R Module 1: Reproducible Reports in Quarto

BMSC 620 | R Workflow Series

Emile Latour

What We're Learning Today

By the end of this module, you will be able to:

1. Understand and modify the **YAML header** in a `.qmd` file
2. Use **chunk options** to control how code and output appear
3. Insert **R results directly into prose** using inline R
4. Produce a report where numbers never need to be manually copy-pasted

> **Note**
>
> These skills apply directly to every homework and future manuscript you write.

The Problem We're Solving

Imagine you run an analysis and write up your results:

> "The mean BMI in our sample was **27.3** kg/m², with a standard deviation of **6.1**."

Then your collaborator sends you a corrected dataset.

You re-run your analysis. The mean is now **27.6**. The SD is now **6.3**.

How many places in your report do you need to update?

Every. Single. One. Manually.

The Solution: Reproducible Reporting

What if your prose *was* your code?

```
The mean BMI in our sample was `r mean(nhanes.samp$BMI, na.rm = TRUE) |> round(1)` kg/m²,
with a standard deviation of `r sd(nhanes.samp$BMI, na.rm = TRUE) |> round(1)`.
```

When the data changes, you re-render. **Everything updates.**

> The mean BMI in our sample was 26.6 kg/m², with a standard deviation of 8.

This is what Quarto makes possible — and it starts with understanding the structure of a `.qmd` file.

Anatomy of a `.qmd` File

A Quarto document has three parts:

```
1  ---                          ← YAML header
2  title: "My Report"
3  format: html
4  ---
5
6  Some prose text              ← Markdown text
7
8  ```{r}                       ← Code chunk
9  mean(x)
10 ```
```

Section	Purpose
YAML	Document settings and metadata
Prose	Your writing (Markdown)
Chunks	Your R code

All three work together to produce your final document.

Part 1: The YAML Header

What Is YAML?

YAML = "YAML Ain't Markup Language" *(don't worry about the name)*

It lives between the two sets of `---` at the **very top** of your file.

```yaml
---
title: "NHANES Sample Analysis"
author: "Your Name"
date: today
format: html
---
```

> **Important**
>
> **YAML is indentation-sensitive.** Misaligned spaces will cause your document to fail to render. Always use spaces, not tabs.

Essential YAML Fields

```yaml
---
title: "NHANES Sample Analysis"        # Document title
subtitle: "BMSC 620 Homework 3"        # Optional subtitle
author: "Your Name"                    # Your name
date: today                            # Auto-fills today's date
format: html                           # Output format
---
```

`date: today` is a Quarto shortcut — it automatically uses the current date when you render. No more hardcoding dates.

Controlling Your Output Format

The `format` field controls what kind of document is produced.

Simple version:

```yaml
format: html
```

With options:

```yaml
format:
  html:
    toc: true
    toc-depth: 2
    number-sections: true
    theme: flatly
    self-contained: true
```

`self-contained: true` bundles everything into one `.html` file — **very useful for submitting homework.**

Useful `format: html` Options

Option	What it does
`toc: true`	Adds a table of contents
`toc-depth: 2`	How many heading levels in the TOC
`number-sections: true`	Numbers your section headings
`theme: flatly`	Changes the visual theme
`self-contained: true`	One file, easy to submit
`code-fold: true`	Hides code by default (reader can expand)

Document-Level Execute Options

You can set default behavior for **all** code chunks in the YAML:

```yaml
---
title: "NHANES Analysis"
format: html
execute:
  echo: true        # Show code by default
  warning: false    # Hide warnings by default
  message: false    # Hide package messages by default
---
```

Tip

Setting `warning: false` and `message: false` at the document level is almost always a good idea. Individual chunks can override these defaults.

Part 2: Chunk Options

What Are Chunk Options?

Chunk options control **how a specific chunk of code behaves** when the document renders.

They go at the top of a chunk using the `#|` prefix (the "hashpipe"):

````
```{r}
#| label: my-analysis
#| echo: false
#| warning: false

mean(nhanes.samp$BMI, na.rm = TRUE)
```
````

Each `#|` line is one option. You can have as many as you need.

The Most Important Chunk Options

Option	Values	What it controls
`label`	any name	Names the chunk (useful for debugging)
`echo`	`true/false`	Show the code in output?
`eval`	`true/false`	Run the code?
`warning`	`true/false`	Show warnings?
`message`	`true/false`	Show messages?
`include`	`true/false`	Include *anything* from this chunk?

echo: Show or Hide Code

`echo: true` — code and output both appear

```
1 nhanes.samp %>%
2   summarise(
3     mean_bmi = mean(BMI, na.rm = TRUE) %>%
4     sd_bmi   = sd(BMI, na.rm = TRUE) %>% ro
5   )
```

```
# A tibble: 1 × 2
  mean_bmi sd_bmi
     <dbl>  <dbl>
1     26.6      8
```

`echo: false` — only the output appears

```
# A tibble: 1 × 2
  mean_bmi sd_bmi
     <dbl>  <dbl>
1     26.6      8
```

Tip

Use `echo: false` in polished reports where the code isn't the point — readers see only the result.

`eval`: Run or Skip Code

````
```{r}
#| eval: false

# This code will be shown but NOT run
install.packages("tidyverse")
```
````

Tip

`eval: false` is perfect for showing installation instructions or example code that you don't want to actually execute when rendering.

`include`: Invisible but Active

````
```{r}
#| label: setup-chunk
#| include: false

# This entire chunk — code AND output — is invisible in the final document
# But the code still runs!
library(tidyverse)
library(oibiostat)
data("nhanes.samp")
```
````

`include: false` is what you use for your **setup chunk** — you need the libraries loaded, but readers don't need to see that.

Chunk Options for Figures

```r
#| label: fig-bmi-hist
#| fig-width: 7
#| fig-height: 4
#| fig-cap: "Distribution of BMI in the NHANES sample (n = 200)"
#| fig-alt: "Histogram showing a roughly normal distribution of BMI"

ggplot(nhanes.samp, aes(x = BMI)) +
  geom_histogram(bins = 30)
```

Option	Purpose
`fig-width` / `fig-height`	Dimensions in inches
`fig-cap`	Caption below figure
`fig-alt`	Accessibility alt text

Why Label Your Chunks?

```
```{r}
#| label: bmi-summary

nhanes.samp |>
  summarise(mean_bmi = mean(BMI, na.rm = TRUE))
```
```

Chunk labels help you in three ways:

1. **Debugging** — error messages tell you exactly which chunk failed
2. **Navigation** — RStudio shows labeled chunks in the document outline
3. **Cross-referencing** — figures labeled `fig-*` can be cited in text

> **Important**
>
> Label names must be **unique** within a document and contain no spaces (use `-` or `_` instead).

Part 3: Inline R

What Is Inline R?

Inline R lets you embed a **single R expression** directly inside your prose.

Syntax:

```
The sample includes `r nrow(nhanes.samp)` participants.
```

Renders as:

> The sample includes 200 participants.

The backtick-r opens it, the closing backtick ends it. Everything between is evaluated as R.

A Realistic Example

Suppose you want to report summary statistics in a sentence:

```r
# These are calculated in a code chunk
n      <- nrow(nhanes.samp)
m_bmi  <- mean(nhanes.samp$BMI, na.rm = TRUE) |> round(1)
sd_bmi <- sd(nhanes.samp$BMI, na.rm = TRUE) |> round(1)
```

In your qmd:

```
The NHANES sample included `r n` participants.
Mean BMI was `r m_bmi` kg/m² (SD = `r sd_bmi`).
```

Rendered output:

> The NHANES sample included 200 participants. Mean BMI was 26.6 kg/m² (SD = 8).

The Smart Workflow: Compute Then Report

Rather than putting complex expressions inline, **compute first in a chunk, then report inline.**

❌ **Hard to read:**

````
Mean BMI was
`r mean(nhanes.samp$BMI, na.rm=TRUE)|>round
````

✅ **Clean:**

````
```{r}
#| include: false
m_bmi <- mean(nhanes.samp$BMI,
              na.rm = TRUE) |> round(1)
```

Mean BMI was `r m_bmi`.
````

The chunk does the work. Inline R just reports it.

Formatting Numbers Inline

The `scales` package formats numbers directly — no more `round()` or `paste0()`:

```
library(scales)

n       <- nrow(nhanes.samp)
m_bmi   <- mean(nhanes.samp$BMI, na.rm = TRUE)
sd_bmi <- sd(nhanes.samp$BMI, na.rm = TRUE)

prop_overweight <- mean(nhanes.samp$BMI >= 25, na.rm = TRUE)
p_val <- 0.0003
```

```
The sample included `r n` participants.
Mean BMI was `r scales::number(m_bmi, accuracy = 0.1)` kg/m²
(SD = `r scales::number(sd_bmi, accuracy = 0.1)`).
`r scales::percent(prop_overweight, accuracy = 0.1)` had BMI ≥ 25.
The association was statistically significant (*p* `r scales::pvalue(p_val)`).
```

Renders as:

> The sample included 200 participants. Mean BMI was 26.6 kg/m² (SD = 8.0). 54.7% had BMI ≥ 25. The association was statistically significant (*p* <0.001).

The `accuracy` argument controls decimal places. `scales::pvalue()` automatically handles the `< 0.001` case.

Putting It All Together

Before: The Fragile Report

```
## Results

We analyzed data from 200 participants. The mean age was
38.2 years (SD = 14.7). Mean BMI was 26.7 kg/m²
(SD = 6.5). Approximately 60.4% of participants had a
BMI ≥ 25.
```

Every number here is hardcoded. If the dataset changes — or if you made an error — **you have to find every number manually.**

After: The Reproducible Report

````
```{r}
#| include: false
n      <- nrow(nhanes.samp)
m_age  <- mean(nhanes.samp$Age, na.rm = TRUE)
sd_age <- sd(nhanes.samp$Age,   na.rm = TRUE)
m_bmi  <- mean(nhanes.samp$BMI, na.rm = TRUE)
sd_bmi <- sd(nhanes.samp$BMI,   na.rm = TRUE)
pct_ow <- scales::percent(mean(nhanes.samp$BMI >= 25, na.rm = TRUE),
                          accuracy = 0.1)
```

We analyzed data from `r n` participants. Mean age was
`r scales::number(m_age, accuracy = 0.1)` years
(SD = `r scales::number(sd_age, accuracy = 0.1)`). Mean BMI was
`r scales::number(m_bmi, accuracy = 0.1)` kg/m²
(SD = `r scales::number(sd_bmi, accuracy = 0.1)`).
Approximately `r pct_ow` had a BMI ≥ 25.
````

Renders as:

> We analyzed data from 200 participants. Mean age was 33.3 years (SD = 21.1). Mean BMI was 26.6 kg/m² (SD = 8.0). Approximately 54.7% had a BMI ≥ 25.

Re-run with new data. **Everything updates.**

Key Takeaways

YAML

- Controls document settings and metadata
- `execute:` sets defaults for all chunks
- `self-contained: true` for easy submission

Chunk options

- `echo` / `eval` / `include` control visibility
- Always label your chunks
- Fig options control plot appearance

Inline R

- Never manually type a number that R can compute
- Compute in a chunk, report inline
- `scales` package for clean formatting

> **Tip**
>
> **The rule:**
>
> If a number in your prose came from an analysis, it should come from R — not from you typing it.

References

YAML

- Good intro to YAML – focus on “Document YAML” not “Project YAML”.
- YAML in R for Data Science
- Bootstrap themes

Chunk options

- Global chunk options in Quarto
- Quarto chunk options
- Quarto documentation: chunk options

Inline R

- Inline R Code in RMarkdown – Applies for Quarto too.
- Inline R code
- Quarto documentation – OK. Have to click on the “Knitr” tab to see R.