

DATA 545 Tutorial

Stacy DeRuiter

2025-08-20

Table of contents

Preface

This book contains notes and materials for the Calvin University course DATA 545 (Applied Regression Modeling).

Part I

Getting Started

1 Installation

1.1 Motivation

One way to access R and RStudio is via an account at <https://r.stem.calvin.edu/> (or if not a Calvin student, at posit.cloud). To use RStudio on one of these servers, all you need to do is log in, with nothing to install or maintain.

For many, this cloud approach is a great way to use RStudio, and they have no reason to install a standalone copy of the software on a personal computer. If you are happy using the server, exit this tutorial now and continue happily using the server!

If you have concerns about your internet bandwidth, speed, usage limits, or firewalls, or if you want to be able to work on assignments for this class somewhere *without* internet access, or if you need to analyze really large datasets or fit complex models, you may want to install R and RStudio and work locally instead of on the server.

1.1.1 Pros and Cons

The **benefits** of downloading your own copy are that **you can work offline** and should not be subject to any (hopefully rare) server-related errors, freezing, etc.

The **negatives** of downloading your own copy are that you have to maintain it yourself, installing and updating packages and software.

1.2 Goal

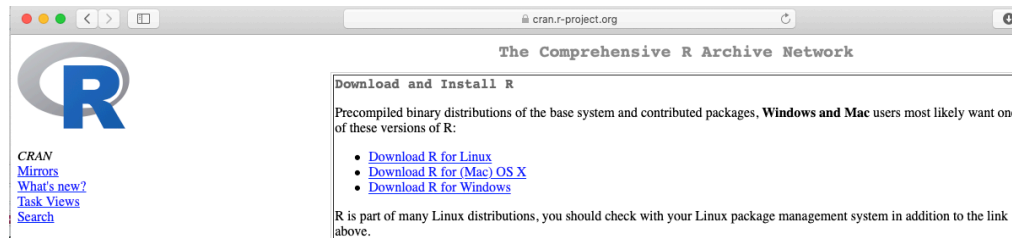
This document will guide you through the process of installing R, RStudio, and other necessary R packages on your own computer, *if you choose to do so*. Again, there is no course requirement to do this.

The process will have three stages, which work best in order:

1. Install R
2. Install RStudio
3. Install necessary R packages from within RStudio

1.3 Download/Install 1 of 2: R

R downloads are available from <https://cran.r-project.org/>.



- Select the download that matches your operating system and hardware (Mac OS, Windows, Linux, etc.)
- You only need the “base” version.
- Download the installer and run it. You may want to choose not to create shortcuts, since you will access R only through RStudio.

1.3.1 Mac with Homebrew

Windows and Linux users: skip this section.

- If working on Mac OS and already using Homebrew to manage software packages, you can skip the manual download above and just run:

```
brew install r
```

- If you want to get Homebrew and install this way **on a mac**, [there are detailed instructions online](#) – scroll down to “Instructions for Mac Users”. Note that you don’t necessarily need OpenBLAS for this course (as recommended on the linked website); it does not really matter either way.

1.4 Download/Install 2 of 2: RStudio

Once you have installed R, you next need to install **RStudio**.

- Downloads of RStudio are available at <https://rstudio.com/products/rstudio/download/>.
- You should select the **free version of RStudio Desktop**.
- Download and install the version that matches your operating system

	RStudio Desktop Open Source License	RStudio Desktop Commercial License	RStudio Server Open Source License	RStudio Server Pro Commercial License
	Free	\$995 /year	Free	\$4,975 /year (5 Named Users)
	DOWNLOAD Learn more	BUY Learn more	DOWNLOAD Learn more	BUY Evaluation Learn more
Integrated Tools for R	✓	✓	✓	✓
Priority Support		✓		✓
Access via Web Browser			✓	✓
Enterprise Security				✓
Project Sharing				✓
Manage Multiple R Sessions & Versions				✓
Admin Dashboard				✓

1.4.1 Mac with Homebrew

Windows and Linux users: you can't use Homebrew.

- If using a mac and Homebrew, you can alternatively install RStudio via:

```
brew cask install rstudio
```

1.5 Install 3/3: Packages

In addition to base R and the RStudio IDE, we use a few add-on packages that you will need to install yourself.

- Open RStudio
- In your RStudio **Console** window, which is on the lower left by default, type (or copy and paste) the code below and click “Enter” to run it:

```
install.packages(c('rmarkdown', # reproducible research documents
                  'tidyverse', 'remotes', # graphics and data wrangling
                  'pander', # formatting tables
                  'glmmTMB', 'mgcv', # fitting regression models
                  'car', 'ggeffects', # working with fitted models
```

```

# optional additions:
# 'mosaic', # formula-based summary stats and resampling
# 'openintro', # datasets
# 'shiny', 'plotly', 'gganimate', 'leaflet' # interactive graphics/maps
))
# if desired, for function to ggplot ACFs:
remotes::install_github('stacyderuiter/s245')

```

- In addition to the packages you listed specifically, a number of dependencies (other packages that the packages you requested require to work) will be installed.
- The amount of time it takes will depend on your computer and internet connection speed, but as long as it finishes without any messages that literally say “Error: ...”, it worked!.
- If RStudio prompts you to update packages or install additional dependencies, it’s usually a good idea to do so.
- If R asks you if you want to install a certain package “from source” blah, blah, “is newer...” usually you can answer yes (or no) and it will work either way.
- If you get an error or have any questions, get in touch with your professor.

1.5.1 TeX for PDF generation

To enable generation of PDF output from Rmarkdown documents, there is a little more code to run. (If you don’t know what this means yet, you will soon - and you *do* probably need to be able to do it.)

This one has two steps: installing the package, and then *using* the package to install the PDF-generation utility.

If you already have TeX/LaTeX/MikTeX installed on your computer, you can probably skip this installation (but it won’t hurt).

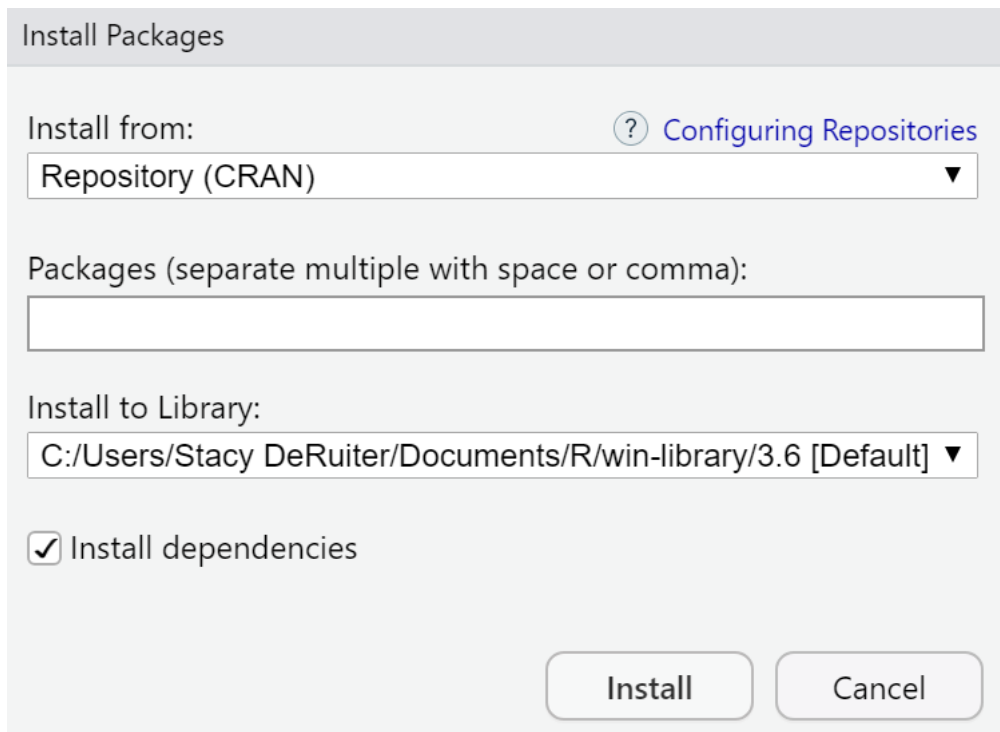
```

install.packages('tinytex')
tinytex::install_tinytex() # install TinyTeX

```

1.5.2 Point-and-Click Option

If you would like to install the packages interactively instead of on the command line (as already shown above), you can click the **Packages** tab on the lower right in RStudio, then click *Install* at the top of the tab. Enter the names of the packages you want to install in the middle “Packages” blank, and leave the rest of the default options, then click “Install”.

The image shows the 'Install Packages' dialog box in RStudio. It has a title bar 'Install Packages'. Below the title bar, there is a section 'Install from:' with a help icon and a link 'Configuring Repositories'. A dropdown menu shows 'Repository (CRAN)'. Below that is a text input field for 'Packages (separate multiple with space or comma):'. Then there is a section 'Install to Library:' with a dropdown menu showing 'C:/Users/Stacy DeRuiter/Documents/R/win-library/3.6 [Default]'. At the bottom left, there is a checked checkbox 'Install dependencies'. At the bottom right, there are two buttons: 'Install' and 'Cancel'.

1.6 You did it!

If you complete all three steps above, you should have a working version of RStudio on your machine. To use it, just open RStudio; it should look nearly identical to the RStudio Server version you have been using online.

You don't ever have to open or access R directly; RStudio does it all for you.

In case of any errors or problems, contact me (stacy.deruiter at calvin.edu) anytime and I'll do my best to help.

(Don't contact school help desks; they don't support this software).

2 R Basics

```
#| setup: true
#| autorun: true
#| include: false
#| exercise:
#|   - find-sqrt
#|   - round-sqrt
#|   - c-and-sum
#|   - name-a-variable
#|   - cat-var
#|   - check-out-data
#|   - get-help
#|   - baseball-mistakes
#|   - look-at-MI_lead
#|   - rename-MI_lead

library(readr)
library(tidyr)
library(dplyr)
library(ggformula)
library(mosaic)
library(mosaicData)

theme_set(theme_bw())

knitr::opts_chunk$set(
  echo = TRUE,
  fig.align = "center",
  fig.width = 6, fig.height = 2.5)

options("readr.edition" = 1)

data(HELPrct, package = "mosaicData")

MI_lead <- read_csv(file='https://sldr.netlify.app/data/MI_lead.csv')
```

2.1 Your Mission

The purpose of this tutorial is to help you start to get familiar with the way R works, and some basic R commands...even if you haven't yet installed R on your computer or made a [posit.cloud](#) account.

This tutorial environment uses webr, which lets you read some helpful information, then immediately practice writing and running your own R code, *all in your web browser*.

Here's hoping it provides a nice, gentle introduction in a controlled environment!

2.2 Communicating with R

You will do most of your work in R with *code* or *commands*. Instead of pointing and clicking, you will type one or more lines of code, which R will *execute* (doing the work you have asked it to do).

Then, R will return the results of whatever operation you asked it to do - sometimes producing a plot, other times creating a plot.

Sometimes executing code has almost no visible effect (no plot or text output is produced), but instead some object is created and stored in R's *environment* for later use.

2.2.1 Two Key Questions

To get R (or any software) to do something for you, there are two important questions you must be able to answer. Before continuing, think about what those questions might be.

2.2.2 The Questions

To get R (or any software) to do a job for you, there are two important questions you must be able to answer:

1. What do you want the computer to do?
2. What must the computer know in order to do that?

2.2.3 Providing R with the information it needs

R *functions* provide R with the answer to the first question: what do you want the computer to do?

Most functions in R have short, but descriptive names that describe what they do. For example, R has some functions to do basic mathematical operations: the function `sqrt()` computes the square root of a number, and the function `round()` rounds a number (by default, it rounds to the nearest integer).

But just giving R a function is not enough: you also need to answer the second question (what information does R need to do the job?). For example, if you want to use the function `round()`, you also need to provide R with the number you want to round!

We will provide answers to our two questions by filling in the boxes of a basic template:

```
function ( information1 , information2 , ...)
```

(The ... indicates that there may be some additional *input arguments* (input information we could provide to R) we could add eventually. Some functions need only one input, but if a function takes more than one argument, they are separated by commas. They have names, and if named (like: `function(input_name = value, input2_name = 'value')`) they can be in any order.

2.2.4 Using simple functions

Let's practice what you just learned, trying out the mathematical `sqrt()` and `round()` functions.

Edit the code below to compute the square root of 64:

```
#| exercise: find-sqrt
function(information_R_needs)
```

Hint 1

Consider using the `sqrt()` function:

```
sqrt(___)
```

Hint 2

The input information that `sqrt()` needs to make your calculation is the number you want the square root of: 64.

Solution.

💡 Solution:

```
sqrt(64)
```

Now try computing the square root of 44, *and then* rounding it to the nearest integer:

```
#| exercise: round-sqrt
```

💡 Hint 1

You'll need to use *two* functions this time:
The `sqrt()` function, and then the `round()` function.

```
sqrt(____)  
round(____)
```

💡 Hint 2

The input information that `sqrt()` needs to make your calculation is the number you want the square root of: 44. Run that code first, to get the input you will need for `round()`...

```
sqrt(44)  
round(____)
```

Solution.

💡 Solution:

```
sqrt(44)  
round(6.63325)
```

Can you do it all in one go? Well...yes!

```
round(sqrt(44))
```

There's also an easier-to-read way to do that, using a *pipe operator* `|>`. It takes the output of one operation and passes it as input to the next. You can read it as `|>` = "and then..." so we could do:

```
sqrt(44) |>  
  round()
```

1. Take the square root of 44, *and then*
2. round the result.

(More on pipes later!)

2.2.5 Storing information in R: variables

In the last section, you computed the square root of 44 and then rounded it, perhaps like this:

```
sqrt(44)
```

```
[1] 6.63325
```

```
round(6.63325)
```

```
[1] 7
```

But to do that, you probably had to first find the root, make a note of the result, and then provide that number to the `round` function. What a pain!

A very useful option, if you have value (or a variable, dataset, or other R object) that you will want to use later on, is to store it as a named object in R. In the previous example, you might want to store the square root of 44 in a variable called `my_root`; then you can provide `my_root` to the `round()` function without checking the result of the `sqrt()` calculation first:

```
my_root <- sqrt(44)
round(my_root)
```

```
[1] 7
```

Notice that to assign a name to the results of some R code, you use the symbol `<-`. You can think of it as an *assignment arrow* – it points *from* a value or item *toward* a *name* and assigns the name to the thing.

Try editing the code to change the name of the variable from `my_root` to something else, then run your new code:

```
#| exercise: name-a-variable
my_root <- sqrt(44)
round(my_root)
```

i Hint

Make sure you change the name `my_root` in *both* places.

Solution.

i Solution

```
your_new_name <- sqrt(44)
round(your_new_name)
```

2.2.6 What if I have a list of numbers to store?

Sometime you might want to create a variable that contains more than one number. You can use the function `c()` to *concatenate* a list of numbers:

```
my_fave_numbers <- c(4, 44, 16)
my_fave_numbers
```

```
[1]  4 44 16
```

(First we stored the list of numbers, calling it `my_fave_numbers`; then we printed the results to the screen by simply typing the variable name `my_fave_numbers`).

Try making a list of your three favorite numbers, then using the function `sum` to add them all up:

```
#| exercise: c-and-sum
```

i Hint

First use `c()` to concatenate your chosen numbers (separated by commas).
Don't forget to use `<-` to assign your list of numbers a name!
Then, use `sum()` to add them up.

```
my_numbers <- c(___, ___, ___)  
sum(___)
```

Solution.

i Solution

This is just one possible solution.

```
my_numbers <- c(4, 16, 44)  
sum(my_numbers)
```

Notice you *could* also nest `sum()` and `c()`, or use the pipe operator `|>` to calculate the numeric answer, *but then you would not have the object `my_numbers` available for later use...*

```
sum(c(4, 16, 44))  
# or  
c(4, 16, 44) |>  
  sum()
```

2.2.7 What about data that are not numeric?

R can work with categorical data as well as numeric data. For example, we could create a list of words and store it as a variable if we wanted (feel free to try changing the words if you want):

```
#| exercise: cat-var  
my_words <- c('RStudio', 'is', 'awesome')  
my_words
```

2.2.8 What if I have a LOT more data to store?

`c()` works great for creating small lists of just a few values, but it is *not* a good way to enter or store large data tables - there is lots of potential for user error. In this course, you will usually be given a dataset already in electronic form; if you need to create one, you would turn to spreadsheet or database software. Either way you read the existing data file into R directly.

In R, these larger datasets are stored as objects called `data.frames`. The next sections will get you started using them.

2.3 How should data tables be organized for statistical analysis?

A comprehensive guide to good practices for formatting data tables is available at <http://kbroman.org/dataorg/>.

A few key points to keep in mind:

- This data table is for the computer to read, not for humans! So eliminate formatting designed to make it “pretty” (color coding, shading, fonts...)
- Use short, simple variable names that do not contain any spaces or special characters (like `?`, `$`, `%`, `-`, etc.)
- Organize the table so there is one column for every variable, and one row for every observation (person/place/thing for which data were collected).
- Use informative variable values rather than arbitrary numeric codes. For example, a variable `Color` should have values `‘red’`, `‘white’`, and `‘blue’` rather than 1, 2, and 3.

You will have chances to practice making your own data files and importing them into R outside this tutorial.

2.4 Using built-in datasets in R

R has a number of built-in datasets that are accessible to you as soon as you start RStudio.

In addition to the datasets that are included with base R, there are add-on *packages* for R that contain additional software tools and sometimes datasets.

To use datasets contained in a package, you have to load the package by running the command:

```
library(packagename)
```

2.4.1 Example of loading a package

For example, we will practice looking at a dataset from the package `mosaic`.

Before we can access the data, we have to load the package. The code might look like this:

```
library(mosaic)
```

(Nothing obvious will happen when you run this code...it basically just gives R permission to access the package, so there is often no output visible.)

2.4.2 Viewing a dataset

The `mosaic` package includes a dataset called `HELPrct`.

If you just run the dataset name (`HELPrct`) as a command, R will print some (or all - egad!) of the dataset out to the screen! (So don't...)

But...how can we extract selected, useful information about a dataset?

2.4.3 Gathering information about a dataset

There are a few functions that make it easier to take a quick look at a dataset:

- `head()` prints out the first few rows of the dataset.
- `names()` prints out the names of the variables (columns) in the dataset
- `dplyr::glimpse()` (function `glimpse()` from package `dplyr`) gives an short list-like overview of the dataset
- `skimr::skim()` (function `skim()` from the package `skimr`) prints out more detailed graphical summary information about a dataset
- `nrow()` reports the number of rows (observations or cases) in the dataset
- `ncol()` reports the number of columns (variables) in the dataset

Try applying each of these functions to the `HELPrct` data and see what the output looks like each time:

```
#| exercise: check-out-data  
function(____)
```

Solution.

Solution

The input for each of the functions is the name of the dataset: `HELPrct`.

```
head(HELPrct)
names(HELPrct)
nrow(HELPrct)
ncol(HELPrct)
skimr::skim(HELPrct)
dplyr::glimpse(HELPrct)
```

In this case, the point is usually to view the information on-screen, not to store it for later use, so we have not used `<-` at all to store any output for later use or reference.

2.4.4 Getting more help

You can get help related to R function, and built-in R datasets, using a special function: `?`. Just type `?` followed by the name of the function or dataset you want help on:

```
#| exercise: get-help
```

Solution.

Solution

For example, if you want to know about the function `nrow()`:

```
?nrow
```

2.5 Reading in data from a file

For this class, you will often be asked to analyze data that is stored in files that are available online - usually in csv format. It's simple to read them into R. For example, we can read in the file `MI_lead.csv`, which is stored at https://sldr.netlify.app/data/MI_lead.csv using the function `read_csv()` (from package `readr` or super-package `tidyverse`):

```
library(readr) # the readr package contains the read_csv() function
MI_lead <- read_csv(file = 'https://sldr.netlify.app/data/MI_lead.csv')
```

2.5.1 The most common mistakes

The code below contains several of the **most common mistakes** students make when they try to read in a data file. See if you can find and correct them **all**!

The code below - if corrected - *would* (on posit.cloud or in standalone R/RStudio) run without an error and read in some baseball statistics from the file <http://stacyderuiter.github.io/teachingdata/data-raw/baseball.csv>.

Here in this tutorial, it may give the error: `! curl package not installed, falling back to using url()` – there’s not a straightforward fix, sorry, but try it on the server if you want to prove to yourself that it works!

```
#| exercise: baseball-mistakes
read_csv(http://stacyderuiter.github.io/teachingdata/data-raw/base)
```

Hints

Think about:

- Is the filename or URL spelled correctly, with no typos?
- Is the filename or URL in quotation marks (either " or ' work equally)?
- Is the URL complete (including the file extension ".csv")
- Was `<-` used to assign a *name* to the dataset once read in? (Otherwise it will just be uselessly printed to the screen and not available for later use!)

Solution.

Solution

```
baseball_data <- read_csv(file = 'http://stacyderuiter.github.io/teachingdata/data-raw/baseball.csv')
```

2.5.2 What about local files?

The same function, `read_csv()`, can be used to read in a local file. You just need to change the input to `read_csv()` – instead of a URL, you provide a path and filename (in quotes). For example, the input `file = 'https://sldr.netlify.app/data/MI_lead.csv'` might become `file = 'C:\\Data\\MI_lead.csv'`.

We won’t do an example in this tutorial because it’s not straightforward to work with local files within a tutorial environment, but you can practice it once you are working independently in RStudio.

If you are working on the server `r.cs.calvin.edu`, you will have to *upload* files to your cloud space on the server before you can read them in (RStudio on the server cannot access files on your computer's hard drive). Look in the “Files” tab on the lower right, and then click “Upload.”

2.5.3 Named input arguments

The input argument we provided to R is the URL (in quotes – either single or double quotes are fine). But notice that this time, we gave the input argument a *name*, “file”, and specified its value with an equal sign.

This is not *required* - the command works fine without it:

```
MI_lead <- read_csv('https://sldr.netlify.app/data/MI_lead.csv')
```

However, if a function has *more than just one* input argument, it's good to get in the habit of providing names for the inputs. If you provide names, then the order in which you list the inputs doesn't matter; without names, **the order matters** and you have to use `?` to figure out what order R expects!

2.5.4 Renaming variables in a dataset

This is an advanced topic, so don't worry if it seems complicated; for now, it's just nice to realize some of the power R has to clean up and reorganize data.

What if we didn't like the names of the `MI_lead` variables? For example, a new user of the dataset might not know that `ELL` stands for “elevated lead levels” and that `ELL2005` gives the *proportion* of tested kids who had elevated lead levels in the year 2005.

If we wanted to use a clearer (though longer) variable name, we might prefer “`prop_elevated_lead_2005`” instead of “`ELL2005`” – more letters to type, but a bit easier to understand for a new user. How can we tell R we want to rename a variable?

We use the code:

```
MI_lead <- MI_lead |>
  rename(prop_elevated_lead_2005 = ELL2005)

glimpse(MI_lead)
```

The code above uses some tools you've seen, and some more advanced ones you haven't seen yet. The symbol `|>` is called a “pipe” and basically means “and then...” Translated into words, the code above tells R:

- Make a dataset called `MI_lead` by starting with the dataset `MI_lead`.
- **Next, take the results do something more with them** (`|>`) ...
- `rename()` a variable. What I want to rename is the variable `ELL2005`. Its new name should be `prop_elevated_lead_2005`.”

See...you can already start to make sense of even some pretty complicated (and useful) code.

Note: If you give R several commands, *not* connected by pipes, it will do the first, then the second, then the third, and so on. R doesn't need the pipe for permission to continue! Instead, the pipe tells R to take the *results* from the first command, and use them as the input or starting material for the next command.

2.5.5 Check out the data

OK, back to business - simple functions and datasets in R.

It's your turn to practice now. Use one of the functions you have learned so far to extract some information about the `MI_lead` dataset.

How many rows are in the dataset? How many variables (columns)?

What are the variables named, and what are their values like?

Remember, ? won't work on `MI_lead` because it's not a built-in R dataset. Also, the dataset `MI_lead` is already read in for you, here...so you don't need to use `read_csv()`.

```
#| exercise: look-at-MI_lead
```

2.6 Review

What have you learned so far? More than you think!

2.6.1 Functions in R

You've learned that R code is made up of functions, which are generally named descriptively according to the job they do. Functions have one or more input arguments, which is where you provide R with all the data and information it needs to do the job. The syntax for calling a function uses the template:

```
function ( information1 , information2 , ...)
```

2.6.2 Variables in R

You’ve practiced creating variables in R using `c()`, and saving information (or the results of a computation) using the assignment arrow `<-`.

2.6.3 Datasets in R

You’ve considered several different ways to get datasets to work with in R: you can use datasets that are built in to R or R packages, or you can use `read_csv()` to read in data files stored in .csv format.

2.6.4 Vocabulary

You should now be able to define and work with some R-related terms:

- *code* or *commands* that R can *execute*
- *function* and *inputs* or *arguments*
- *assignment arrow*: `<-`
- *pipe* = “and then...”: `|>` (note: `|>` is an older way of writing a pipe, and it does basically the *same* thing as `|>`)

2.7 Congratulations!

You just completed your first tutorial on R, and wrote some of your own R code. I *knew* you could do it...

Want more help and practice? Consider checking out outside resources from posit: <https://posit.cloud/learn/primers>

3 Using Quarto

3.1 Instructions

While you work through this chapter, you will create a Quarto (.qmd) document.

Quarto lets you combine R code, output, and text in a single document that can be rendered in HTML, PDF, Word and more formats.

It's like magic: you save all your text and R code in a simple file; when you're ready, push a button and it's compiled into an output document with nicely formatted text, code (optional to include, but for this class you always will), and **all the figures and tables generated by your code**.

Since all the data analysis and results are automatically included in the compiled output document, your work is *reproducible* and it's easy to re-do analysis if the data change, or if a mistake is uncovered.

3.2 Reference Materials

For more details on using Quarto, and detailed documentation, see <https://Quarto.org/docs/guide/>.

Quarto and posit also provide substantial resources for learners. This tutorial is tailored to our course, including just the stuff you need and not much you won't use frequently. But if you want *even more* about Quarto, you might check out:

- Tutorials for beginners at <https://Quarto.org/docs/get-started/hello/rstudio.html> (Hello, Quarto! and Computations are most relevant.)
- Detailed documentation at <https://Quarto.org/docs/guide/>.

3.2.1 Optional Video

If you love video introductions, consider also this [23-minute offering from posit and Mine Cetinkaya-Rundel](#):

3.3 Logistics

To create a .qmd file, you will have to work in RStudio (outside this tutorial environment). So, as you work on this tutorial, you will probably switch back and forth between the tutorial itself and an RStudio session on your computer or on the server at <https://r.stem.calvin.edu> (or if not at Calvin, at posit.cloud).

Historical Note: The precursor of the Quarto document is the Rmarkdown (.rmd) document (and even older - the Sweave document). If you know and love one of those, you may use it, but probably best to upgrade to Quarto, which is superceding them.

3.4 Getting Started

3.4.1 Logging in to RStudio

Log in to your account at <https://r.stem.calvin.edu> (or if not at Calvin, at posit.cloud).

Or, if you installed R on your own machine, open RStudio.

3.4.2 Panels

When you open RStudio, you will see at least three different panels: The *Console* is on the left. On the upper right are *Environment*, *History* and maybe more; on the lower right are *Files*, *Plots*, and *Packages*. Explore a little to try to see what is there!

Files shows you the files saved in your personal space on the server. You can organize, upload, and delete files and folders.

3.4.3 Executing code in R

You can *do* things in R by typing commands in the *Console* panel.

However, working that way makes it hard to keep a record of your work (and hard to redo things if anything changes or if a mistake was made).

For this class, you will instead **work in Quarto files, which can contain text, R code, and R output (such as figures).**

After you have opened a file (like an RMarkdown file) on the RStudio server, the *Console* panel will be on the lower left and the newly opened file will be on the top left. Let's learn how to do it...

3.5 Quarto (qmd) Files

3.5.1 Quarto files are stand-alone!

Every Quarto file (qmd file) must be completely stand-alone. It doesn't share any information with the *Console* or the *Environment* that you see in your RStudio session. **All** R code that you need to do whatever you are trying to do must be included in the qmd file itself!

For example, if you use the point-and-click user interface in the RStudio *Environment* tab to import a data file, that dataset will *not* be available when rendering your qmd file.

Similarly, if you load the `mosaic` package by typing in the *Console* window,

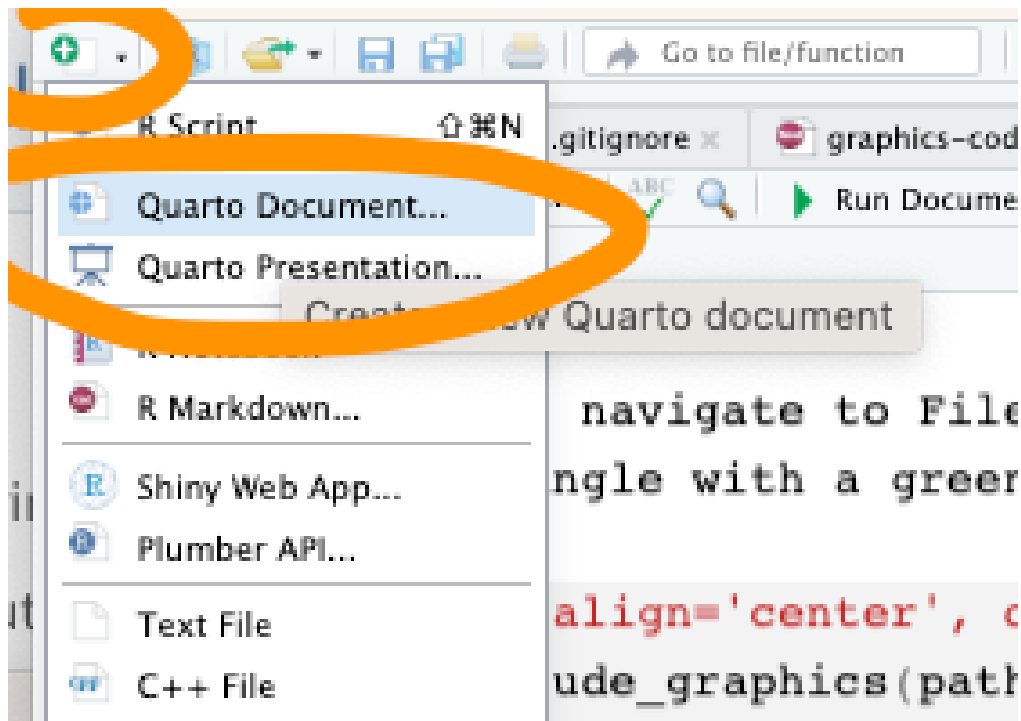
```
library(mosaic)
```

mosaic functions and data will not be available to use within the qmd file.

So: Keep your qmd files stand-alone! (You have no choice, actually...)

3.5.2 Create a Quarto file

In RStudio, navigate to File -> New File -> Quarto Document..., or click on the white rectangle with a green circle+ :



and select Quarto from the drop-down menu.

Choose html or pdf output.

(Why not Word? Too much temptation to make changes and do formatting after the fact in Word...which makes your work no-longer-reproducible. In qmd, you have documented everything you've done. If you make changes after rendering to Word, that's not true anymore.)

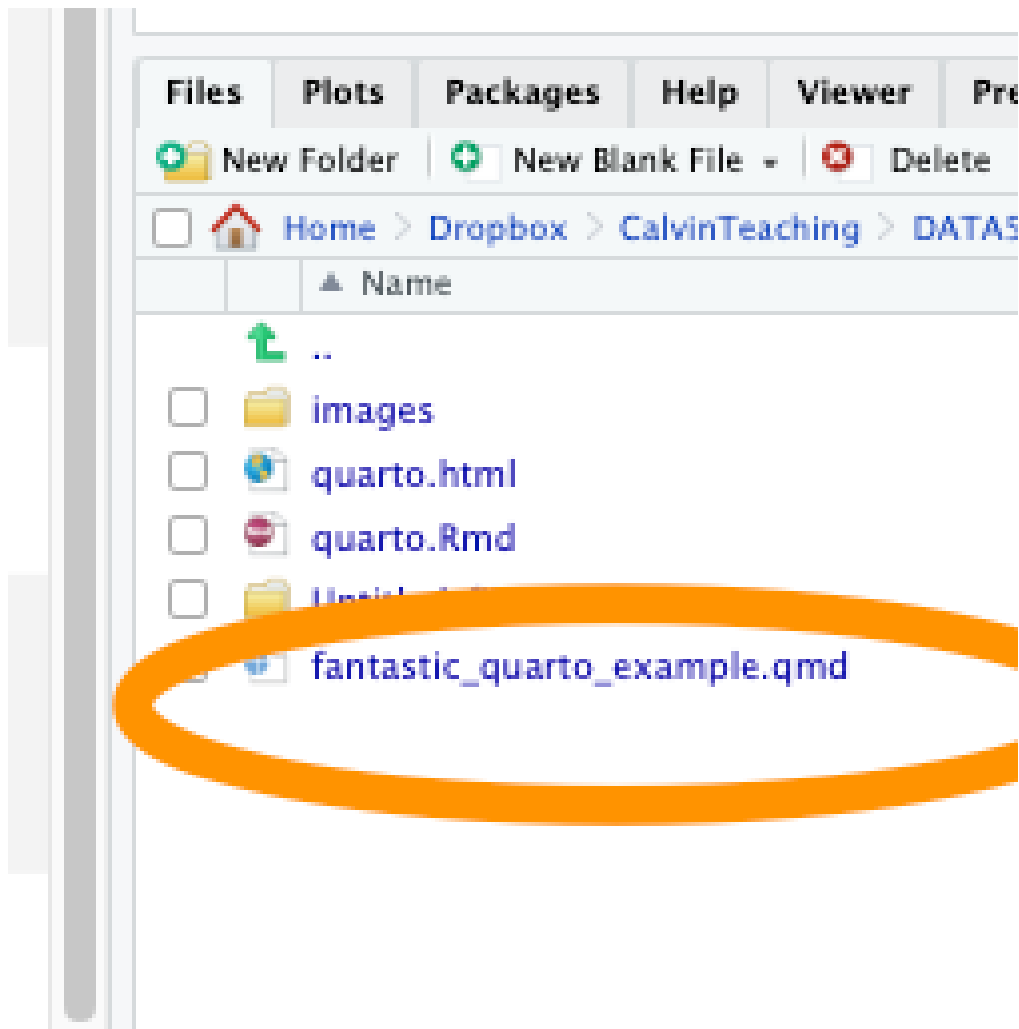
3.5.3 Save your qmd file

Save your file by clicking on the disk icon at the top of the file tab (give it a clear file name like `deruiter_quarto_practice.qmd`).

Do your best to avoid spaces and special characters in your file names.

If on a server, the file will be saved to the cloud, not to your computer.

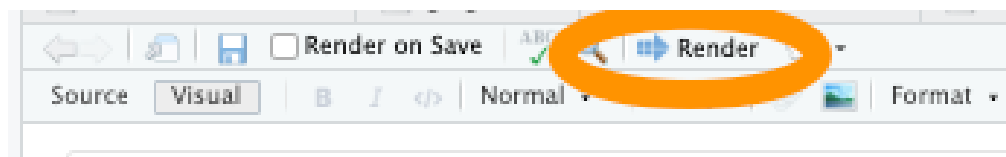
All your files will be accessible in the RStudio *Files* tab (lower right panel) whenever you log into RStudio, regardless of which computer you are using. You may organize them into directories (folders) if you want.



3.5.4 Render!

How do qmd files actually work? What's so cool about them?

Click on the *fat blue arrow* next to the word “Render” at the top of the file window.

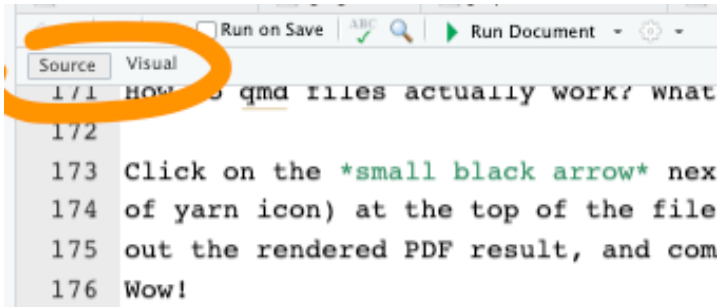


Check out the rendered html or pdf result, and compare it to the original Quarto file.

Wow!

3.5.5 Source vs. Visual Editor

Look to the upper right corner of your qmd file. You should see some buttons that allow you to toggle between “Source” and “Visual” editor modes.



In your own file, toggle back and forth a few times. The **Source** mode lets you see (and type) the straight-up markdown – which is probably nice if you’re already used to it, and annoying or mystifying if not. The **Visual** mode is more of a what-you-see-is-what-you-get (like the rendered version), point-and-click type interface. You may use whichever you prefer.

Be aware that if you are going to copy/paste between documents, you probably want to do so in **Source** mode.

3.5.6 Personalize your Markdown file

At the top of the Quarto file, there is a section called the “YAML header”. It starts and ends with 3 dashes - - -.

In this part of the file, be very careful what you type: a stray space or character will lead to an error.

This is where you can enter an appropriate title, author(s), and date (within the quotation marks). You can also choose the format you want to render to (usually pdf or html – not in quotes).

```

v ---
  title: "My Title"
  author: "Stacy DeRuiter"
  date: "June 26, 2023"
  format: html
  editor: visual
  ---

```

Customize your YAML header in your own Quarto doc, and then render again to see the effect.

Make sure you do this for every assignment! (No prof or boss likes getting submissions called “Untitled”...)

3.6 Quarto YAML settings

3.6.1 PDF or html?

For our course, you can choose to render to *either* an html file or a PDF file.

So, you’ll have either `format: pdf` or `format: html` in your YAML header. You can also try `format: typst` to render PDF files a bit faster (learn more about [typst output format](#) online).

But *if* you choose html, there’s an important change you have to make to the YAML header to ensure your html file is stand-alone. Meaning: you want all images, etc. to be embedded in the one file rather than stored in an accompanying folder. Otherwise, when you (say) upload the file on Moodle or email it, all the images and graphs will be omitted...yikes! Yes, embedding these makes the file larger, but if you are sharing the rendered html document, you need to.

If rendering to html, it is *essential* that you specify the setting `embed-resources: true`!

So, *make sure* you add `embed-resources: true` after the entry `format: html:` in your YAML header, exactly as shown below.

Make sure to keep the spacing and line breaks just as shown.

The indents are each two spaces, so there are 2 spaces before `html:` and 4 before `embed-resources:`.

```

---
title: "Lego Challenge - Follow-up"
author: "Stacy DeRuiter"
format:
  html:
    embed-resources: true
    code-tools: true
editor: visual
---

```

3.6.2 Code tools

Note that the YAML header shown above also had a second option activated for rendered html files: `code-tools: true`.

```

---
title: "Lego Challenge - Follow-up"
author: "Stacy DeRuiter"
format:
  html:
    embed-resources: true
    code-tools: true
editor: visual
---

```

What does this one do?

It adds a button “Code” at the top right of your file.

Lego Challenge - Follow-up

</> Code

AUTHOR
Stacy DeRuiter

Data

If you click it, you can view and copy the source code (basically, the contents of the original qmd file before rendering). This is not a bad option, for example for homework, as it allows me to see every detail of the settings you used and may help me troubleshoot any issues.

A screenshot of a 'Source Code' window with a close button (X) in the top right corner. The window contains the following text:

```
---
title: "Lego Challenge - Follow-up"
author: "Stacy DeRuiter"
format:
  html:
    embed-resources: true
    code-tools: true
  editor: visual
---

```{r setup, include=FALSE}
library(mgcV)
library(glmTMB)
library(mosaic)
```

## 3.7 Text and Code in Quarto

### 3.7.1 Text

The Quarto file is where you save all the R commands you want to use, plus any text commenting on the work you are doing and the results you get. Parts of the file with a plain white background are normal text.

You can format the text. For example, enclosing a word in asterisks will generate italics, so *\*my text\** in the qmd file will become *my text* in the PDF. Using two asterisks instead of one will generate boldface, so **\*\*my text\*\*** becomes **my text**. You can also make bulleted lists, numbered lists, section headers, and more. For example,

#### Some Text

becomes

**Some Text**

(a sub-section header). Fewer hashtags make the text even larger, and more make it smaller.

Caution! Forgetting the space after the last hashtag will format your text verbatim rather than as a header (*#fail*). Failing to leave a blank line before the header can also make formatting fail.

Check out the Quarto Markdown Basics reference at <https://quarto.org/docs/authoring/markdown-basics.html> for more examples of how to format text in Quarto.

Before moving on, try a few of the tricks you just learned in your qmd file. Make it pretty!



### 3.7.2 qmd file anatomy: R code chunks

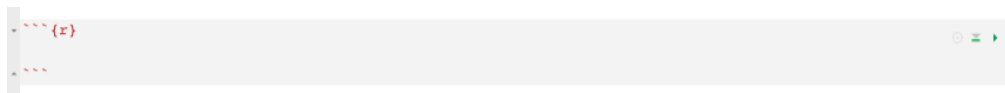
An qmd file can (of course!) contain one or more **R code chunks**. These sections of the file have a grey background onscreen. In Source mode, each one begins with

“`{r}`

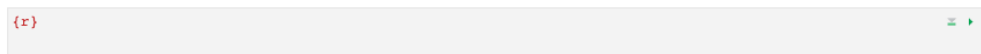
and ends with

“

like so:



In Visual mode you can't see the ‘:

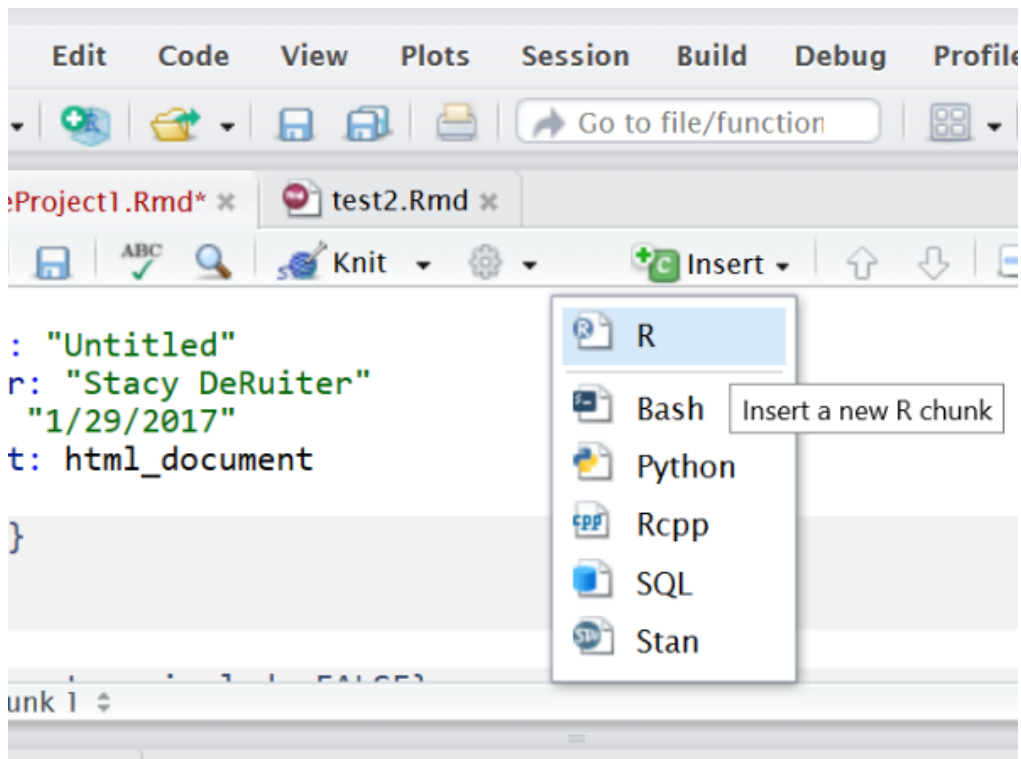


### 3.7.3 How to add a new R code chunk to your file

To add a code chunk to your file in Source editor mode, you have three options.

1. You can type in the header and footer by hand to start and end the chunk.
2. You can click on the “add chunk” button at the top right. It’s a green box with the C inside (at the top of the qmd file; choose the first option, “R”, in the pulldown) to insert an empty chunk.
3. You can use a keyboard shortcut: Windows, Ctrl + Alt + I or OS X, Cmd + Option + I

When you click the **Render** button, code in code chunks will be run, and any output will be included in the document.



### 3.7.4 Setup Chunk

Consider using the first R code chunk in a qmd file to specify settings (for graphics, display, etc.). In this chunk, you can also give R permission to use certain packages (software toolkits) with

```
library(packagename)
```

For example, we will use the `ggformula` package for graphics. So, verify that the first R code chunk in your file includes the line `library(ggformula)`.

You can also specify options for each R code chunk - these go at the top, prefaced by `#|`. A typical setup chunk for our course might look like:

```

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

library(tidyverse)
library(ggformula)
library(glmTMB)
library(ggeffects)

theme and font size for graphics
theme_set(theme_bw(base_size = 14))

options for code echoing, figure alignment, and figure size
knitr::opts_chunk$set(
 echo = TRUE,
 fig.align = "center",
 fig.width = 6, # in inches
 fig.height = 2.5)

```

Notice that several packages are loaded (that we will use frequently). `theme_set()` is used to specify some settings for graph output, and `knitr::opts_chunk$set()` is used to specify whether or not to include R code in the rendered file (Yes please: use `echo: true!`) and specify the default figure size.

There are *tons* more options and settings, and you can explore them at <https://yihui.org/render/options/#chunk-options>.

But for now, if you use something like the setup chunk shown above, it should work well and have what you need for almost all work in this course.

### 3.7.5 The settings chunk is invisible!

If you look carefully at the rendered output, you will see that the setup chunk does *not* appear there. That's intentional - when you load packages with `library()`, they often print a lot of long and pretty useless messages, which you want to omit from your rendered document.

This is achieved by having the setting `include: false`

However, for our course, **no chunk other than the setup chunk should have the setting “include: false”** (or `echo: false` for that matter). Generally, anyone evaluating your coursework needs to see all the code you used, not just its output.

### 3.7.6 Clean Up

At this point, you probably want to get rid of all the extra content in the template.

If you haven't put a setup chunk into your own qmd file...do it now! Here's another reminder of how it would look:

```

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

library(tidyverse)
library(ggformula)
library(glmTMB)
library(ggeffects)

# theme and font size for graphics
theme_set(theme_bw(base_size = 14))

# options for code echoing, figure alignment, and figure size
knitr::opts_chunk$set(
  echo = TRUE,
  fig.align = "center",
  fig.width = 6, # in inches
  fig.height = 2.5)
```

```

Next, Delete **everything** in the file *other than* the YAML header and your setup R code chunk.

Now the clutter is gone and you have space to include your own R code and text.

(Before going further, make sure it still renders.)

## 3.8 Run R Code

There are multiple ways to run and test R code from a markdown file. Sometimes you want to render the whole file and get the PDF or HTML; other times you want to run just a specific bit of code to make sure it's working correctly.

### 3.8.1 Running R Code from a qmd file: Render the file

Every time you render the file, all R code will be run automatically.

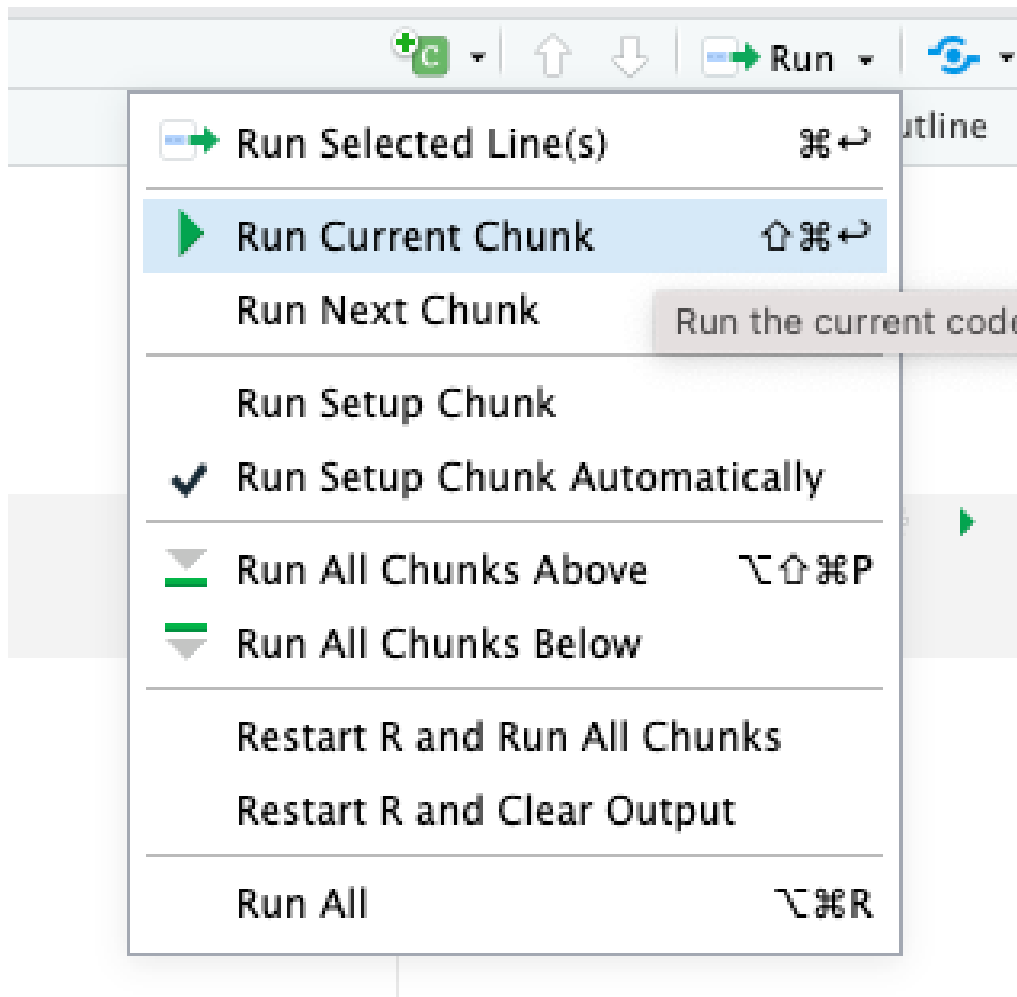
A side note: PDF or HTML? Which is preferable?

I think PDFs are a little more portable and a good default option, and their formatting is best for anything you are going to print out or share via email (especially with less technically inclined folks).

However, later in the semester we may see how to create some pretty cool interactive graphics and/or tables in R, and these can only be rendered in HTML. For this class, you may use either one. (But not Word, remember? Because you'll lose reproducibility...)

### 3.8.2 Running R Code from a qmd file: Run Menu

You can also use shortcuts/buttons to run specific chunk(s). Here is one way to do it (option 1): Use the *Run* pulldown menu at the top of the file. (Choose the option you want based on what you are trying to do).



### 3.8.3 Running Code from a qmd file: Shortcut Button

Here is another way to use shortcuts/buttons to run only a specific chunk (option 2): Click on the green arrow at the upper right of a code chunk to run the chunk.

```
##{r}
this is a comment (a short note to yourself)
hi_birds <- read_csv('https://sldr.netlify.app/data/hawaii_birds.csv')
##
```

### 3.8.4 Running Code from a qmd file: Copy and Paste

Finally, here's a third way to use shortcuts/buttons (option 3):

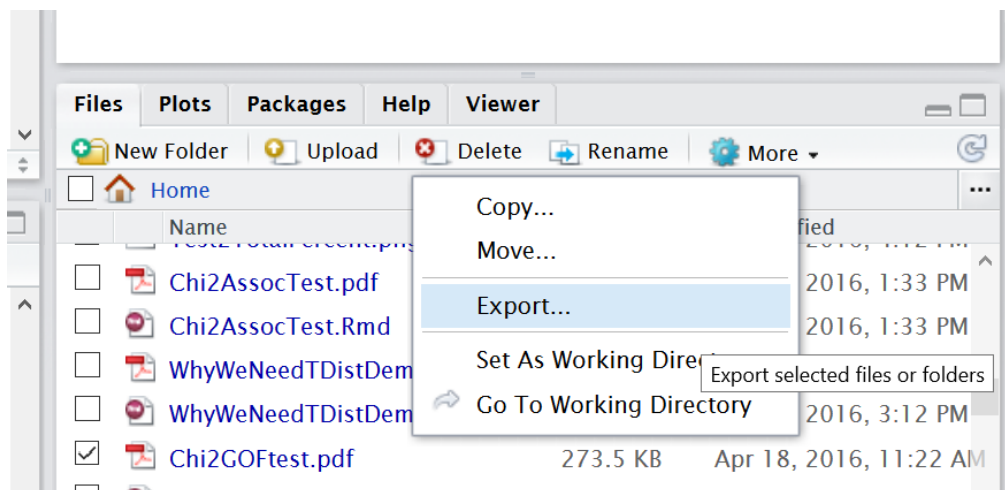
Copy the code you want to run, paste to the console window, and hit Enter.

(Or, place your cursor in the line you want to run and hit ctrl + enter (Windows) or cmd + enter (Mac).)

## 3.9 Downloading files from RStudio

You will have to download your files if you want a copy on your own computer, or to be able to upload a copy to Moodle to turn in.

To download, go to the *File* tab, check the box for the file you want, then select More - Export from the menu at the top of the *File* tab.



## 3.10 Quarto Files Stand Alone!

We already covered this once, but we’re covering it again because it’s one of the most common student mistakes in qmd files!

If you run R code in the console or the RStudio GUI (for example, reading in a data set by pasting code into the console or using the *Import Dataset* button in the *Environment* tab), **you won’t be able to use the results in your markdown file.**

Any and all commands you need, including reading in data, need to be included in the file.

The reverse is also true. If you run just one R code chunk in a qmd file using the “run” buttons mentioned above, or by copy-pasting into the console, you are effectively running that code in the console.

If R gives an error saying it cannot find a certain function, variable, or dataset, the most likely fix is to run the *preceding* code chunks (especially **setup!**) before the one you’re stuck on.

## 3.11 Data from a URL

You can load online datafiles in .csv format into R using the function `read_csv()`. The input to `read_csv()` is the full url where the file is located, in quotation marks. (Single or double quotes – it doesn’t matter which you choose, as they are equivalent in R.)

For example, we will consider a dataset with counts of the numbers of birds of different species seen at different locations in Hawai’i. It is stored at [https://sldr.netlify.app/data/hawaii\\_birds.csv](https://sldr.netlify.app/data/hawaii_birds.csv), and can be read into R using the command below.

```
hi_birds <- read_csv('https://sldr.netlify.app/data/hawaii_birds.csv')
```

### 3.11.1 When you read in data, store it to a named object

Note that we didn’t just run the `read_csv()` function – we assigned the results a **name** so that instead of printing the data table to the screen, R stores the dataset for later use.

```
hi_birds <- read_csv('https://sldr.netlify.app/data/hawaii_birds.csv')
```

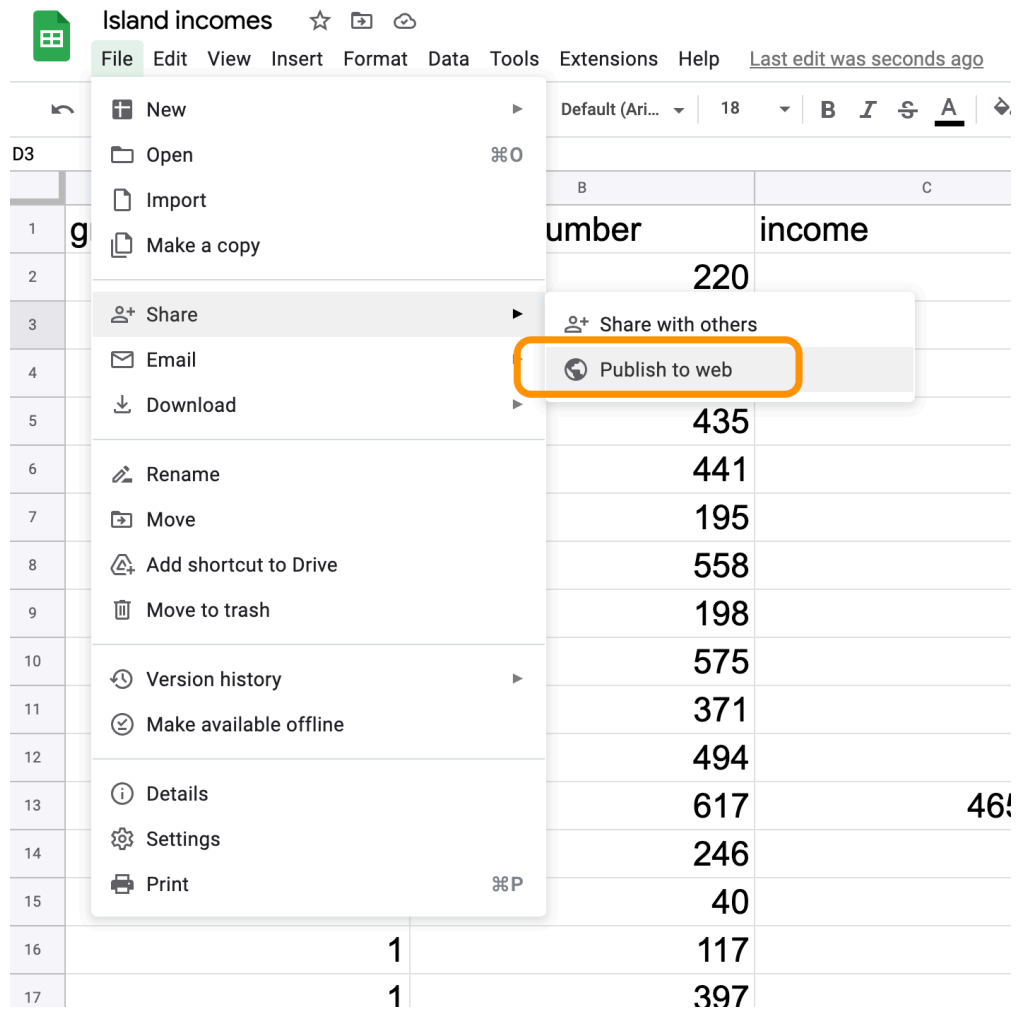
Here, we assigned the name **hi\_birds** to the dataset using an “assignment arrow” `<-` (the “arrow” points from the object toward the name).

## 3.12 Data from Google Sheets

There's also a simple way to read in data from a Google Sheet.

First, go to the Google Sheet online to prepare it by “publishing it online”.

In the **File** menu, choose “Publish to the Web”:



In the pop-up window, choose to publish your “Entire Document” as a .csv file:



## Publish to the web

This document is not published to the web.

Make your content visible to anyone by publishing it to the web. You can link to or embed your document. [Learn more](#)

**Link** Embed

---

Entire Document ▾ Comma-separated values (.csv) ▾

Publish

---

► Published content & settings

Finally, copy the resulting link.

**Publish to the web** ×

This document is published to the web.

Make your content visible to anyone by publishing it to the web. You can link to or embed your document. [Learn more](#)





**Link** Embed

---

Entire Document ▾ Comma-separated values (.csv) ▾

Press Ctrl+C to copy.

DwaM47JRO9wXZ\_UM1mUcCF94dlm50RH4uEgo8-qsSNRdGWp/pub?output=csv

Or share this link using:    

**Note:** Viewers may be able to access the underlying data for published charts. [Learn more](#)

Published

You can use `read_csv()` with this link as input to read your data into R.

### 3.13 Data from a File

You can also upload your own data file to `posit.cloud`, or save it to your computer if you installed R/RStudio, and then read it in to R using `read_csv()`. The basic process is:

- Use spreadsheet software to create the data table
- Save the file as a csv file
- Upload the csv file if working on `posit.cloud`
- Use the `read_csv()` function to read the file into R

### 3.14 R functions

After reading the data in, you can use R functions to have a look at it, for example:

```
head(hi_birds)
glimpse(hi_birds)
nrow(hi_birds)
```

Try each of the lines of code above in R. What do the functions `head()`, `glimpse()`, and `nrow()` **do**? Try to figure it out based on the output they produce.

If you get stuck, consult R's built-in help files. Remember, you can access the help for a function by running the code `?functionName` – for example, if you want help on `head()`, run:

```
?head
```

## 4 R Results in Quarto Text

```
library(mosaic) # for computing summary stats w/formula interface
```

### 4.1 Including results of R calculations in your text

You may want to include the results of R calculations in the TEXT part of a report. Then, if the calculated value changes, the text can be automatically updated to match.

Let's say you compute the mean of some kids' foot lengths:

```
mean(~length, data = KidsFeet)
```

```
[1] 24.72308
```

#### 4.1.1 Simple but Inefficient

You may want to cite the result in the text part of your file...so you would type:

The mean length of the kids' feet was ' r mean(~length, data=KidsFeet) ' cm.

To get:

The mean length of the kids' feet was 24.7230769 cm.

#### 4.1.2 Side Note: back-ticks

*Those accent marks (before the “r” and at the end of the R-code stuff) are not normal single quotes or apostrophes; they are “back-ticks” or “graves” ( ‘ ), just like those used to help define the start and end of R code chunks in your Quarto file. There should not actually be a space between the ‘ and the r.*

### 4.1.3 More Efficient

It's annoying (and sometimes not really practical) to (re)type the entire R command in the text part of your file. An option is to save the quantity you want to refer to as a variable in R:

```
mean_length <- mean(~length, data = KidsFeet)
```

Then you can write: The mean foot length of the kids was ' r mean\_length' cm.

To get: The mean foot length of the kids was 24.7230769 cm.

## 4.2 Rounding

What if you want to report numeric values with a more reasonable number of decimal places? Use `round()`: The mean foot length of the kids was ' r round(mean\_length, digits = 2)' cm

and you get: The mean foot length of the kids was 24.72 cm

You can also consider using the function `signif()` if you want to specify the number of significant digits rather than the number of decimal places.

## 4.3 R results with more than one value inside

What if you want to cite a value from an object that contains more than one value?

### 4.3.1 Vectors

For example, what if you computed means for both boys and girls? The output would be a vector of two means, then.

You can use hard brackets ( `[ ... ]` ) to refer to the first, second, etc. entries. For example:

```
girlboy.means <- mean(~ length | sex,
 data = KidsFeet)
```

You type: The girls' mean foot length was ' r girlboy.means["G"] ' , and the boys' was ' r girlboy.means["B"] ' ,

to get: The girls' mean foot length was 24.3210526, and the boys' was 25.105.

You can also use numeric indices – for example, ‘`r girlboy.means[2]`’ instead of ‘`r girlboy.means[“G”]`’ to get the girls’ value – but using names when you can is often safer because you don’t have to worry about whether things are stored in the order you think that they are!

### 4.3.2 Matrices, Tables, data.frames, tibbles...

If you are referring to a data table or other object with multiple rows and columns, you can use the syntax `[row.numbers, column.numbers]` to extract a row, a column, or a specific value of interest. If you leave either `row.numbers` or `column.numbers` blank, all rows/columns will be included.

For example, consider a table showing some data from a survey of intro stat students (`Ticket` tells whether they have gotten a speeding ticket while driving a car, and `Texted` tells whether they have texted while driving a car):

```
student_survey <- read.csv('https://sldr.netlify.app/data/IntroStatStudents.csv',
 na.strings = list('', 'NA'))
tally(~Ticket | Texted,
 data = student_survey,
 format = 'proportion')
```

|               | Texted     |            |
|---------------|------------|------------|
| Ticket        | No         | Yes        |
| I don't drive | 0.03703704 | 0.00000000 |
| No            | 0.77777778 | 0.71900826 |
| Yes           | 0.14814815 | 0.28099174 |
| <NA>          | 0.03703704 | 0.00000000 |

What if we want to print just the first column of data?

(Note: Don’t count the row and column names when numbering the rows and columns.)

```
tally(~Ticket | Texted,
 data = student_survey,
 format = 'proportion')[,1]
```

|               |            |            |            |
|---------------|------------|------------|------------|
| I don't drive | No         | Yes        | <NA>       |
| 0.03703704    | 0.77777778 | 0.14814815 | 0.03703704 |

Or better (and clearer...)

```
tally(~Ticket | Texted,
 data = student_survey,
 format = 'proportion')[, "No"]
```

| I don't drive | No         | Yes        | <NA>       |
|---------------|------------|------------|------------|
| 0.03703704    | 0.77777778 | 0.14814815 | 0.03703704 |

What about the third row (for people who have gotten a ticket)?

```
tally(~Ticket | Texted,
 data = student_survey,
 format = 'proportion')["Yes",]
```

| No        | Yes       |
|-----------|-----------|
| 0.1481481 | 0.2809917 |

What about the proportion of students with tickets, among those who've texted while driving? (Row 3, Column 2 = row "Yes" and column "Yes")? Let's first save the table so we don't have to recompute...

```
driver_table <- tally(~Ticket | Texted,
 data = student_survey,
 format = 'proportion')
```

Type: The proportion of students who have texted while driving who have gotten a speeding ticket is `r driver_table["Yes", "Yes"]`.

To get: The proportion of students who have texted while driving who have gotten a speeding ticket is 0.2809917.

*(Like before, if it's possible to use names instead of numeric indices, try to do so!)*

## 5 Math Notation in Quarto

### 5.1 Greek Letters, common symbols, subscripts and superscripts

You might be wondering...

How can I include Greek letters and other symbols in the text part of my Quarto (or RMarkdown) document?

Basically, you enclose the name of the symbol you want with `$...$`

(if you use LaTeX, this will be very familiar):

| Type this in qmd:                              | To get this when rendered:      |
|------------------------------------------------|---------------------------------|
| <code>\$\hat{p}\$</code>                       | $\hat{p}$                       |
| <code>\$\bar{x}\$</code>                       | $\bar{x}$                       |
| <code>\$\alpha\$</code>                        | $\alpha$                        |
| <code>\$\beta\$</code>                         | $\beta$                         |
| <code>\$\gamma\$</code>                        | $\gamma$                        |
| <code>\$\Gamma\$</code>                        | $\Gamma$                        |
| <code>\$\mu\$</code>                           | $\mu$                           |
| <code>\$\sigma\$</code>                        | $\sigma$                        |
| <code>\$\sigma^2\$</code>                      | $\sigma^2$                      |
| <code>\$\rho\$</code>                          | $\rho$                          |
| <code>\$\epsilon\$</code>                      | $\epsilon$                      |
| <code>\$\sim\$</code>                          | $\sim$                          |
| <code>\$\mu_D\$</code>                         | $\mu_D$                         |
| <code>\$\mu_{longsubscript}\$</code>           | $\mu_{longsubscript}$           |
| <code>\$\hat{p}_{longsubscript}\$</code>       | $\hat{p}_{longsubscript}$       |
| <code>\$\mu \neq 0\$</code>                    | $\mu \neq 0$                    |
| <code>\$\mu \geq 5\$</code>                    | $\mu \geq 5$                    |
| <code>\$\mu \leq 1\$</code>                    | $\mu \leq 1$                    |
| <code>\$\cup\$</code>                          | $\cup$                          |
| <code>\$\cap\$</code>                          | $\cap$                          |
| <code>\$\vert\$</code>                         | $\mid$                          |
| <code>\$\sim\$</code>                          | $\sim$                          |
| <code>\$\frac{numerator}{denominator}\$</code> | $\frac{numerator}{denominator}$ |

---



---

Type this in qmd:    To get this when rendered:

---



---

For other Greek letters, just spell out the name of the letter that you want (following the models above). If you want a capital Greek letter, capitalize the first letter of its name when you write it out (e.g. Sigma instead of sigma).

*Note: Avoid spaces before the final \$ or after the initial \$.*

## 5.2 Summations and Products

---



---

Type This:    To get this in your PDF:

---



---

`$$\sum_{i=1}^n x_i$`     $\sum_{i=1}^n x_i$   
`$$\prod_{i=1}^n f(i)$`     $\prod_{i=1}^n f(i)$

---



---

These will format as seen above if used in inline math mode (enclosed in single \$s). If you put them in display math mode by using two \$\$ at the start and end instead of just one, then the result will be displayed centered on its own line and the limits of the summation/product will be above/below the  $\Sigma$  or  $\Pi$ :

$$\prod_{i=1}^n f(i)$$

## 5.3 Long equations

You can use double \$ to bracket equations you want to display on a line of their own. Inside can be multiple mathematical expressions. For example:

```

$$\hat{y} = \beta_0 + \beta_1 x_1, $$

$$\hat{\epsilon} = \hat{y}_i - y_i $$

$$\epsilon \sim N(0, \sigma) $$

```

gives

$$\hat{y} = \beta_0 + \beta_1 x_1$$



$$\hat{\epsilon}_i = \hat{y}_i - y_i$$

$$\epsilon \sim N(0, \sigma)$$

*Note: Avoid spaces before the final \$ or after the initial \$. Also note, the equation will NOT be inside a code chunk...I only did that here because it's hard to get the **un-rendered source version** to appear neatly in the text part of a rendered Quarto file otherwise.*

## **Part II**

# **Designing Effective Visualizations**

## Section Learning Outcomes

After this section, you will be able to:

1. Critique statistical graphics based on design principles.
2. Recognize common misleading design choices for data visualizations
3. Recognize data visualization that tells a true story, identifying elements that emphasize the main finding and make the figure easy to interpret at a glance

## Reference Materials

- [\*Beyond Multiple Linear Regression\* Ch. 1.5](#)
- *Ecological Models & Data in R* Ch. 2 discusses graphics, but is not recommended as the approach to reading in data, writing R code, and generating graphs in R is very different to that used in this course.
- A comprehensive, and free, supplemental reference is [Fundamentals of Data Visualization](#) by Claus Wilke

It's suggested that you refer to the above materials as needed *after* doing this section, with particular focus on the topics you found most challenging.

## Inspiration

Above all, show the data.

E. Tufte, [\*The Visual Display of Quantitative Information\*](#)

But...

The Numbers Don't Speak for Themselves.

C. D'Ignazio and L. Klein, [\*Data Feminism\*](#)

In visualizing data, we use graphics to gain and communicate an honest understanding of data in context.

## Motivation: Imagine First!

Figures are a crucial tool for exploring your data *and* communicating what you learn from the data.

Whether you are doing a quick check to assess basic features of a dataset or creating a key figure for an important presentation, the best practice is to work thoughtfully.

### The I.C.E.E. method:

- **I**magine how you want your graph to look, *before* you
- **C**ode. Once you have the basic starting point,
- **E**valuate your work, and
- **E**laborate (refine it).

Repeat until the figure is as awesome as it needs to be.

## NO To Mindless Copy/Paste

Too many of us fall into the trap of starting to write code (or copy/pasting it!) *before* pausing to think carefully about the desired outcome, then settling for the first vaguely relevant result (or [delighting in the unintended outcome...](#)).

You can do better than mindless copying! Only *mindful* copy-pasting allowed.

This section provides some advice to get you started. It can also provide inspiration for constructive critique of others' graphics.

Here we focus only on the **I\_EE** parts of the process, where you design and assess graphics. Code will come later.

## Appearance Goals

Specifically, how *exactly* should a graphic look? There are so many choices: color, size, text and more. What are best practices for creating something beautiful, that represents the data honestly, and is easy to understand?

This section will provide some rules of thumb to help you **Evaluate** statistical graphics. It will also teach you to spot common problems and suggest ways to fix them, allowing you to provide *constructive* critique (to yourself or to others!) about how to **Elaborate** and refine data visualizations.

## You still have your freedom!

As you digest all these rules and tips, you may wonder: “Do I *have* to always obey every one?” Well...No, of course not. Be creative!

Sometimes it's OK to break these rules *when you have thought it through* and *with a good justification*.

A good justification means that in your particular case, breaking a certain rule will make your graph more informative, easier to understand, or better at telling the story you're highlighting.

## Learning Objectives

This section will give you some basic tools to:

1. **Graph data with integrity**, avoiding misleading design choices
2. **Tell the right story**, including elements that *emphasize your main finding* and make your figure *easy to interpret at a glance*

## 6 What is *Necessary*?

### 6.1 Bye, Junk!

Our first principle is: if it doesn't *need* to be in your graph, it shouldn't be there. Keep things as simple as possible. What are some justifications for a *need* to include an element in a plot?

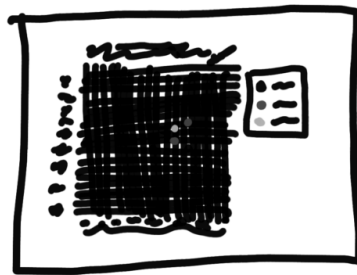
- It is crucial to the story you are telling, or the research question you are answering.
- It emphasizes your main point. For example, some plots may not need color, and in others it may add crucial visual contrast to highlight a main point.
- It makes the graph easier to read and understand
- It makes the main message of the graph more memorable

**If you need it, include it, but if not, keep it simple!**

Imagine you are using *very* expensive ink to print every element of the graph. Is every drop of ink you're using really worth it? If not, take it out. As influential data visualization thinker Edward Tufte put it in *The Visual Display of Quantitative Information*,

A large share of ink on a graphic should present data-information, the ink changing as the data change. Data-ink is the non-erasable core of a graphic...

In other words, don't let annotations, labels, grids, etc. overwhelm the visual impact of your data – Don't do this:



Following Tufte's  
advice, I greatly  
decreased our  
chart's data-ink-ratio.



I think you  
got that  
backwards.  
/

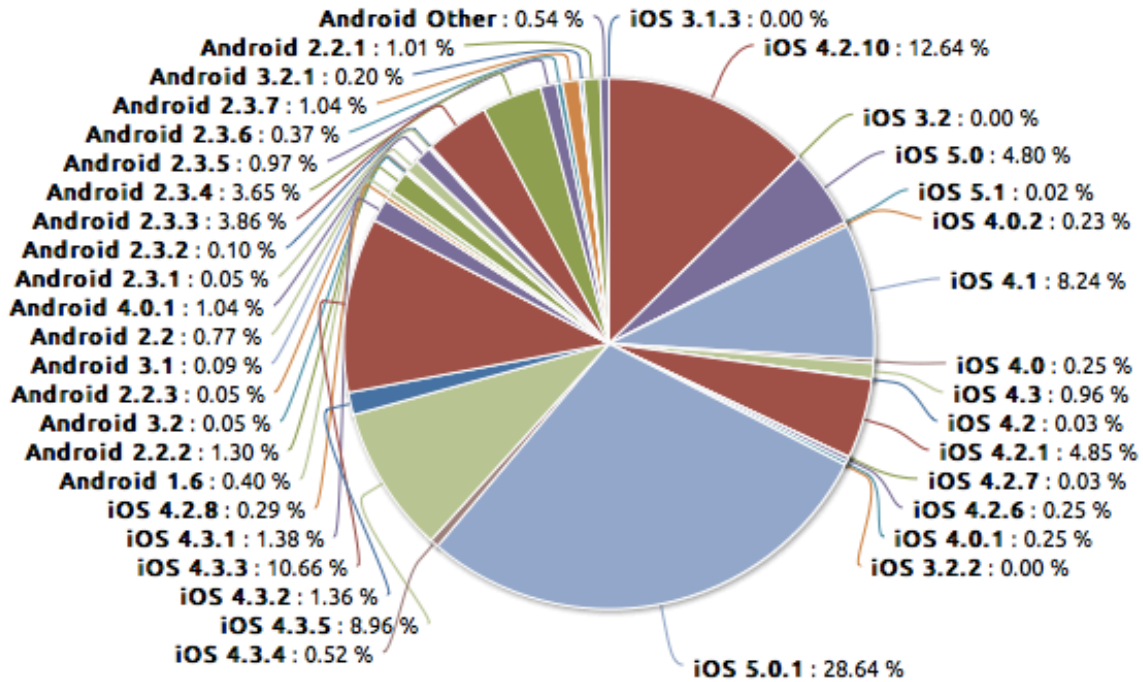
fresh spectrum

Cartoon source: <https://freshspectrum.com/data-ink-ratio/>

### 6.1.1 Check: Critique the Pie

The figure below, from [a Forbes article on mobile operating system crashes](#), is pretty awful.

### Crashes by OS Version Normalized (12/1 - 12/15)



## A Conclusion?

What is **one main conclusion** from the graph above?

(It's pretty confusing to interpret, so you may have to study carefully to find something...)

## Remove...What?

Now that you have identified one main conclusion from the graph...

What is one element of the plot that:

- obscures that conclusion,
- is NOT necessary, and
- could be removed to improve the plot?

Answer constructively - as if the person who made the plot was incredibly smart and someone you admire, and to whom you wanted to be kind but helpful.



## 6.2 Grids and Boxes

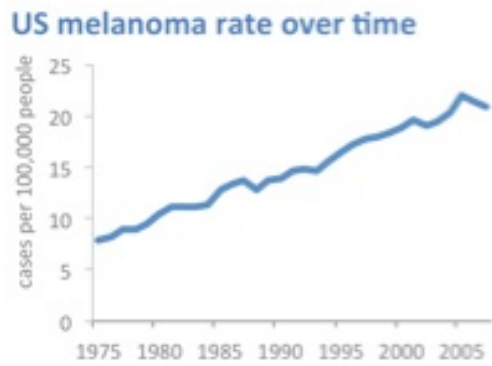
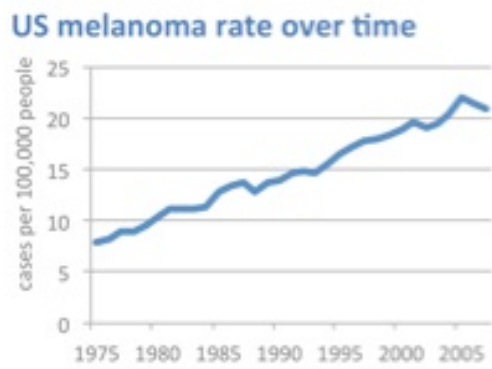
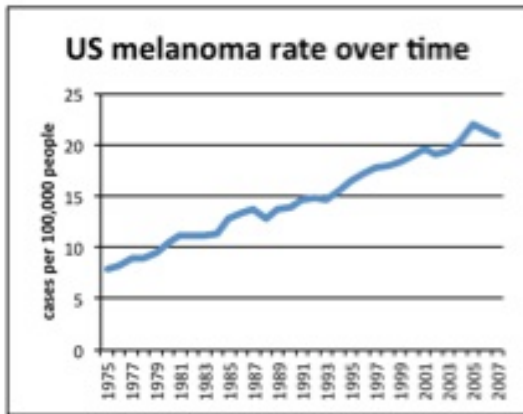
Should your graphics include boxes, axis lines, and grid lines?

Well, it depends...

- Removing unnecessary axes, grids, and labels yields a cleaner plot that may be easier to take in at a glance – there is less to distract from the main story
- **But...** omitting needed baselines, tick marks, gridlines, and labels can cause confusion and make it hard to identify categories or estimate numeric values
- Scientific graphics usually need axis lines, with tick marks
- If a viewer will need to refer to an axis to estimate heights of bars or locations of points, then consider using gridlines for that axis.
- Instead of an entire grid, it may be more effective to include single lines indicating important threshold values
- Consider using a color that nearly blends into the background for grid lines, so that they detract as little attention as possible from the data

### 6.2.1 Example

[Storytelling with Data](#) provide an example of a cluttered figure where the trend over time pops out more as unnecessary grids and boxes are made less visible, then removed:



Optionally, if you'd like more examples, read [S. Few's 3.5-page article on when grid lines are helpful](#).

# 7 Color

## 7.1 Using Color

Color - used with care - can be an incredibly effective part of a data visualization.

- Ensure your color choices **highlight the story** you want to tell
  - Consider using black and grey to help some elements fade into the background - for example, grid lines and labels that must be present but aren't the most important elements.
  - Or, color all groups but the one you want to highlight in grey, and use a bold color for the “main” group...
- Choose color combinations that look good and are **distinguishable by color-blind viewers and in greyscale**
  - Defaulting to pre-defined color palettes provided by your software may be better than haphazardly choosing colors manually
  - Consider being redundant - use size and/or shape as well as color to indicate groups so figures are legible in greyscale, too.
- Use color **consistently**.
- Example: if “young” cases are red in one graph, don't use red for “old” in the next graph. And if many graphs are colored by the same grouping variable, use the same colors in all of them.

The video below, created by [Storytelling with Data](#), gives explanations and examples.

- If you have time, watch from 11:48 to 28:41 (about 17 minutes). This segment will play automatically in the clip below.
- If you're in a rush, the most important sections (about 10 minutes) are:
  - 13:57 - 15:12 (Sparing use of color)
  - 18:44 - 21:25; see also the [infographic of color in culture](#)
  - 22:25 - 23:10 (Color blindness - to view your graphs as someone with color blindness would, take a screen shot and try the [simulator online](#))
  - 23:50 - 28:41 (Consistency)

**Which of the following are lessons from the Storytelling with Data video on Being Clever with Color? Mark all correct answers “TRUE”.**

Color grabs attention. TRUE / FALSE

Color signals where to look. TRUE / FALSE

Color should be used sparingly. TRUE / FALSE

Too much color, and everything is highlighted - the viewer does not know what to pay attention to. TRUE / FALSE

Color can show quantitative values, too, not just categories. TRUE / FALSE

Colors have tone and meaning. TRUE / FALSE

Not everyone can see colors. TRUE / FALSE

Use color consistently. TRUE / FALSE

Simple black and white is always the best choice. TRUE / FALSE

Click for explanations of solutions above.

- The human eye is naturally drawn to colors.
- Since color grabs attention, we expect it to direct us toward the most important stuff that is worthy of our attention.
- But...Too much color, and everything is highlighted - the viewer does not know what to pay attention to.
- Also, remember that the meaning and interpretation of colors varies by culture.
- Since some people can not see color, use color-blind friendly palettes and redundant coding (shape, text) where possible without cluttering the figure.
- Inconsistent use of color can be confusing and distracting.
- Sometimes black and white is great - but often color helps you tell a story!

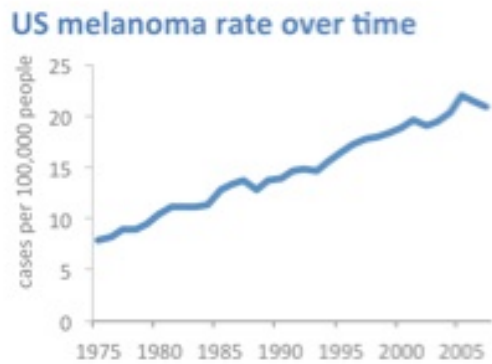
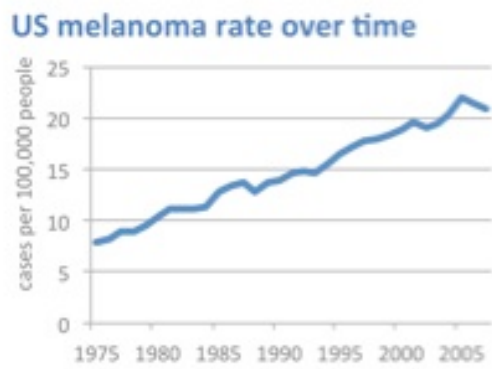
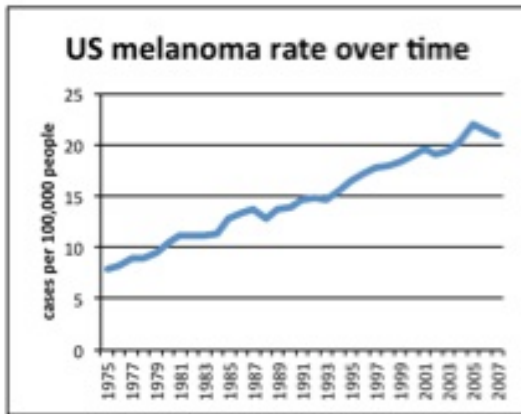
## 8 Text Elements

### 8.1 Titles, Labels, Size

When using text in a figure, ensure it is easy to read. Make sure no unnecessary text is included.

- Default size of text in figures produced by statistical software *is almost always too small*. Make sure your text is *big enough to be easily legible in the context where you will present it* (on the page in a report, on a slide for a presentation, etc.)
- Other than the title of the vertical (y) axis, all the text in a plot should be horizontal. This makes it easier to read.
- Axis labels should be self-explanatory
  - Viewers should be able to guess what they mean accurately without looking at anything but the figure
  - Use words instead of numeric codes or cryptic abbreviations
- Axis labels should also be as short as possible while remaining easy to understand
- Every plot should have a title. Sometimes this might be a literal title at the top of the graph, but those are relatively rare. More often in scientific work, a text caption appears below the figure. The first phrase/sentence of the caption acts as the figure's title

Remember the melanoma rates over time figure we saw earlier?



What helpful changes did Storytelling with Data make to the text labels as they improved the figure?

The x axis labels are rotated so they are horizontal. TRUE / FALSE

The title color is changed to blue and the axis labels to grey. TRUE / FALSE

The box around the plot is removed. TRUE / FALSE

Click for explanations of solutions above.

- Rotating axis labels so they are horizontal is generally an improvement. To make this happen, the number of tick marks and labels on the x axis was also reduced. Notice the labels are much easier to read.
- The color changes helped too. The blue links the title with the trend it describes, and the grey makes the axis titles less prominent and lets the viewer focus on the data. Continue to the next section for more on using color...
- The box is gone, and it is a big improvement to the plot! But technically, you were asked about changes to the text labels...

## 8.2 When Things Overlap

Especially when graphing variables with long category values, you may end up with ugly, illegible overlapping labels.

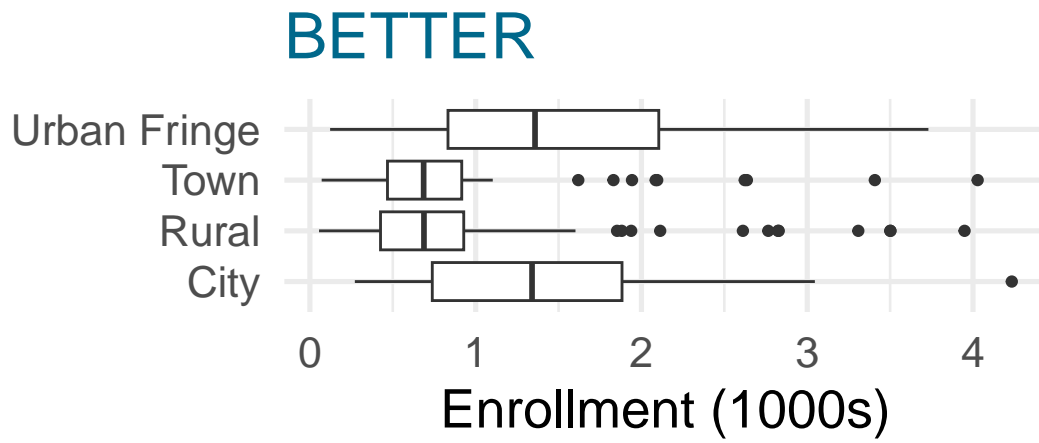
Some solutions, in rough order of preference, are to:

- adjust the figure width or height so everything fits
- switch x and y coordinates so the “long” labels are on the y axis (in R, this resizes the plot area so that labels fit); or,
- rotate the too-long labels, which eliminates the overlap *but makes them harder to read than horizontal text*.
- make the font smaller (*but this might make it annoyingly hard to read, or make your viz feel cluttered!*)

## 8.3 When Text Runs “off the edge”

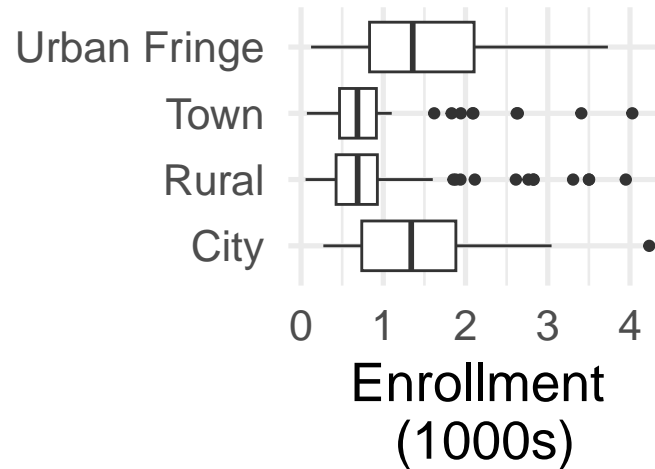
Sometimes a title or axis label is too long and runs off the edge of the figure. Using a smaller font is *not* often an ideal solution. If you can’t just use a shorter label, consider adding line breaks.

## 8.4 Examples





ALSO  
BETTER



## 8.5 Let's Talk About Titles

Should your graph have a title?

Well...*maybe*.

Hear me out.

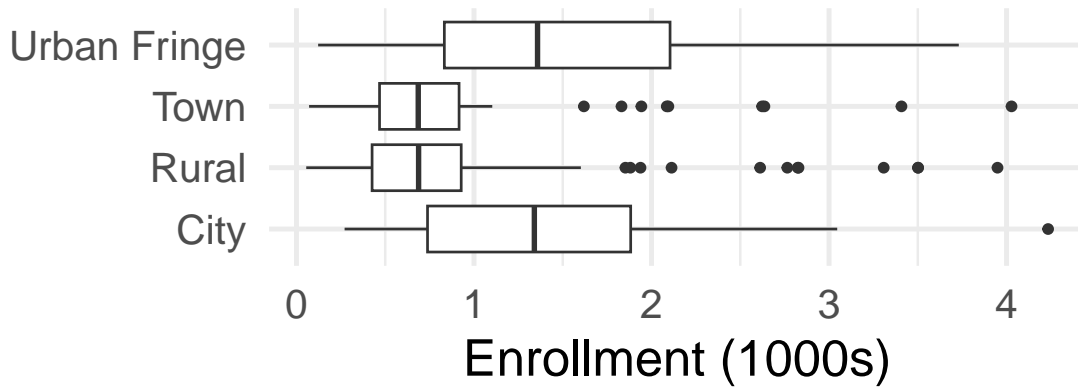
Whether or not titles are useful, allowed, or expected is pretty discipline- and audience-specific. In journalism and some parts of business, they are used very often. In the peer-reviewed scientific literature, they are exceedingly rare. In a slide deck, they are often useful...unless they say the same thing as your slide title! Know your goals and your audience.

### 8.5.1 Avoid Redundant Titles

If you do want to use a title, make *sure* it provides information or details that are *not* already present in other text elements. A title that restates the axis labels is usually a waste:

# Enrollment by Location

**AVOID titles like the one above that repeat axis labels!**



Notice that we can argue about this a little. Could we have the title *instead* of the axis labels? Maybe, but if it's a scientific publication, having the units of measure is going to be important.

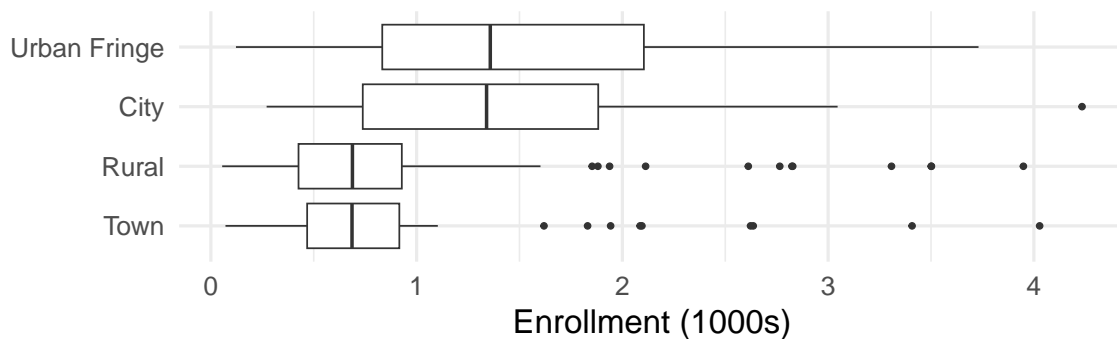
Is this title *actually adding info* because it clarifies that “Town,” “City,” etc. are “Locations”? I think it's a stretch, but sometimes you can make such a case.

Generally, my advice is...

If the title is not adding something *new and crucial* and helping the reader decipher the main *story* of the plot, then omit it!

So what might an “informative” title look like?

## Small Schools Dominate in More-Rural Areas



## 9 Summary

After reviewing the preceding sections, you should be able to articulate some principles for designing good visualizations with respect to use of space, axis limits, use of color, and text elements.

In fact, you probably already *knew* these principles - at least, you knew 'em when you saw 'em (you could have easily sorted some terrible and better graphs even if you couldn't have said *exactly* what was terrible, or better, about them).

Now, think about how you could apply these principles in your own work, or in providing feedback or advice to others...

### 9.1 Critique Practice

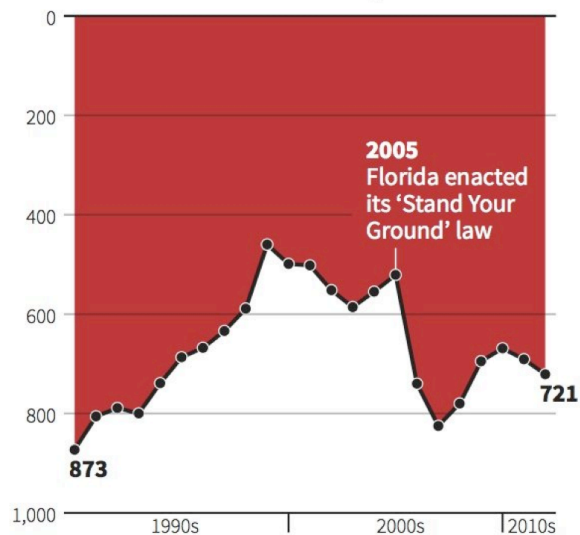
Try using what you have learned to provide a *constructive* critique of an example. That might mean pointing out specific successes or positives as well as areas for improvement, with concrete advice about how to improve and why.

Consider the graphic below. At a glance, what do you think it means? Looking more carefully, what do you notice?

---

## Gun deaths in Florida

Number of murders committed using firearms



Source: Florida Department of Law Enforcement

C. Chan 16/02/2014

REUTERS

**Pause to think:** What changes, if any, would you suggest to the figure's creator to make it clearer and easier to understand? Be sure to be constructive - gently explain any problems and suggest solutions.

## 9.2 Video Review

Wow, that was quite a lot of information! If you could use a brief review from a different point of view, check out the [optional video from Kristen Sosulski](#)

## 9.3 Recap & Reflect: 12 Tips

The 4-minute video below summarizes design principles for data visualization in the form of 12 tips.

As you watch, make note of one or two tips that strike you (you'll report your thoughts in the next section). Is there one that nicely summarizes an idea introduced earlier in the section? One you're not sure about? One that you think is incredibly important? One that makes you say "Aha! *Now* I see why I loved/hated that visualization!"

### 9.3.1 Pause for Reflection

Take a moment to reflect on what you learned. Which Tip do you remember most clearly, think is most important, or want to challenge? Consider making a few notes for yourself for the future (you'll have to make and critique plenty of graphics in your homework assignments).

## 9.4 A Critique Checklist

If working to improve your own visualization or attempting to give feedback on one another analyst made, you might consider using a checklist to guide you.

Based heavily on the advice and template provided by [Evergreen Data...](#)

Check out our [Graphics Critique Checklist!](#)

## **Part III**

# **Choose a Graph Type**

## Section Learning Outcomes

After this tutorial, you will:

1. Distinguish variable types: quantitative, categorical (nominal, ordinal, interval, ratio); explanatory, response, covariate.
2. Choose an appropriate graphical display for a specified combination of variables.
3. (Continue to) critique statistical graphics based on design principles.

**Note:** You do NOT have to memorize all the information in this tutorial. Review it now, but know you will probably return to this tutorial for later reference. Your goal should be to finish with a basic idea of which graph types should be used for which variable types. Notice that the “Gallery” sections in the navigation bar are labeled by which variable types are to be shown!

At the end, you might want to finish with your own notes filling in a table like the one below:

|                  | Variables | Graphs                       |
|------------------|-----------|------------------------------|
| One Quantitative |           | histogram, density plot, ... |
| One Categorical  |           | ...                          |
| ...              |           | ...                          |

## Text Reference

- [Beyond Multiple Linear Regression Ch. 1.5](#)
- *Ecological Models & Data in R* Ch. 2 discusses graphics, but is not recommended as the approach to reading in data, writing R code, and generating graphs in R is very different to that used in this course.
- A comprehensive, and free, supplemental reference is [Fundamentals of Data Visualization by Claus Wilke](#)

## Motivation: Imagine First!

Figures are a crucial tool for exploring your data *and* communicating what you learn from the data.

Whether you are doing a quick check to assess basic features of a dataset or creating a key figure for an important presentation, the best practice is to work thoughtfully. You already learned about creating graphics by I.C.E.E:

## The I.C.E.E. method:

- *Imagine* how you want your graph to look, *before* you
- *Code*. Once you have the basic starting point,
- *Evaluate* your work, and
- *Elaborate* (refine it).

Repeat until the figure is as awesome as it needs to be.

## Limiting Your Imagination

There is really no limit to the creative data visualizations you might dream up.

But there is a set of basic, workhorse graphics that statisticians and data scientists use most frequently. What are the common options and how do you choose among them?

The best choice depends on what kind of data you have, and also on what you want to do with it: what question are you trying to answer? What story will you tell?

## Goals

Specifically, you will now focus on **choosing the right type of visualization for the task at hand**.

Note that the graphs shown in this tutorial are over-simplified versions - icons, really - with missing labels, huge titles, and huge data elements. This is intentional, to evoke the look of each plot type rather than to present actual data.



# 10 Variable Types

Before designing a graphic, you need some data. Ideally, it will be in a tidy table, with one row per case and one column per variable.

Different plots may be appropriate, depending on whether the variable is:

- *Categorical* (either nominal or ordinal) or
- *Quantitative* (interval or ratio)
- Beware categorical variables that are stored using numeric codes: they are still categorical!
- *Note:* Variables that take on discrete numeric values can be treated as either, depending mainly on whether there are a lot of possible values (treat like numeric) or few (treat like categorical)
- Other courses or disciplines may distinguish carefully between ordinal and nominal data. We often won't, since we don't learn distinct methods for them, but treat both as *categorical*.

The video below gives a concise explanation of the different variable types you need to be able to recognize.

[https://youtu.be/eghn\\_\\_C7JLQ?si=Oozu0Z6p4xTBh64z](https://youtu.be/eghn__C7JLQ?si=Oozu0Z6p4xTBh64z)

## 10.1 Distributions

Sometimes, you need a plot that lets you *see* the **distribution** of a single variable:

- What *values* does it take on?
- How *often* does each value occur?

Sometimes these graphs present the answer to a scientific question of interest, but often they are used during exploration or model assessment to better understand a dataset and:

- Check the data
  - Are there lots of missing values?
  - Are missing values encoded as 999 or -1000 or some other impossible value, instead of being marked as missing via NA?

- Verify whether the variable's distribution matches expectations (for example, symmetry, etc.)

# 11 Gallery: One Categorical Variable

## 11.1 Consider your Audience

To show one categorical variable, we will mainly use bar charts. You might also consider lollipop/Cleveland dot plots, or pie graphs.

| Graph         | Show Proportions? | Show Counts? | Statisticians |
|---------------|-------------------|--------------|---------------|
| Bar           | ✓                 | ✓            | 😬             |
| Lollipop      | ✓                 | ✓            | 😎             |
| Cleveland Dot | ✓                 | ✓            | 😎             |
| Pie           | ✓                 | ✗            | 😬             |

## 11.2 Bar Graph

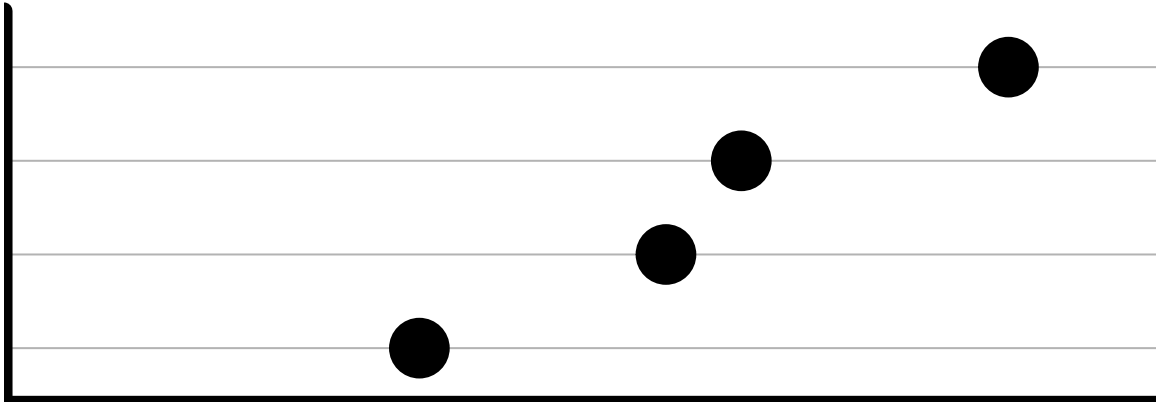
### BAR



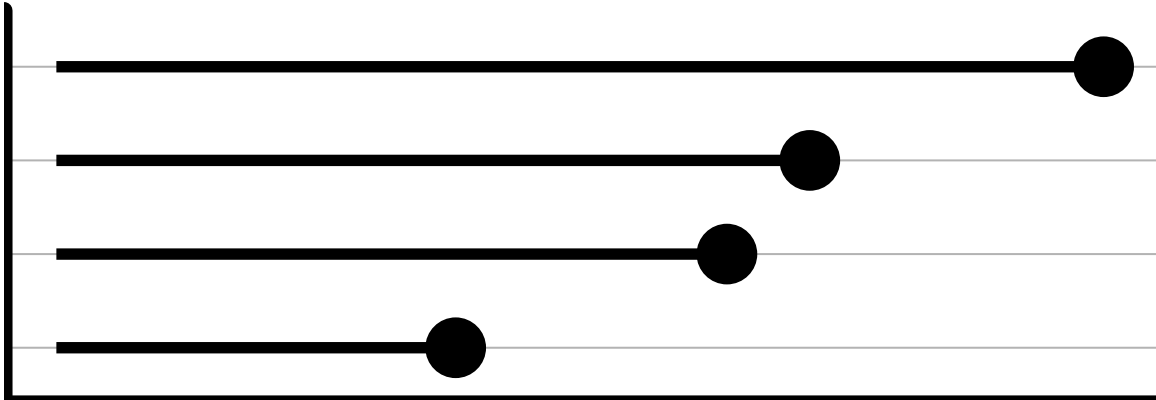
- Can show either counts, proportions, or percentages
- Easy to see which categories have more/fewer observations

### 11.3 Cleveland Dot / Lollipop Plot

#### CLEVELAND DOT



#### LOLLIPOP



- Main difference is whether the “sticks” are drawn (Lollipop) or not (Cleveland Dot)
- Much like a bar chart, but using dots or lollipops to mark the count or proportion in each category
- Work well when there are many categories to be ranked by frequency

## 11.4 Pie Chart

# PIE



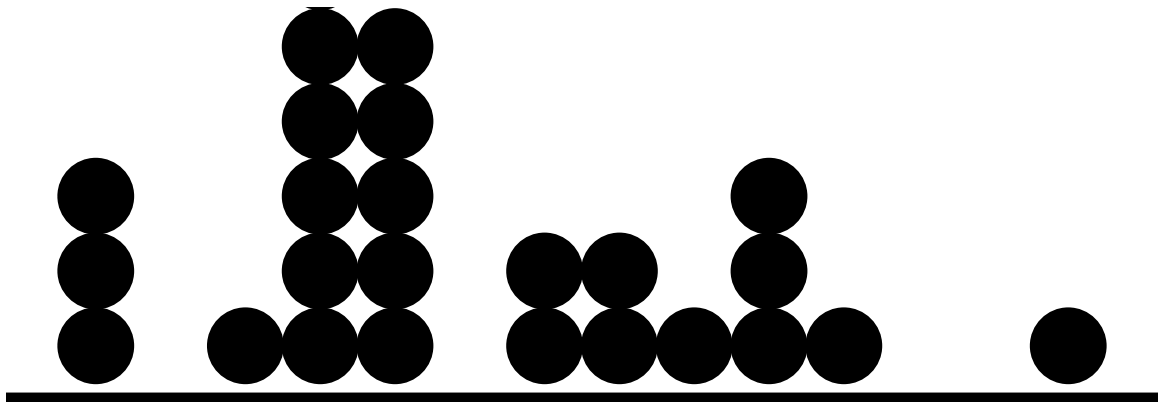
- Display proportions, not counts
- Unpopular with many statisticians and data scientists because...
  - Hard to see which categories have more/fewer observation when proportions similar
  - Temptation to clutter them up with annotation (for example, percentage for each slice)
  - Can be inefficient use of space on rectangular page
- Often easier to interpret when there are few categories

## 12 Gallery: One Quantitative Variable

What are some ways to display the distribution of one quantitative variable?

### 12.1 Dotplot

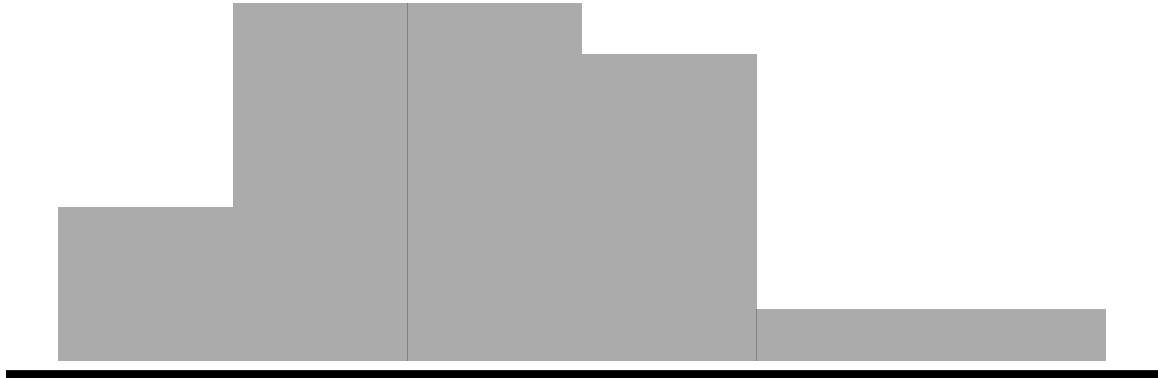
#### DOTPLOT



- Intuitive representation: x-axis shows range of variable values, and dots are data points
- As the idea is to show one dot per observation, may not work well for huge datasets

## 12.2 Histogram

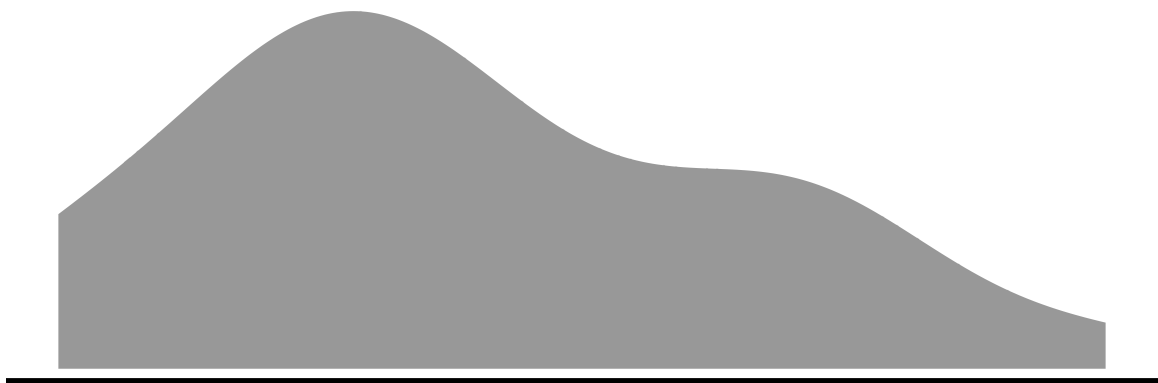
# HISTOGRAM



- Range of variable values is divided into bins, then height of each bar corresponds to the number of observations in the bin
- Effective way to examine the *shape* of a distribution
- Choosing the number of bins to use is tricky: too many, and the shape is jagged; too few over-smooths (peaks blend together). Not sure? Find a number of bins that is definitely too few, and one that is definitely too many, and then try to settle on an in-between value that best shows the real shape of the distribution without over-smoothing.

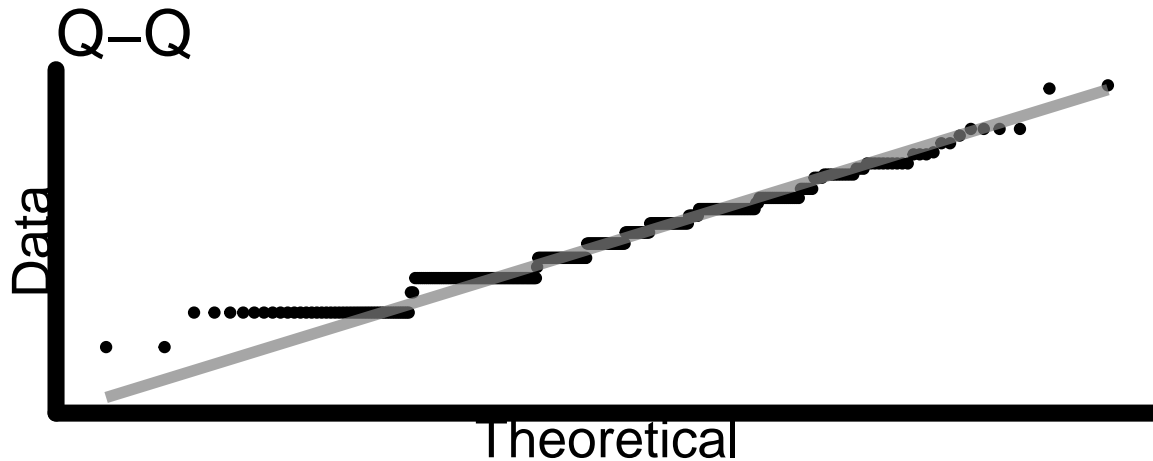
## 12.3 Density Plot

# DENSITY



- Like a smoothed version of a histogram (obtained by kernel density estimation, if you want to look up mathematical details)
- Caution: for small datasets, the density plot may show “peaks” that really correspond to one or a few observations
- Can only show *density* (relative frequency of observation), *not counts*

## 12.4 QQ Plot



- “Q-Q Plot” is short for “Quantile-Quantile Plot”
- In some cases, we want to examine the shape of a variable’s distribution *to see if it matches a theoretical expectation*. For example: do the regression residuals match a normal distribution? (If that example doesn’t make sense to you now - it will later in the course, don’t worry.)
- Quantile-quantile plots are one way to make this comparison. They plot the quantiles of the data as a function of the same quantiles of the expected theoretical distribution; if there’s a good match, the points should follow a line with slope = 1.
- How close to the straight line is “close enough”? That’s the tricky part...

## 12.5 Check Your Understanding: One-variable plots

Which plot would work best to show the distribution of 75 families’ household incomes?

- (A) Lollipop plot
- (B) Histogram



- (C) Bar chart

Which plot would work best to show the distribution of 75 families' postal codes?

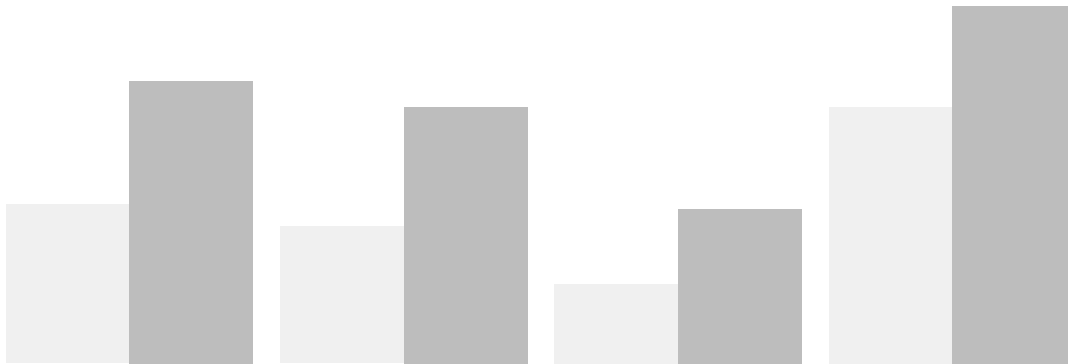
- (A) Bar chart
- (B) Density plot
- (C) Histogram
- (D) Scatter plot

Click for explanations of solutions above.

Lollipop plots and bar graphs work better for categorical variables – they show counts or proportions (or some other summary of counts in categories). By default, there would be one lollipop or bar for each unique value of income - what a mess! Histograms and density plots, on the other hand, show the distribution of one quantitative variable. (Scatter plots are usually used to show 2 quantitative variables.)

## 13 Gallery: 2-3 Categorical Variables

### SIDE-BY-SIDE BAR



- One set of bars per “category”, colored by “groups” – shows *two* categorical variables at once
- Good for showing *counts* in each combination of categories/groups
- It is hard to compare proportion in each group across categories, if the total number in each category differs.

## 13.1 Stacked Bar Graph

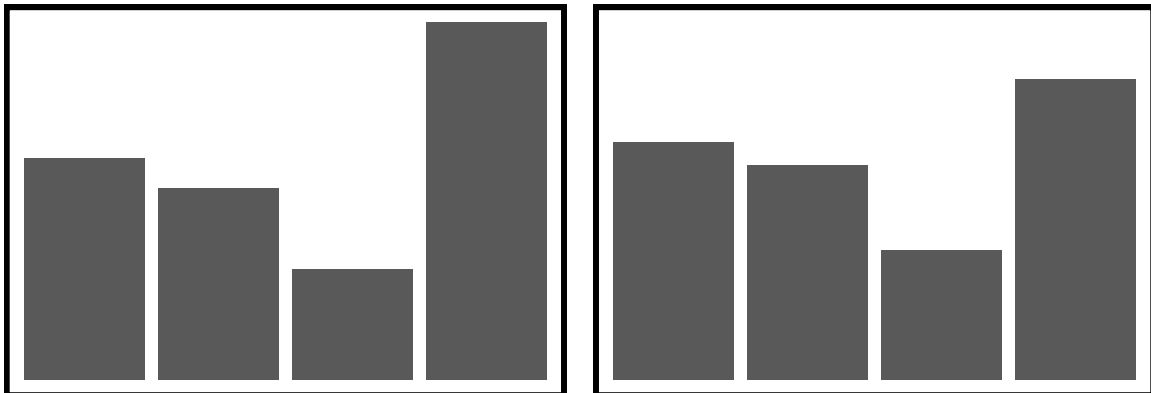
### STACKED BAR



- Similar to side-by-side bar
- Compared to side-by-side, it's *harder* to compare proportions in each group within a category, but *easier* to estimate the proportion in each category.

## 13.2 Faceted Bar Graph

### FACETED BAR



- One plot box – usually called a “panel” or “facet” – for each of a set of groups
- Think carefully about the question of interest and the relationship you want to highlight as you choose: should bar heights correspond to...
  - Number of observations?

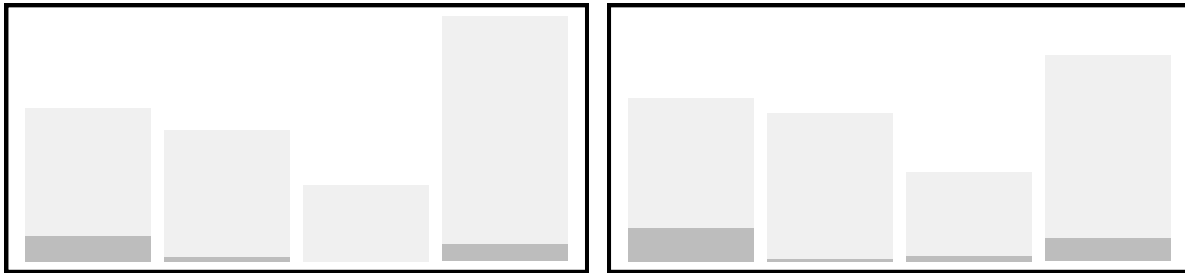
- Proportion of observations *overall in the whole dataset*?
- Proportion of observations *in the panel*?
- Something else?

### 13.3 Combinations (Stacked bars + Facets, etc.)

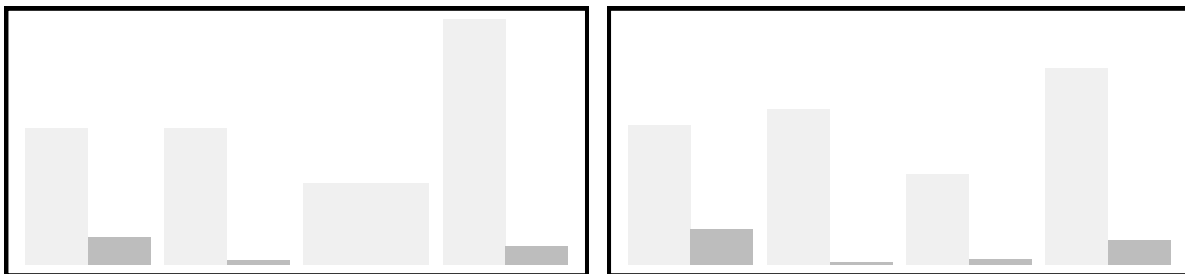
Of course, if you have 3 variables instead of just two, you *can* combine methods. Avoid it unless you are sure it is necessary and communicates clearly.

- **Be sure that the resulting graph is not too complex to understand quickly, at a glance.** Packing too much information into one graph sometimes means *none* of the info is actually communicated!
- And if showing proportions or percentages in such a display, **be sure you understand what denominator is being used in the calculations** – is it the fraction of the whole dataset, within facets, etc.?

#### STACKED BAR + FACETS

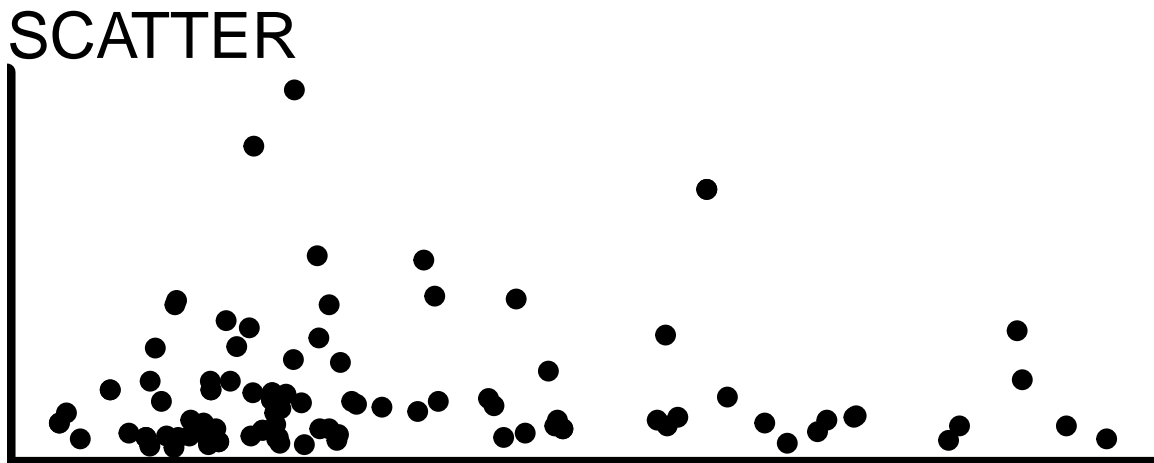


#### SIDE-BY-SIDE + FACETS



## 14 Gallery: Multiple Quantitative Variables

### 14.1 Scatter Plot



- A scatterplot is the default for visualizing the relationship between two quantitative variables
- Be sure you actually have *two quantitative variables*! If not, another plot may be a better option.

Let's just reiterate: if one of your variables is *actually* or *effectively* categorical, a basic scatterplot is usually not ideal!

## 14.2 Line Plot



- If the x-axis variable is **Time** (or it otherwise makes sense to join the dots), a line can replace the dots, or be added to them
- Make sure connecting the dots makes sense in context and does not guide the eye to incorrect interpretations (for example, emphasizing outliers)

## 14.3 >2 Quantitative Variables

What if you have three or four quantitative variables whose relationships you're curious about?

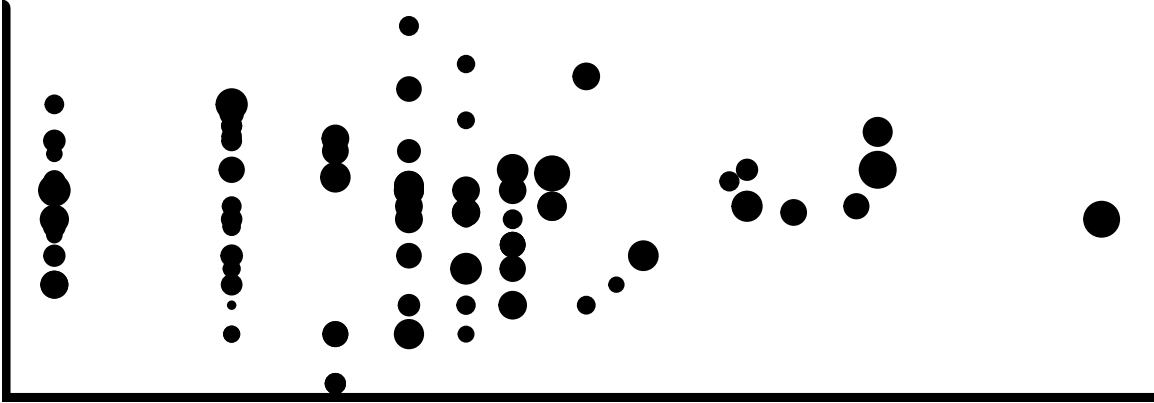
*Proceed with caution!*

It's possible to include 3+ variables on one plot, but ideally **it should still be interpretable at a glance:**

- What is the main point of the figure? Is it possible to make the point without showing all 3+ variables together?
- Keep things as simple as you can while still telling the story you want to tell.

### 14.3.1 Scatter + Size

## SCATTER + SIZE



- You can adjust the size of each dot in a scatter plot to correspond to the value of a third variable
- This is especially useful when the third variable measures the size of the population being represented – for example, a scatter plot of life expectancy vs income for many countries, with point size indicating population of each country

### 14.3.2 Scatter + Color

You can also color by a third quantitative variable:

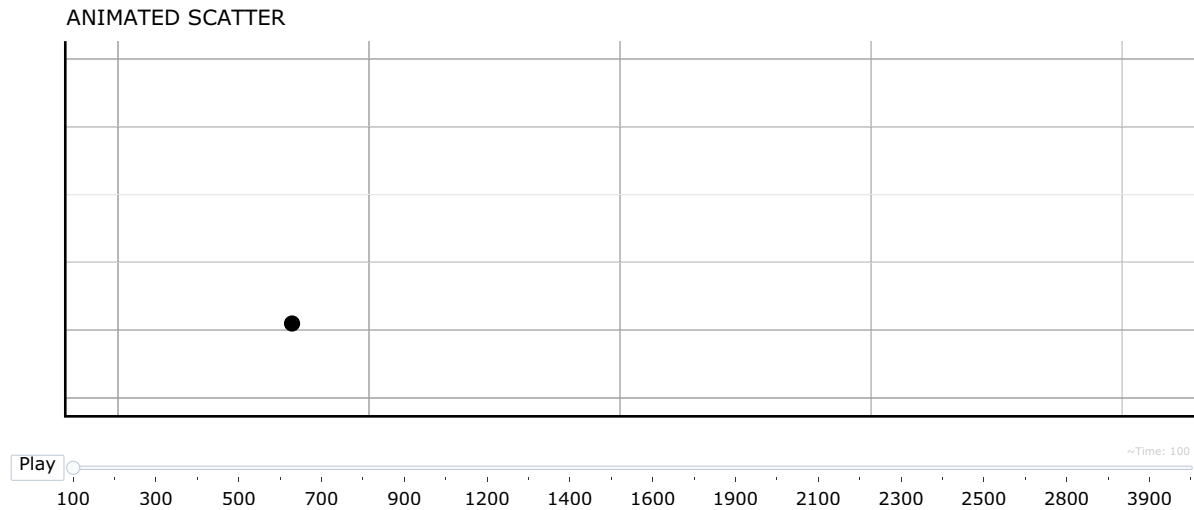
## SCATTER + COLOR



This usually only works well visually if all three variables are clearly associate with each other, so that certain colors are clearly dominant in certain regions of the graph. Otherwise, you get a mishmash of colors all over, which can be distracting.

### 14.3.3 Animation

It may be possible to show a third quantitative variable via animation (this often works especially well if that third variable is actually time!)





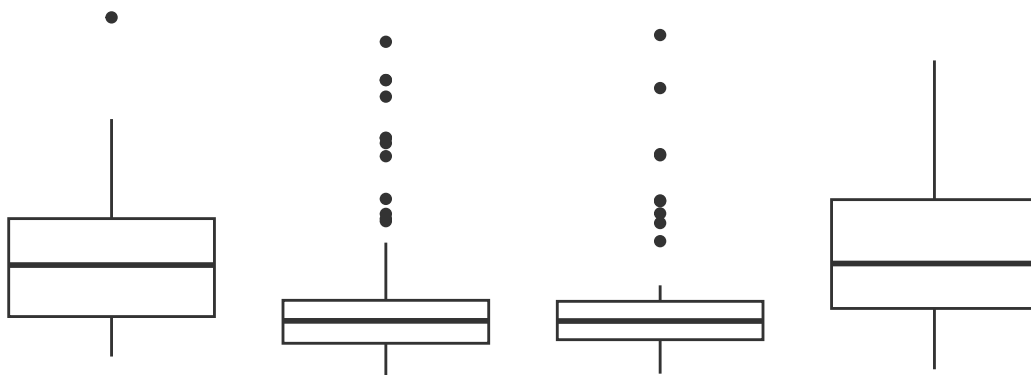
# 15 Gallery: Mixed Quantitative + Categorical

## 15.1 Distribution by Groups

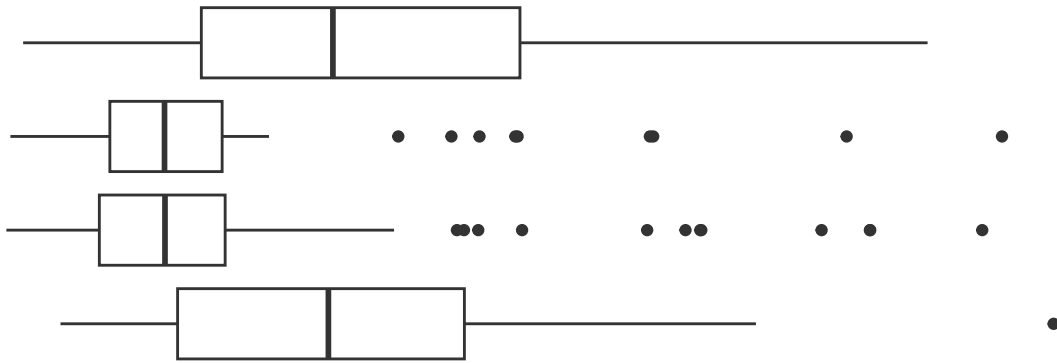
There are several plots designed specifically to look at the distribution of a quantitative variable, grouped by a categorical variable.

### 15.1.1 Boxplots

## BOXPLOTS



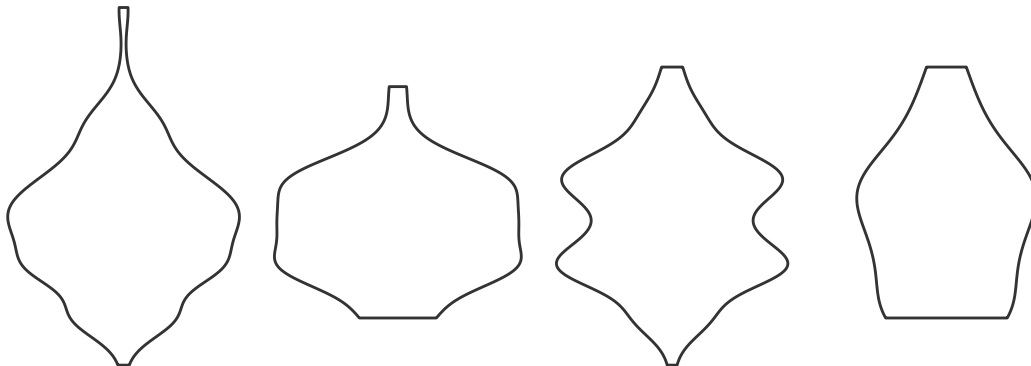
# BOXPLOTS (HORIZONTAL)



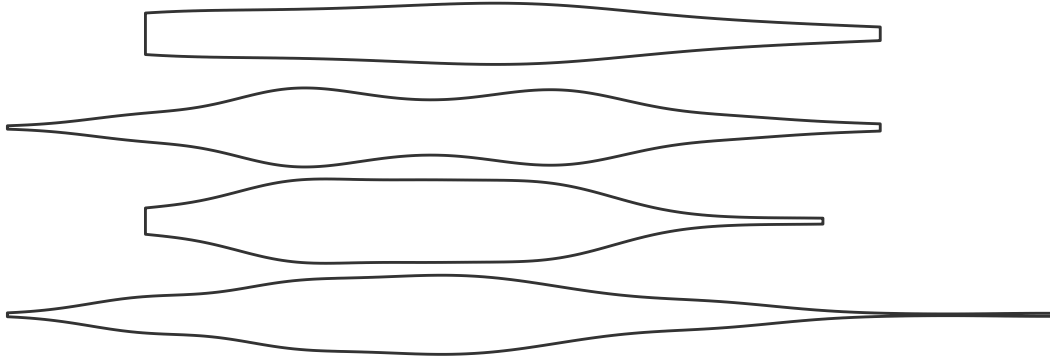
- The boxplot shows a *summary* of the distribution. The box spans the middle half of the data, with the line marking the median. The “whisker” lines extend to cover the range of “most of” the data, with outliers shown individually
- For details, check out [this optional explanation of how boxplots are constructed](#) from [Introduction to Modern Statistics](#) by Mine Çetinkaya-Rundel and Johanna Hardin.
- If your dataset is too small to estimate the median and quartiles of the data accurately, consider showing all the observations (for example, using or overlaying a jitter plot)

## 15.1.2 Violin Plots

# VIOLIN PLOTS



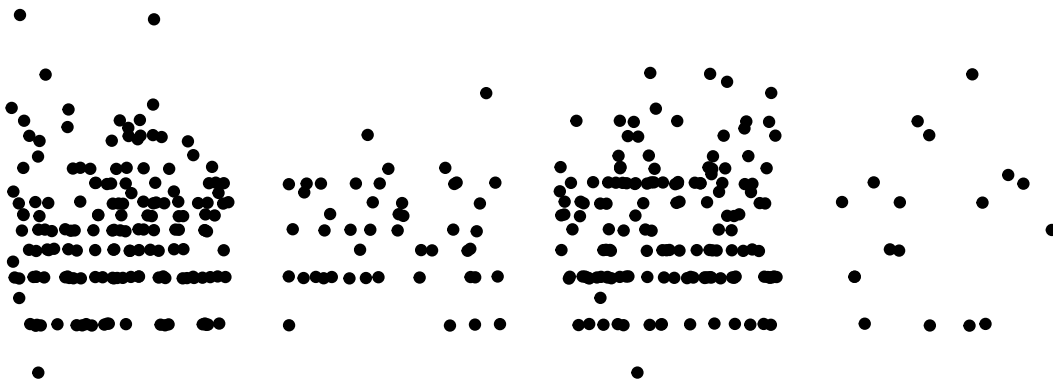
# VIOLIN (HORIZONTAL)



- These show a mirrored density plot for each group
- As for density plots, make sure you have a large enough dataset so that the bumps in the density curve don't represent just one or a couple of observations

## 15.1.3 Jitter Plots

# JITTER PLOTS



- These show all the points in every category, “jittered” (moved slightly away from the category axis) to reduce overplotting
- If the dataset is too large, overplotting may still be a big problem
- Jitter plots are often used as an additional layer on top of boxplots or violin plots to make the size of the dataset, and the locations of individual datapoints, more visible

#### 15.1.4 Sina Plots

## SINA PLOTS



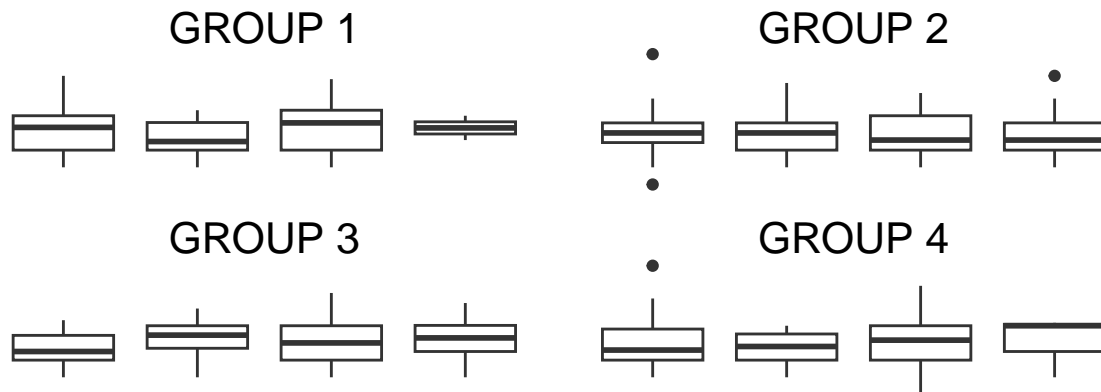
- These show all the points in every category, arranged so that the width of the point cloud corresponds to the density of observations
- If the dataset is too large, overplotting may become an issue
- A sina plot is a bit of a hybrid between a violin plot and a jitter plot; or, a more organized, less random version of a jitter plot.

## 15.2 Facets?

You can also consider making multi-panel plots with one histogram, density plot, or dotplot per facet, but comparing between facets is usually harder than comparing boxplots or violin plots on a single axis.

Multi-facet plots can show one panel per group, for *any* kind of plot seen so far: a bar graph for each group, a stacked bar for each group, a scatterplot for each group, a set of boxplots for each group, etc. etc.

# FACETED BOXPLOTS



## 15.2.1 Check Your Understanding: Quant. + Cat.

There are some errors and inconsistencies in the chart below!

Check it out – can you find them?

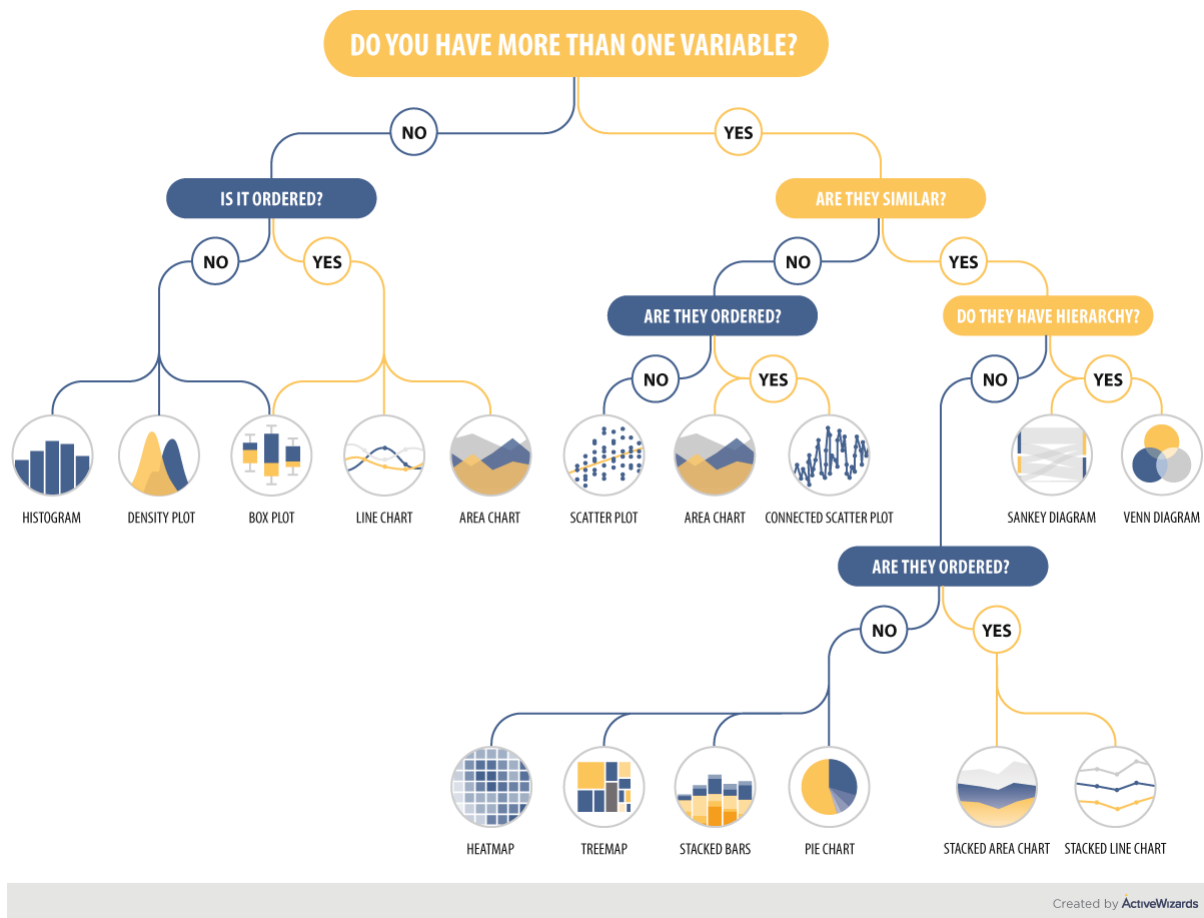


Figure 15.1: chart choice infographic