Notes 3 - Quarto

Math 3190 - Fundamentals of Data Science

Rick Brown

Southern Utah University

Table of Contents

Section 1

Quarto

Reproducible Reports

The final product of a data analysis project is often a report: scientific publications, news articles, an analysis report for your company, or lecture notes for a class.

Now imagine after you are done you realize you:

- had the wrong data set
- have a new data set for the same analysis
- made a mistake and fix the error, or
- your boss or someone you are training wants to see the code and be able to reproduce the results

Situations like these are common for a data scientist.

Quarto

Quarto is a format for **literate programming** documents.

It is based on **markdown**, a markup language that is widely used to generate html pages.

You can learn more about markdown here:
https://www.markdowntutorial.com/

Quarto is similar to environments like Jupityer Notebooks, Google Colab, and R Markdown (R Markdown is very much like Quarto, but is dependent on **R**). However, Quarto is language agnostic. This means it natively works in both **R** and Python (and Julia and Observable JavaScript).

You can download Quarto here: https://quarto.org/docs/download/.

Quarto Setup

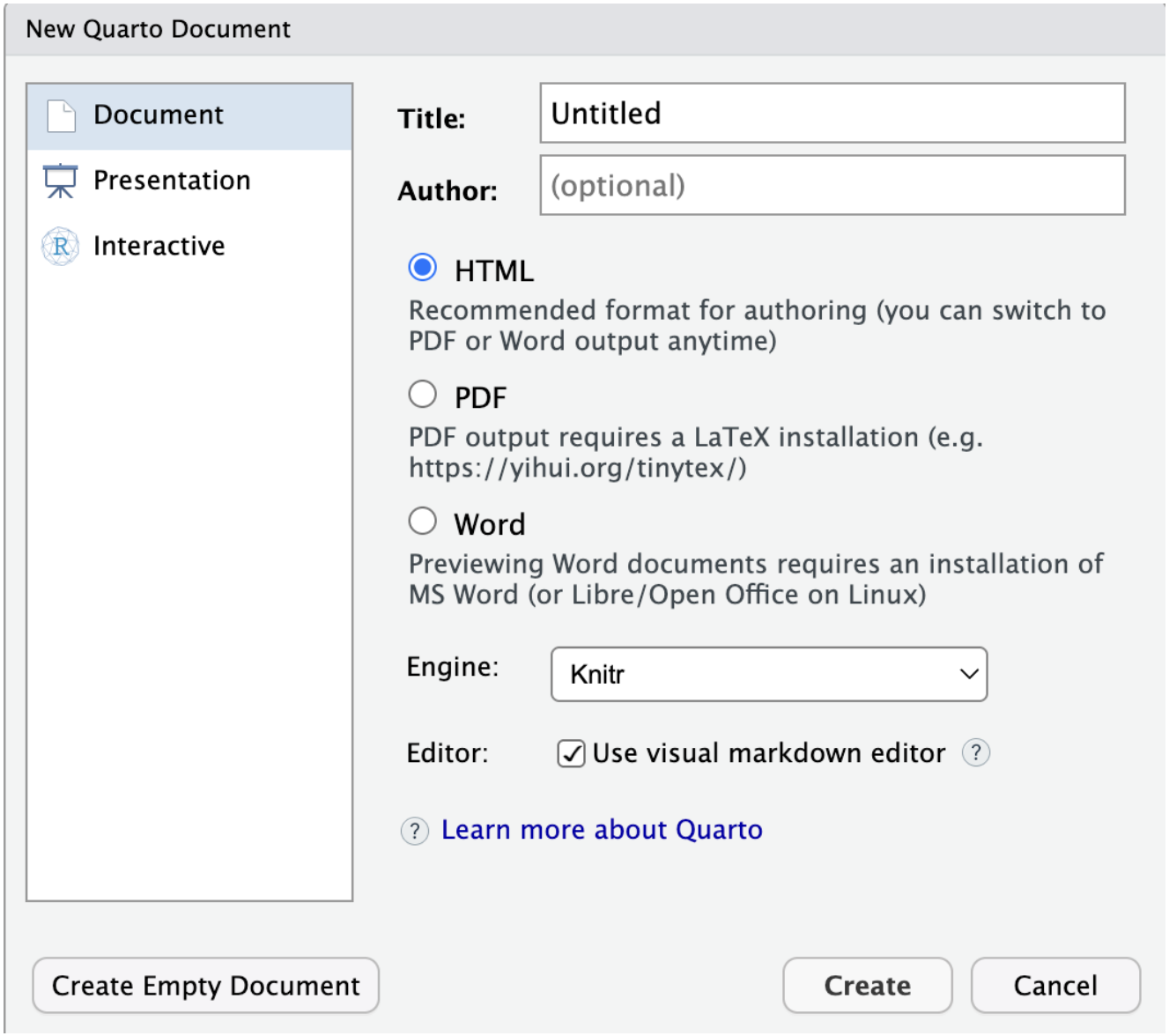
Literate programming weaves instructions, documentation, and detailed comments in between machine executable code.

With Quarto, you need to **compile** the document into the final report. This allows the code to run in a clean workspace each time, so it is reproduceable and gives the same results each time.

You can start a Quarto document in Positron by clicking on **New** then **Quarto Document**. You will then be asked for a title and author.

Final reports can be to be in: HTML, PDF, Microsoft Word, or presentation formats like PowerPoint, Beamer, or HTML slides.

Quarto Setup



Quarto Setup

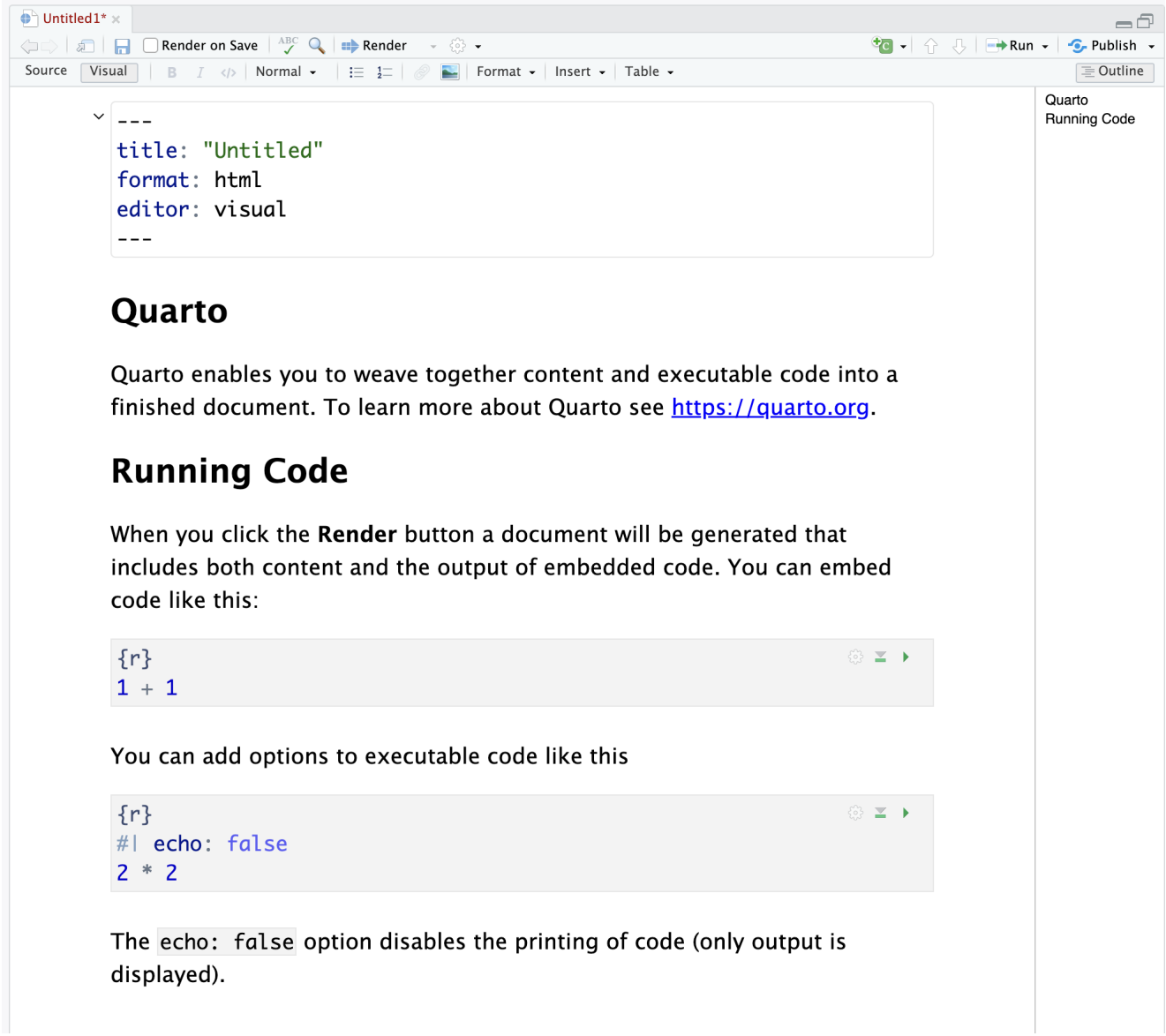
As a convention, we use the `.qmd` extension for these files.

Once you gain experience with Quarto, you will be able to do this without the template and can simply start from a blank template.

In the template, you will see several things to note (in the following slides).

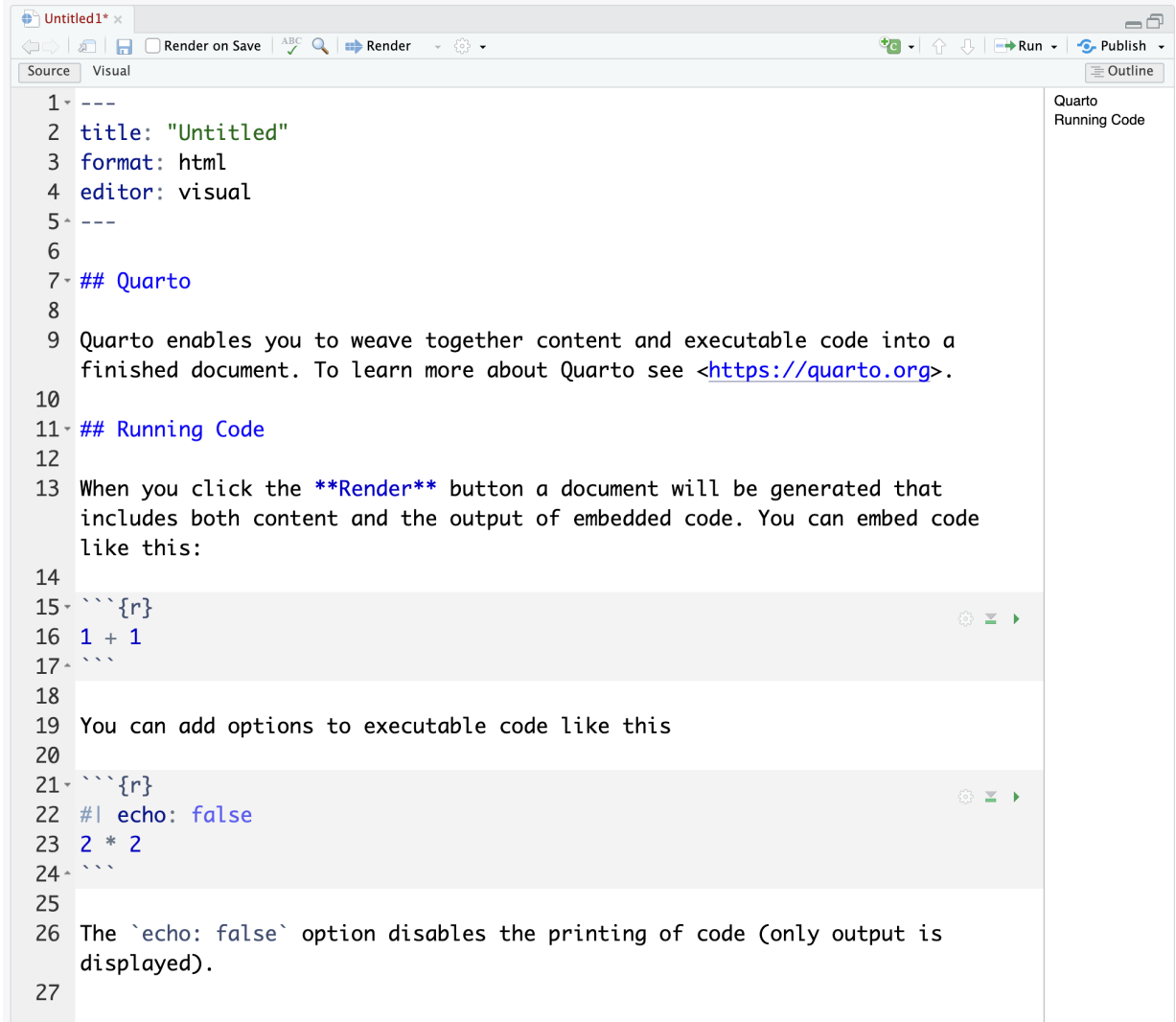
First, the default when opening a Quarto document is that it is in **Visual** editing mode. This will likely be a little more comfortable for you if you are not used to coding in something like LaTeX.

Quarto Visual



Quarto Source

However, we will focus on the source editor since it is much more customizable (although you are not prohibited from using the visual editor).



The Header

At the top of either the visual or source editor, you will see the following:

```
---
title: "Untitled"
format: html
editor: visual
---
```

This is known as the YAML header. YAML stands for YAML Ain't Markup Language.

One parameter that we will highlight is `format`. By changing this to, say, `pdf` or `docx`, we can control the type of output that is produced.

The Header

There are many, many options that can be adjusted in the YAML.

```yaml
---
title: "Project Title"
author: "Your Name"
date: "1/1/2026"
format:
  html:
    code_folding: hide
    toc: true
    theme: "flatly"
editor: source # Makes the source editor the default
               #   when file is opened.
# Like R, you can put comments in the YAML.
---
```

R Code Chunks

In various places in the document, we see something like this:

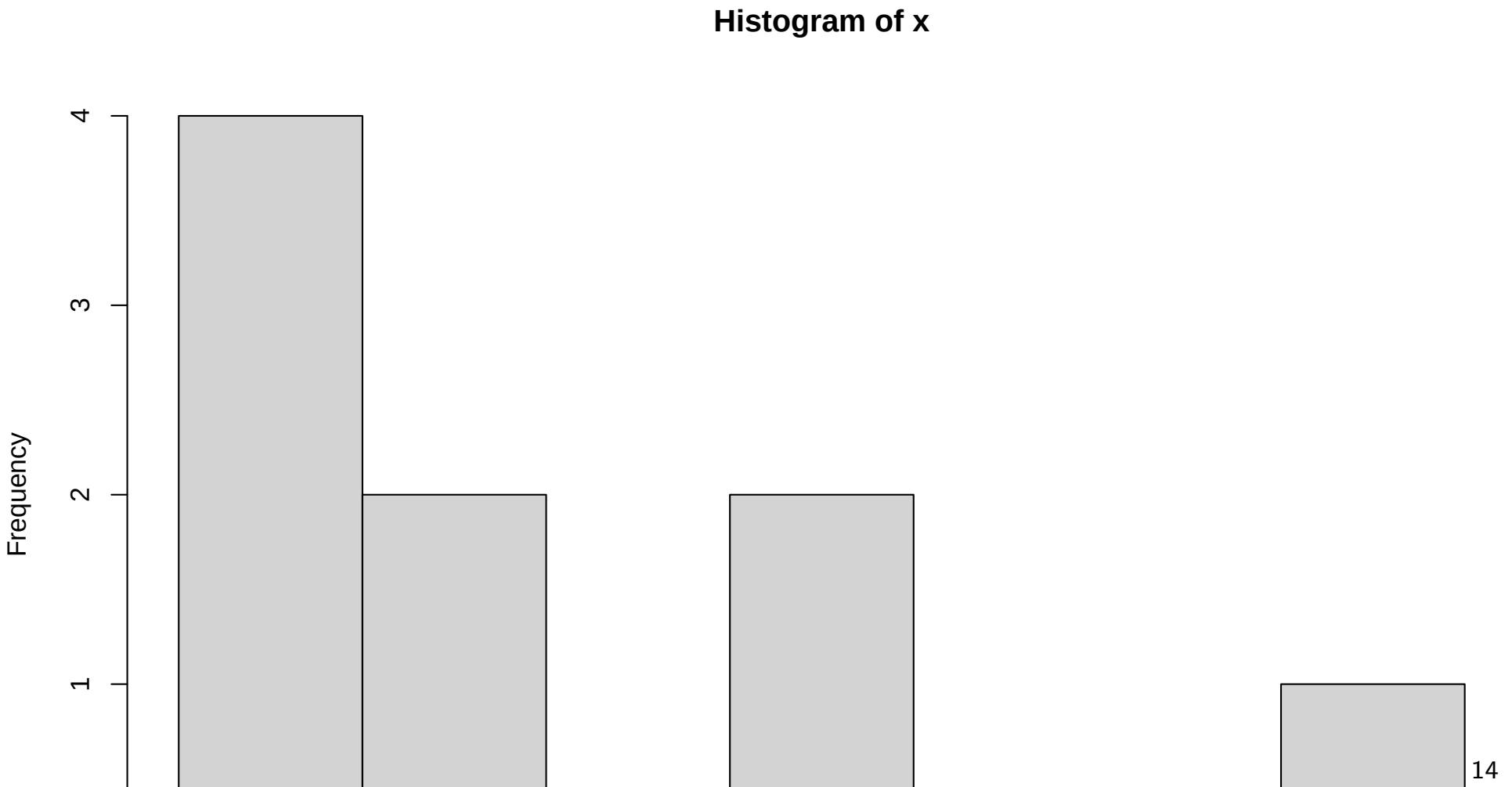
````
```{r}
1 + 1
```
````

These are the code chunks. When you compile the document, the **R** code inside the chunk, in this case 1+1, will be evaluated and the result included in that position in the final document.

To add your own **R** chunks, you can type the characters above quickly with the key binding Command-Option-I (that is a capital i) on the Mac and Ctrl-Alt-I on Windows. You can also click on the green code chunk button in the toolbar at the top.

R Code Chunks

These code chunks can be used to show code, but that code will also run. Even plots can be made, although this plot is too big to fit on this slide. We will see soon how to adjust the size of plots.



R Code Chunks

By default, both the code and the output will show up in your document. To avoid having the code show up, you can use an argument. To avoid this, you can set options for that code chunk using a special #| comment in the chunk. For example:

````
```{r}
#| echo: false
x <- c(10, 15, 2, 4, 10, 5, 3, 5, 2)
hist(x)
```
````

These comments need to be put at the beginning of the code chunk. Notice that this is YAML, not **R** code. That is why `false` is lowercase while in **R** it would be all capitalized.

R Code Chunks

Its a good habit to add a label to the **R** code chunks. This can be useful when debugging, among other situations. You do this by using the `#| label:` comments:

````
```{r}
#| label: x_hist
#| echo: false
x <- c(10, 15, 2, 4, 10, 5, 3, 5, 2)
hist(x)
```
````

Global Options and knitR

If there are certain options you want to apply to all code chunks, you can specify them in the YAML header with `execute:`. Be aware that white space matters in the YAML. The `echo` and `eval` options need to be indented, need to have a colon, and need to have one space after that colon.

```
---
title: "Untitled"
format: html
execute:
  echo: false
  eval: true
---
```

We use the `knitR` package to compile Quarto documents based on **R**, so we need to install that package. The specific function used to compile is the `knit()` function, which takes a filename as input. RStudio provides a Render button that does that for you, though.

Adding Equations

You can add formatted equations using between dollar signs $. Example, if you type

```
$x^2+\frac{1}{2}$
```

It will look like $x^2 + \frac{1}{2}$. In fact, nearly all LaTeX code works in Quarto.

If you put two dollar signs, the equation will appear on its own line.

```
$$\sum_{i=1}^{10}\left(\lambda_i+\frac{10}{\alpha}\right)$$
```

$$\sum_{i=1}^{10}\left(\lambda_i+\frac{10}{\alpha}\right)$$

Adding and Adjusting Images

We can add images to our Quarto file by using

```
![image_caption_goes_here](image_name.ext){width=50%}
```

if the image is already in the same directory. Otherwise, you can put the path to the file in those parentheses. We can adjust the width of the image as well by changing the number in the `width=50%`.

Adding and Adjusting Images

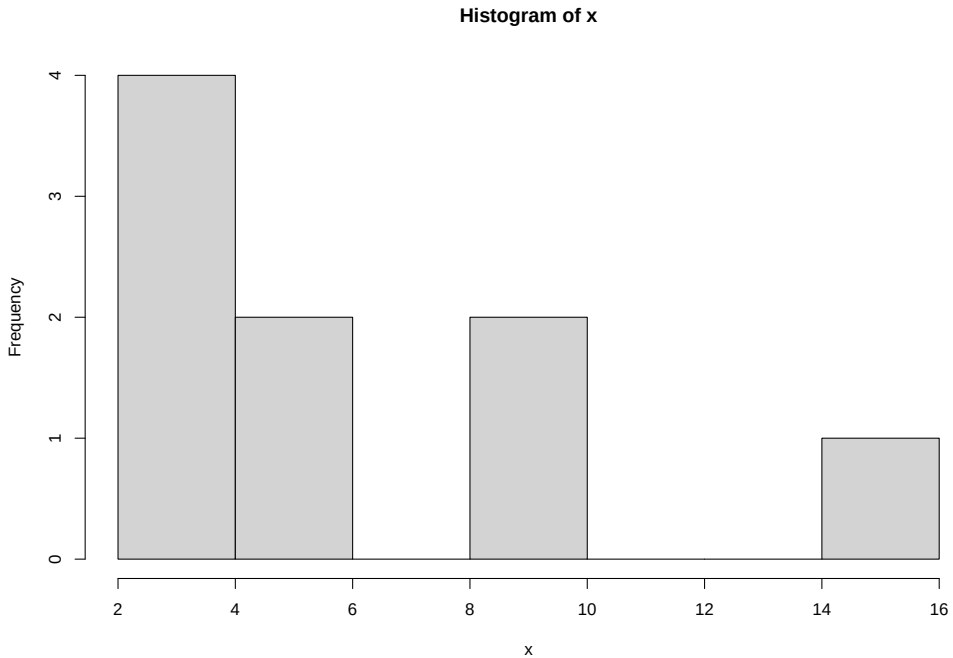
If the image is generated by **R** code, we can adjust the size by using `fig-width`, `fig-height`, `out-width` and/or `out-height` options in the code chunk.

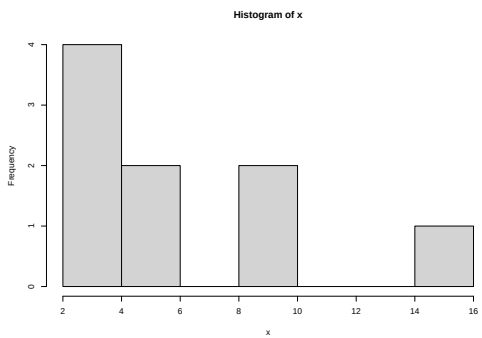
````
```{r}
#| label: image_code
#| label: image_code
#| eval: true
#| echo: true
#| out-width: "25%"
x <- c(10, 15, 2, 4, 10, 5, 3, 5, 2)
hist(x)
```
````

The result is shown othe next slide.

Adding and Adjusting Images

The first image has `out-wdith: "50%"` and the second has `out-wdith: "25%"`





Other Quarto Options

We will explore other Quarto options: headers, tabsets, and latex equations in class and in your homework.

Note: From now on, all homework should be completed using a Quarto document (uploaded to GitHub). Your document should include headers, descriptive text, **R** code, and plots/figures!

More on Quarto

There is a lot more you can do with Quarto. I highly recommend you continue learning as you gain more experience writing reports in **R**. There are many free resources on the internet including:

1. R Studio's tutorial: https://quarto.org
2. The Quarto guide: https://quarto.org/docs/guide/
3. The Quarto definitibe guide: https://bookdown.org/yihui/rmarkdown
4. The knitR book: https://yihui.name/knitr/
5. LaTeX command guide: https://www.bu.edu/math/files/2013/08/LongTeX1.pdf

Extra - Tips

- Make sure you Render your document often. That way you will find any errors quickly without having to search for them.
- Comments in **R** chunks still are the familiar `#`. But comments in Quarto documents outside of **R** chunks are `<!-- comment here -->`.
- You can bulk comment by highlighting all lines you want commented and then typing `Cmd-Shift-C` on a Mac or `Ctrl-Shift-C` on Windows.

Extra - Mac Quick Look Previewing

If you use the preview feature in macOS (pressing the space bar when highlighting a file in preview), you'll notice that there is no preview available for `.md` files, `.Rmd` files, or files without an extension. You can install QLMarkdown and Syntax Highlight to enable quick viewing of those files.

Section 2

Python in Quarto

Python Intro

One useful feature in Quarto is the ability to run code chunks from other languages. One of the most useful is the ability to run Python inside of Quarto.

There are two ways to do this:

1. Run Python natively.
2. Run Python through **R**.

Run Python in Quarto Natively

If you have Python installed, you can put `engine: jupyter` or `engine: jupyter3` in the YAML. Then you can put Python code chunks like this:

````
```{python}
import numpy as np
np.array([[1, 2, 3], [4, 5, 6]])
```
````

Note that **R** code chunks won't work when using the `jupyter` engine unless other options are specified.

Run Python in Quarto through **R**

We can run Python in Quarto through **R** using the `reticulate` package. This will make it a bit slower than running Python natively.

```
install.packages("reticulate")
library(reticulate)
```

Installing Python and Python Packages

To install Python, we can run the `install_python()` function from the `reticulate` package. Just indicate which Python version you want to install. As of the time these notes were created, the latest version is 3.12.1, but you can look up the latest version or go with the default.

```
install_python("3.14.0")
```

This can take a while to install. Please be patient!

To install Python packages, we need to use an **R** code chunk using the `py_install()` function.

```
py_install(packages = "matplotlib")
py_install(packages = "numpy")
py_install(packages = "pandas")
```

Running Python

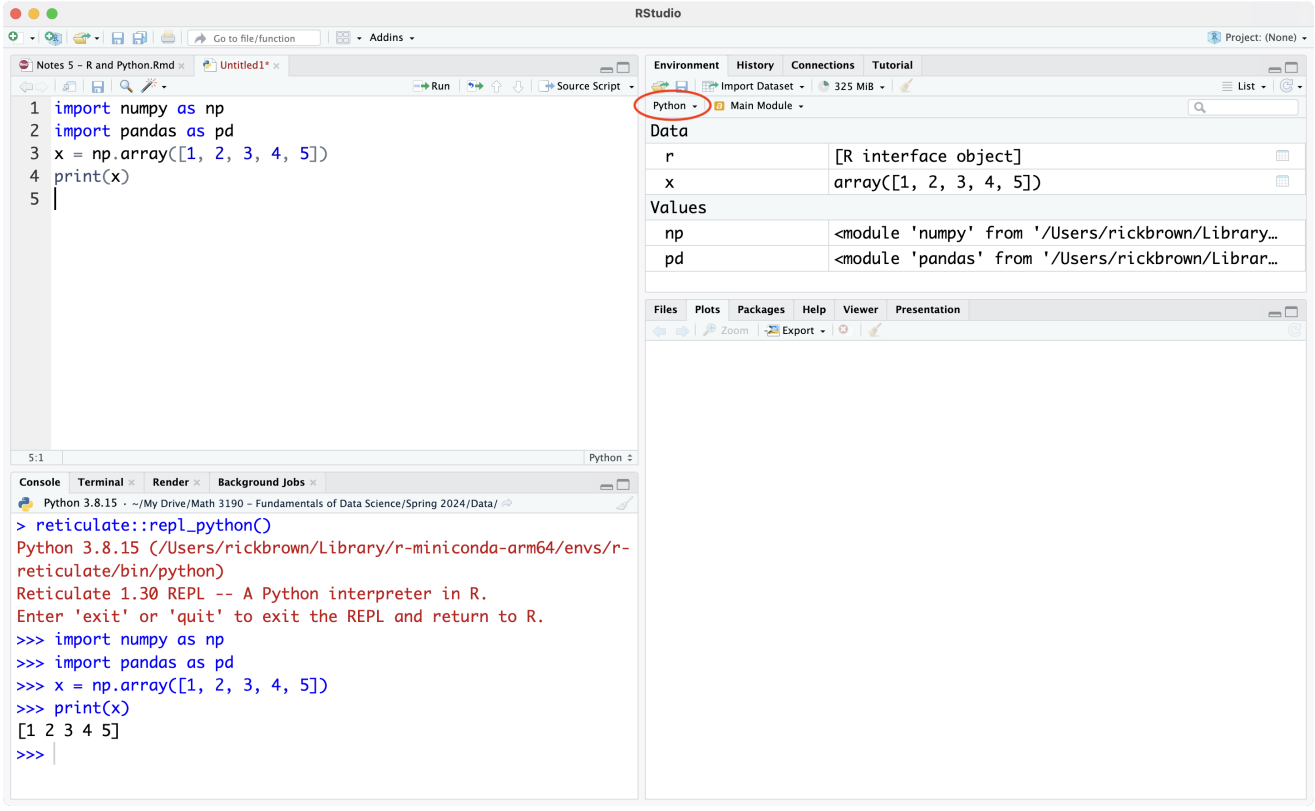
Once we have installed Python, we can insert a Python code chunk just like an **R** code chunk:

````
```{python}
import numpy as np
import pandas as pd
x_python = np.array([1, 2, 3, 4, 5])
print(x_python)
```
````

```
[1 2 3 4 5]
```

Python Envirnoment

When running Python in RStudio, the arrows in the console change and the environment switched to a Python environment. We can view what objects are stored in the **R** and Python envirnoments by clicking on the dropdown button.



Switch Between Python and **R**

We can switch from an **R** enviornment (indicated by a single >) to a Python one (indicated by three arrows >>>) by

- Running a Python code chunk.
- Running the line `reticulate::repl_python()`.
- Double clicking on a Python object in the Python environment list.
- Clicking on the **R** icon at the top left of the Console to bring up a menu and then select Python.

We can switch from a Python environment back to **R** by

- Running an **R** code chunk.
- Typing `exit`.
- Clicking on the Python icon at the top left of the Console to bring up a menu and then select **R**.

Using Python Objects in **R** and Vise-Versa

We can send objects from Python to **R** and vise-versa. Everything defined in Python is stored in the `py` object, which can be called from **R** using `py$name_of_object`. To use an **R** object in Python, we can use preface it with `r.`, so it would be `r.name_of_object`.

```
# R Code. x_python was defined earlier.
library(reticulate)
py$x_python
```

```
[1] 1 2 3 4 5
```

```
y = 2 * py$x_python # Still R code
```

```
# Python Code
print(r.y)
```

```
[ 2.  4.  6.  8. 10.]
```

Type Conversions

While many objects are stored similarly in **R** and Python, it is good to know how things are stored when they are passed back and forth. This table shows the conversion:

R	**Python**
Single-element vector	Scalar
Multi-element vector	List
List of multiple types	Tuple
Named list	Dict
Matrix/Array	NumPy ndarray
Data Frame	Pandas DataFrame
Function	Python function
Raw	Python bytearray
NULL, TRUE, FALSE	None, True, False

Session info

```
R version 4.5.1 (2025-06-13)
Platform: aarch64-apple-darwin20
Running under: macOS Sequoia 15.7.2

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;  LAPACK version 3.12

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/Denver
tzcode source: internal

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

other attached packages:
[1] reticulate_1.44.0

loaded via a namespace (and not attached):
 [1] digest_0.6.37     fastmap_1.2.0     xfun_0.52         Matrix_1.7-3
 [5] lattice_0.22-7    knitr_1.50        htmltools_0.5.8.1 png_0.1-8
 [9] rmarkdown_2.29    tinytex_0.57      cli_3.6.5         grid_4.5.1
[13] compiler_4.5.1    rstudioapi_0.17.1 tools_4.5.1       evaluate_1.0.3
[17] Rcpp_1.0.14       yaml_2.3.10       rlang_1.1.6       jsonlite_2.0.0
```