\\beta$ = ", round(var_tidy$estimate, 2),
                   ", $se$ = ", round(var_tidy$std.error, 2),
                   ", $p$ = ", p_value
  )

  return(report)
}
```

Reporting the results of the regression model created in the previous chunk *with static text* would look like:

A simple linear regression was conducted to predict fuel efficiency from vehicle weight.

The model significantly predicted miles per gallon, $F(1, 30) = 91.38$, $p < .001$, $R^2 = .75$. Vehicle weight was a significant predictor ($\beta = -5.34$, $p < .001$), indicating that for each additional thousand pounds of weight, fuel efficiency decreased by 5.34 miles per gallon.

Reporting the results of the regression model created in the previous chunk *with inline R code* would look like:

A simple linear regression was conducted to predict fuel efficiency from vehicle weight. The model significantly predicted miles per gallon, $F(1, 30) = 91.38$, $p = < .001$, $R^2 = 0.75$. Vehicle weight was a significant predictor ($\beta = -5.34$, $se = 0.56$, $p = < .001$), indicating that for each additional thousand pounds of weight, fuel efficiency decreased by 5.34 miles per gallon.

Common Issues

Are you getting errors referring to packages, installation, CRAN, or other library-related things? First check that all necessary packages are installed on your system. Then check that nothing in your document *or sourced files* includes code that would prompt R to install or update a package that is already loaded. It's good practice to load all necessary packages at the beginning of your document. If your code requires uncommon or non-CRAN packages that a user is likely to need to install before running, include a comment noting that at the beginning of your document. Optionally (and required in D2MR), include a **commented out** line of code that will install the package if it is not already installed *if the user chooses to uncomment the line.*

Something knitting in a way you didn't expect? Check that the class of your output is what you think it should be. In particular, if you want something to appear as regular-old in-line text, it needs to be a *value* data type, like numeric, character, or logical. The `cor()` function, for example, results in a simple numeric value, while the `cor.test()` function results in an "htest" object. That object contains numeric values you can extract and reference, but if you try to reference it directly it won't knit.

Are inline references to tables, figures, sections, or citations rendering as "??" and/or the literal reference in bold (e.g., "Figure **??**" or **smith2017??**)? Check that the key you're using is identical to the chunk label or bibtex citation key. Did you use the `fig-` or `tab-` prefix for both the chunk label and the reference? Did you use the `@` symbol before the citation key? Did a Zotero update the citation key format in your .bib file?

Tables or figures being weird or not displaying at all? Tables are the worst, and there's a lot of things that can go wrong with them. Figures aren't as bad, but still prone to many of the same problems. The first things to check are:

1. Did you use the `tbl-` / `fig-` prefix in the chunk label?
2. Did you include the `tbl-cap` / `fig-cap` option in the chunk?
3. Is your chunk actually returning a table/figure at all, or is it assigning the table/figure to an object that you didn't actual call in the chunk?
4. Is the chunk returning HTML, LaTeX, or text in any other format or markup language? Add `results = "asis"` to the chunk header to ensure that the output is rendered "as-is," i.e., actually processing the markup rather than displaying the literal output. (Personally I find this very counterintuitive. Wouldn't it make more sense if "asis" meant "don't process the markup"? Is what it is.)

If you're still having trouble, try running the chunk in a separate R script to see if it's a problem with the chunk or with the document. If it runs fine in the script, try running the script in the document to see if it's a problem with the chunk options or the document structure.

Aust, F., & Barth, M. (2024). *papaja: Prepare reproducible APA journal articles with R Markdown*. https://doi.org/10.32614/CRAN.package.papaja

Barth, M. (2023). *tinylabels: Lightweight variable labels*. https://cran.r-project.org/package=tinylabels

Bavelas, J. B., Chovil, N., Lawrie, D. A., & Wade, A. (1992). Interactive gestures. *Discourse Processes*, *15*(4), 469–489. https://doi.org/10.1080/01638539209544823

Chang, W., Cheng, J., Allaire, J., Sievert, C., Schloerke, B., Xie, Y., Allen, J., McPherson, J., Dipert, A., & Borges, B. (2024). *Shiny: Web application framework for r*. https://CRAN.R-project.org/package=shiny

Dowling, N. (2022). *Obviously I Don't Know but Whatever: Emblematic and Pragmatic Uses of Shrug Gestures in Early Childhood and Adolescence* [PhD thesis, The University of Chicago]. https://doi.org/10.6082/uchicago.4897

Dowling, N. (2025). *From Data to Manuscript in R – D2MR* [Course Website]. https://nrdowling.com/d2mr/

Gohel, D., & Skintzos, P. (2024). *Flextable: Functions for tabular reporting*. https://CRAN.R-project.org/package=flextable

Grolemund, G., & Wickham, H. (2011). Dates and times made easy with lubridate. *Journal of Statistical Software*, *40*(3), 1–25. https://www.jstatsoft.org/v40/i03/

Heritage, J. (2012a). Epistemics in Action: Action Formation and Territories of Knowledge. *Research on Language & Social Interaction*, *45*(1), 1–29. https://doi.org/10.1080/08351813.2012.646684

Heritage, J. (2012b). The Epistemic Engine: Sequence Organization and Territories of Knowledge. *Research on Language & Social Interaction*, *45*(1), 30–52. https://doi.org/10.1080/08351813.2012.646685

Müller, K., & Wickham, H. (2023). *Tibble: Simple data frames*. https://CRAN.R-project.org/package=tibble

R Core Team. (2024). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Robinson, D., Hayes, A., & Couch, S. (2024). *Broom: Convert statistical objects into tidy tibbles*. https://CRAN.R-project.org/package=broom

Schneider, W. J. (2025). *Apaquarto*. https://github.com/wjschne/apaquarto

Wickham, H. (2016b). *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org

Wickham, H. (2016a). *Ggplot2: Elegant Graphics for Data Analysis* (2nd ed. 2016 edition). Springer.

Wickham, H. (2023a). *Conflicted: An alternative conflict resolution strategy*.

https://CRAN.R-project.org/package=conflicted

Wickham, H. (2023b). *Forcats: Tools for working with categorical variables (factors)*. https://CRAN.R-project.org/package=forcats

Wickham, H. (2023c). *Stringr: Simple, consistent wrappers for common string operations*. https://CRAN.R-project.org/package=stringr

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., … Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, *4*(43), 1686. https://doi.org/10.21105/joss.01686

Wickham, H., Cetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data* (2nd edition). O'Reilly Media.

Wickham, H., François, R., Henry, L., Müller, K., & Vaughan, D. (2023). *Dplyr: A grammar of data manipulation*. https://CRAN.R-project.org/package=dplyr

Wickham, H., & Henry, L. (2025). *Purrr: Functional programming tools*. https://CRAN.R-project.org/package=purrr

Wickham, H., Hester, J., & Bryan, J. (2024). *Readr: Read rectangular text data*. https://CRAN.R-project.org/package=readr

Wickham, H., Vaughan, D., & Girlich, M. (2024). *Tidyr: Tidy messy data*. https://CRAN.R-project.org/package=tidyr

Xie, Y. (2015). *Dynamic documents with R and knitr* (2nd ed.). Chapman; Hall/CRC. https://yihui.org/knitr/

Xie, Y. (2018). *Bookdown: Authoring Books and Technical Documents with R Markdown*. https://bookdown.org/yihui/bookdown/

Yasumoto, A. (2024). *ftExtra: Extensions for 'flextable'*. https://CRAN.R-project.org/package=ftExtra