Day 2, Lecture Two: Data Management and Visualization

Andie Creel

January, 2023

1 Goal

The goal of this lecture is:

1) Learn how to set up an R Project
2) Learn best practices for file structures
3) Importing data
4) Exporting results
5) Basic data visualization with ggplot

2 Creating an R Project

2.1 What's an R Project?

An R project contains multiple scripts, your data, results you export, and any other files you have associated with that project.

R projects are an extremely useful organizational tool, and you should use them for every research project.

They act as an address for your computer to recognize that everything in that project stays together.

2.2 New R Project

1) Open R Studio
2) File/New Project… (alternatively, little clear box with R button)
3) Navigate to the file you want the r project to exist in. **Make sure you can find this again so that we can drag folders into it.**
4) Give it a short, informative name
5) Done! Notice you're top right hand corner.. you're R Studio session has loaded your R Project

Do this as a group together.

3 File Structure

- File structure can make or break a research project
  - Acts as an outline that gives you forward momentum
  - Allows your collaborators to clearly follow what you did

  - Allows you to return to a project after and quickly remember what step you were on and what remains to be done
- There are best practices for file structure that are worth following now

3.1 Best practices

Matt Wibbenmeyer (fellow at RFF) wrote a Notes on Project Management and Collaboration for RFF RAs and interns that I still use. It is worth reading and you can find it here. I share it with all my RAs and collaborators.

There will be your `project/` folder. This is the file your R Project files lives. Everything else will be in this folder or sub folders. I typically have the following sub folders.

- `data/`
  - `raw_data/`: The original data. Do not edit this! You never know when what you originally thought was a good idea was actually bad. Always keep a copy of your original data here.
  - `clean_data/`: Where you store the clean data sets that you'll use for your models
- `scripts/`: all of the r scripts you write
  - `1_data_cleaning.R`
  - `2_descriptives.R`
  - `3_analysis.R`
  - `4_figures.R`
- `results/`: where you store final graphs and tables
- `manuscript/`: where you store your writing that (figures crossed) becomes your paper
- `presentations/`: Any presentations you prepare for the research project

In our new r project, let's create a clean and raw ***`data`****,* ***`script`****, and* ***`results`*** *sub folder. You can do this with the file-green-plus button in the files tap (bottom right corner).*

4 Importing Data

You can import a lot of different data sets in a lot of different ways. The most common data sets I import are CSVs.

4.1 Note on Excel Files

I try to avoid going back and forth between excel and R. Excel is useful for data entry (especially when you're collecting your data in the field). But once you're done with data collection, I would avoid going between R and Excel.

My advice would be to export your excel sheet to a csv. CSVs are more memory efficient and easier to read into R. However, if you edit the Excel file remember you'll need to re-export your csv.

Code for if you do need to read in an excel file instead of a csv.

```
# ----------------------------------------------------------------------
# Install (only once) and load the readxl package
# ----------------------------------------------------------------------
# install.packages("readxl")
```

```r
library(readxl)

# --------------------------------------------------------------------------
# read in an excel sheet
# --------------------------------------------------------------------------
myData <- read_excel("raw_data/an_excel_file.xls",
                       sheet = "the_name_of_the_sheet_I_am_importing")
```

4.2 Code for reading in a CSV so you have it

There are a million ways to read in CSVs. In this pdf is the code so you have it.

```r
# --------------------------------------------------------------------------
# Base r
#   Pro: no need to load package
#   Con: less efficient, slower, and worse at getting variable types right
#   Use case: when you have a small and simple data set
# --------------------------------------------------------------------------
myData <- read.csv("/raw_data/an_imaginary_csv.csv")

# --------------------------------------------------------------------------
# Tidyverse package: readr
#   Pro: faster, intuitive at predicting variables types
#   Con: Requires a package
#   Use case: almost all the time
# --------------------------------------------------------------------------
# install.packages("readr")
library(readr)

myData <- read_csv("/raw_data/an_imaginary_csv.csv")

# --------------------------------------------------------------------------
# Another package: vroom
#   Pro: excellent for big data
#   Con: a bit clunkier than readr
#   Use case:  big data
# --------------------------------------------------------------------------
# install.packages("vroom")
library(vroom)

myData <- vroom("/raw_data/an_imaginary_csv.csv")
```

5 Today's Example

5.1 Data Download and Clean

First step: download csv and drag it into `raw_data` folder.

- Here is the link: https://github.com/a5creel/intro_to_programming/blob/main/lecture_material/4_data_manage_vis/raw_data/mpg.csv

- Click the download button

First, pull data into the `raw_data` *file we created while going through file structure. Next, create a script called* `1_data_clean.R` *in the* `scripts` *folder*

```r
# Andie Creel / Goal: data cleaning / Started: January 11 2023

# ------------------------------------------------------------------------------
# load libraries
# ------------------------------------------------------------------------------
library(readr) # reading in csv
library(dplyr) # data cleaning
# library(ggplot2) # if we're fighting to get data loaded

# ------------------------------------------------------------------------------
# read in: reading in data is slow, so i always call it _og in case
#     I regret something later and want to return to this step
# ------------------------------------------------------------------------------
myData_og <- read_csv("data/raw_data/mpg.csv")

# Look at the dataset so we know what variables we have etc
# View(myData_og)

# alternatives in case this isn't working
# myData_og <- read_csv("https://raw.githubusercontent.com/a5creel/intro_to_programming/main/lecture_ma

# data(mpg)
# myData_og <- mpg
# rm(mpg)

# ------------------------------------------------------------------------------
# clean data: what we learned this morning
#   - the variables you do or don't need should be informed by your research Q
# ------------------------------------------------------------------------------

# our research Q is:
#   - only interested in Ford, dodge and toyota
#   - not about the type of fuel "fl"
myData <- myData_og %>%
  filter(manufacturer == "ford" | manufacturer == "dodge" | manufacturer == "toyota") %>%
  select(-fl) %>%
  mutate(cyl = as.factor(cyl)) # data manipulation: factor datatypes are discrete categories



# write clean data so it's saved as a seperate file we can load
write_csv(myData, "data/clean_data/my_clean_data.csv")
```

- writing clean data can save memory and be fast for when you're doing analysis

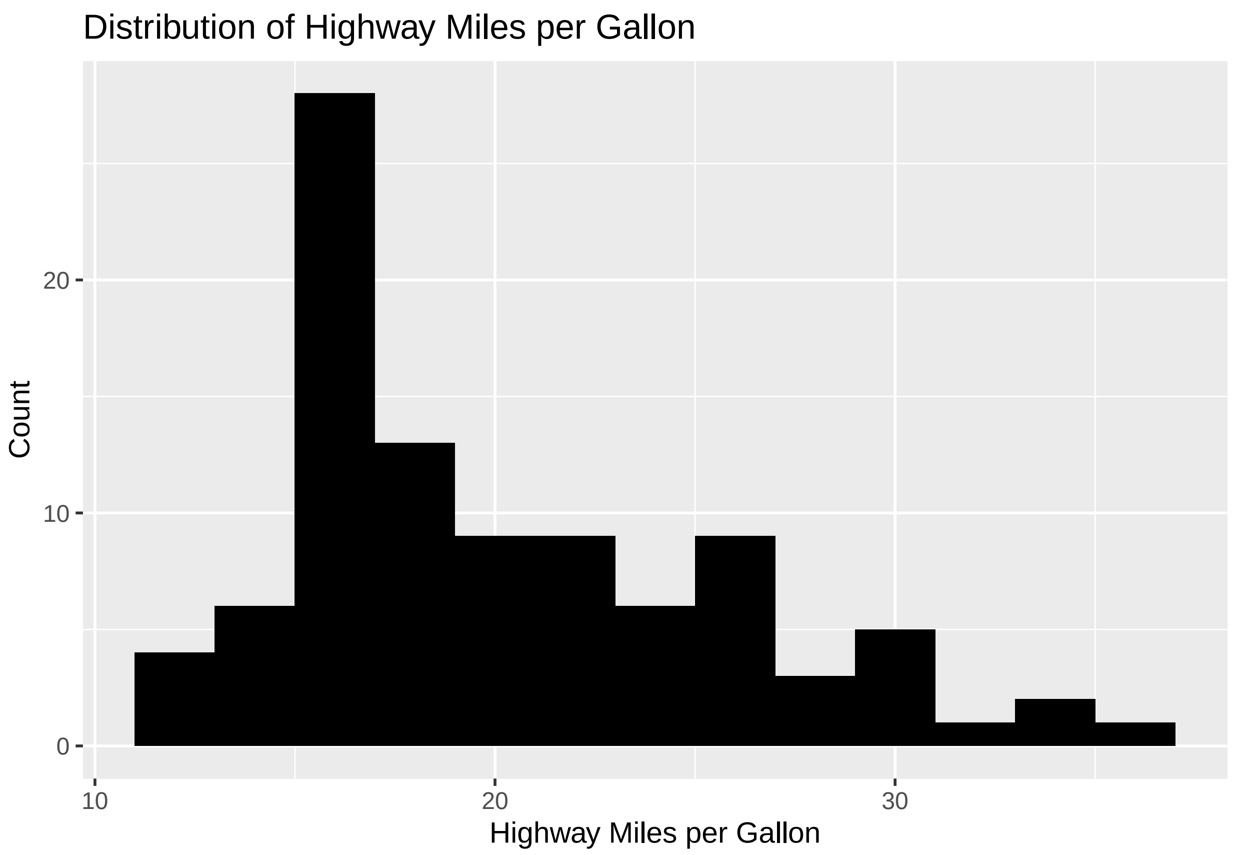
5.2 Data Visualization

- Normally, you'd do some more exploratory analysis in another file

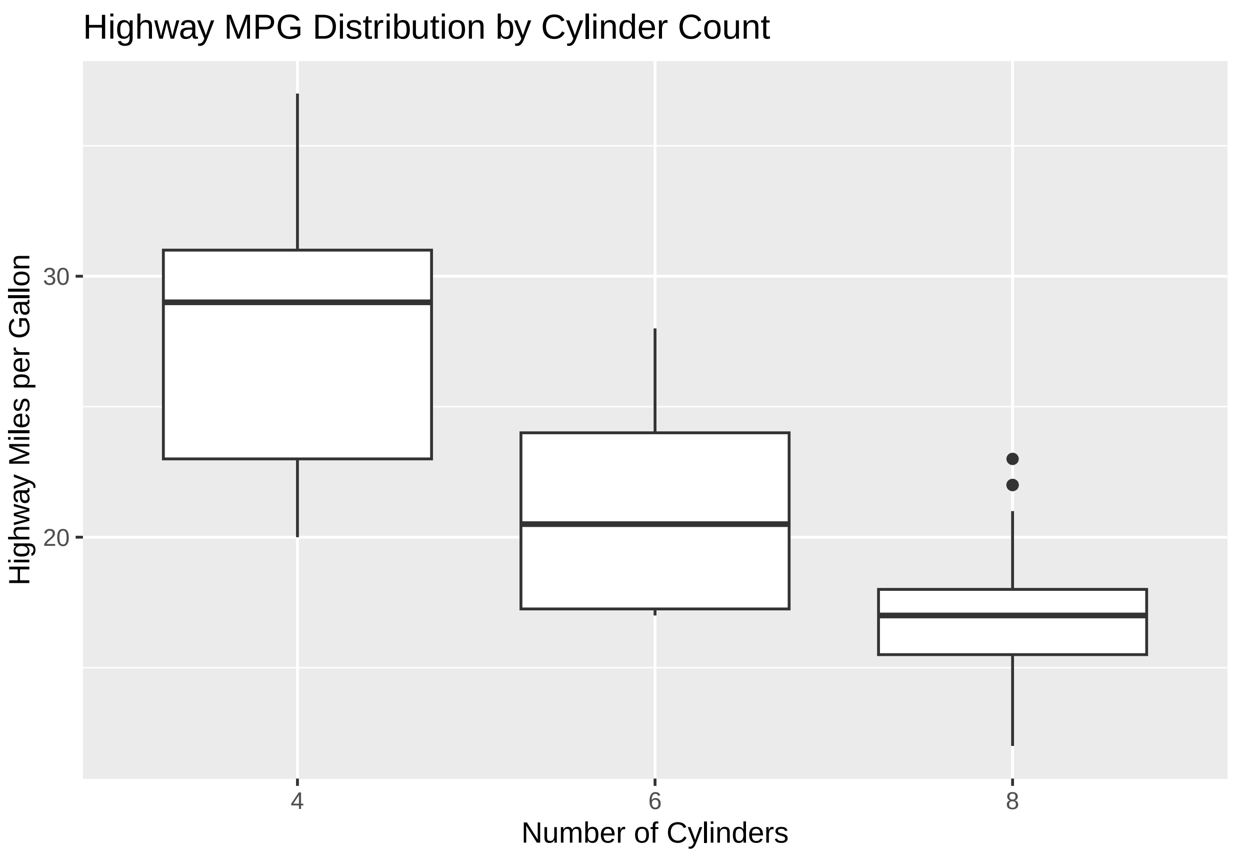
- You'd also do more data cleaning than this
- But we're going to move onto data visualization for times sake
- We'll do 6 charts together
    - Good data visualization requires choosing the right chart
    - Get a feel for it
- Then we'll polish one
    - see the types of things we can change
    - get some best practices for nice figures

Create a `2_figures.R` script in the `scripts` folder

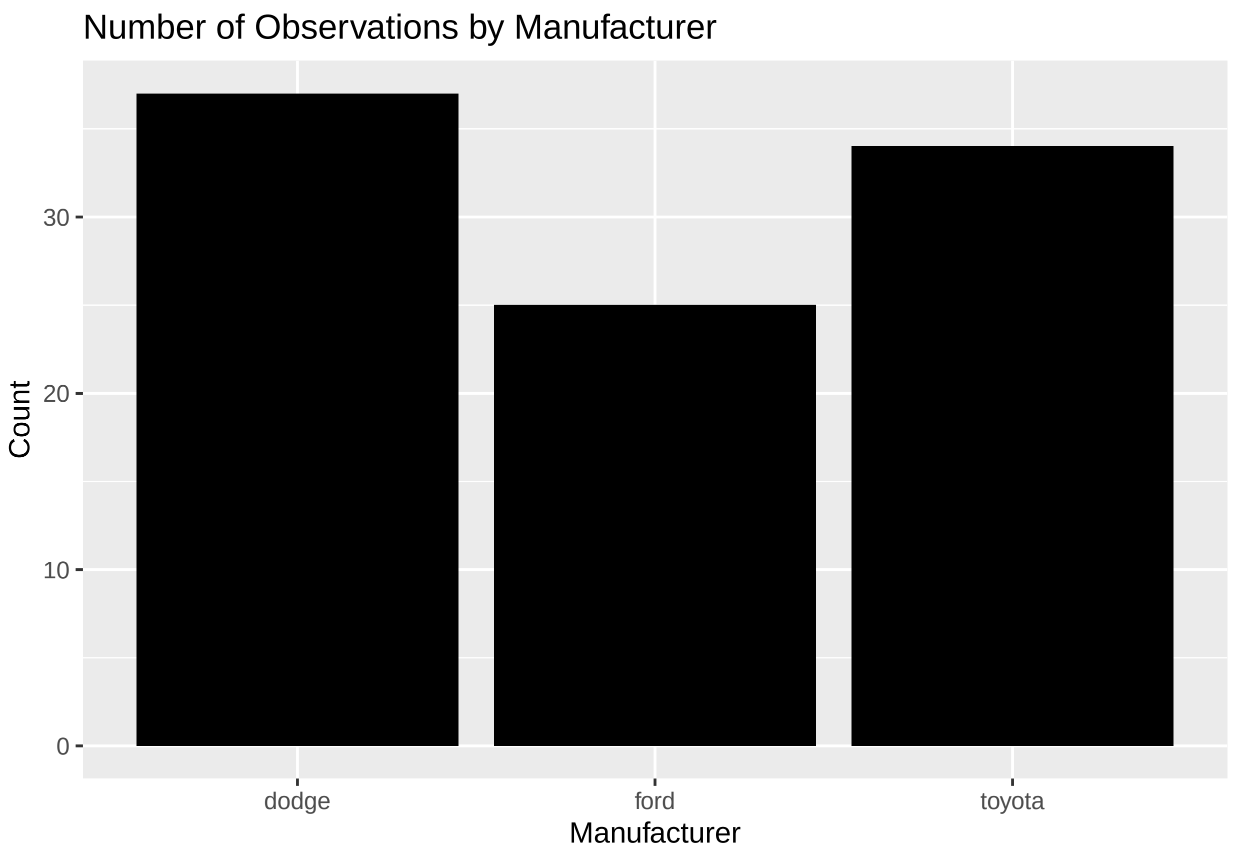
```
# Andie Creel / Goal: create figures / Started: January 11 2023

# -------------------------------------------------------------------------
# load libraries
# -------------------------------------------------------------------------
rm(list = ls()) # clear work space at beginning of script
library(dplyr)
library(readr)
library(ggplot2)

# -------------------------------------------------------------------------
# read in clean data
# -------------------------------------------------------------------------
myData <- read_csv("data/clean_data/my_clean_data.csv")

# -------------------------------------------------------------------------
# Histogram -- the distribution of a single numerical variable: geom_histogram()
# -------------------------------------------------------------------------
#AC: run these one line at a time
#   - aes: aesthetics
#   - tells you the axis you'll be plotting
#       - x axis: hwy mpg

# dataframe is an input to the plot function
ggplot(myData, aes(x = hwy)) +
  # geom_histogram() +
  geom_histogram(binwidth = 2, fill = "black") + # inputs change look
  labs(title = "Distribution of Highway Miles per Gallon",
       x = "Highway Miles per Gallon",
       y = "Count")
```



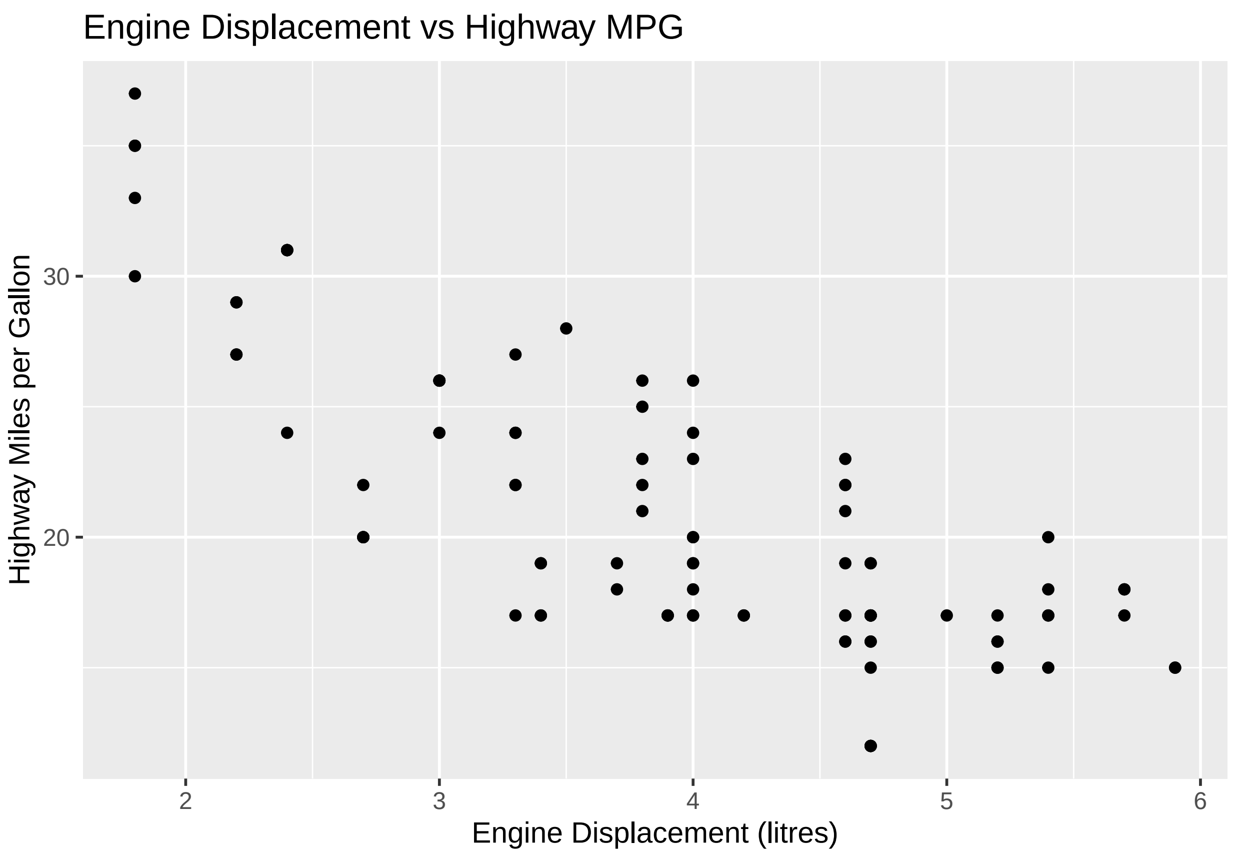
```
# ----------------------------------------------------------------------
# Box plot -- continuous variable for different categories: geom_boxplot()
#   cyl: number of cylinders
# ----------------------------------------------------------------------
myData %>% # Pipe data
  mutate(cyl = as.factor(cyl)) %>%  # didn't read in as a factor
  ggplot(aes(x = cyl, y = hwy)) +  # make the plot
    geom_boxplot() +
    labs(title = "Highway MPG Distribution by Cylinder Count",
         x = "Number of Cylinders",
         y = "Highway Miles per Gallon")
```



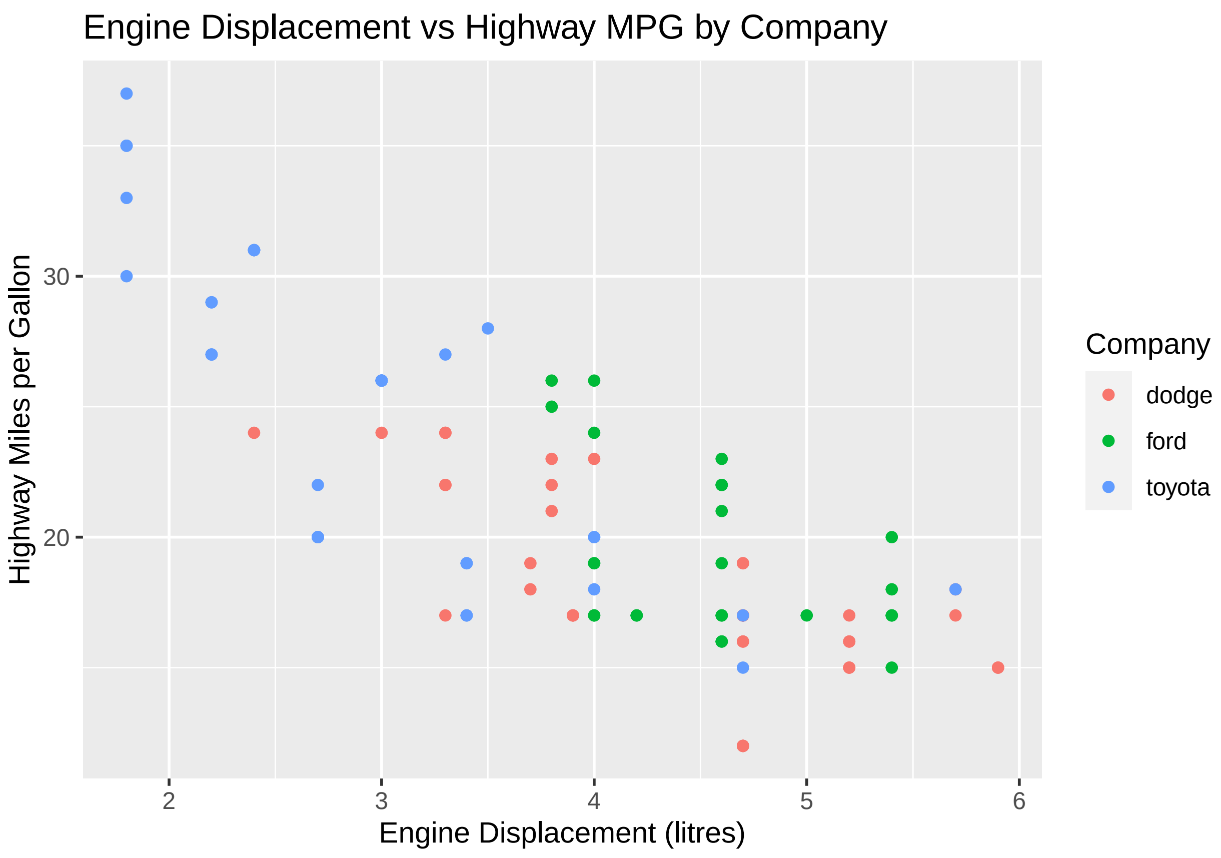
```r
# ---------------------------------------------------------------------------
# Bar chart -- count of observations in different categories: geom_bar()
# ---------------------------------------------------------------------------
ggplot(myData, aes(x = manufacturer)) +
  geom_bar(fill = "black") +
  labs(title = "Number of Observations by Manufacturer",
       x = "Manufacturer",
       y = "Count")
```



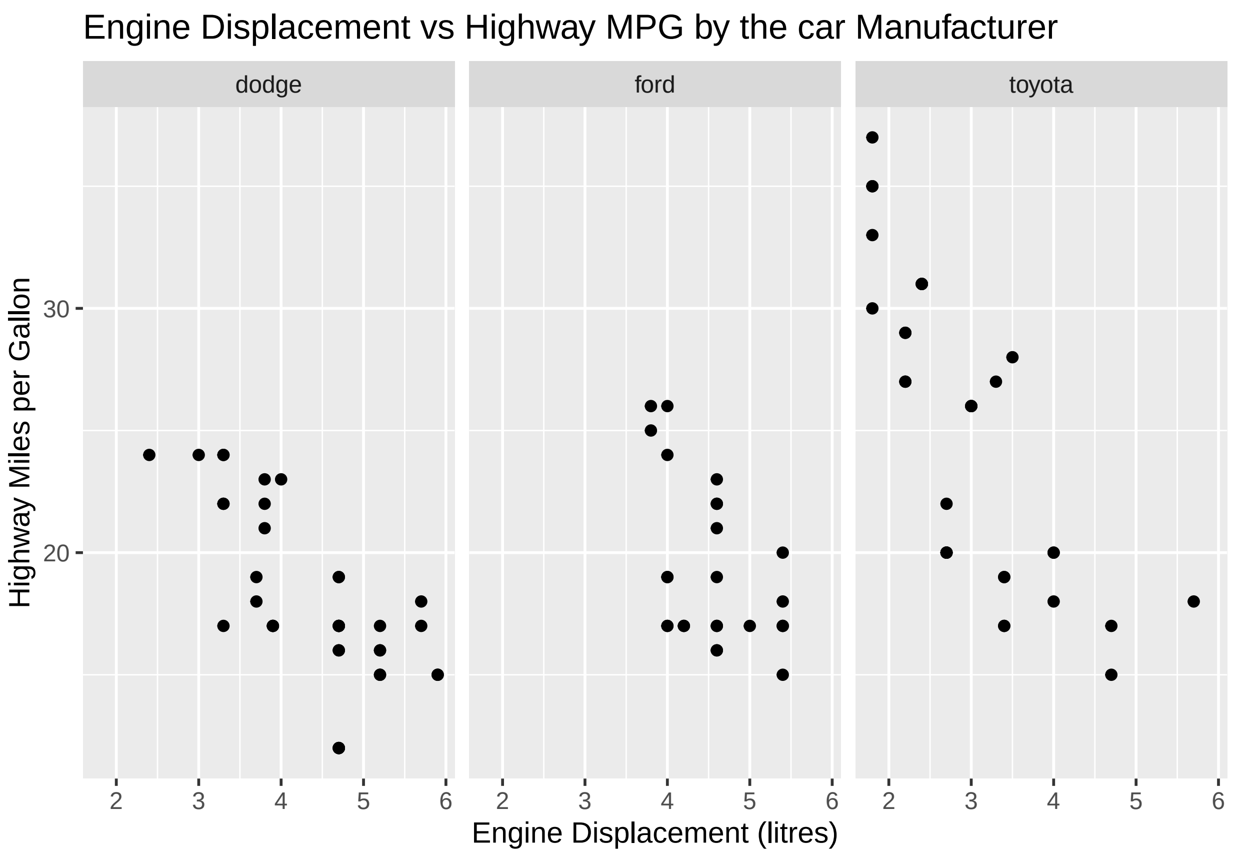
```r
# ------------------------------------------------------------------------
# Scatter plot -- two continuous variables: geom_point()
#   hwy: highway miles per gallon
#   displ: engine displacement which is approx. engine size
# ------------------------------------------------------------------------


ggplot(myData, aes(x = displ, y = hwy)) +
  geom_point() +
  labs(title = "Engine Displacement vs Highway MPG",
       x = "Engine Displacement (litres)",
       y = "Highway Miles per Gallon")
```



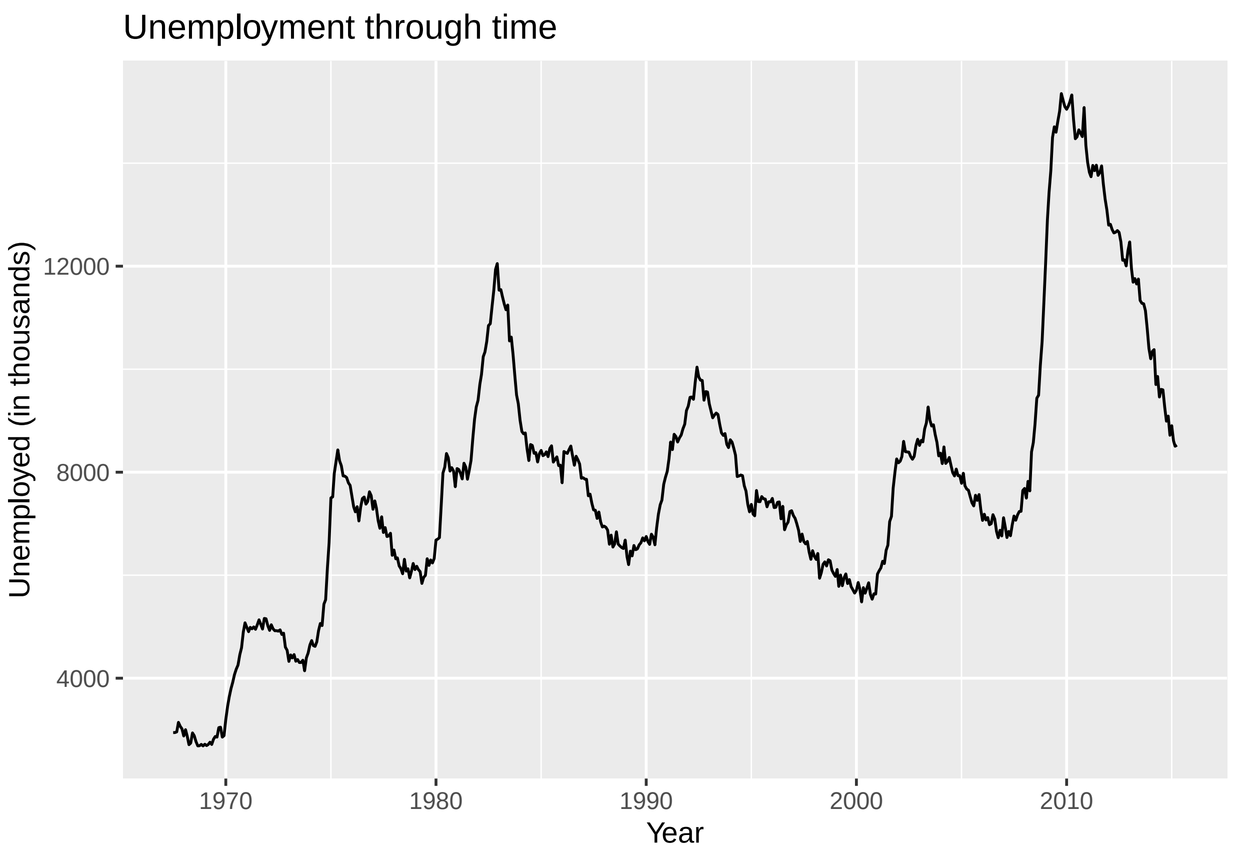
```r
# Third color axis: groups that you want shown in different colors.
ggplot(myData, aes(x = displ, y = hwy, color = manufacturer)) +
  geom_point() +
  labs(title = "Engine Displacement vs Highway MPG by Company",
       x = "Engine Displacement (litres)",
       y = "Highway Miles per Gallon",
       color = "Company")
```



```r
# ------------------------------------------------------------------------
# Multiple plots -- same graph for different categories: facet_wrap()
#     - same information last chart with the color
# ------------------------------------------------------------------------

# can store a graph as an object
myFacet <-  ggplot(myData, aes(x = displ, y = hwy)) +
  geom_point() + # type of graph you wanna see multiple times
  facet_wrap(~manufacturer) + # seperated by what?
  labs(title = "Engine Displacement vs Highway MPG by the car Manufacturer",
       x = "Engine Displacement (litres)",
       y = "Highway Miles per Gallon")

# display the graph
myFacet
```



```r
# ------------------------------------------------------------------------
# Line chart -- How a numeric variable changed through time: geom_line()
# ------------------------------------------------------------------------

# Use the economics data set (ggplot2 package)
data(economics)

myLine <- economics %>%
  ggplot(aes(x = date, y = unemploy)) +
  geom_line() +
    labs(title = "Unemployment through time",
       x = "Year",
       y = "Unemployed (in thousands)")

# Print chart
myLine
```



```
# ---------------------------------------------------------------------------
# Save a chart
# ---------------------------------------------------------------------------

# Line graph
ggsave("results/line_chart.png", # file path and name
       plot = myLine, # plot you want to save
       width = 10, # width of plot (inches)
       height = 8, # height of plot (inches)
       dpi = 300) # dots per square inch

# Facet
ggsave("results/facet.png",
       plot = myFacet,
       width = 10,
       height = 4, # specialize the height for the figure
       dpi = 300)
```

Open up one of the plots in our results folder.

These are okay, but not presentation ready.

6 A few of the best practices for data visualization

Why is data visualization so important?

- You put hundreds of hours into data collection and analysis
- Most people are just going to look at your figures
- If bad, they won't read your paper
- If good, they'll know your key results from a glance
- Figures make or break whether people know what you found

Let's take one of our charts and make it polished following a few basic best practices. Data visualization is an art all on it's own, and it's worth taking advantage of other resources in data visualization. However, these best practices will get you pretty far.

Best Practices

1) Choose the right graph: let your research question guide this
2) Write clear labels and titles
3) Simple as possible while not becoming reductive
4) All parts of graph are legible
5) Colors are not horrible and also work for people who are color blind
6) Use ggplot/code for as much as possible (saves so much time in long run if you get new data, change your mind about something, etc)

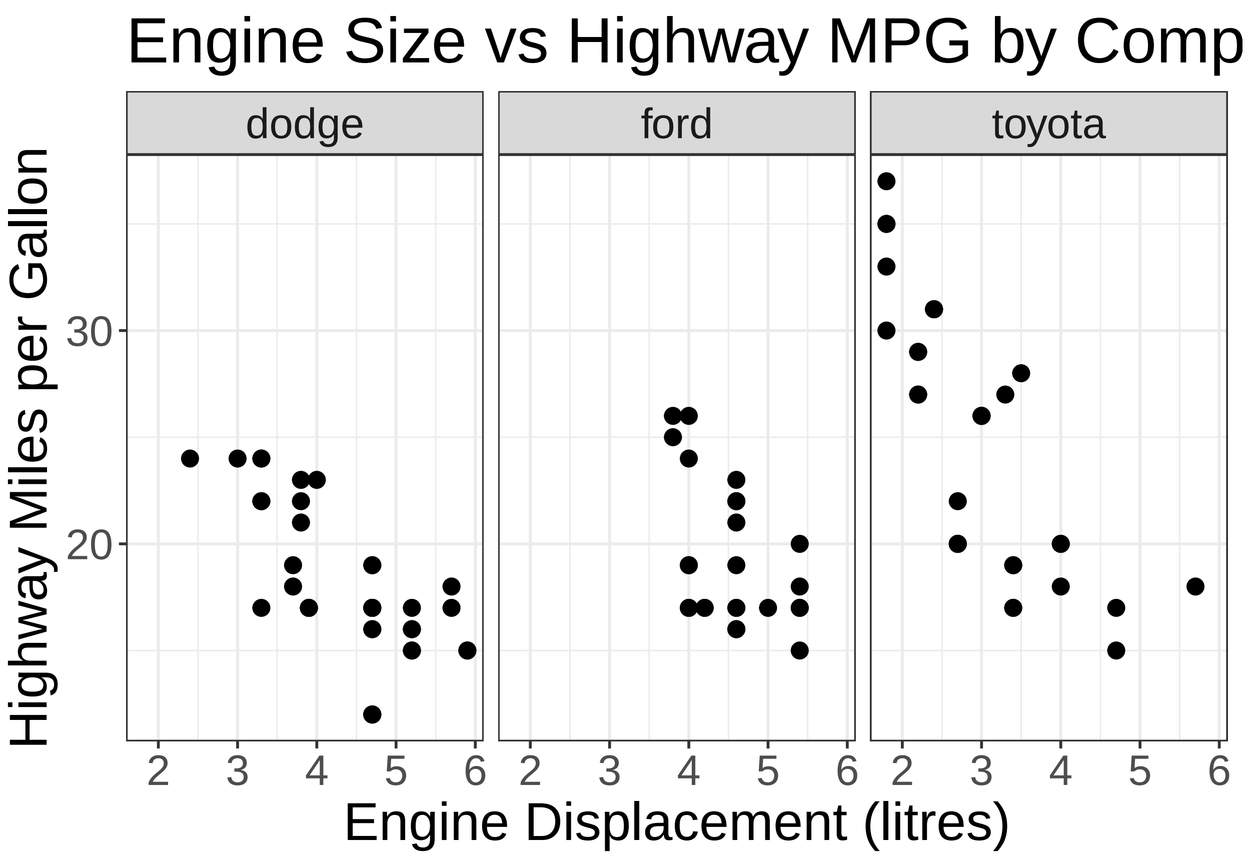
These six rules will get your pretty far.

Let's work with a plot we already made in the `2_results.R` file.

6.1 Facet chart

Consider if our research questions gas mileage relative to engine size, compared across companies.

```r
# -------------------------------------------------------------------------
# Work with out facet plot
# -------------------------------------------------------------------------

myFacet <- ggplot(myData, aes(x = displ, y = hwy)) +
  geom_point(size = 2.5) + # make points legible
  facet_wrap(~manufacturer) +
  labs(title = "Engine Size vs Highway MPG by Company", # clean up our title
       x = "Engine Displacement (litres)",
       y = "Highway Miles per Gallon") +
  # theme_minimal() +
  theme_bw() +                                        # choose simple theme
  theme(text = element_text(size=20))                 # make it legible

# print chart
myFacet
```



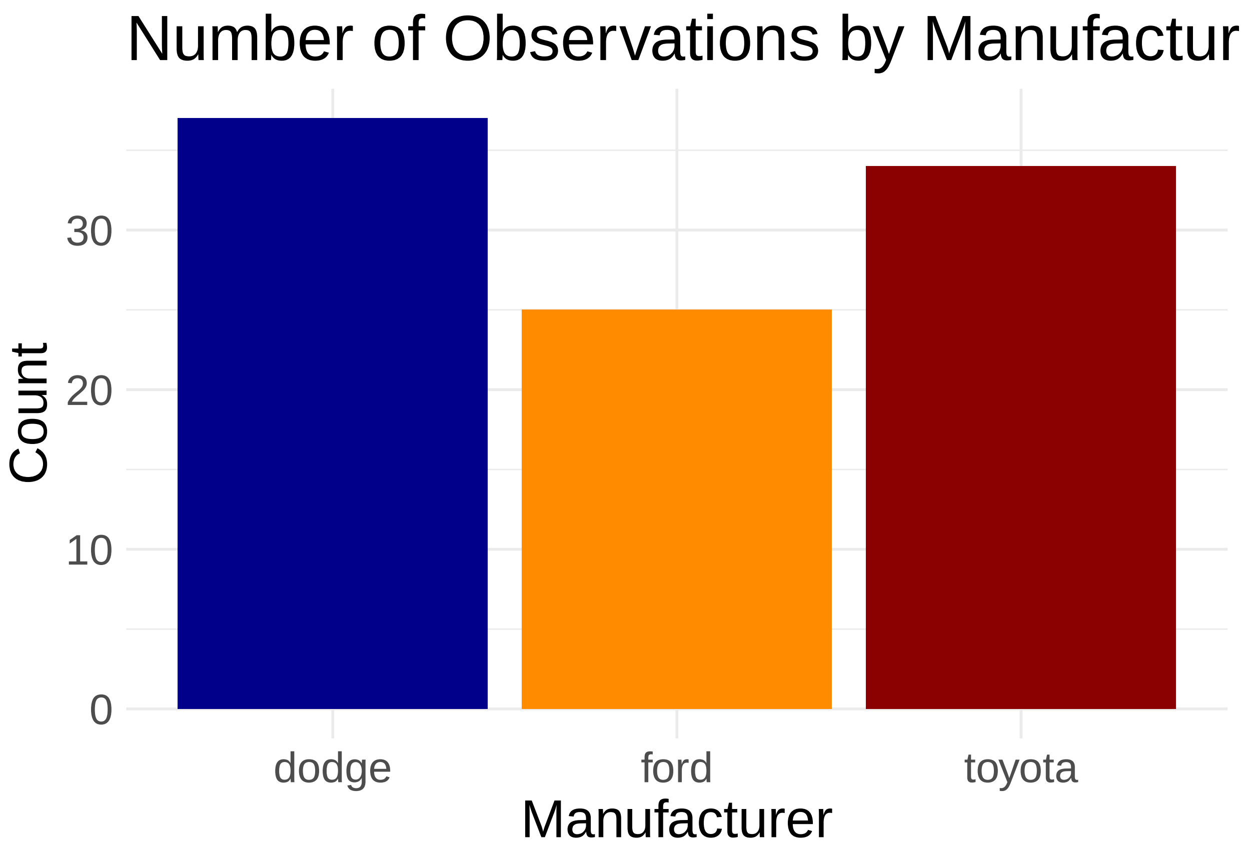
```
ggsave("results/facet.png",
       plot = myFacet,
       width = 10,
       height = 4,
       dpi = 300)
```

6.2 Bar chart (colors)

My main advise is to keep your colors as simple as possible.

Consider our bar chart.

```
# -----------------------------------------------------------------------
# Working with our old bar chart
# -----------------------------------------------------------------------

# colors I use: darkred, darkblue, darkgreen, darkorange a lot.

myBar <-  ggplot(myData, aes(x = manufacturer)) +
  geom_bar( fill = c("darkblue", "darkorange", "darkred")) + # change colors
  labs(title = "Number of Observations by Manufacturer",
       x = "Manufacturer",
       y = "Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  # theme_bw() +
  theme_minimal() +                                       # make it simple
  theme(text = element_text(size=20))                     # make it legible

# print chart
myBar
```



```
# show the difference between png and jpeg
ggsave("results/bar.jpeg",
       plot = myBar,
       width = 10,
       height = 8,
       dpi = 300)

ggsave("results/bar.png",
       plot = myBar,
       width = 10,
       height = 8,
       dpi = 300)
```

6.3 Wrap up

There are million ways to tweak charts, and you'll probably spend loads of time tweaking your figures in R. However, once you get over the learning curve, making them in R will be so much faster than if you're editing them in Power Point or Excel.

When it's all code, you can regenerate your figures extremely quick. This is so valuable when you get more data, or in the worst case where you realize you made a mistake data cleaning and need to fix that then regenerate all your data.