Building an `R` Report

Matt Malis

Spring 2024

Objective

This document gives an overview on how to produce an `R` report through RStudio. In addition to running `R` code (like you can in a regular `R` script), an `R` report can also include the output of the code, plus blocks of text and mathematical notation, in a single well-formatted and presentable document. This is the format you will use to turn in all of your problem sets and lab reports.

When reading this document, you'll want to look back and forth between the `R` script and the PDF document that it generates.

Workflow

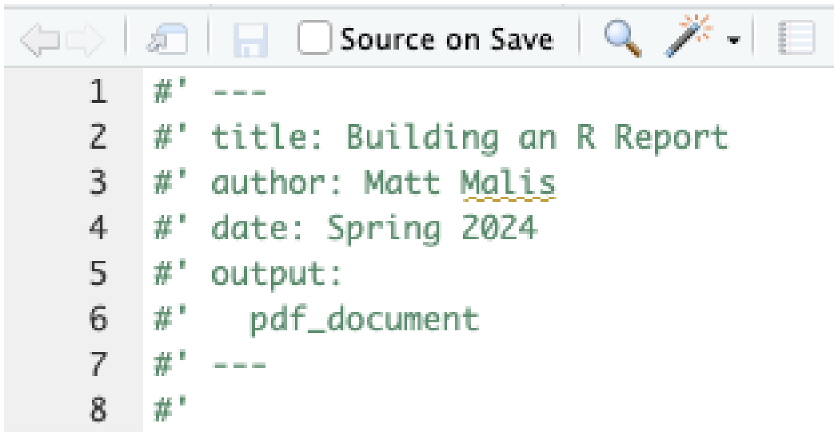
I recommend the following workflow when writing up your `R` reports:

- Have two separate `.R` files open, one code and one report, e.g.: "`PS1_code_Lastname.R`" and "`PS1_report_Lastname.R`".
- Start by writing up your code in the code file (with comments), building it up piece by piece and debugging along the way.
- Once your code runs successfully in the code file, copy it over (piece by piece) into the report file and build the report (piece by piece), with the selected `R` output and the surrounding text.

The general idea here is that you *will* run into problems when writing up your report. This is inevitable with any kind of programming you do. If you try to do everything in the report file at once, when you get an error, it will be hard to figure out whether the error is due to the `R` code, or the markup/typsetting, or some combination of the two. It will make things easier if you break the task down into smaller pieces, so that you can isolate and fix the problems that arise.

Formatting basics

First, your report needs to have a title block, like you see here (and at the top of this `R` script):[1]



```
#' ---
#' title: Building an R Report
#' author: Matt Malis
#' date: Spring 2024
#' output:
#'   pdf_document
#' ---
#'
```

To compile the report:

1. Save your report, with an informative title, e.g. "`sample_report.R`"
2. Click the notebook icon that you see in the top right corner of the image.

Then a PDF file will be created (in the same folder where your `R` script is saved), and a viewer window will pop up with the PDF.

Each line in an `R` report script starts with one of three options:

1. `#'` This line is text (including math and title block)
2. Nothing at the start of a line is `R` code.
3. `#+` This line sets options for the code chunk that follows it

More details on each of these below.

Line type 1: Text

First, as you can see in the script, anything following `#'` (including this line!) gets processed as text.[2] Within a text line, there are a few basic formatting options you'll use frequently:

- You can create a bullet list (like this one) by starting a text line with `*`
  - Then you can go to a second level of bullet list by including two more spaces before the `*`
    - Or a third, and so on…
  - (Note that you need an empty line between the previous text and the start of a bullet list)
- You can start a numbered list by typing "`1.`" (as we did above)
- (For math notation, see the LaTeX section below)
- Make your text look like `R` code by wrapping it in backticks, `‘like this‘`
- Wrap text with a single asterisk to make it *italicized*
- Wrap text with a double asterisk to make it **bold**

[1] In the `R` script you will see that the end of the header actually says `pdf_document: extra_dependencies: ["bbm","setspace"]` . This last line is not necessary in all of your reports; ignore it for now, but see the discussion in the LaTeXsection below.

[2] Note that in the `R` script, in order for `#’` to appear in the text, we had to surround it with backticks.

Line type 2: Code

Anything on a line without `#'` gets processed as `R` code, like this:

```
3 + 5
```

```
## [1] 8
```

The “3+5” (on the gray shaded background) is the code that was run; the “## [1] 8” is the output of the code; 8 is the result of the code, and [1] is indexing the the result. Below, we see that “35” is the 26th element of the output, hence the [26] right before it.

```
10:50
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
## [26] 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

If we start a line with just `#` instead of `#'`, it will appear as commented-out code, e.g.:

```
#  multiplying 3 times 5
3*5 # we can also include a comment after # in the same line as the code
```

```
## [1] 15
```

Line type 3: Code options

Finally, starting a line with `#+` does something different entirely: it provides options for the `R` code that follows it. Look in the `R` script to see some examples of how we’re using these options:

- with `#+ warning=F`:

```
1:2+3:5
```

```
## [1] 4 6 6
```

- with `#+ warning=T`:

```
1:2+3:5
```

```
## Warning in 1:2 + 3:5: longer object length is not a multiple of shorter object
## length
```

```
## [1] 4 6 6
```

- with `#+ eval=T`:

```
a <- 3
```

- with `#+ eval=F`:

```
a <- 4
```

- now we’ll print `a` and see that the value stored is 3, not 4 (because the “`a <- 4`” line was not run)

```
a
```

```
## [1] 3
```

The options we're interested in are outlined in Table 1.

	Code runs?	Results print?	Code prints?	Message prints?	Warning prints?
`warning = F`	✓	✓	✓	✓	
`message = F`	✓	✓	✓		✓
`echo = F`	✓	✓		✓	✓
`include = F`	✓				
`eval = F`			✓		

Table 1: `R` Code Chunk Options

Notes:

- `F` is equivalent to `FALSE`
- the default for each of these instructions is `T`, which is, you guesed it, `TRUE`
- messages: generally useless, print when you load a package (see below)
- warnings: more useful, print when the code runs but `R` suspects it didn't run how you intended
- you can use these instructions in combination, e.g. `message=F, echo=F`, which would run the code, print the results and any warnings, but not print the code or any messages

Packages

R has a lot of built-in functionality (we refer to this as "base R"). But there is also a lot of functionality we'll be using throughout the course that comes from external, user-created packages.

To create the document we're looking at right now, we're relying on a pretty complicated web of interconnected software packages (knitr, pandoc, rmarkdown,LaTeX). (See here for an explanation.) Fortunately, we don't need to understand a lot of what's going on under the hood in order to use the functionality that these packages provide.

For our purposes, we just need to do the following:

First, we need to install a few packages. You only need to do this once on your computer this semester. To install packages, un-comment the following lines of code (i.e. delete te # at the start of each line), compile the report, and then comment out these lines again (i.e. add the # back to the start of each line).

```r
# install.packages('tinytex')
# tinytex::install_tinytex()
# install.packages('tidyverse')
# install.packages('knitr')
```

Note that if you run an install.packages('...') line for a package that's already installed, R will just (re-)install the most recent version of the package. This is usually fine, it just wastes time and is unnecessary.

Next, whenever you open a new R session, you need to load the packages you'll be using in that session (you have to do this every time).

```r
library(knitr); library(tidyverse)
```

$\LaTeX$

$\LaTeX$is a markup language that we can incorporate into our `R` reports. It's a very versatile language that allows us to typeset mathematical notation, among other things. (The footnotes on the second page, and the table on the third page, were created with LaTex.)

For inline math, we wrap `$` symbols around the math. So `$y_i = \alpha + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$` gives us $y_i = \alpha + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$.

Using two dollar signs gives us "display math", which puts the math expression on its own line separate from the surrounding text. (Save this for important definitions, main points, etc.) For example: `$$E[\hat{\mu}] = E[\bar{X}] =   E \left[ \frac{1}{n} \sum_i^n X_i \right] = \mu$$` produces

$$E[\hat{\mu}] = E[\bar{X}] = E \left[\frac{1}{n} \sum_i^n X_i \right] = \mu$$

(Note that in the context of an `R` report, typed math is text, not code. Any line that includes math starts with `#'`.)

You'll notice that just the two lines of math above included a bunch of different pieces of notation. You'll figure these out as you go, and you'll get more familiar with them over time. Google and ChatGPT are your friends.

- Detexify and Mathpix are two apps that convert hand-drawn math symbols to LaTex notation (with varying degrees of success).
- Here is a simple scratchpad app that lets you experiment with LaTex math markup.

A couple more examples, for your reference. Here we'll use the `align` environment to write multiple aligned lines of math.

The LaTex markup:

```
\begin{align}
 \rho_{X,Y} &= \frac{Cov[X,Y]}{\sigma_X \sigma_Y} = \frac{E[(X-E[X])(Y-E[Y])]}{\sqrt{V[X]}\sqrt{V[Y]}}\\
 %
 f(x) &= \frac{1}{\sigma \sqrt{2\pi}} e^{- \frac{(x - \mu)^2}{2\sigma^2}}, \forall x \in \mathbbm{R}  \label{eq:normal}
\end{align}
```

The output:

$$\rho_{X,Y} = \frac{Cov[X,Y]}{\sigma_X \sigma_Y} = \frac{E[(X-E[X])(Y-E[Y])]}{\sqrt{V[X]}\sqrt{V[Y]}} \tag{1}$$

$$f(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{- \frac{(x - \mu)^2}{2\sigma^2}}, \forall x \in \mathbb{R} \tag{2}$$

A few points:

- To use the `align` environment, you simply wrap the text in `\begin{align} ... \end{align}` (and likewise for any other environment)
  - The `&` symbol determines where the two lines of equations will align vertically.

- To show the LaTex markup above the actual output, we had to wrap the markup in the `verbatim` environment, as in: `\begin{verbatim} \begin{align} ... \end{align} \end{verbatim}`
- The `align` environment automatically numbers your equations. This is useful as it lets you refer back to your equations by number.
  - For instance, we can say stuff like “Equation (2) is the probability density function (pdf) for the Normal distribution”, without having to hard-code in the number 2.
  - Referencing the equation takes two steps: First, in the line of the equation, you have to give the equation a label, as in `\label{eq:normal}`. Then, you reference the equation with `\ref{eq:normal}` or `\eqref{eq:normal}` (the latter automatically puts parentheses around the equation number).
    - You can reference a lot of things besides equations. For instance, Table 1, or Footnote 1.
  - If you don't want the equations numbered, you can simply alter the environment by adding an asterisk, as in: `\begin{align*} ... \end{align*}`
- The fancy $\mathbb{R}$ in Equation~(2) required an external package, `bbm`.
  - This is an external LaTeXpackage, which is loaded differently from an external `R` package: this LaTeXpackage was loaded in the title block of this script, with the line `extra_dependencies: ["bbm"]`.
- Notice the `%` in between the two equation lines. `%` functions like the `#` in `R`, “commenting out” whatever follows on the same line.

Presenting and Visualizing Data

We'll spend a lot more time in future lab sessions on data visualization and presentation. Here we just want to go over a couple points that pertain specifically to incorporating these in your R reports.

Creating tables with the `kable` function

First, the `kable` function is useful for creating clean and professional looking tables from `R` data structures. Let's see how this works:

```r
# load in the mtcars data (this comes built-in with base R)
data(mtcars)
# if we just run head(mtcars), or equivalently, print(head(mtcars))..
#  the output will appear as usual R code output
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```r
# This is fine for inspecting ourselves, but it's not good for presenting to an audience
#
# The following simple command will incorporate the table more cleanly into the report
kable(head(mtcars), format="pandoc",
      caption='First Six Rows of the \\texttt{mtcars} Dataset \\label{tab:mtcars}')
```

Table 2: First Six Rows of the `mtcars` Dataset

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

(Notice how we snuck in a label into the table caption, which we can reference as: Table 2.)

```r
# We can make tables not just with the raw data,
#   but also with more informative summaries or other transformations of the data
mtcars_summary = mtcars %>%
  group_by(cyl) %>%
  summarise(
    avg_mpg = mean(mpg),
    n=n()
  )
mtcars_summary %>% kable(format='pandoc')
```

cyl	avg_mpg	n
4	26.66364	11
6	19.74286	7
8	15.10000	14

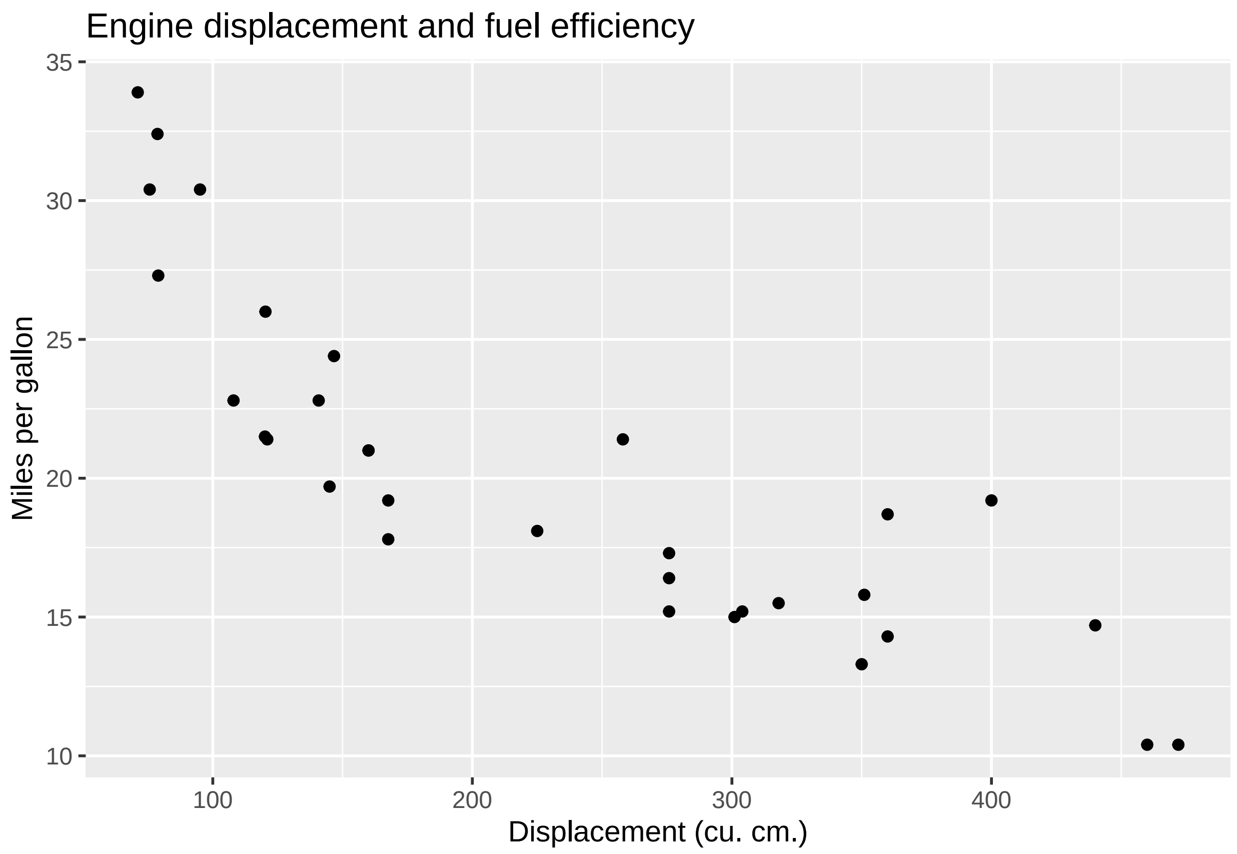
```r
# The column names are ok but not great. And we don't need so many decimals.
#  And let's say we want the columns centered. Can we improve this?
mtcars_summary %>%
  kable(format='pandoc',
        digits=2,
        align='c',
        col.names=c('Cylinders','Avg. MPG', 'Num. Obs.'))
```

Cylinders	Avg. MPG	Num. Obs.
4	26.66	11
6	19.74	7
8	15.10	14

Plotting

We can plot data and have the figures appear in the text.

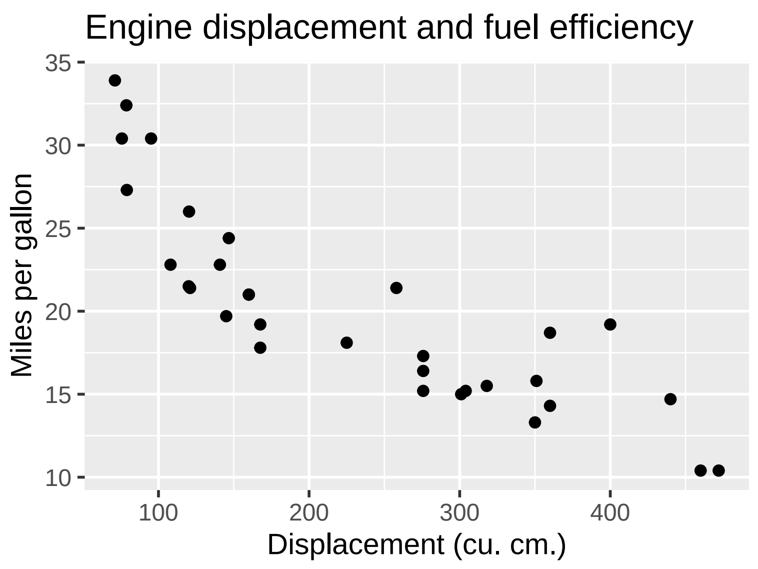
```r
ggplot(mtcars)+
  geom_point(aes(x=disp, y=mpg))+
  ylab("Miles per gallon")+
  xlab("Displacement (cu. cm.)")+
  ggtitle("Engine displacement and fuel efficiency")
```



This plot is a bit too big, we can can change the size options by using a comment that starts with `#+`. For example, to change the plot size, we can specify `#+ fig.width=4, fig.height=3` before plotting. Note that these changes only apply to the figure you're currently plotting. It also often looks better to center the figure on the page, by adding in the command `#+ fig.align='center'`

Here is an example of changing the size of the figure using `#+ fig.width=4, fig.height=3, fig.align='center'`

```r
ggplot(mtcars)+
  geom_point(aes(x=disp, y=mpg))+
  ylab("Miles per gallon")+
  xlab("Displacement (cu. cm.)")+
  ggtitle("Engine displacement and fuel efficiency")
```



A final point on plotting:

Mac users sometimes have trouble with saving R plots. The reason for this is often to do with a piece of graphing software called xquartz that used to come standard on Macs and then didn't. You can download it here: https://www.xquartz.org/.