Reports and manuscripts with Quarto

What is `quarto`?

- *Open-source document format and computational notebook system*
- Integrates text, code, and outout
- Can create multiple different types of products (documents, slides, websites, books)

Why not R Markdown?

Only because quarto is newer and more featured!

- Anything you already know how to do in R Markdown you can do in quarto, and more!
- All of these slides, website, etc. are all made in quarto.
- If you know and love R Markdown, by all means keep using it!

Quarto workflow

1. Create a Quarto document
2. Write code
3. Write text
4. Repeat 2-3 in whatever order you want
5. Render repeatedly as you go

How does it work?

- You text in markdown and code in R
- `knitr` processes the code chunks, executes the R code, and inserts the code outputs (e.g., plots, tables) back into the markdown document
- `pandoc` transforms the markdown document into various output formats

Text and code...

````
# My header

Some text

Some *italic text*

Some **bold text**

- Eggs
- Milk

```{r}
x <- 3
x
```
````

... becomes ...

My title

Some text

Some *italic text*

Some **bold text**

- Eggs
- Milk

```
1 x <- 3
2 x
```

```
[1] 3
```

If you prefer, you can use the visual editor

My title

Some text

Some *italic text*

Some **bold text**

```{r}
x <- 3
x
```

R chunks

Everything within the chunks has to be valid R[1]

````
```{r}
x <- 3
```
````

````
```{r}
x + 4
```
````

```
[1] 7
```

Chunks run in order, continuously, like a single script

YAML

At the top of your Quarto document, a header written in *yaml* describes options for the document

```yaml
---
title: "My document"
author: Louisa Smith
format: html
---
```

There are a *ton* of possible options, but importantly, this determines the document output

Output

https://quarto.org/docs/output-formats/all-formats.html

HTML is easiest

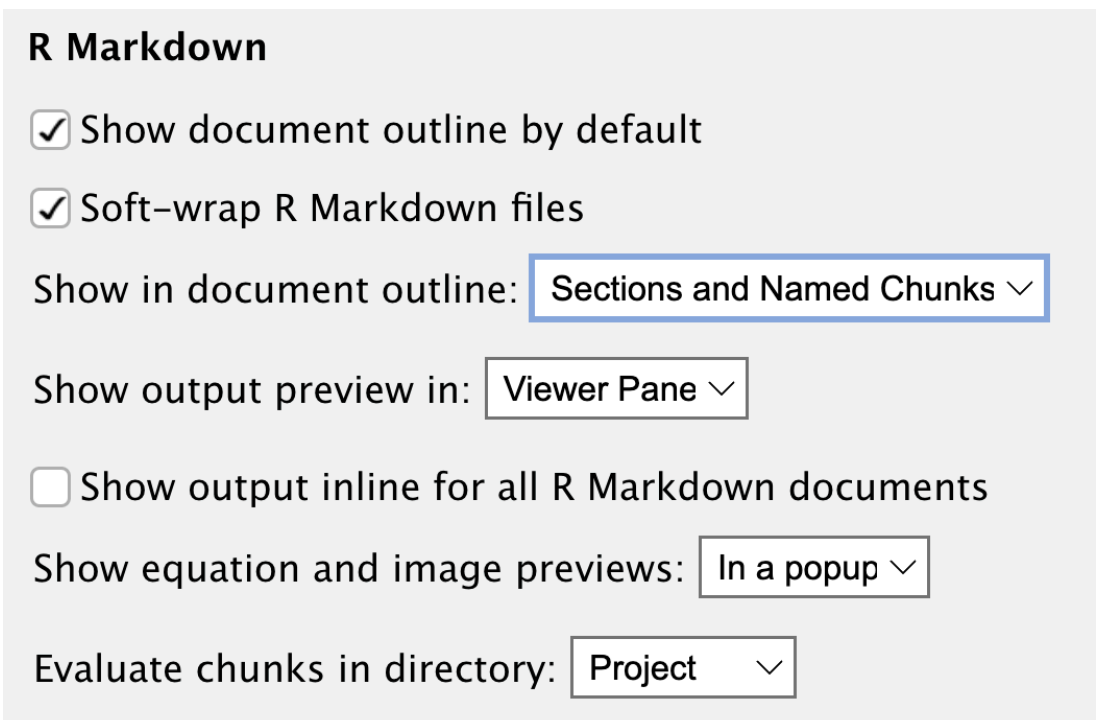
We're going to focus on html output

- It's easy to transition to Word (`format: docx`) but it's not as good for constant re-rendering
- You need a LaTeX installation for pdf (`format: pdf`)
  - I recommend `{tinytex}` (just run `install.packages("tinytex")`!)

Interactive output

You can choose whether you want to have chunk output show up within the document (vs. just the console) when you are running Quarto (and RMarkdown) documents interactively

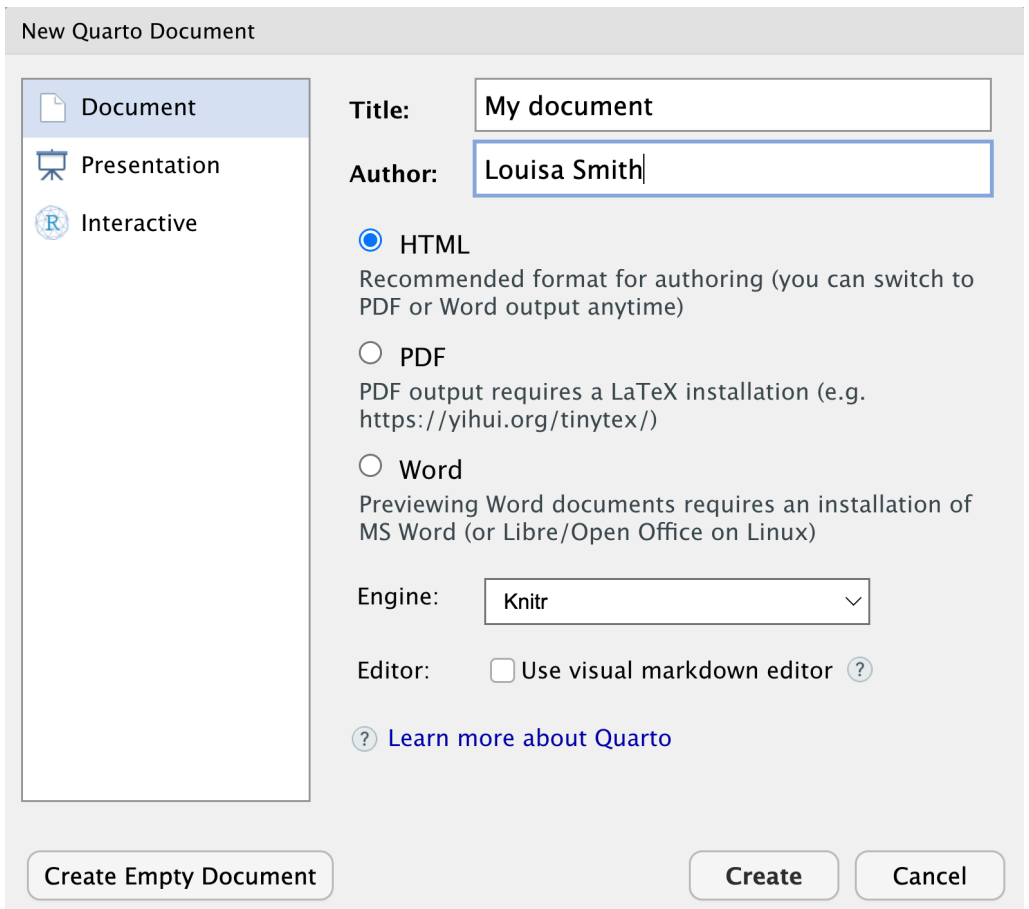
Tools > Global options



Exercises

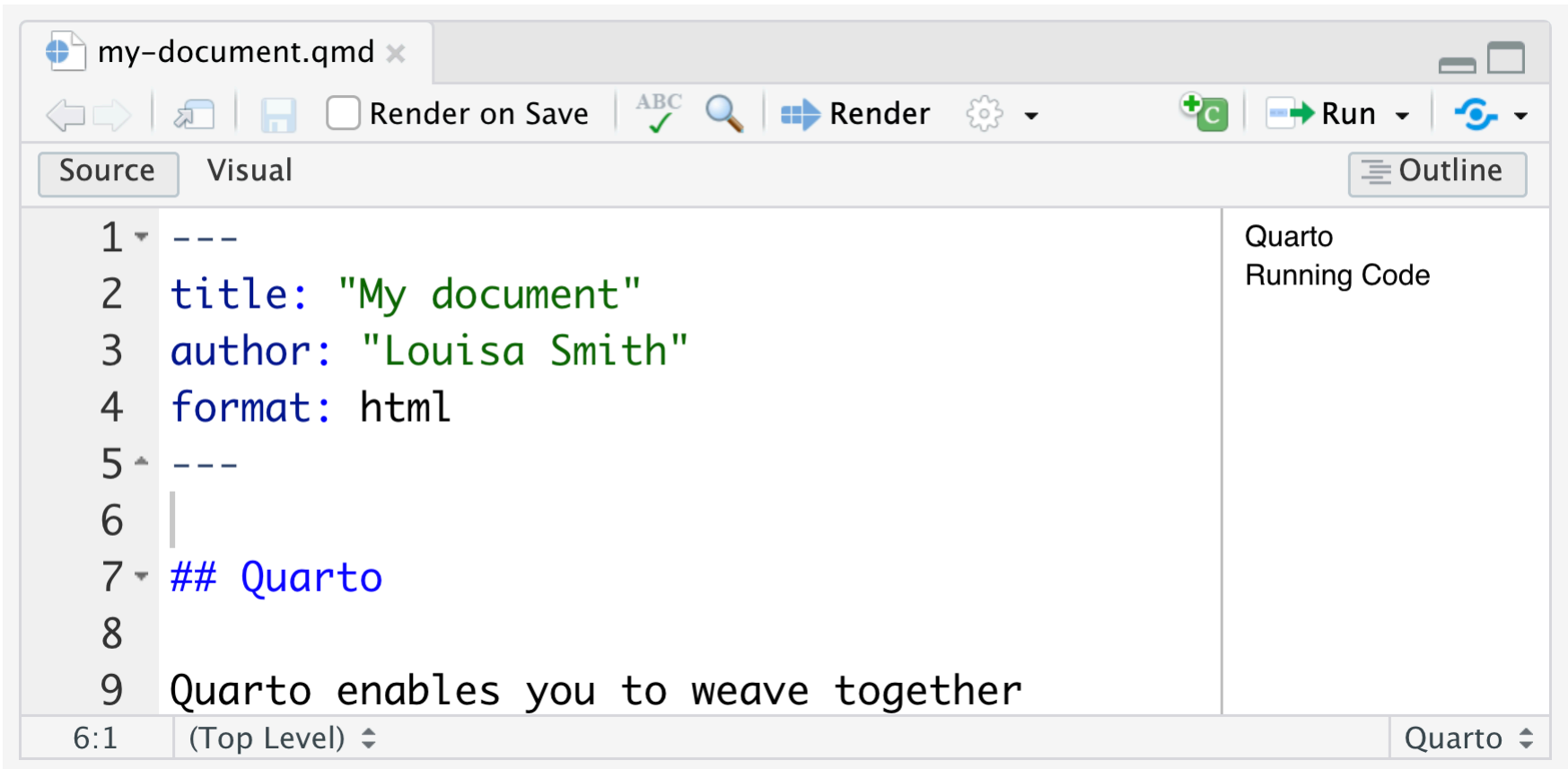
Open up your in class R project!

File > New File > Quarto Document



Exercises

- Try toggling between Source and Visual views
- Toggle on and off the Outline
- Click Render and look at the output



Quarto options

Chunk options

In your Quarto document, you had a chunk:

````
```{r}
#| echo: false
2 * 2
```
````

`#| echo: false` tells `knitr` not to show the code within that chunk

In RMarkdown, you would have written this `{r, echo = FALSE}`. You can still do that with Quarto, but it's generally easier to read, particularly for long options (like

Chunk options

Some of the ones I find myself using most often:

- `#| eval: false`: Don't evaluate this chunk! Really helpful if you're trying to isolate an error, or have a chunk that takes a long time
- `#| error: true`: Render this *even if* the chunk causes an error
- `#| cache: true`: Store the results of this chunk so that it doesn't need to re-run every time, as long as there are no changes
- `#| warning: false`: Don't print warnings
- `#| message: false`: Don't print messages

Document options

You can tell the *entire* document not to evaluate or print code (so just include the text!) at the top:

```yaml
---
title: "My document"
author: Louisa Smith
format: html
execute:
  eval: false
  echo: false
---
```

Careful! YAML is *really* picky about spacing.

Document options

There are lots of different options for the document

- For example, you can choose a theme:

```yaml
---
format:
  html:
    theme: yeti
---
```

- Remember the pickiness: when you have a format option, `html:` moves to a new line and the options are indented 2 spaces

Exercises

Download the quarto document with some `{gtsummary}` tables from yesterday

- Change some options to hide code and output
- Add another code chunk
- Play with themes
- Deal with errors
- Add some text

Quarto tables, figures, and stats

Chunks can produce figures and tables

```r
#| label: tbl-one
#| tbl-cap: "This is a great table"
knitr::kable(mtcars)
```

Table 1: This is a great table

	mpg	cyl	disp	hp	drat	wt	qse
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.0
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.6
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.4

	mpg	cyl	disp	hp	drat	wt	qse
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.0
Valiant	18.1	6	225.0	105	2.76	3.460	20.2
Duster 360	14.3	8	360.0	245	3.21	3.570	15.8
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.0
Merc 230	22.8	4	140.8	95	3.92	3.150	22.9
Merc 280	19.2	6	167.6	123	3.92	3.440	18.3
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.9
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.4
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.6

	mpg	cyl	disp	hp	drat	wt	qse
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.0
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.9
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.8
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.4
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.5

	mpg	cyl	disp	hp	drat	wt	qse
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.9
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.0
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.8
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.3
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.0

	mpg	cyl	disp	hp	drat	wt	qse
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.9
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.7
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.9
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.5
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.5
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.6
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.6

Chunks can produce figures or tables

```
```{r}
#| label: fig-hist
#| fig-cap: "This is a histogram"
hist(rnorm(100))
```
```

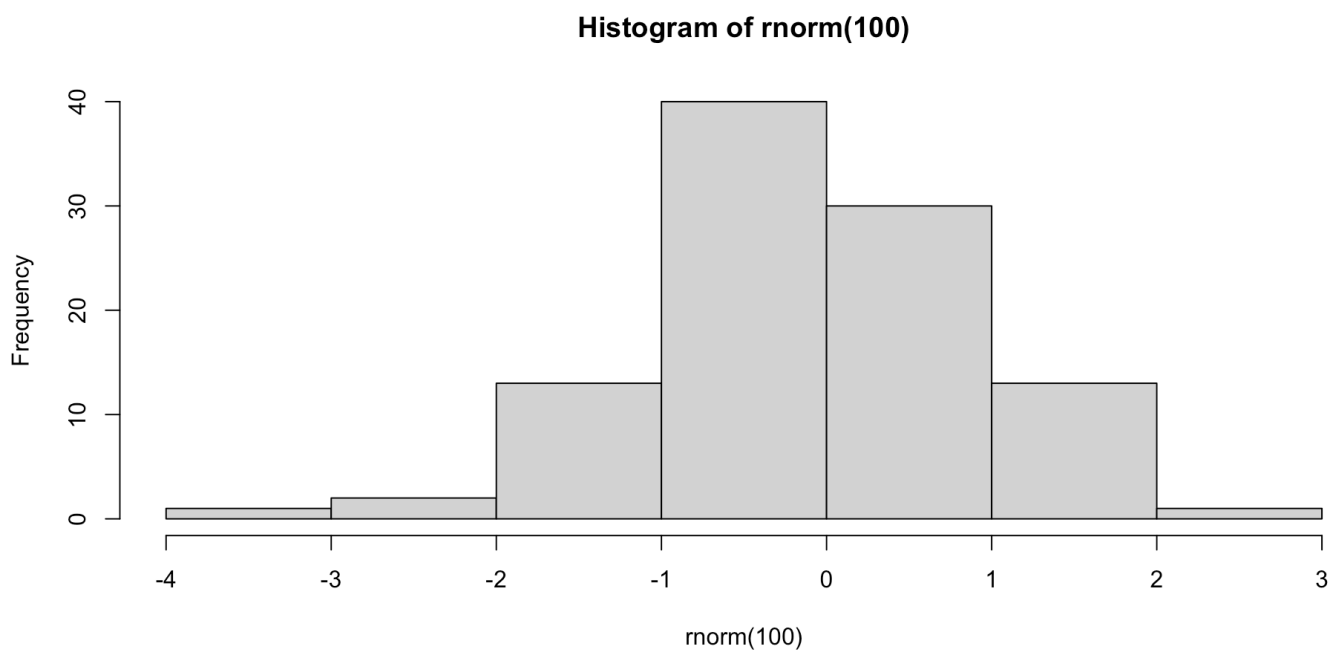


Figure 1: This is a histogram

Cross-referencing

You can then refer to those with `@tbl-one` and `@fig-hist` and the Table and Figure ordering will be correct (and linked)

```
@fig-hist contains a histogram and @tbl-one a table.
```

gets printed as:

Figure 1 contains a histogram and Table 1 a table.

Inline R

Along with just regular text, you can also run R code *within* the text:

```
There were `r 3 + 4` participants
```

becomes:

There were 7 participants

Inline R

This is helpful for reporting statistics, e.g. the sample size:

```
There were `r nrow(nlsy)` participants
```

becomes:

There were 1.2686^{4} participants

Inline stats

You can also create an object in a chunk and then reference it later in the text

````
```{r}
total_sample <- nrow(nlsy)
```
````

````
There were `r total_sample` participants
````

Inline stats (aside)

I often create a list of stats that I want to report in a manuscript:

```
stats <- list(n = nrow(data),
              mean_age = mean(data$age))
```

I can then print these numbers in the text with:

There were `` `r stats$n` `` participants with a mean age of `` `r stats$mean_age` ``.

which turns into:

There were 1123 participants with a mean age of 43.5.

Inline stats from `{gtsummary}`

We saw briefly yesterday:

```
library(gtsummary)
income_table <- tbl_uvregression(
  nlsy,
  y = income,
  include = c(
    sex_cat, race_eth_cat,
    eyesight_cat, income, age_bir
  ),
  method = lm
)
```

```
inline_text(income_table, variable = "age_bir")
```

```
[1] "595 (95% CI 538, 652; p<0.001)"
```

We pulled a statistic from our univariate table

If we're making a table, we probably want to report numbers from it

````
```{r}
#| label: tbl-descr
#| tbl-cap: "Descriptive statistics"
#| output-location: slide
table1 <- tbl_summary(
  nlsy,
  by = sex_cat,
  include = c(sex_cat, race_eth_cat, region_cat,
              eyesight_cat, glasses, age_bir)) |>
  add_overall(last = TRUE)
table1
```
````

If we're making a table, we probably want to report numbers from it

Table 2: Descriptive statistics

Characteristic	**Male** N = 6,403[1]	**Female** N = 6,283[1]	**Overall** N = 12,686[1]
race_eth_cat			
Hispanic	1,000 (16%)	1,002 (16%)	2,002 (16%)
Black	1,613 (25%)	1,561 (25%)	3,174 (25%)
Non-Black, Non-Hispanic	3,790 (59%)	3,720 (59%)	7,510 (59%)
region_cat			
Northeast	1,296 (21%)	1,254 (20%)	2,550 (20%)
North Central	1,488 (24%)	1,446 (23%)	2,934 (24%)
South	2,251 (36%)	2,317 (38%)	4,568 (37%)
West	1,253 (20%)	1,142 (19%)	2,395 (19%)
Unknown	115	124	239
eyesight_cat			
Excellent	1,582 (38%)	1,334 (31%)	2,916 (35%)
Very good	1,470 (35%)	1,500 (35%)	2,970 (35%)

[1] n (%); Median (Q1, Q3)

Characteristic	Male N = 6,403[1]	Female N = 6,283[1]	Overall N = 12,686[1]
Good	792 (19%)	1,002 (23%)	1,794 (21%)
Fair	267 (6.4%)	365 (8.5%)	632 (7.5%)
Poor	47 (1.1%)	85 (2.0%)	132 (1.6%)
Unknown	2,245	1,997	4,242
glasses	1,566 (38%)	2,328 (54%)	3,894 (46%)
Unknown	2,241	1,995	4,236
age_bir	25 (21, 29)	22 (19, 27)	23 (20, 28)
Unknown	3,652	3,091	6,743

[1] n (%); Median (Q1, Q3)

I want to report some stats!

The help file for `inline_text()` is helpful and tells us that we can look at `table1$table_body` to help figure out what data to extract.

How about the median (IQR) age of the male participants at the birth of their first child?

```
1 inline_text(table1, variable = "age_bir", column = "stat_1")
```

```
[1] "25 (21, 29)"
```

Formatting

We can add sample sizes for the overall stats on people who wear glasses using the `pattern =` argument:

```
inline_text(table1, variable = "glasses", column = "stat_0",
            pattern = "{n}/{N} ({p}%)")
```

```
3,894/8,450 (46%)
```

Formatting for regression statistics

Remove some details:

```
inline_text(income_table, variable = "age_bir")
```

```
[1] "595 (95% CI 538, 652; p<0.001)"
```

```
inline_text(income_table, variable = "age_bir",
            pattern = "{estimate} ({conf.low}, {conf.high})")
```

```
[1] "595 (538, 652)"
```

Better yet...

We can integrate these into the text of our manuscript:

```
A greater proportion of female (`r
inline_text(table1, variable = "glasses", column
= "stat_2")`) than male

(`r inline_text(table1, variable = "glasses",
column = "stat_1")`) participants wore glasses.
```

Which becomes:

A greater proportion of female (2,328 (54%)) than male (1,566 (38%)) participants wore glasses.

Readability

Because this can be hard to read, I'd suggest storing those stats in a chunk before the text:

````
```{r}
glasses_f <- inline_text(table1, variable = "glasses",
                         column = "stat_2")
glasses_m <- inline_text(table1, variable = "glasses",
                         column = "stat_1")
```
A greater proportion of female (`r glasses_f`) than male (`r glass
````

Exercises

Return to the quarto document with the tables and complete the exercises on the website.

Functions

Writing functions to reduce copy and paste

If you find yourself doing something over and over, you can probably write a function for it.

Functions in R

I've been denoting functions with parentheses: `func()`

We've seen functions such as:

- `mean()`
- `tbl_summary()`
- `init()`
- `create_github_token`

Functions take **arguments** and return **values**

Looking inside a function

If you want to see the code within a function, you can just type its name without the parentheses:

```
1 usethis::create_github_token
```

```
function (scopes = c("repo", "user", "gist", "workflow"), description = "DESCRIBE THE TOKEN'S USE
CASE",
    host = NULL)
{
    scopes <- glue_collapse(scopes, ",")
    host <- get_hosturl(host %||% default_api_url())
    url <- glue("{host}/settings/tokens/new?scopes={scopes}&description={description}")
    withr::defer(view_url(url))
    hint <- code_hint_with_host("gitcreds::gitcreds_set", host)
    message <- c(`_` = "Call {.run {hint}} to register this token in the local Git\n
credential store.")
    if (is_linux()) {
        message <- c(message, `!` = "On Linux, it can be tricky to store credentials
persistently.",
            i = "Read more in the {.href ['Managing Git(Hub) Credentials' article]
(https://usethis.r-lib.org/articles/articles/git-credentials.html)}.")
    }
    message <- c(message, i = "It is also a great idea to store this token in any\n
password-management software that you use.")
    ui_bullets(message)
```

```
    invisible()
}
<bytecode: 0x11726e178>
<environment: namespace:usethis>
```

Structure of a function

```
1  func <- function()
```

You can name your function like you do any other object

- Just avoid names of existing functions

Structure of a function

```
1 func <- function(arg1,
2                  arg2 = default_val)
3 }
```

What objects/values do you need to make your function work?

- You can give them default values to use if the user doesn't specify others

Structure of a function

```
1 func <- function(arg1,
2                  arg2 = default_val) {
3
4 }
```

Everything else goes within curly braces

- Code in here will essentially look like any other R code, using any inputs to your functions

Structure of a function

```
1 func <- function(arg1,
2                  arg2 = default_val) {
3   new_val <- # do something with args
4 }
```

- One thing you'll likely want to do is make new objects along the way
- These aren't saved to your environment (i.e., you won't see them in the upper-right window) when you run the function
- You can think of them as being stored in a temporary environment within the function

Structure of a function

```
1 func <- function(arg1,
2                  arg2 = default_val) {
3   new_val <- # do something with args
4   return(new_val)
5 }
```

Return something new that the code has produced

- The `return()` statement is actually optional. If you don't put it, it will return the last object in the code. When you're starting out, it's safer to always explicitly write out what you want to return.

Example: a new function for the mean

Let's say we are not satisfied with the `mean()` function and want to write our own.

Here's the general structure we'll start with.

```
1 new_mean <- function() {
2
3 }
```

New mean: arguments

We'll want to take the mean of a vector of numbers.

It will help to make an example of such a vector to think about what the input might look like, and to test the function. We'll call it `x`:

```
1 x <- c(1, 3, 5, 7, 9)
```

We can add `x` as an argument to our function:

```
1 new_mean <- function(x) {
2
3 }
```

New mean: function body

Let's think about how we calculate a mean in math, and then translate it into code:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

So we need to sum the elements of `x` together, and then divide by the number of elements.

We can use the functions `sum()` and `length()` to help us.

We'll write the code with our test vector first, before inserting it into the function:

```
1 n <- length(x)
2 sum(x) / n
```

```
[1] 5
```

New mean: function body

Our code seems to be doing what we want, so let's insert it. To be explicit, I've stored the answer (within the function) as `mean_val`, then returned that value.

```
1 new_mean <- function(x) {
2   n <- length(x)
3   mean_val <- sum(x) / n
4   return(mean_val)
5 }
```

Testing a function

Let's plug in the vector that we created to test it:

```
new_mean(x = x)
```

```
[1] 5
```

And then try another one we create on the spot:

```
new_mean(x = c(100, 200, 300))
```

```
[1] 200
```

Exercises

Create some functions!

Create an R script in your project called `functions.R` to save your work!

Functions, continued

Adding another argument

Let's say we plan to be using our `new_mean()` function to calculate proportions (i.e., the mean of a binary variable). Sometimes we'll want to report them as as percentage by multiplying the proportion by 100.

Let's name our new function `prop()`. We'll use the same structure as we did with `new_mean()`.

```
1 prop <- function(x) {
2   n <- length(x)
3   mean_val <- sum(x) / n
4   return(mean_val)
5 }
```

Testing the code

Now we'll want to test on a vector of 1's and 0's.

```
1 x <- c(0, 1, 1)
```

To calculate the proportion and turn it into a percentage, we'll just multiply the mean by 100.

```
1 multiplier <- 100
2 multiplier * sum(x) / length(x)
```

```
[1] 66.66667
```

Testing the code

We want to give users the option to choose between a proportion and a percentage. So we'll add an argument `multiplier`. When we want to just return the proportion, we can just set `multiplier` to be 1.

```
1 multiplier <- 1
2 multiplier * sum(x) / length(x)
```

```
[1] 0.6666667
```

```
1 multiplier <- 100
2 multiplier * sum(x) / length(x)
```

```
[1] 66.66667
```

Adding another argument

If we add `multiplier` as an argument, we can refer to it in the function body.

```
1 prop <- function(x, multiplier) {
2   n <- length(x)
3   mean_val <- multiplier * sum(x) /
4   return(mean_val)
5 }
```

Adding another argument

Now we can test:

```
1  prop(x = c(1, 0, 1, 0), multiplier = 1)
```

```
[1] 0.5
```

```
1  prop(x = c(1, 0, 1, 0), multiplier = 100)
```

```
[1] 50
```

Making a default argument

Since we don't want users to have to specify `multiplier = 1` every time they just want a proportion, we can set it as a **default**.

```
1 prop <- function(x, multiplier = 1) {
2   n <- length(x)
3   mean_val <- multiplier * sum(x) / n
4   return(mean_val)
5 }
```

Now we only need to specify that argument if we want a percentage.

```
1 prop(x = c(0, 1, 1, 1))
```

```
[1] 0.75
```

```
1 prop(x = c(0, 1, 1, 1), multiplier = 100)
```

Caveats

- This is obviously not the best way to write this function!
- For example, it will still work if `x = c(123, 593, -192)`…. but it certainly won't give you a proportion or a percentage!
- We could also put `multiplier =` *any number*, and we'll just be multiplying the answer by that number – this is essentially meaningless.
- We also haven't done any checking to see whether the user is even entering numbers! We could put in better error messages so users don't just get an R default error message if they do something wrong.

```
1 prop(x = c("blah", "blah", "blah"))
```

Functions, continued

Let's improve our proportion function

```r
prop <- function(x, multiplier = 1) {
  n <- length(x)
  mean_val <- multiplier * sum(x) / n
  return(mean_val)
}
```

The `multiplier =` argument is a little silly. Let's just give one option to make it a percentage.

TRUE/FALSE

`TRUE` and `FALSE` are special terms in R (no quotes)

- They are also treated as 1 and 0 in many situations

Let's say we want to replace the `multiplier =` argument with `percentage =`, where the user can set `TRUE` or `FALSE`.

```
1 prop(c(1, 0, 1, 0, 1))
```

```
[1] 0.6
```

```
1 prop(c(1, 0, 1, 0, 1), percentage = TRUE)
```

```
[1] 60
```

TRUE/FALSE lead naturally to `if` statements

```
if (3 > 8) {
  print("This is true")
} else {
  print("This is false")
}
```

```
[1] "This is false"
```

```
percentage <- TRUE
if (percentage) {
  print("print percentage")
} else {
  print("print proportion")
}
```

```
[1] "print percentage"
```

Incorporate it into the function

```r
prop <- function(x, percentage = FALSE) {
  n <- length(x)
  mean_val <- sum(x) / n
  if (percentage) {
    mean_val <- mean_val * 100
  } else {
    # don't actually need this else statement!
    mean_val <- mean_val
  }
  return(mean_val)
}
```

Let's try it out

```
1 prop(c(1, 0, 1, 0, 1))
```

```
[1] 0.6
```

```
1 prop(c(1, 0, 1, 0, 1), percentage = TRUE)
```

```
[1] 60
```

```
1 prop(c(1, 0, 1, 0, 1), percentage = FALSE)
```

```
[1] 0.6
```

```
1 nlsy_cc <- read_csv(here::here("data", "clean", "nlsy.rds"))
2 prop(nlsy_cc$glasses)
```

```
[1] 51.78423
```

Exercises

1. Create a function that takes a vector of numbers and returns the standard deviation manually (like we did the mean). Use `if` statements to check if the vector has only one (or fewer) elements and return `NA` if so. (Hint: the `length()` function will be helpful!) You don't need any extra arguments besides the vector of numbers.

$$\text{sd}(x) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$$

Exercises

2. Modify your function to remove the NA values before calculating the standard deviation. (Hint: the `na.omit()` function will be helpful!) Add an argument `na.rm =` that defaults to `TRUE` (the *opposite* of the `na.rm` argument in the built-in R function `sd()`). If `na.rm = FALSE`, then the function should return `NA` if there are any NA values in the vector.
3. What is the standard deviation of income in (all of) NLSY? Compare with the built-in R function `sd()`.

{renv}

Package management for R

What is `{renv}`?

`{renv}` is an R package for managing project dependencies and creating reproducible environments

Benefits of using {renv}

1. **Isolation:** Creates project-specific environments separate from the global R library.
2. **Reproducibility:** Ensures consistent package versions for code reproducibility.
3. **Collaboration:** Facilitates sharing and collaborating on projects with others.

Getting Started with `{renv}`

1. Install `{renv}`

```
install.packages("renv")
```

2. Initialize a project

```
renv::init()
```

3. Install packages

```
install.packages("other_package")
# only an option when using renv!
install.packages("github_user/github_package")
```

4. Track dependencies via a lockfile

```
renv::snapshot()
```

Behind the scenes

- Your project `.Rprofile` is updated to include:

```
1 source("renv/activate.R")
```

- This is run every time R starts, and does some management of the library paths to make sure when you call `install.packges("package")` or `library(package)` it does to the right place (`renv/library/R-{version}/{computer-specifics}`)
- A `renv.lock` file (really just a text file) is created to store the names and versions of the packages.

renv.lock


```
{
  "R": {
    "Version": "4.3.0",
    "Repositories": [
      {
        "Name": "CRAN",
        "URL": "https://cran.rstudio.com"
      }
    ]
  },
  "Packages": {
    "R6": {
      "Package": "R6",
      "Version": "2.5.1",
      "Source": "Repository",
      "Repository": "CRAN",
      "Requirements": [
        "R"
      ],
      "Hash": "470851b6d5d0ac559e9d01bb352b4021"
    },
    base64enc": {
      "Package": "base64enc",
      "Version": "0.1-3",
      "Source": "Repository",
```

```
    "Repository": "CRAN",
    "Requirements": [
      "R"
    ],
    "Hash": "543776ae6848fde2f48ff3816d0628bc"
  },
```

Using {renv} later

Restore an environment

```
renv::restore()
```

Install new packages

```
install.packages("other_package")
```

Update the lockfile

```
renv::snapshot()
```

Collaboration with `{renv}`

- Share the project's `renv.lock` file with collaborators to ensure consistent environments
- When they run `renv::restore()`, the correct versions of the packages will be installed on their computer

```
1  renv::restore()
```

Other helpful functions

Remove packages that are no longer used:

```
1  renv::clean()
```

Check the status of the project library with respect to the lockfile:

```
1  renv::status()
```

This will tell you to `renv::snapshot()` to add packages you've installed but haven't snapshotted, or `renv::restore()` if you're missing packages you need but which aren't installed

Conclusion

`{renv}` benefits:

- Isolation, reproducibility, and collaboration

Getting started with `{renv}`

1. Initialize a project using `renv::init()`
2. Install packages and store with `renv::snapshot()`
3. Restore later or elsewhere with `renv::restore()`

Finer control over statistics

We fit a series of univariate regressions

```
1  income_table <- tbl_uvregressio
2    nlsy,
3    y = income,
4    include = c(
5      sex_cat, race_eth_cat,
6      eyesight_cat, income, age_b
7    ),
8    method = lm
9  )
10 income_table
```

Characteristic	N	Beta	95% CI	p-value
sex_cat	10,195			
Male		—	—	
Female		-358	-844, 128	0.15
race_eth_cat	10,195			
Hispanic		—	—	
Black		-1,747	-2,507, -988	<0.001
Non-Black, Non-Hispanic		3,863	3,195, 4,530	<0.001
eyesight_cat	6,789			
Excellent		—	—	
Very good		-578	-1,319, 162	0.13
Good		-1,863	-2,719, -1,006	<0.001
Fair		-4,674	-5,910, -3,439	<0.001
Poor		-6,647	-9,154, -4,140	<0.001
age_bir	4,773	595	538, 652	<0.001

But a table is a limited form of output

We might want to dig in a little more to those regressions

- One helpful option from `{gtsummary}` was to extract data from the table directly
- This can be reported in a manuscript (rather than copying and pasting from the table)

```
1 inline_text(income_table, variable = "age_bir")
```

```
[1] "595 (95% CI 538, 652; p<0.001)"
```

What if we want *all* the numbers, say to create a figure?

- Under the hood, `{gtsummary}` is using the `{broom}` package to extract the statistics from the various models
- We can also use that package directly!

Statistical models in R can be messy

```r
mod_sex_cat <- lm(income ~ sex_cat, data = nlsy)
```

We could look at the model summary:

```r
summary(mod_sex_cat)
```

```
Call:
lm(formula = income ~ sex_cat, data = nlsy)

Residuals:
   Min     1Q Median     3Q    Max
-14880  -8880  -3943   5477  60478

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    14880.3      172.6  86.237   <2e-16 ***
sex_catFemale   -357.8      247.8  -1.444    0.149
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 12510 on 10193 degrees of freedom
  (2491 observations deleted due to missingness)
```

Statistical models in R can be messy

If we want to do something with the various values, we could extract each statistic individually:

```
coef(mod_sex_cat)
```

```
 (Intercept) sex_catFemale
  14880.3152     -357.8029
```

```
confint(mod_sex_cat)
```

```
                2.5 %     97.5 %
(Intercept)   14542.079 15218.5512
sex_catFemale  -843.608   128.0022
```

```
summary(mod_sex_cat)$r.squared
```

```
[1] 0.0002044429
```

```
summary(mod_sex_cat)$coefficients
```

```
              Estimate Std. Error   t value  Pr(>|t|)
(Intercept)   14880.3152   172.5521 86.236672 0.0000000
sex_catFemale  -357.8029   247.8349 -1.443715 0.1488499
```

{broom} has three main functions: augment(), glance(), tidy()

augment() adds fitted values, residuals, and other statistics to the original data

```
1 library(broom)
2 augment(mod_sex_cat)
```

```
# A tibble: 10,195 × 9
   .rownames income sex_cat .fitted  .resid     .hat .sigma   .cooksd .std.resid
   <chr>      <dbl> <fct>     <dbl>   <dbl>    <dbl>  <dbl>     <dbl>      <dbl>
 1 1          30000 Female   14523.  15477. 0.000202 12506.   1.55e-4      1.24
 2 2          20000 Female   14523.   5477. 0.000202 12507.   1.94e-5      0.438
 3 3          22390 Female   14523.   7867. 0.000202 12507.   4.01e-5      0.629
 4 4          22390 Female   14523.   7867. 0.000202 12507.   4.01e-5      0.629
 5 5          36000 Male     14880.  21120. 0.000190 12505.   2.72e-4      1.69
 6 6          35000 Male     14880.  20120. 0.000190 12505.   2.46e-4      1.61
 7 7           8502 Male     14880.  -6378. 0.000190 12507.   2.48e-5     -0.510
 8 8           7227 Female   14523.  -7296. 0.000202 12507.   3.44e-5     -0.583
 9 9          17000 Male     14880.   2120. 0.000190 12507.   2.74e-6      0.170
10 10          3548 Female   14523. -10975. 0.000202 12506.   7.79e-5     -0.878
# ℹ 10,185 more rows
```

{broom} has three main functions: augment(), glance(), tidy()

glance() creates a table of statistics that pertain to the entire model

```
1 glance(mod_sex_cat)
```

```
# A tibble: 1 × 12
  r.squared adj.r.squared  sigma statistic p.value    df   logLik     AIC    BIC
      <dbl>         <dbl>  <dbl>     <dbl>   <dbl> <dbl>    <dbl>   <dbl>  <dbl>
1  0.000204      0.000106 12506.      2.08   0.149     1 -110644. 221295. 2.21e5
# ℹ 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

`{broom}` has three main functions: `augment()`, `glance()`, `tidy()`

`tidy()` is the most useful to me and probably you!

It extracts coefficients and confidence intervals from models

```
1 tidy(mod_sex_cat, conf.int = TRUE)
```

```
# A tibble: 2 × 7
  term          estimate std.error statistic p.value conf.low conf.high
  <chr>            <dbl>     <dbl>     <dbl>   <dbl>    <dbl>     <dbl>
1 (Intercept)     14880.      173.     86.2    0       14542.    15219.
2 sex_catFemale    -358.      248.     -1.44   0.149    -844.      128.
```

`tidy()` works on over 100 statistical methods in R!

Anova, ARIMA, Cox, factor analysis, fixed effects, GAM, GEE, IV, kappa, kmeans, multinomial, proportional odds, principal components, survey methods, ...

- See the full list here
- All the output shares column names
- This makes it really easy to work with the output and reuse code across analyses

Some models have additional arguments

For example, we might want exponentiated coefficients:

```
logistic_model <- glm(glasses ~ eyesight_cat + sex_cat + income,
                      data = nlsy, family = binomial())
tidy(logistic_model, conf.int = TRUE, exponentiate = TRUE)
```

```
# A tibble: 7 × 7
  term                estimate std.error statistic  p.value conf.low conf.high
  <chr>                  <dbl>     <dbl>     <dbl>    <dbl>    <dbl>     <dbl>
1 (Intercept)            0.499   5.96e-2   -11.7   1.74e-31    0.444     0.560
2 eyesight_catVery good  0.920   5.96e-2    -1.39  1.64e- 1    0.819     1.03
3 eyesight_catGood       0.916   6.91e-2    -1.27  2.04e- 1    0.800     1.05
4 eyesight_catFair       0.802   1.00e-1    -2.20  2.77e- 2    0.658     0.976
5 eyesight_catPoor       1.03    2.01e-1     0.147 8.83e- 1    0.694     1.53
6 sex_catFemale          2.04    5.00e-2    14.2   5.46e-46    1.85      2.25
7 income                 1.00    1.93e-6     7.49  6.95e-14    1.00      1.00
```

We can also combine the results of lots of regressions

```r
# we already made mod_sex_cat
mod_race_eth_cat <- lm(income ~ race_eth_cat, data = nlsy)
mod_eyesight_cat <- lm(income ~ eyesight_cat, data = nlsy)
mod_age_bir <- lm(income ~ age_bir, data = nlsy)

tidy_sex_cat <- tidy(mod_sex_cat, conf.int = TRUE)
tidy_race_eth_cat <- tidy(mod_race_eth_cat, conf.int = TRUE)
tidy_eyesight_cat <- tidy(mod_eyesight_cat, conf.int = TRUE)
tidy_age_bir <- tidy(mod_age_bir, conf.int = TRUE)
```

There are of course more efficient ways to do this instead of copy/pasting 4 times...

With a little finagling, we have the same data as in the original univariate regression table...

```
dplyr::bind_rows(
  sex_cat = tidy_sex_cat,
  race_eth_cat = tidy_race_eth_cat,
  eyesight_cat = tidy_eyesight_cat,
  age_bir = tidy_age_bir, .id = "model") |>
  dplyr::mutate(
    term = stringr::str_remove(term, model),
    term = ifelse(term == "", model, term))
```

With a little finagling, we have the same data as in the original univariate regression table...

```
# A tibble: 12 × 8
   model        term    estimate std.error statistic  p.value conf.low conf.high
   <chr>        <chr>      <dbl>     <dbl>     <dbl>    <dbl>    <dbl>     <dbl>
 1 sex_cat      (Inter…   14880.     173.      86.2  0          14542.    15219.
 2 sex_cat      Female     -358.     248.      -1.44 1.49e- 1    -844.      128.
 3 race_eth_cat (Inter…   12867.     302.      42.7  0          12276.    13459.
 4 race_eth_cat Black     -1747.     387.      -4.51 6.58e- 6   -2507.     -988.
 5 race_eth_cat Non-Bl…    3863.     341.      11.3  1.20e-29    3195.     4530.
 6 eyesight_cat (Inter…   17683.     270.      65.6  0          17155.    18212.
 7 eyesight_cat Very g…    -578.     378.      -1.53 1.26e- 1   -1319.      162.
 8 eyesight_cat Good      -1863.     437.      -4.26 2.05e- 5   -2719.    -1006.
 9 eyesight_cat Fair      -4674.     630.      -7.42 1.35e-13   -5910.    -3439.
10 eyesight_cat Poor      -6647.    1279.      -5.20 2.07e- 7   -9154.    -4140.
11 age_bir      (Inter…    1707.     733.       2.33 1.99e- 2     270.     3143.
12 age_bir      age_bir     595.      29.1     20.4  3.71e-89     538.      652.
```

Even easier cleanup!

We could instead clean up the names and add reference rows with the `{tidycat}` package:

```
tidy(logistic_model, conf.int = TRUE, exponentiate = TRUE) |>
  tidycat::tidy_categorical(logistic_model, exponentiate = TRUE) |>
  dplyr::select(-c(3:5))
```

```
# A tibble: 9 × 8
  term                estimate conf.low conf.high variable level effect reference
  <chr>                  <dbl>    <dbl>     <dbl> <chr>    <fct> <chr>  <chr>
1 (Intercept)            0.499    0.444     0.560 (Interc… (Int… main   Non-Base…
2 <NA>                   1        1         1     eyesigh… Exce… main   Baseline…
3 eyesight_catVery …     0.920    0.819     1.03  eyesigh… Very… main   Non-Base…
4 eyesight_catGood       0.916    0.800     1.05  eyesigh… Good  main   Non-Base…
5 eyesight_catFair       0.802    0.658     0.976 eyesigh… Fair  main   Non-Base…
6 eyesight_catPoor       1.03     0.694     1.53  eyesigh… Poor  main   Non-Base…
7 <NA>                   1        1         1     sex_cat  Male  main   Baseline…
8 sex_catFemale          2.04     1.85      2.25  sex_cat  Fema… main   Non-Base…
9 income                 1.00     1.00      1.00  income   inco… main   Non-Base…
```

This makes it easy to make forest plots, for example

```
library(ggplot2)
tidy(logistic_model, conf.int = TRUE, exponentiate = TRUE) |>
  tidycat::tidy_categorical(logistic_model, exponentiate = TRUE) |
  dplyr::slice(-1) |> # remove intercept
  ggplot(mapping = aes(x = level, y = estimate,
                       ymin = conf.low, ymax = conf.high)) +
  geom_point() +
  geom_errorbar() +
  facet_grid(cols = vars(variable), scales = "free", space = "free
  scale_y_log10()
```

This makes it easy to make forest plots, for example