Welcome!

H. Sherry Zhang

This is an R Markdown document and it will be the primary document format for your labs, homework, and projects in this course. For assignments, you will need to knit the R Markdown file into a PDF document (or knit into HTML and save as a PDF document). Use the "Knit" button in the RStudio toolbar or the shortcut: Command/Ctrl + K.

You may get an error such as "Package xxx required but is not installed" or "there is no package called xxx". These mean a required R package is missing.

1. If you haven't already, install the `tidyverse` package using `install.packages(tidyverse)`
2. You will also need to install the `rmarkdown` package in order to knit documents to HTML or PDF.

NOTE: **DO NOT** put the `install.packages()` code in your R Markdown file. You should only ever call `install.packages()` in the console in RStudio.

Exercise 1. Knit this document. You should get a PDF document with the same name as this R Markdown file (i.e. `hellow-world.pdf`).

R Markdown 101

An R Markdown document allows you to combine text, codes, and their outputs in a single document:

```
head(mtcars, 5)
```

```
##                   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
```

The code above prints the `mtcars` dataset, which is a built-in dataset in R.

When working with an R Markdown document, it's important to knit the document frequently. This helps you catch errors early and see the effects of your changes as you make them. In what follows, you'll read and edit the document while observing how the output updates in response. Along the way, you'll also learn some R Markdown basics.

Edit text

To edit the text in this document, you can use the visual editor (i.e. click the "Visual" bottom on the top left of the document) or write directly in the Markdown format (i.e. click the "Source" bottom). The visual editor allows you to format text using a graphical interface, like Microsoft Word, while the source editor allows you to write text in a plain text format that can be easily converted to HTML, PDF, or Word documents.

There are lots of Markdown cheatsheets online for syntax references, but here are a few to get you started:

- Different levels of headings are controlled by the number of `#`. Check how first level heading (`R Markdown basics`) and second level heading (`Edit text`) are created.

- *Italic text* is created by a single asterisk `*` and **Bold text** is created by two asterisks `**`.
- This list is an ordered list, created by `-` or `*` at the beginning of each line. You can also create an ordered list by using numbers followed by a period, like `1.`, `2.`, etc.
- Content inside the backticks (`` ` ` ``) are formatted as code, like `head(mtcars, 5)` - they are printed in monospaced font.
- Of course, you can create link to the R website. Click me - this is a hyperlink.

Edit code

R code are written in code chunks that are enclosed by three backticks and the language name (e.g. `r` for R code). This is an example to load the `tidyverse` package:

```r
# I am an empty code chunk and you can put code comment after the `#` sign
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.2     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

You may have noticed some printed messages in the PDF about attached packages. This message tells you which packages were loaded with `tidyverse`. These are useful messages to know as you are doing analysis,**however, for assignment submissions, you generally don't want to show these messages and warnings.** You can control this using the `warning` and `message` chunk options. For example, you can set `warning = FALSE` and `message = FALSE` to suppress warnings and messages in one particular chunk. To apply an option to all code chunks in a document, you can set these options in the setup chunk at the beginning of the document:

````
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE, eval = TRUE, warning = FALSE, message = FALSE)
```
````

Exercise 2. Set the `warning` and `message` options to `FALSE` in the setup chunk on line 10 of this document. *If you did this correctly, you should not see the messages about the attached packages in the output document.*

You may have also noticed the option `echo = TRUE` in the code chunk above. This controls whether the R code is shown in the output document. As exercises, you should keep `echo = TRUE` so your code is visible, but for formal reports or presentations, you might prefer `echo = FALSE` to hide the code and show only the results.

Exercise 3. Compare the outputs of the two code chunks below: one with `echo = TRUE` and one with `echo = FALSE`. How do they differ?

```r
head(mtcars, 5)
```

```
##                   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
```

```
##                    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
```

YAML (Yet Another Markup Language)

At the top of the document, the YAML header, wrapped in three dashes (`---`), specifies the title, author, and output format of the document. You can modify these fields to customize your document.

Exercise 4. Change the author to your name.

If you have reached this point, you have successfully graduated from R Markdown 101! For now, you can seat back and read through the following sections to know what we will be learning in this course, which we will go through in the lecture on Friday.

What to Expect in Week 1–8

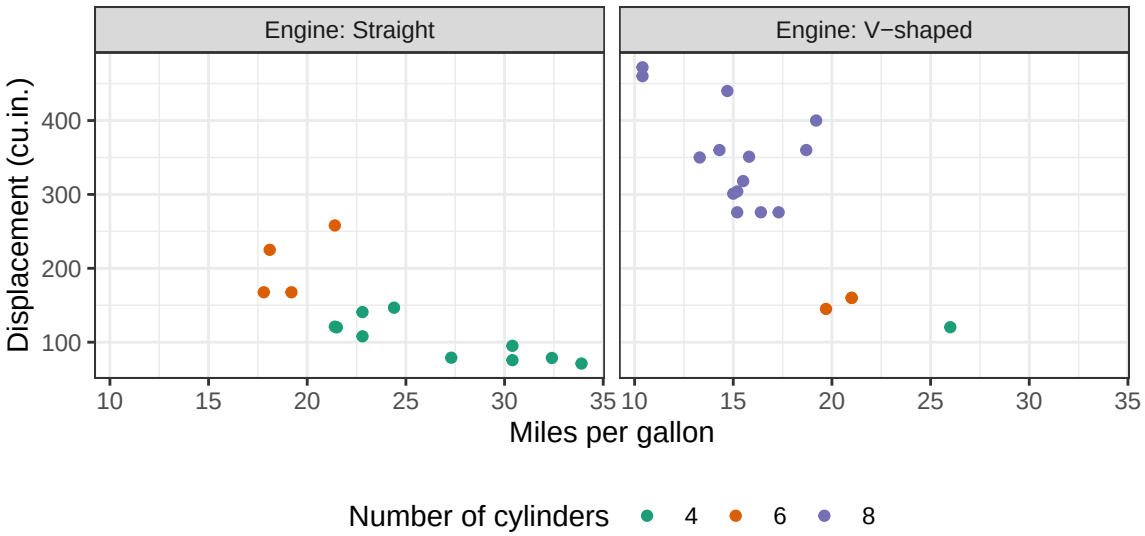
The first part of the course (week 1-8) will center on statistical computing, where you will learn how to explore data through wrangling and visualization. We'll begin with a simple 2D dataset like `mtcars`, and in week 2-3, you'll be able to code something like this and know what each function does:

```
mtcars |>
  mutate(kpl = mpg * 0.425144) |>
  filter(vs == 0) |>
  group_by(cyl) |>
  summarize(disp = mean(disp, na.rm = TRUE),
            kpl = mean(kpl, na.rm = TRUE)) |>
  arrange(disp)
```

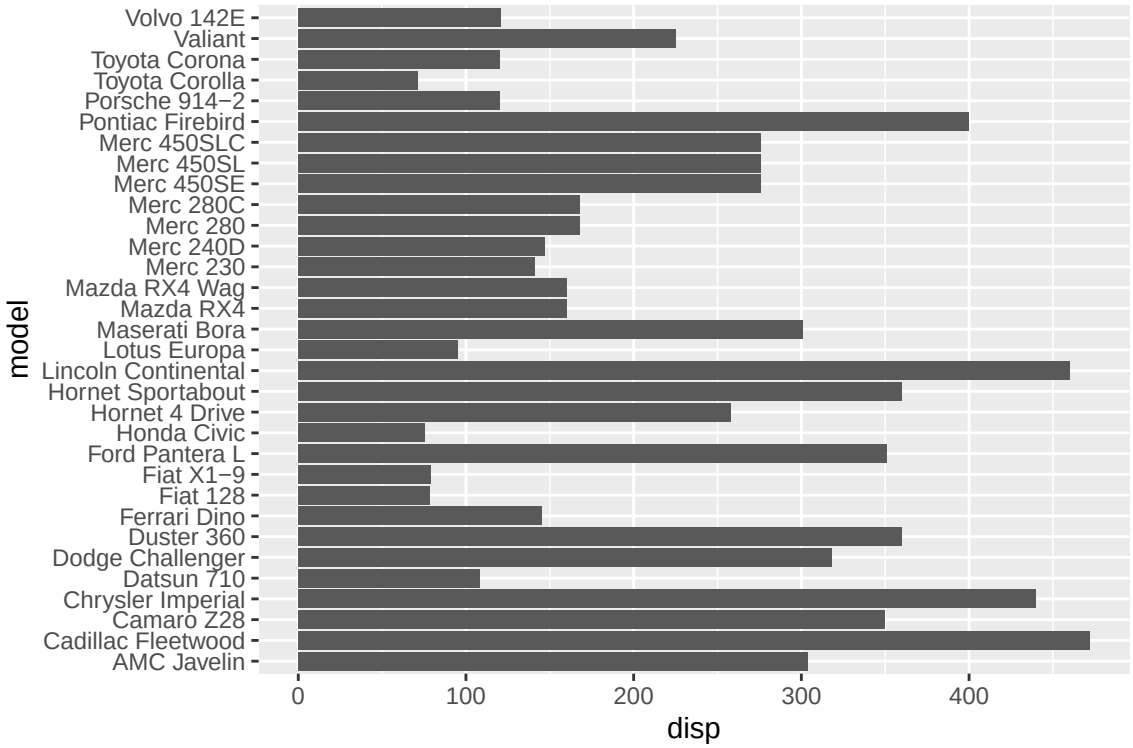
Then we will learn how to plot data in R with the `ggplot2` package (week 3-4) and you will be able to create plots like this:

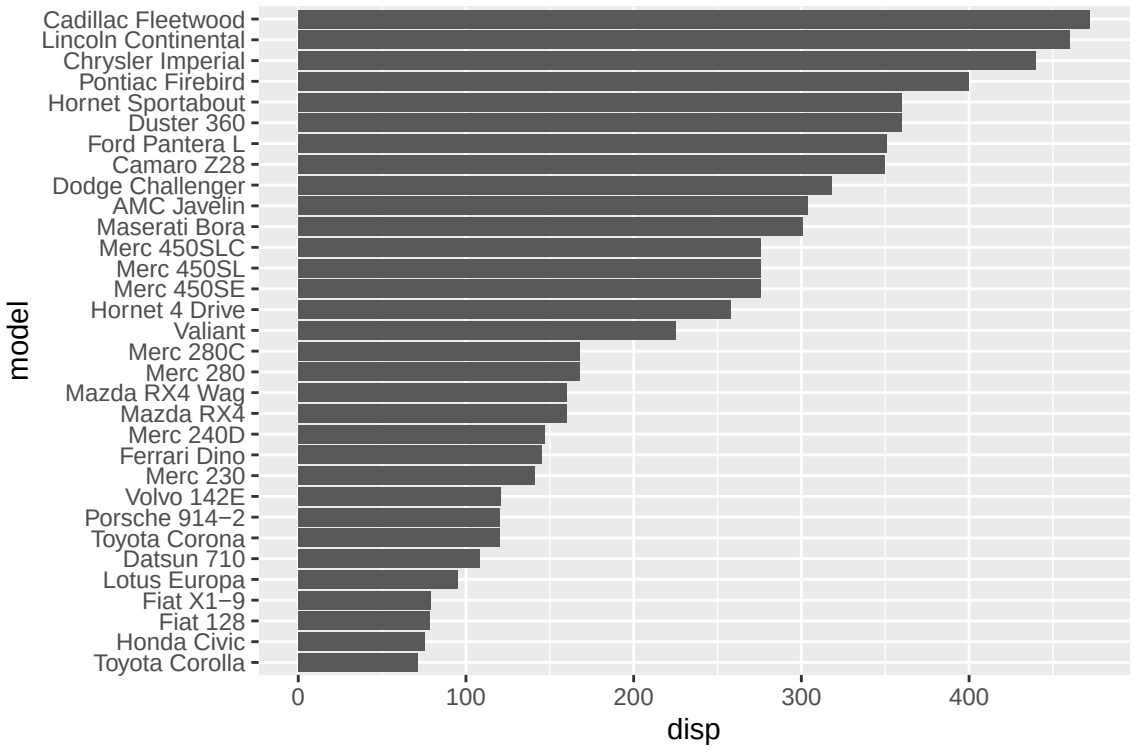
```
mtcars2 <- mtcars |>
  mutate(vs = ifelse(vs == 0, "V-shaped", "Straight")) |>
  rename(Engine = vs)

mtcars2 |>
  ggplot(aes(x = mpg, y = disp, color = as.factor(cyl))) +
  geom_point() +
  facet_wrap(vars(Engine), labeller = label_both) +
  scale_color_brewer(palette = "Dark2", name = "Number of cylinders") +
  theme_bw() +
  theme(legend.position = "bottom") +
  xlab("Miles per gallon") +
  ylab("Displacement (cu.in.)")
```



In addition to learning the syntax of the `ggplot2` package, we'll also cover key principles of data visualization to help you create plots that effectively communicate the information you want to convey. For example, consider the two bar charts below, both showing the exact same information of car models against engine displacement. I hope you would prefer the second plot, where car models are ordered by engine displacement instead of alphabetically. We'll discuss why some visual choices are better communication and how to create them using `ggplot2`.





Once you have learned the basic tools, the key skill is to be able to translate what you want to do with the data into "tidyverse language" and choose the right functions to make it happen. We don't expect you to memorize every function argument (we can't either!), or become a tidyverse wizard in just four weeks. Instead, the goal is to help you build fluency by practicing repeatedly with case studies.

From Weeks 5–8, we'll slightly adjust the datasets and pipelines so you can reinforce what you've learned in Weeks 2–4, while picking up some new tools along the way. For example, we'll introduce:

- specialized wrangling tools for specific data types: time series (lubridate), strings (stringr), and multiple datasets (dplyr joins)
- spatial data, which may look daunting but can be explored with tools sharing the `tidyverse` principles
- web scraping, a common method to collect real-world data for analysis, and
- text data, using the `tidytext` package.

The aim of these weeks is to help you become confident and comfortable using `tidyverse` tools across a variety of data types and analysis tasks.