R BOOTCAMP EINSTEIN 2017

MICHOEL SNOW

AUGUST 8 & 10, 2017

SECTION 1 - THE BASICS

INSTALLING R

If you don't have them already installed you need to download and install R and R studio

- Download R
  - **https://cran.r-project.org/**
- R Studio
  - **https://www.rstudio.com/products/rstudio/download/**

INTRODUCTION TO R

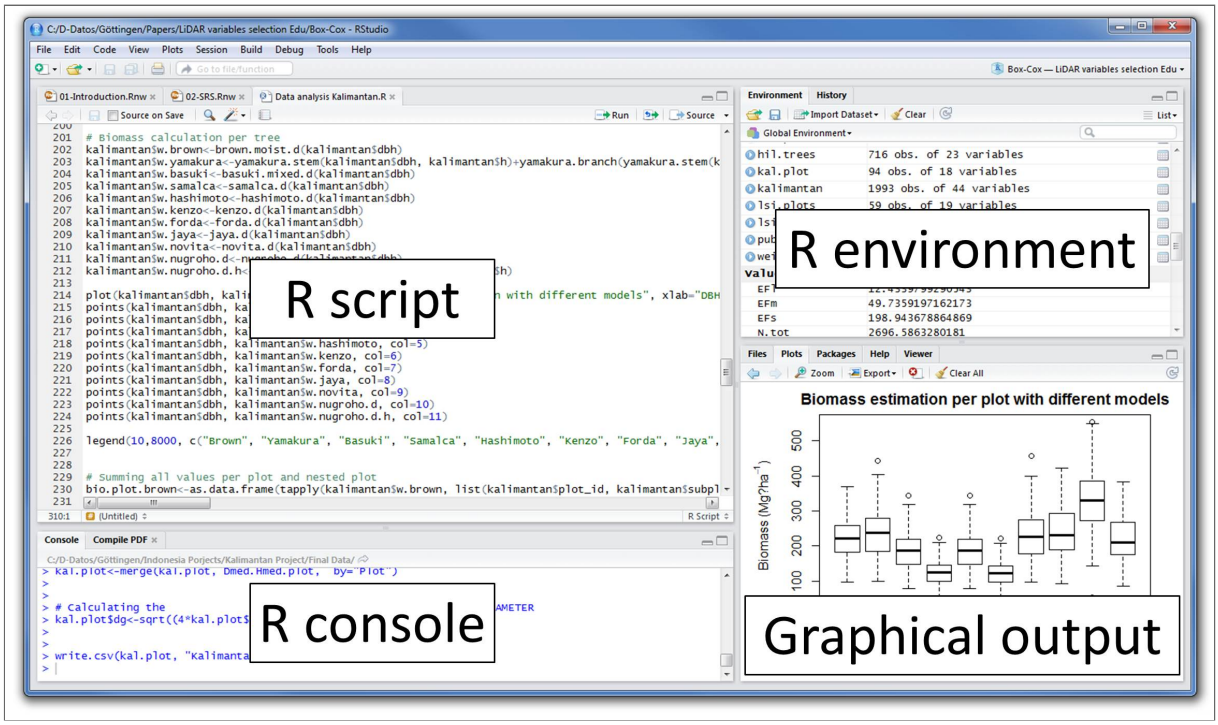
- R is an open source programming environment
- It is most commonly used for statistical computing and graphics
- The basic functionality of R can be extended with ‘packages’
    - Reusable R functions and their documentation
    - Often times will include sample data.

RESOURCES FOR R

- R for data science **http://r4ds.had.co.nz**
  - Served as inspiration for a lot of this bootcamp
  - Written by programmers at R Studio
- R Programming for Data Science **https://bookdown.org/rdpeng/rprogdatascience/**
- R Bootcamp **https://www.jaredknowles.com/r-bootcamp/**

INTRODUCTION TO R STUDIO

- R studio is an IDE within which you can write/edit code, run code, display plots/graphics and a whole host of other functions



- R Console
  - Can directly write and run code here
  - Shows the output of code you run
  - Mostly non recollectable/reproducible
- R Script
  - Here is where you write code and functions (scripts) which are run in the console
  - Recollectable and reproducible
- R Environment
  - Environment tab: variables, vectors, functions
  - History: Searchable history of all commands run in console
  - Other tabs as necessary, such as Git or Building
- Graphical Output
  - Files: List files in the current directory (useful when working with projects)
  - Plots: Displays plots and figures
  - Help : Displays R documentation

GETTING STARTED

- Go to **https://github.com/mseinstein/** to download the Presentions > RBootCamp > r_bootcamp_einstein_2017.R file
- All the code presented here is within that R file

WHAT CAN YOU DO WITH R

- Analytics
  - Mathematics, Probability and Statistics
  - Big Data Analytics
  - Statistical Modeling
  - Machine Learning
- Visualizations
  - Simple but powerful graphics packages
- Applications and Extensions
  - Web applications (Shiny)
  - Bioinformatics/Genomics (Bioconductor)
- Research Friendly
  - Reproducible Results
  - Strong Link to Academia

OBJECTIVES OF THIS BOOTCAMP

By the end of this bootcamp, students should be able to

1. Identify projects and tasks suitable to R
2. Use Rstudio as a frontend for R programming
3. Import data from outside sources into R
4. Manipulate data and perform basic analysis
5. Create and export visualizations of data
6. Modify pre-written R code for their own applications

STRUCTURE OF THIS BOOTCAMP

1. Visualization of Data
2. Manipulation of Data
3. Importing/cleaning of Data

RUNNING CODE IN R

- To run any line of code use CTRL+Enter or Cmd+Enter
  - R will run the highlighted code OR
  - R will run the line of code which the cursor is on

INSTALLING AND LOADING PACKAGES

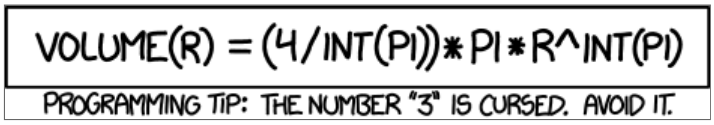
- The power of R can be extended with packages
- You install a package with the `install.packages("Pkg Name")`, e.g., `install.packages("tidyverse")`
  - Only needs to be done once per machine
  - Tidyverse is a collection of R packages for data manipulation and visualization that are designed to work together
  - This is the main package we will be using
- To load a package call the `library` function, e.g., `library(tidyverse)`
  - Needs to be loaded every session you want to use it (usually everytime you open R studio)

KEY SYMBOLS

- `< -`
  - Assignment operator, e.g. x <- 5
  - The equals sign, =, will work as a substitution in most but not all cases
- `#`
  - Denotes a comment
  - All text following the # will be ignored
- `?function_name`
  - Opens the Documentation for the function
- `??term`
  - Searches the help documentation for the queried string

BRIEF ASIDE

RXKCD



PROGRAMMING TIP: THE NUMBER "3" IS CURSED. AVOID IT.

SECTION 2 - DATA VISUALISATIONS

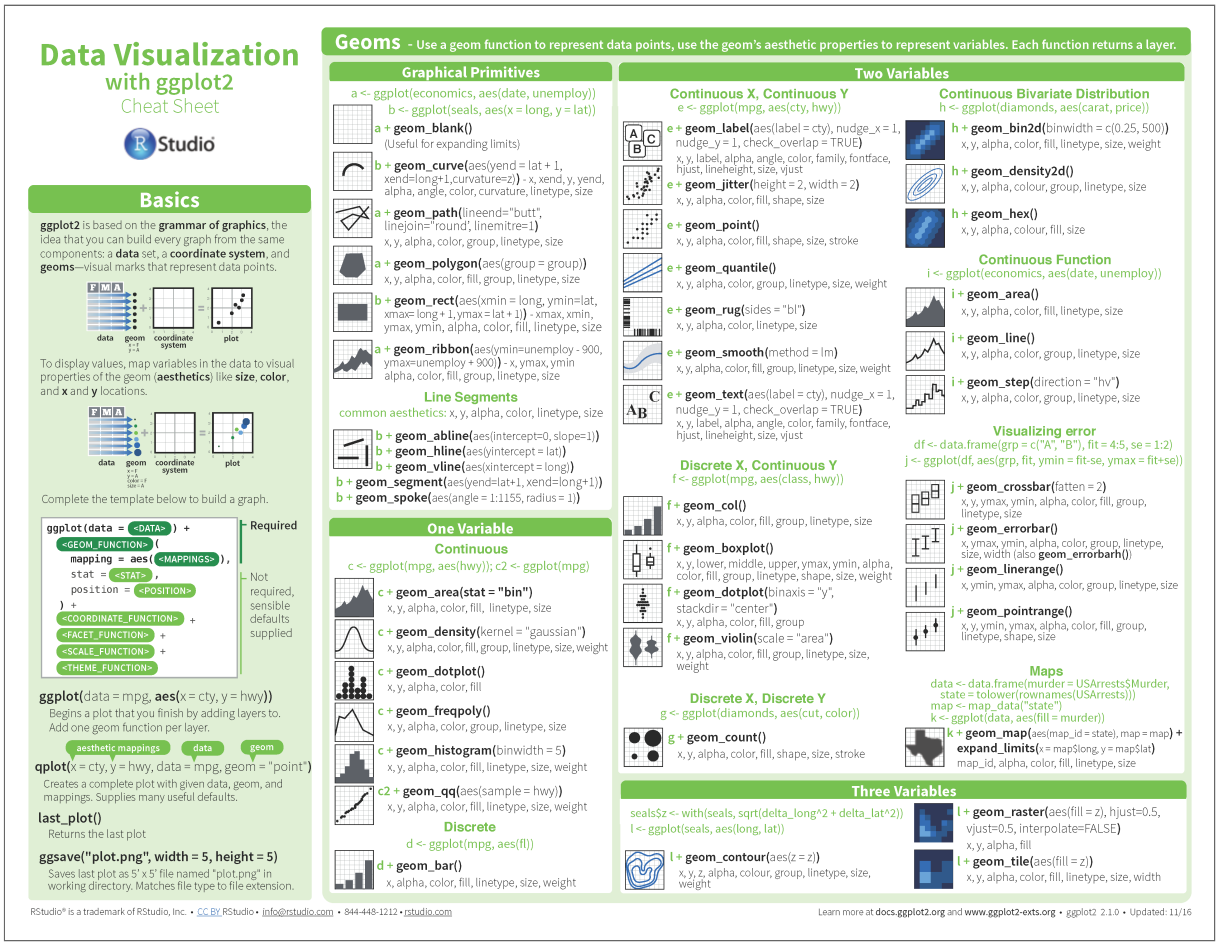
GGPLOT2

- ggplot2 is a plotting system for R, based on the grammar of graphics
- See **http://vita.had.co.nz/papers/layered-grammar.pdf** for an in-depth discussion

```
library(tidyverse)
```

DATA VISUALIZATION CHEAT SHEET

- R studio provides helpful cheat sheets which you can bring up from the Help toolbar
- Open the Data visualization for ggplot2 cheat sheet



Data Visualization with ggplot2 Cheat Sheet

RStudio

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

```
ggplot(data = <DATA>) +                    Required
  <GEOM_FUNCTION>(
    mapping = aes(<MAPPINGS>),
    stat = <STAT>,                         Not required, sensible defaults supplied
    position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <THEME_FUNCTION>
```

ggplot(data = mpg, **aes**(x = cty, y = hwy))
Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings · data · geom

qplot(x = cty, y = hwy, data = mpg, geom = "point")
Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot()
Returns the last plot

ggsave("plot.png", width = 5, height = 5)
Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms - Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

Graphical Primitives

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

- a + **geom_blank()** (Useful for expanding limits)
- b + **geom_curve**(aes(yend = lat + 1, xend=long+1,curvature=z)) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size
- a + **geom_path**(lineend="butt", linejoin="round", linemitre=1) x, y, alpha, color, group, linetype, size
- a + **geom_polygon**(aes(group = group)) x, y, alpha, color, fill, group, linetype, size
- b + **geom_rect**(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size
- a + **geom_ribbon**(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin alpha, color, fill, group, linetype, size

Line Segments

common aesthetics: x, y, alpha, color, linetype, size

- b + **geom_abline**(aes(intercept=0, slope=1))
- b + **geom_hline**(aes(yintercept = lat))
- b + **geom_vline**(aes(xintercept = long))
- b + **geom_segment**(aes(yend=lat+1, xend=long+1))
- b + **geom_spoke**(aes(angle = 1:1155, radius = 1))

One Variable

Continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

- c + **geom_area(stat = "bin")** x, y, alpha, color, fill, linetype, size
- c + **geom_density**(kernel = "gaussian") x, y, alpha, color, fill, group, linetype, size, weight
- c + **geom_dotplot()** x, y, alpha, color, fill
- c + **geom_freqpoly()** x, y, alpha, color, group, linetype, size
- c + **geom_histogram**(binwidth = 5) x, y, alpha, color, fill, linetype, size, weight
- c2 + **geom_qq**(aes(sample = hwy)) x, y, alpha, color, fill, linetype, size, weight

Discrete

d <- ggplot(mpg, aes(fl))

- d + **geom_bar()** x, alpha, color, fill, linetype, size, weight

Two Variables

Continuous X, Continuous Y

e <- ggplot(mpg, aes(cty, hwy))

- e + **geom_label**(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust
- e + **geom_jitter**(height = 2, width = 2) x, y, alpha, color, fill, shape, size
- e + **geom_point()** x, y, alpha, color, fill, shape, size, stroke
- e + **geom_quantile()** x, y, alpha, color, group, linetype, size, weight
- e + **geom_rug**(sides = "bl") x, y, alpha, color, linetype, size
- e + **geom_smooth**(method = lm) x, y, alpha, color, fill, group, linetype, size, weight
- e + **geom_text**(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

f <- ggplot(mpg, aes(class, hwy))

- f + **geom_col()** x, y, alpha, color, fill, group, linetype, size
- f + **geom_boxplot()** x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight
- f + **geom_dotplot**(binaxis = "y", stackdir = "center") x, y, alpha, color, fill, group
- f + **geom_violin**(scale = "area") x, y, alpha, color, fill, group, linetype, size, weight

Discrete X, Discrete Y

g <- ggplot(diamonds, aes(cut, color))

- g + **geom_count()** x, y, alpha, color, fill, shape, size, stroke

Continuous Bivariate Distribution

h <- ggplot(diamonds, aes(carat, price))

- h + **geom_bin2d**(binwidth = c(0.25, 500)) x, y, alpha, color, fill, linetype, size, weight
- h + **geom_density2d()** x, y, alpha, colour, group, linetype, size
- h + **geom_hex()** x, y, alpha, colour, fill, size

Continuous Function

i <- ggplot(economics, aes(date, unemploy))

- i + **geom_area()** x, y, alpha, color, fill, linetype, size
- i + **geom_line()** x, y, alpha, color, group, linetype, size
- i + **geom_step**(direction = "hv") x, y, alpha, color, group, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

- j + **geom_crossbar**(fatten = 2) x, y, ymax, ymin, alpha, color, fill, group, linetype, size
- j + **geom_errorbar()** x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)
- j + **geom_linerange()** x, ymin, ymax, alpha, color, group, linetype, size
- j + **geom_pointrange()** x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

- k + **geom_map**(aes(map_id = state), map = map) + **expand_limits**(x = map$long, y = map$lat) map_id, alpha, color, fill, linetype, size

Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
l <- ggplot(seals, aes(long, lat))

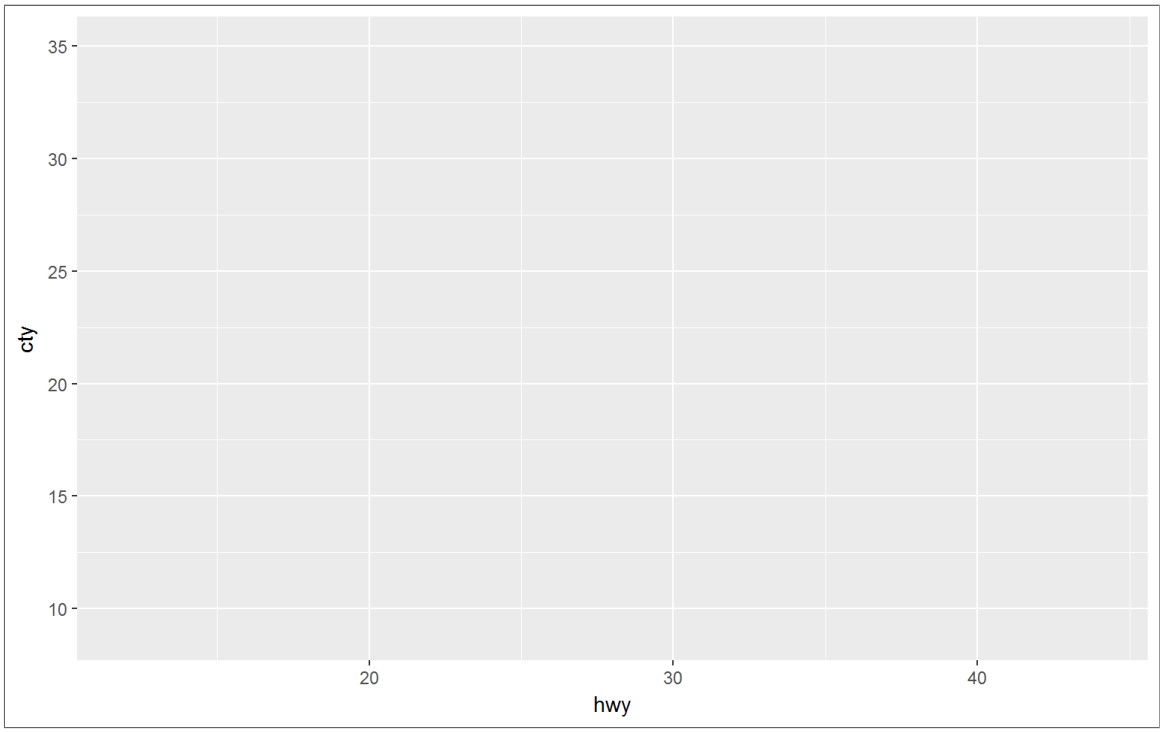
- l + **geom_contour**(aes(z = z)) x, y, z, alpha, colour, group, linetype, size, weight
- l + **geom_raster**(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE) x, y, alpha, fill
- l + **geom_tile**(aes(fill = z)) x, y, alpha, color, fill, linetype, size, width

Learn more at docs.ggplot2.org and www.ggplot2-exts.org • ggplot2 2.1.0 • Updated: 11/16

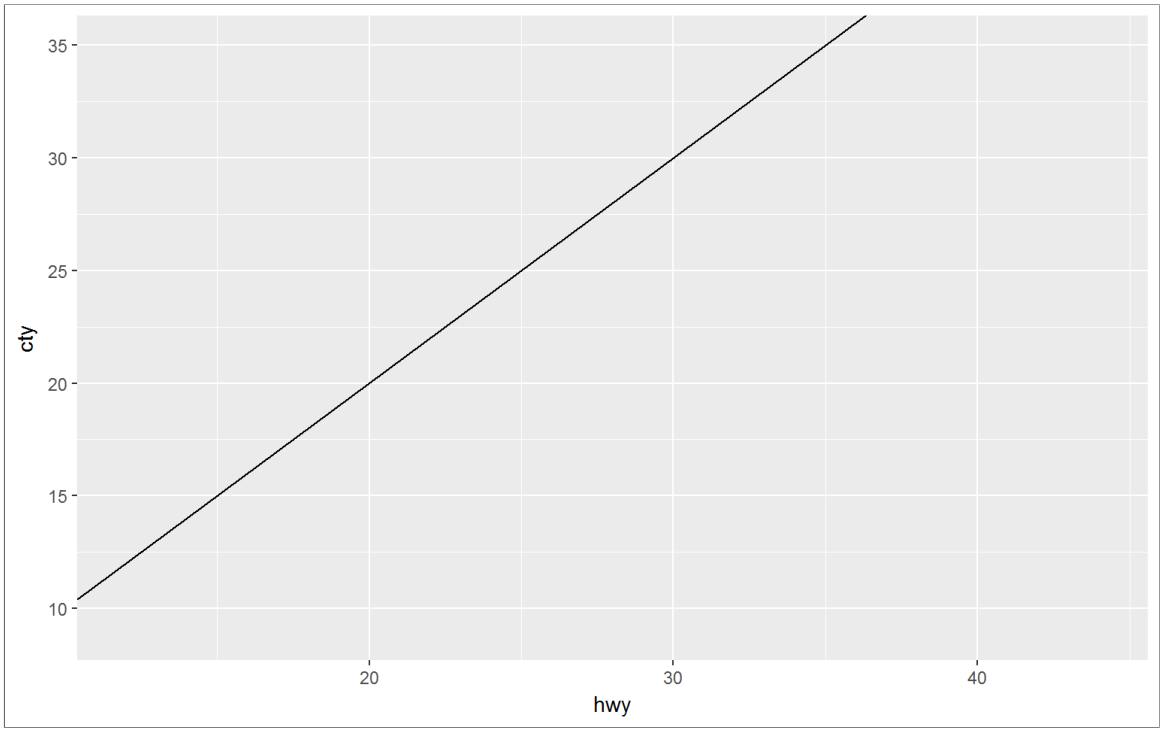
GRAPHICS IN GGPLOT ARE BUILT IN LAYERS

Use the data visualization cheatsheet in the RStudio Help menu as a reference

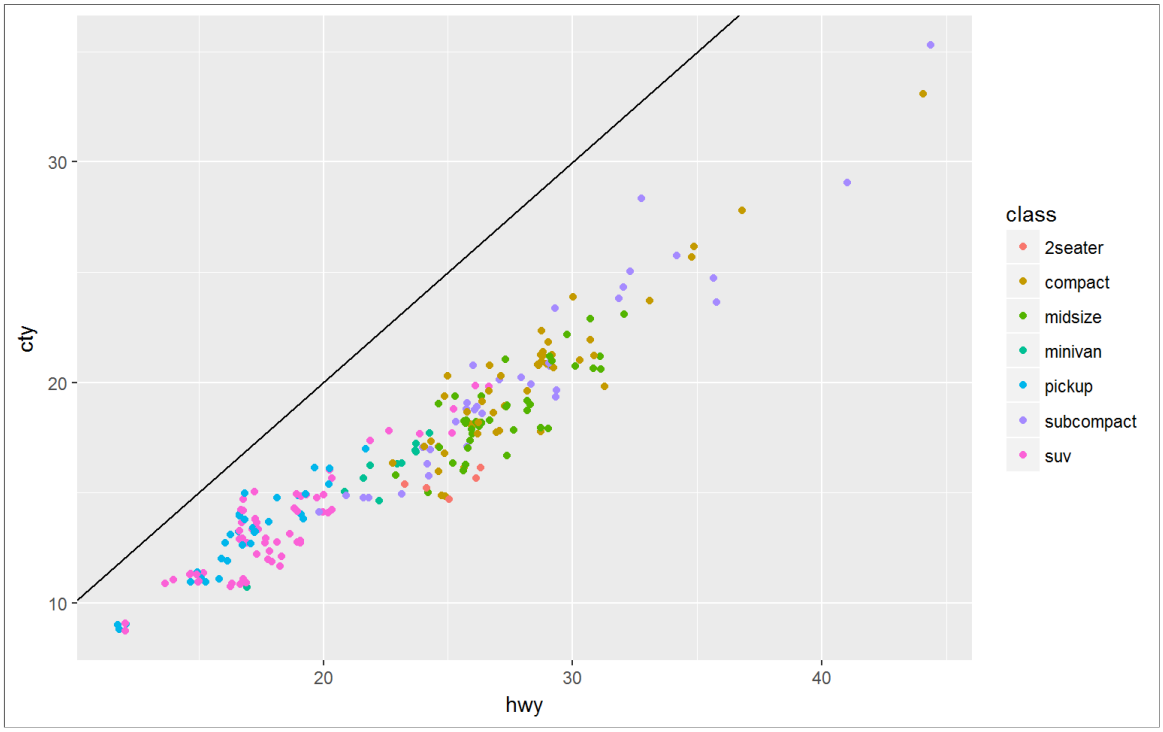
```
ggplot(mpg,aes(x=hwy,y=cty)) +
  geom_blank()
```



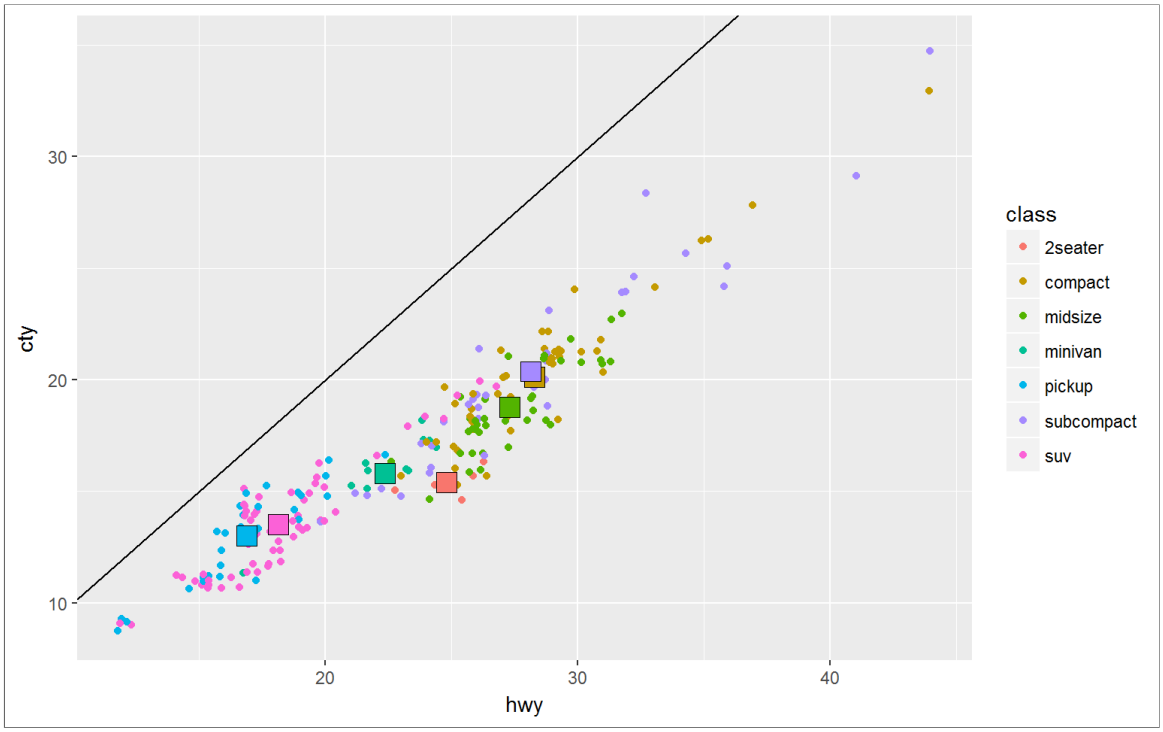

```
ggplot(mpg,aes(x=hwy,y=cty)) +
  geom_blank() +
  geom_abline()
```




```
ggplot(mpg,aes(x=hwy,y=cty)) +
  geom_blank() +
  geom_abline() +
  geom_jitter(aes(color=class))
```




```
ggplot(mpg,aes(x=hwy,y=cty)) +
  geom_blank() +
  geom_abline() +
  geom_jitter(aes(color=class)) +
  geom_point(data=mpg_class,aes(x=hwy_mean,y=cty_mean,fill=class),color="black",size=5,shape
```


cty
hwy
30
20
10
20
30
40
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

MPG DATASET

```
# mpg is a ggplot dataset of fuel economy data from 1999 and 2008 for 38 popular models of c
mpg
```

```
## # A tibble: 234 × 11
##    manufacturer      model displ  year   cyl      trans   drv   cty   hwy    fl   class
##           <chr>      <chr> <dbl> <int> <int>      <chr> <chr> <int> <int> <chr>   <chr>
## 1          audi         a4   1.8  1999     4   auto(l5)     f    18    29     p compact
## 2          audi         a4   1.8  1999     4 manual(m5)     f    21    29     p compact
## 3          audi         a4   2.0  2008     4 manual(m6)     f    20    31     p compact
## 4          audi         a4   2.0  2008     4   auto(av)     f    21    30     p compact
## 5          audi         a4   2.8  1999     6   auto(l5)     f    16    26     p compact
## 6          audi         a4   2.8  1999     6 manual(m5)     f    18    26     p compact
## 7          audi         a4   3.1  2008     6   auto(av)     f    18    27     p compact
## 8          audi a4 quattro   1.8  1999     4 manual(m5)     4    18    26     p compact
## 9          audi a4 quattro   1.8  1999     4   auto(l5)     4    16    25     p compact
## 10         audi a4 quattro   2.0  2008     4 manual(m6)     4    20    28     p compact
## # ... with 224 more rows
```

```
summary(mpg)
```

```
##  manufacturer          model               displ            year           cyl
##  Length:234         Length:234         Min.   :1.600   Min.   :1999   Min.   :4.000   Len
##  Class :character   Class :character   1st Qu.:2.400   1st Qu.:1999   1st Qu.:4.000   Cla
##  Mode  :character   Mode  :character   Median :3.300   Median :2004   Median :6.000   Mod
##                                        Mean   :3.472   Mean   :2004   Mean   :5.889
##                                        3rd Qu.:4.600   3rd Qu.:2008   3rd Qu.:8.000
##                                        Max.   :7.000   Max.   :2008   Max.   :8.000
##      drv                 cty              hwy             fl                class
##  Length:234         Min.   : 9.00   Min.   :12.00   Length:234         Length:234
##  Class :character   1st Qu.:14.00   1st Qu.:18.00   Class :character   Class :character
##  Mode  :character   Median :17.00   Median :24.00   Mode  :character   Mode  :character
##                     Mean   :16.86   Mean   :23.44
##                     3rd Qu.:19.00   3rd Qu.:27.00
##                     Max.   :35.00   Max.   :44.00
```

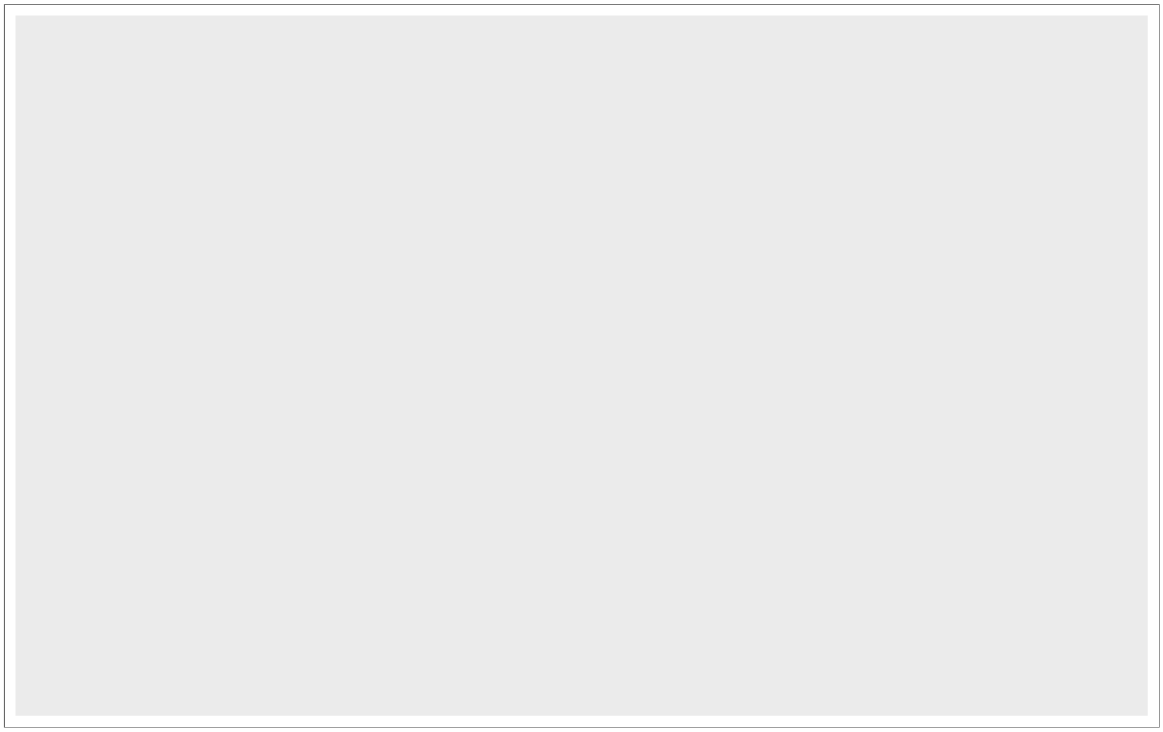
```
mpg$drv
```

```
##   [1] "f" "f" "f" "f" "f" "f" "f" "4" "4" "4" "4" "4" "4" "4" "4" "4" "4" "4" "r" "r" "r"
##  [29] "4" "4" "4" "4" "f" "f" "f" "f" "f" "f" "f" "f" "f" "f" "f" "f" "f" "f" "f" "f" "4"
##  [57] "4" "4" "4" "4" "4" "4" "4" "4" "4" "4" "4" "4" "4" "4" "4" "4" "4" "4" "r" "r" "r"
##  [85] "4" "4" "4" "4" "4" "4" "r" "r" "r" "r" "r" "r" "r" "r" "r" "f" "f" "f" "f" "f" "f"
## [113] "f" "f" "f" "f" "f" "f" "f" "f" "f" "f" "4" "4" "4" "4" "4" "4" "4" "4" "4" "4" "4"
## [141] "4" "f" "f" "f" "f" "f" "f" "f" "f" "f" "f" "4" "4" "4" "4" "f" "f" "f" "f" "f" "4" "4"
## [169] "4" "4" "4" "4" "4" "4" "4" "4" "4" "4" "4" "f" "f" "f" "f" "f" "f" "f" "f" "f" "f" "f"
## [197] "f" "f" "4" "4" "4" "4" "4" "4" "4" "4" "4" "f" "f" "f" "f" "f" "f" "f" "f" "f" "f" "f"
## [225] "f" "f" "f" "f" "f" "f" "f" "f" "f" "f"
```

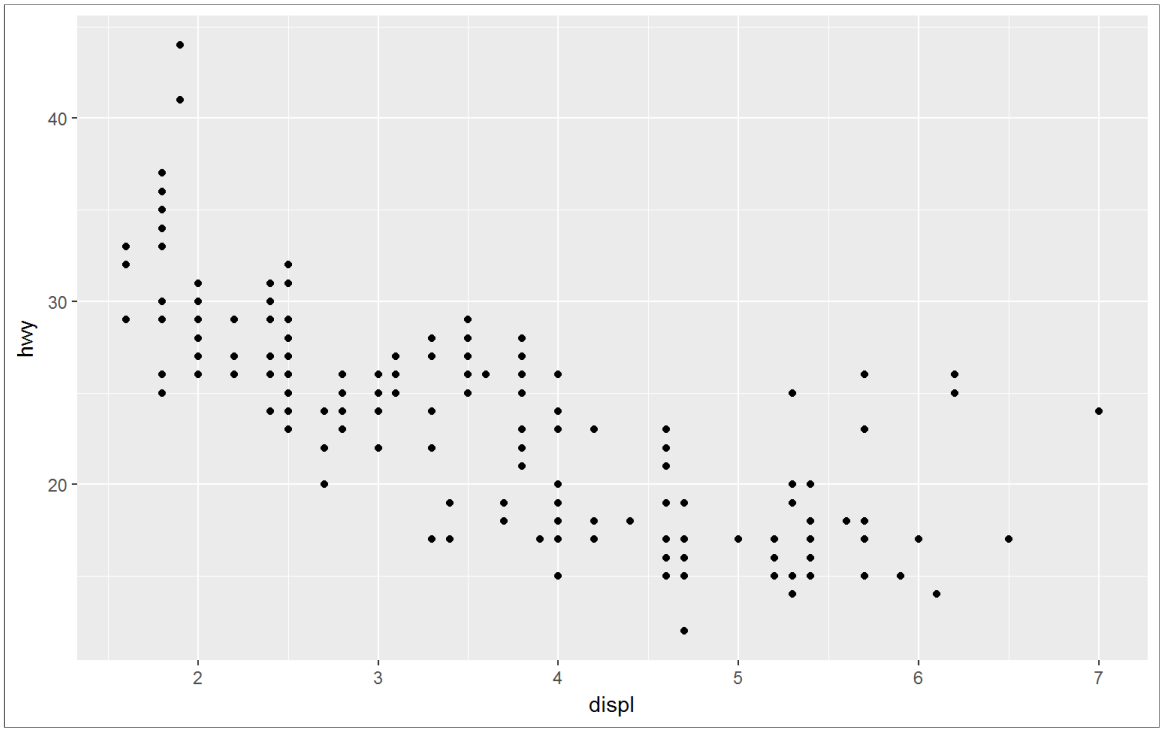
GGPLOT COMMAND

The ggplot command begins a plot to which you can add layers

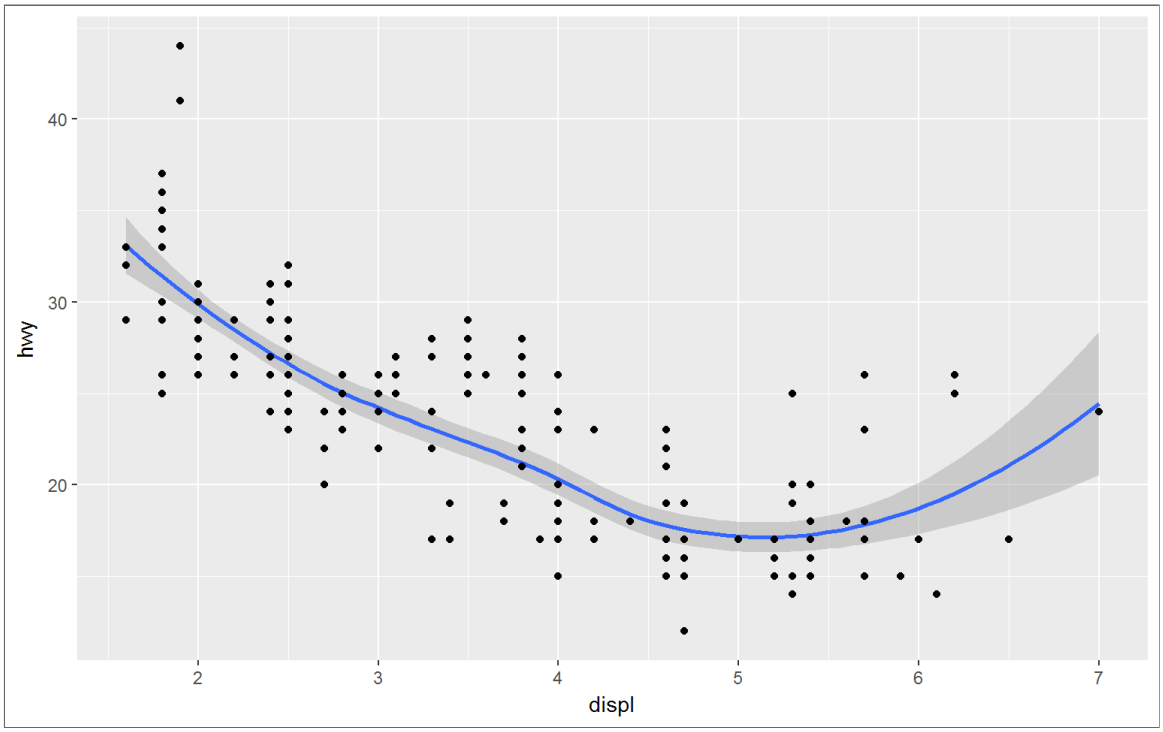
```
ggplot(data = mpg)
```



```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```



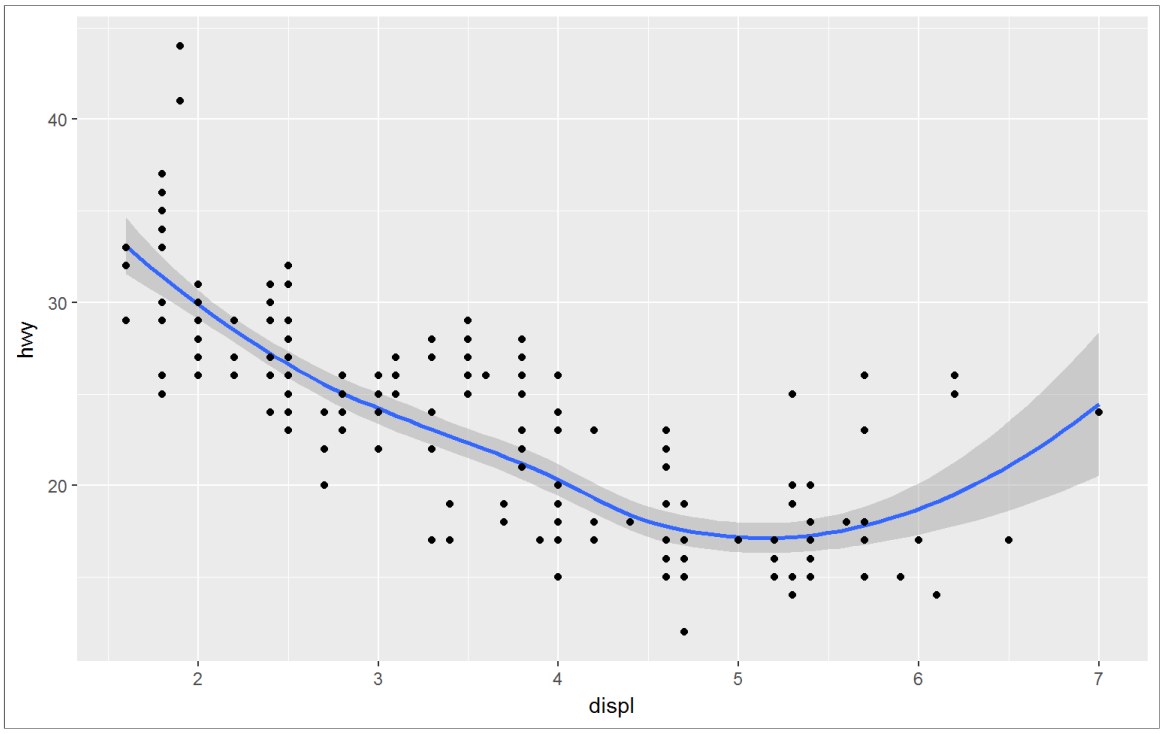

```
ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(x = displ, y = hwy))
```



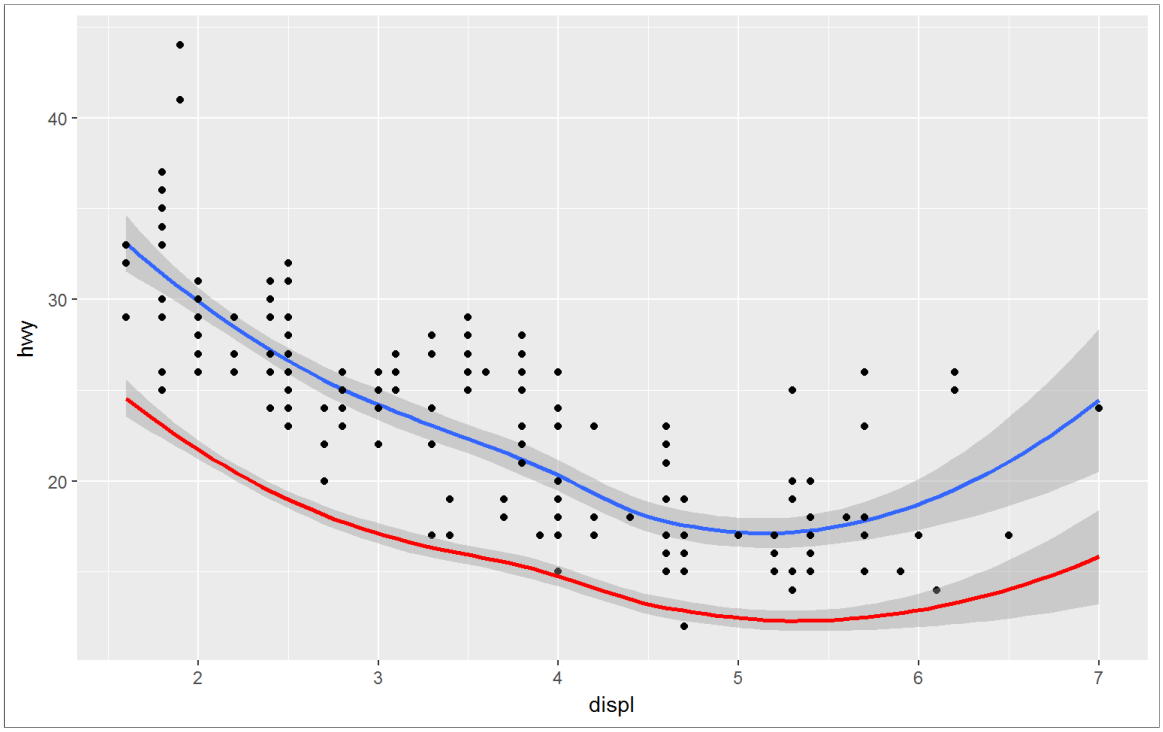
VARIABLE PROPAGATION

By inserting x and y into ggplot, all layers will use those parameters unless otherwise specified

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_smooth() +
  geom_point()
```



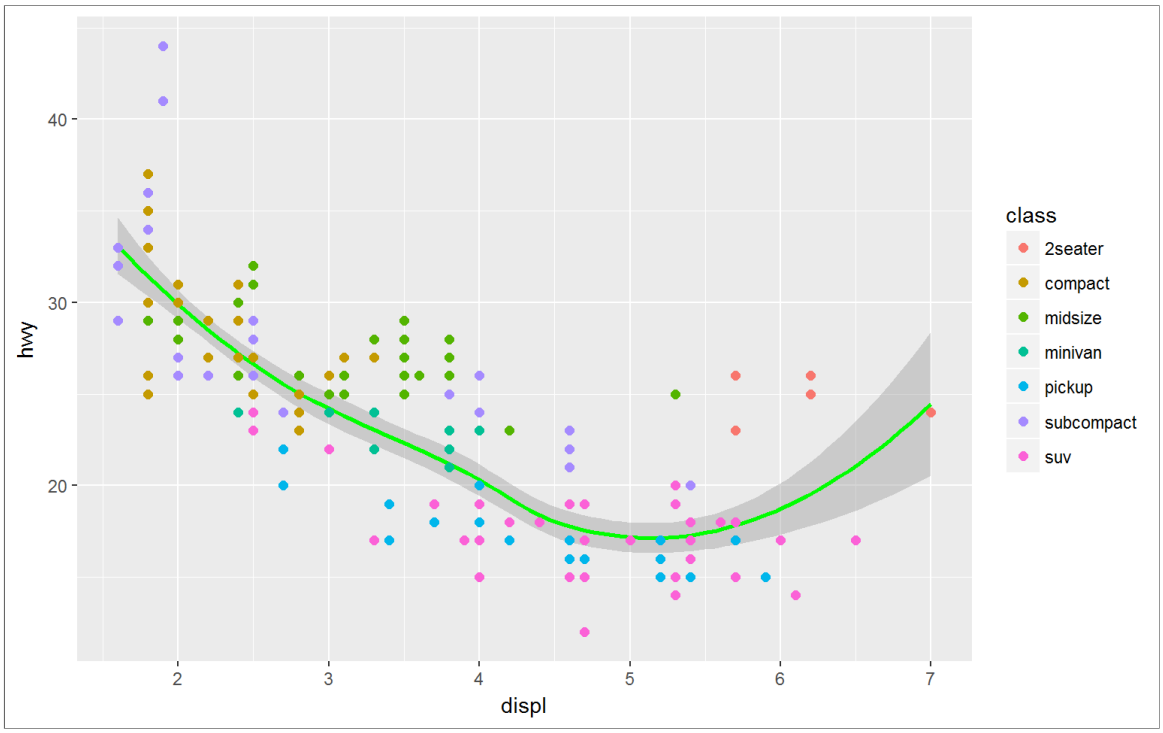

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_smooth() +
  geom_point() +
  geom_smooth(mapping = aes(x = displ,y = cty),color = "red")
```


40
30
20
hwy
2
3
4
5
6
7
displ

FEATURES VS VARIABLES

Note: if you want to plot a variable to a feature like color or size, it must go in the aes() term, if you just want to set them at a certain value they go outside the aes

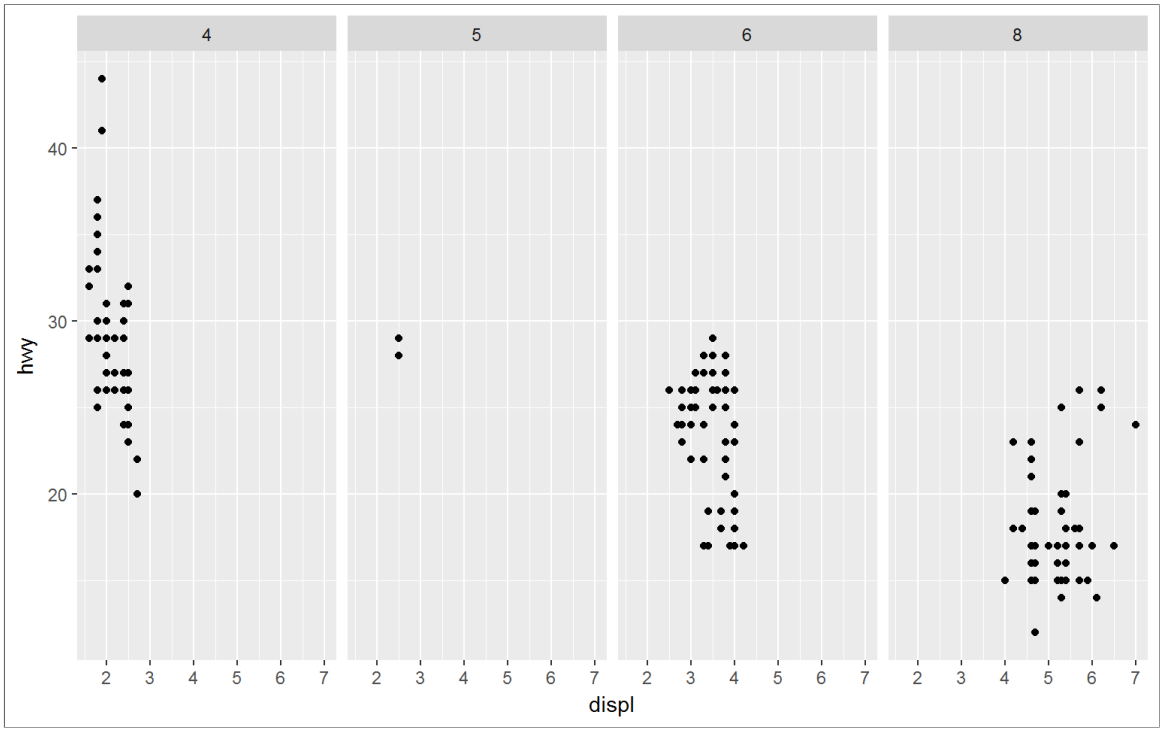
```
ggplot(mpg, aes(displ,hwy)) +
  geom_smooth(color = "green") +
  geom_point(aes(color=class),size=2)
```



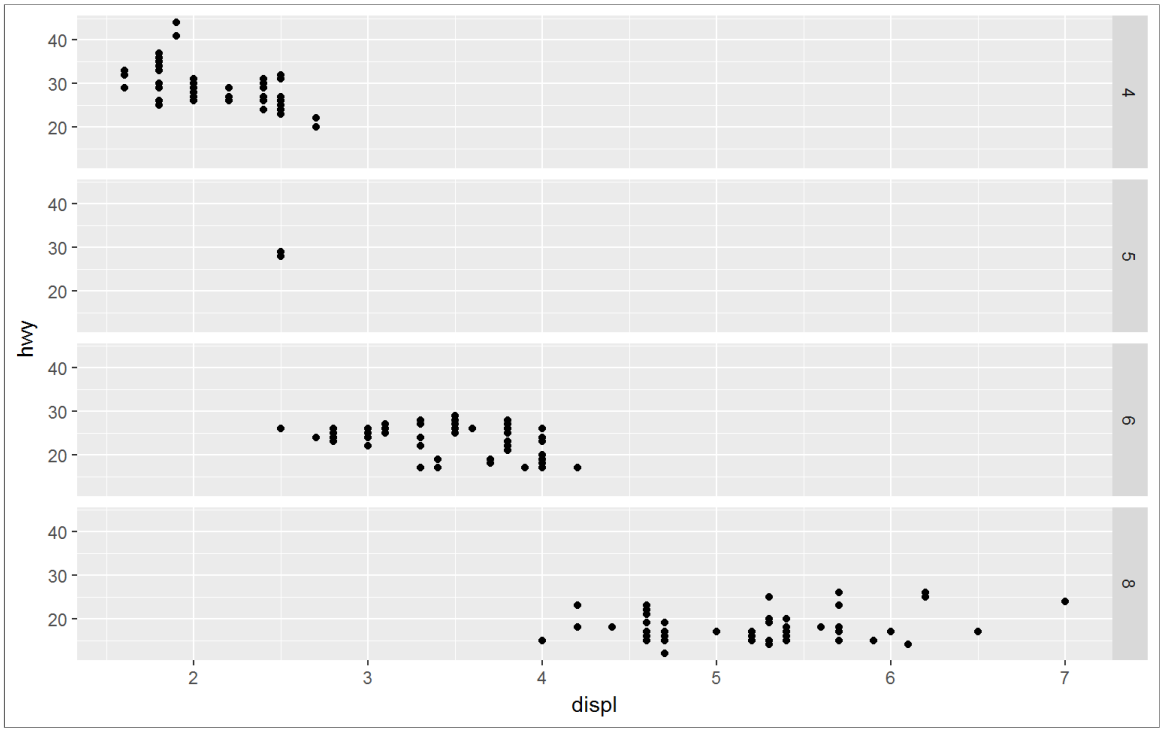
FACETS

You can also split the plot into subplots based on a varible using `facet`

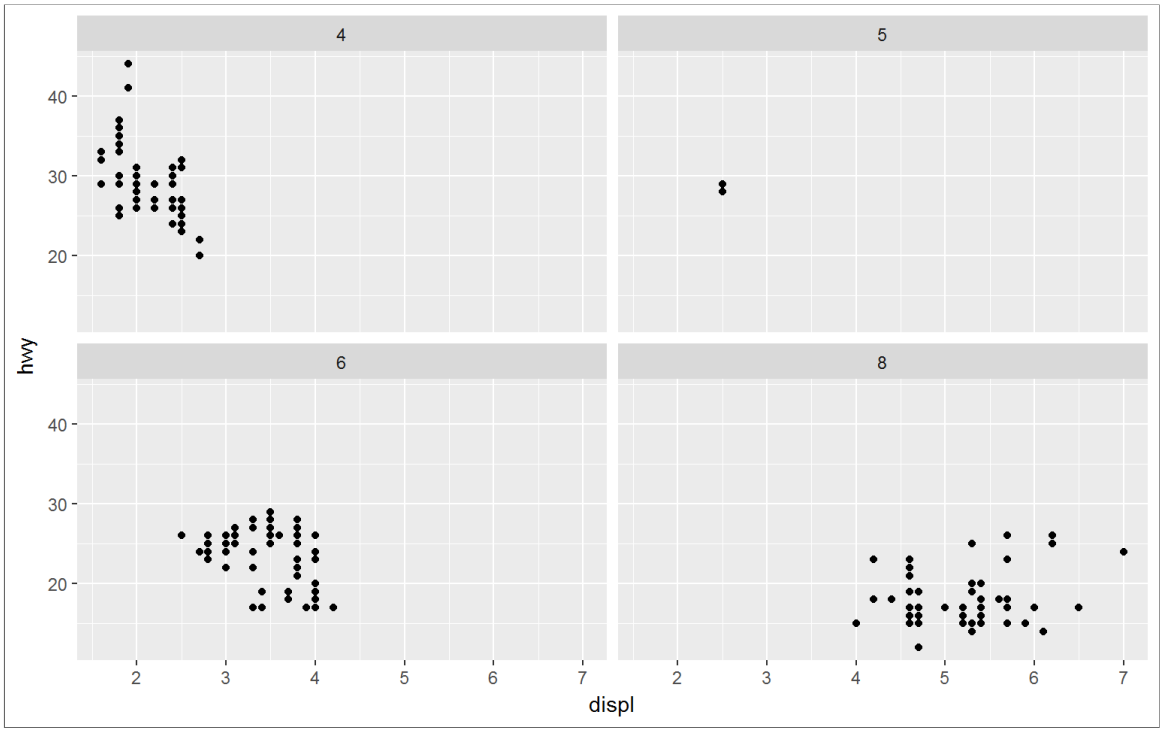
```
ggplot(mpg, aes(displ,hwy))  +
  geom_point() +
  facet_grid(.~cyl)
```



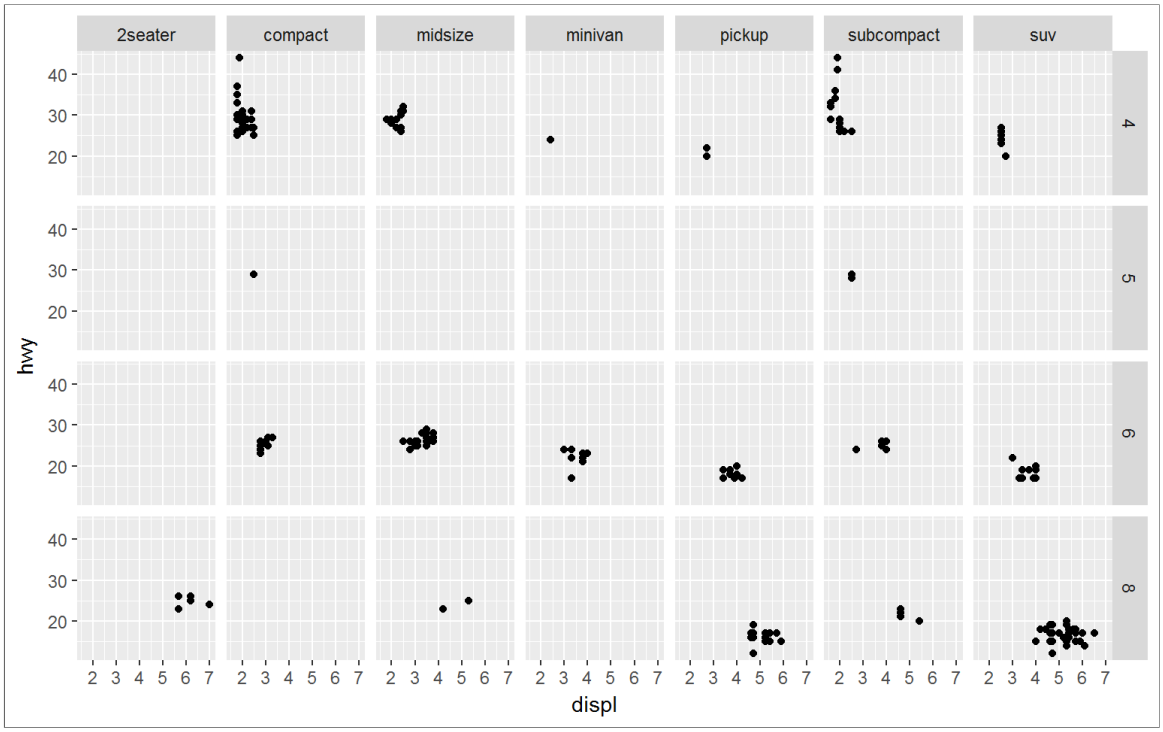

```
ggplot(mpg, aes(displ,hwy))  +
  geom_point() +
  facet_grid(cyl~.)
```




```
ggplot(mpg, aes(displ,hwy))  +
  geom_point() +
  facet_wrap(~cyl)
```

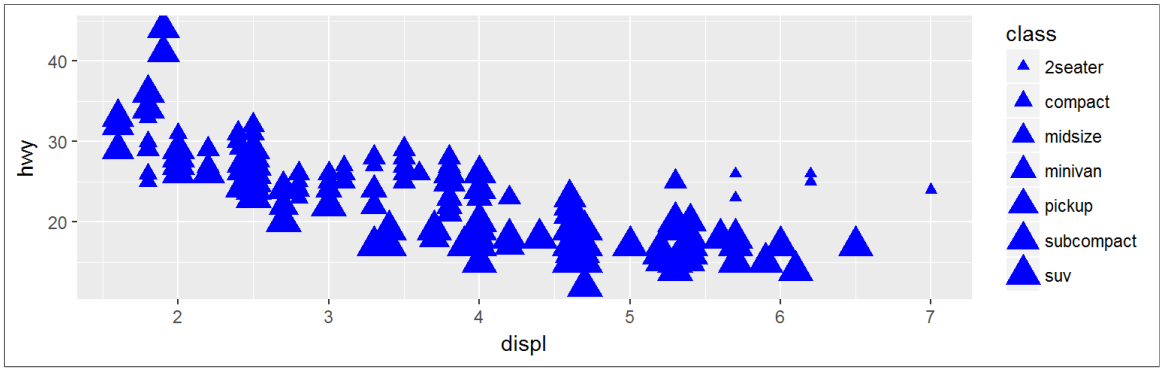


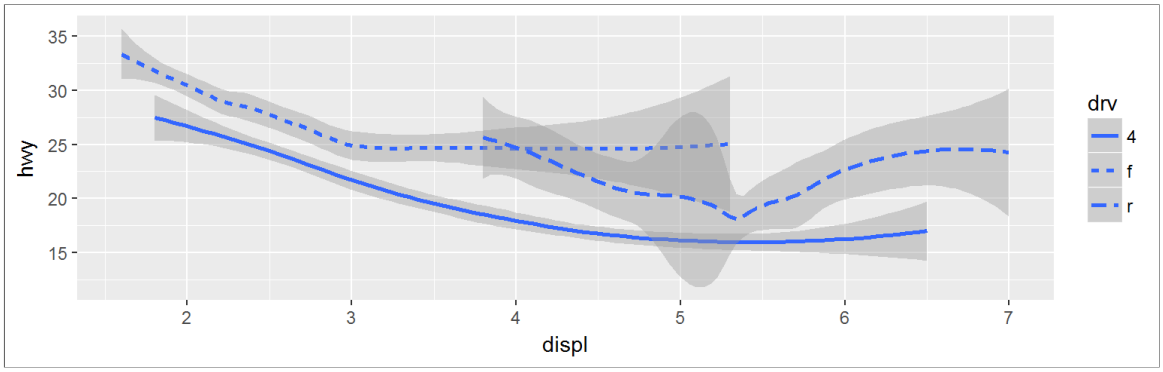

```
ggplot(mpg, aes(displ,hwy))  +
  geom_point() +
  facet_grid(cyl~class)
```

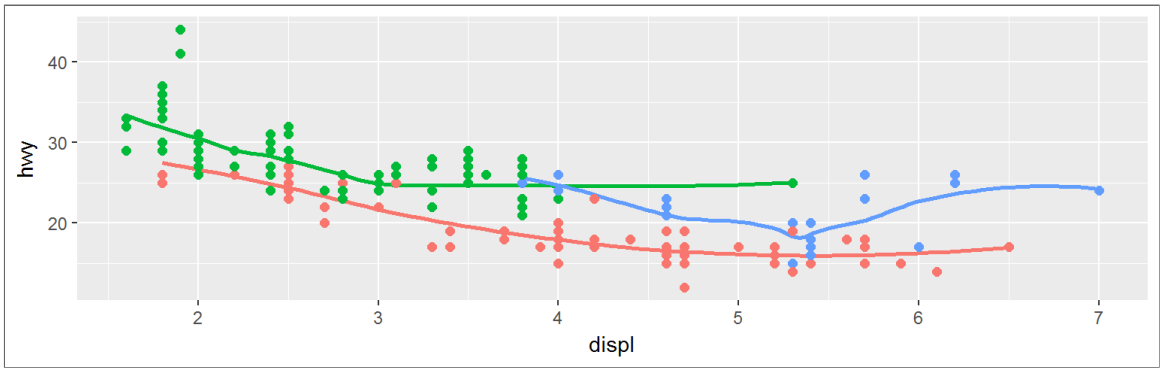

2seater
compact
midsize
minivan
pickup
subcompact
suv
4
5
6
8
hwy
displ
40
30
20
2 3 4 5 6 7

EXERCISES

Try to make some of the following plots:







GRAPHICS AND STATISTICAL TRANSFORMATIONS

This time we will be using the `diamonds` dataset

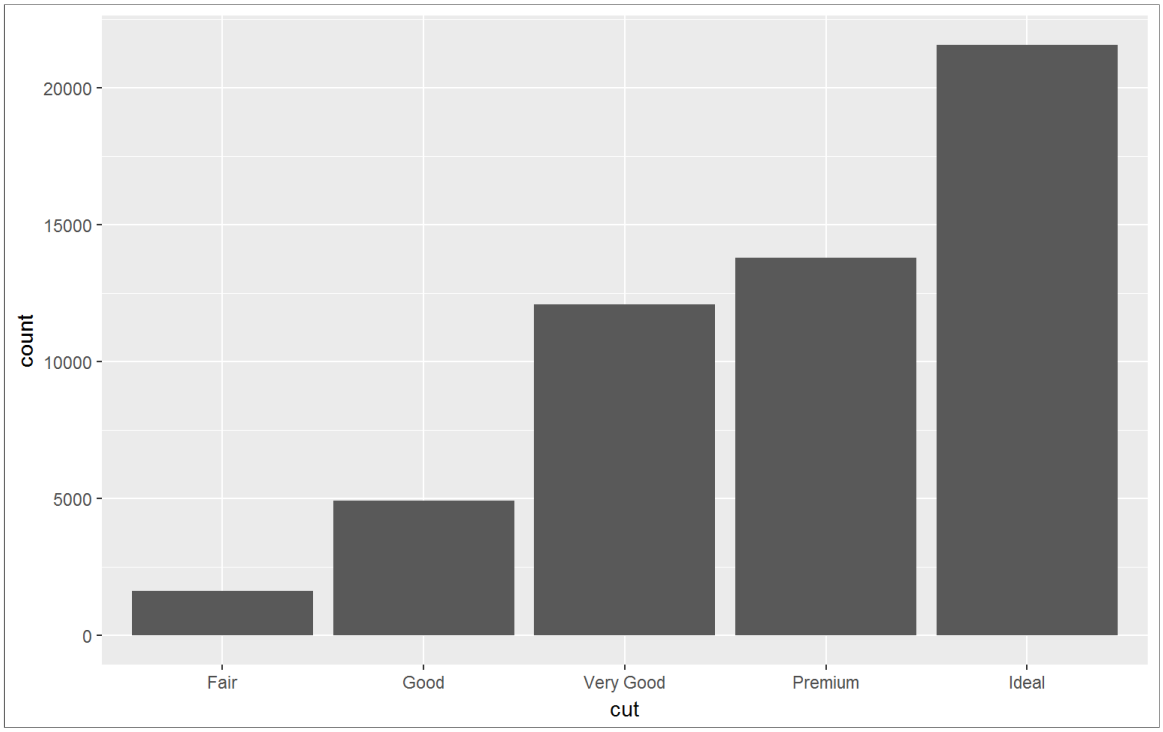
```
diamonds
```

```
## # A tibble: 53,940 × 10
##    carat       cut color clarity depth table price     x     y     z
##    <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1   0.23     Ideal     E     SI2  61.5    55   326  3.95  3.98  2.43
## 2   0.21   Premium     E     SI1  59.8    61   326  3.89  3.84  2.31
## 3   0.23      Good     E     VS1  56.9    65   327  4.05  4.07  2.31
## 4   0.29   Premium     I     VS2  62.4    58   334  4.20  4.23  2.63
## 5   0.31      Good     J     SI2  63.3    58   335  4.34  4.35  2.75
## 6   0.24 Very Good     J    VVS2  62.8    57   336  3.94  3.96  2.48
## 7   0.24 Very Good     I    VVS1  62.3    57   336  3.95  3.98  2.47
## 8   0.26 Very Good     H     SI1  61.9    55   337  4.07  4.11  2.53
## 9   0.22      Fair     E     VS2  65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good     H     VS1  59.4    61   338  4.00  4.05  2.39
## # ... with 53,930 more rows
```

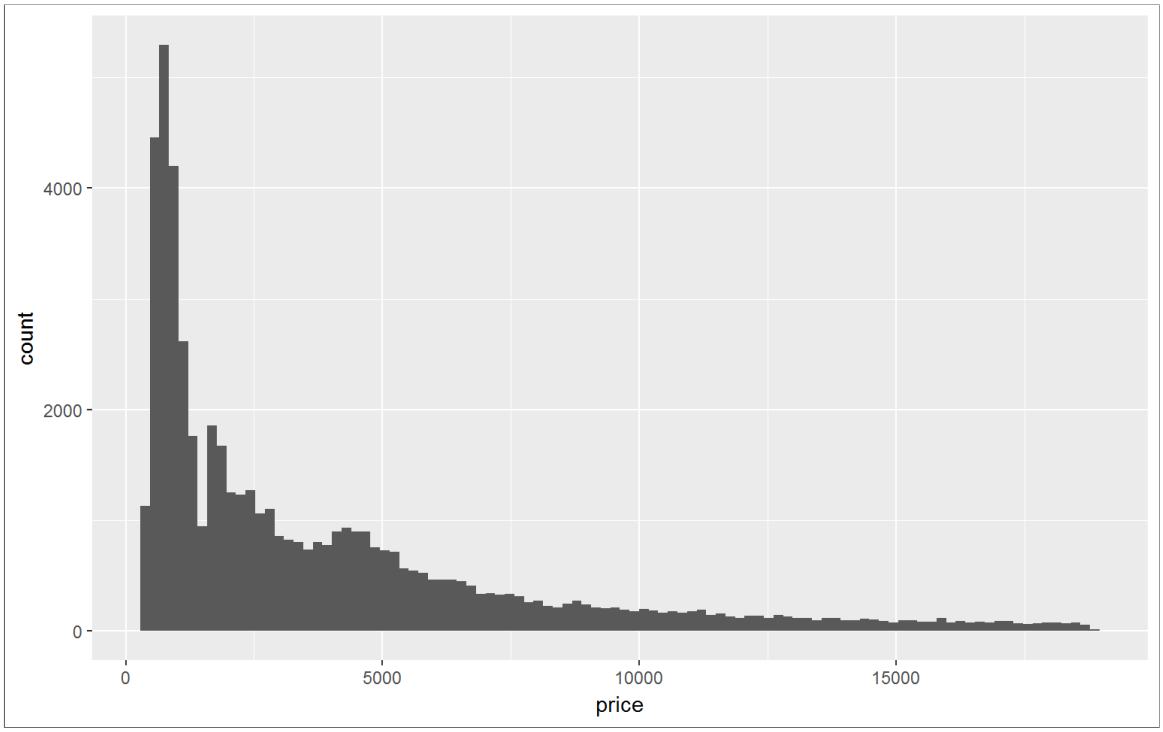
```
summary(diamonds)
```

```
##      carat               cut        color        clarity          depth           table
##  Min.   :0.2000   Fair      : 1610   D: 6775   SI1     :13065   Min.   :43.00   Min.   :43.
##  1st Qu.:0.4000   Good      : 4906   E: 9797   VS2     :12258   1st Qu.:61.00   1st Qu.:56.
##  Median :0.7000   Very Good:12082   F: 9542   SI2     : 9194   Median :61.80   Median :57.
##  Mean   :0.7979   Premium   :13791   G:11292   VS1     : 8171   Mean   :61.75   Mean   :57.
##  3rd Qu.:1.0400   Ideal     :21551   H: 8304   VVS2    : 5066   3rd Qu.:62.50   3rd Qu.:59.
##  Max.   :5.0100                     I: 5422   VVS1    : 3655   Max.   :79.00   Max.   :95.
##                                     J: 2808   (Other): 2531
##        x                y               z
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
##  Median : 5.700   Median : 5.710   Median : 3.530
##  Mean   : 5.731   Mean   : 5.735   Mean   : 3.539
##  3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
##  Max.   :10.740   Max.   :58.900   Max.   :31.800
##
```

```
ggplot(diamonds, aes(cut)) + geom_bar()
```



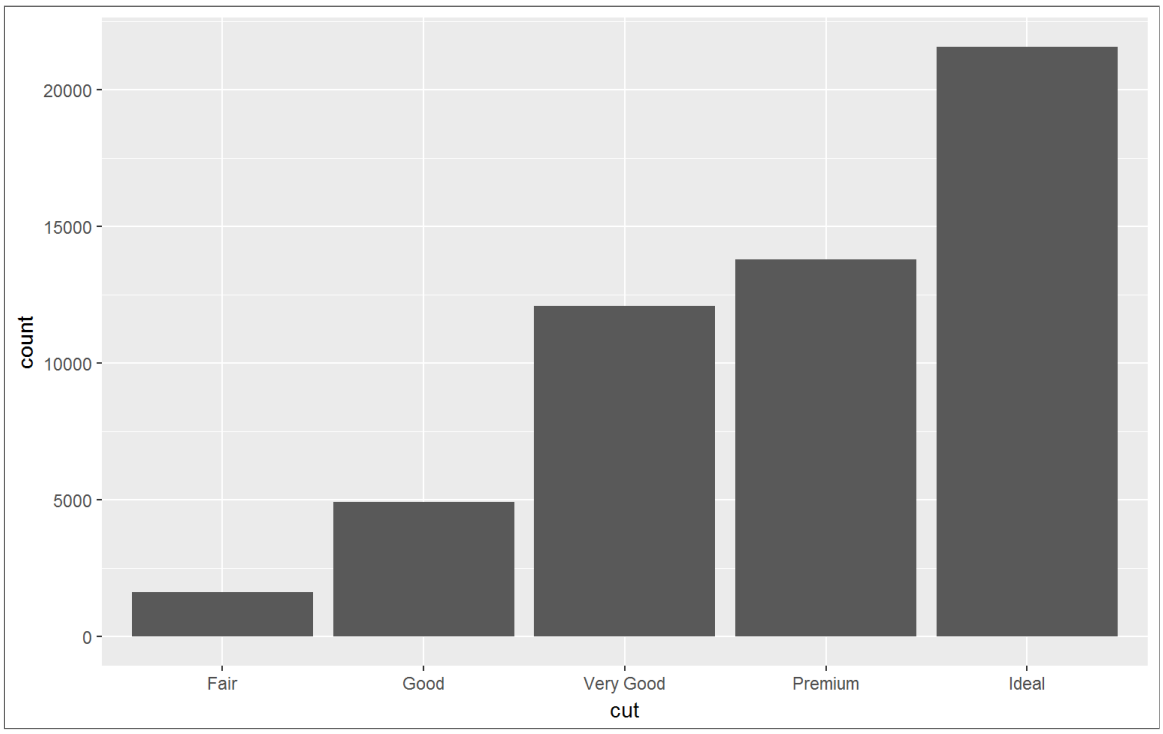

```
ggplot(diamonds, aes(price)) + geom_histogram(bins = 100)
```



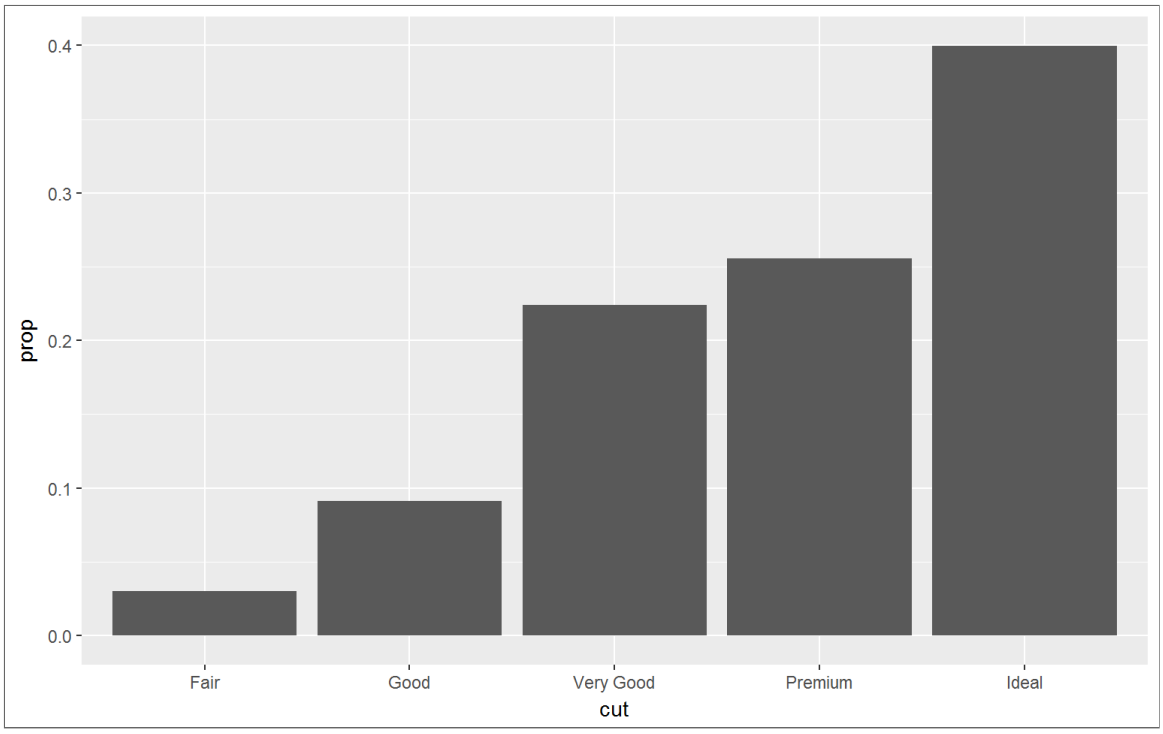
COMPUTED VARIABLES

- Bar charts, histograms and the other plots in the one variable section of the ggplot2 cheat sheet bin your data based on a single variable
- You can determine the computed variables of a graphic by using the help function:
    - Computed variables
        - count
            - Number of points in bin
        - prop
            - Groupwise proportion

```
ggplot(diamonds) + geom_bar(aes(x = cut, y = ..count.., group = 1))
```

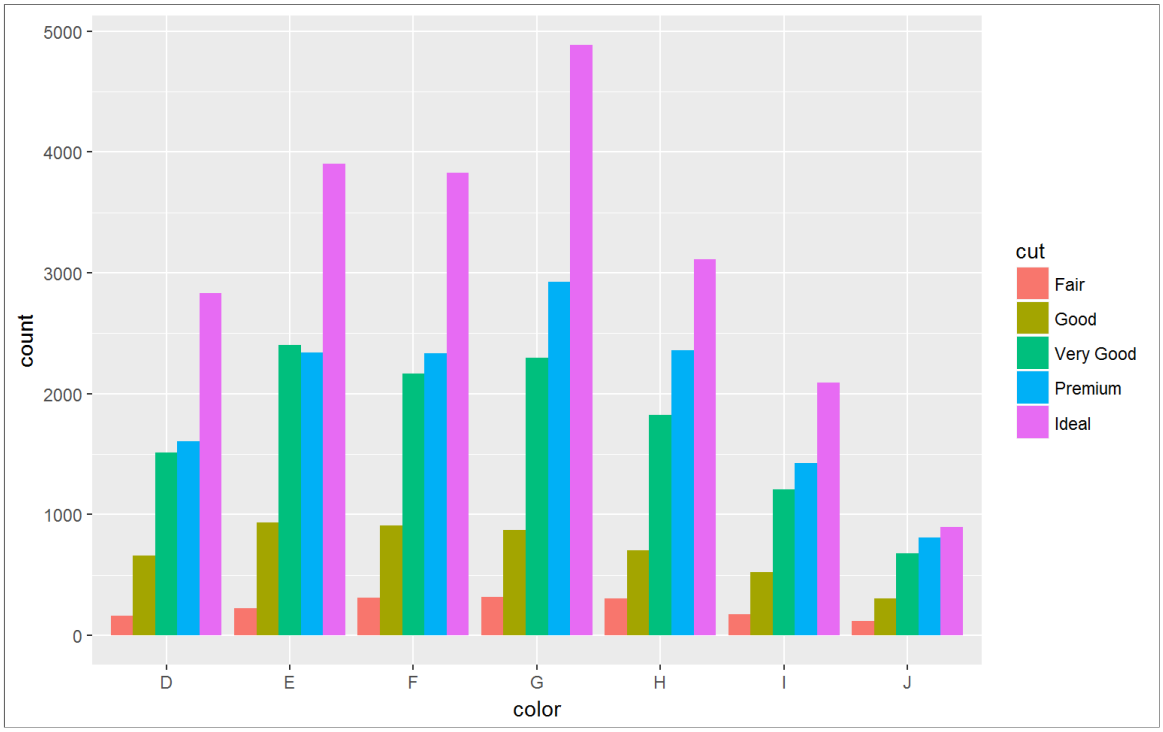


```
ggplot(diamonds) + geom_bar(aes(x = cut, y = ..prop.., group = 1))
```

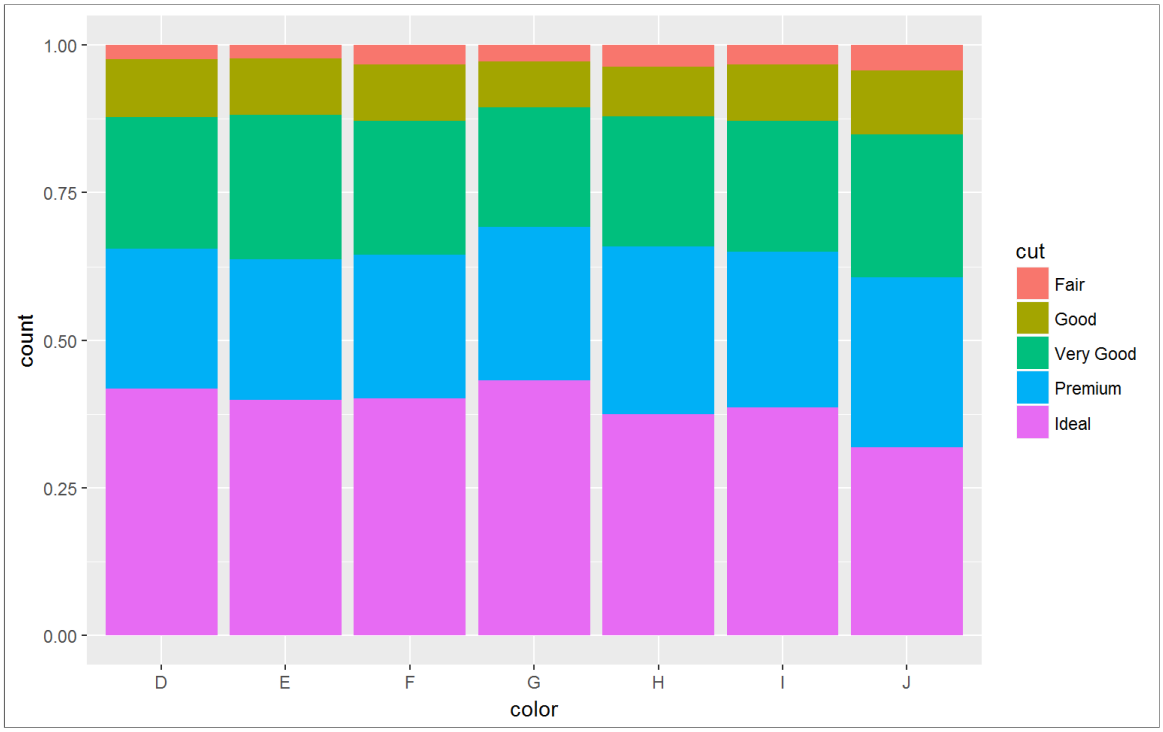


POSITION ADJUSTMENTS

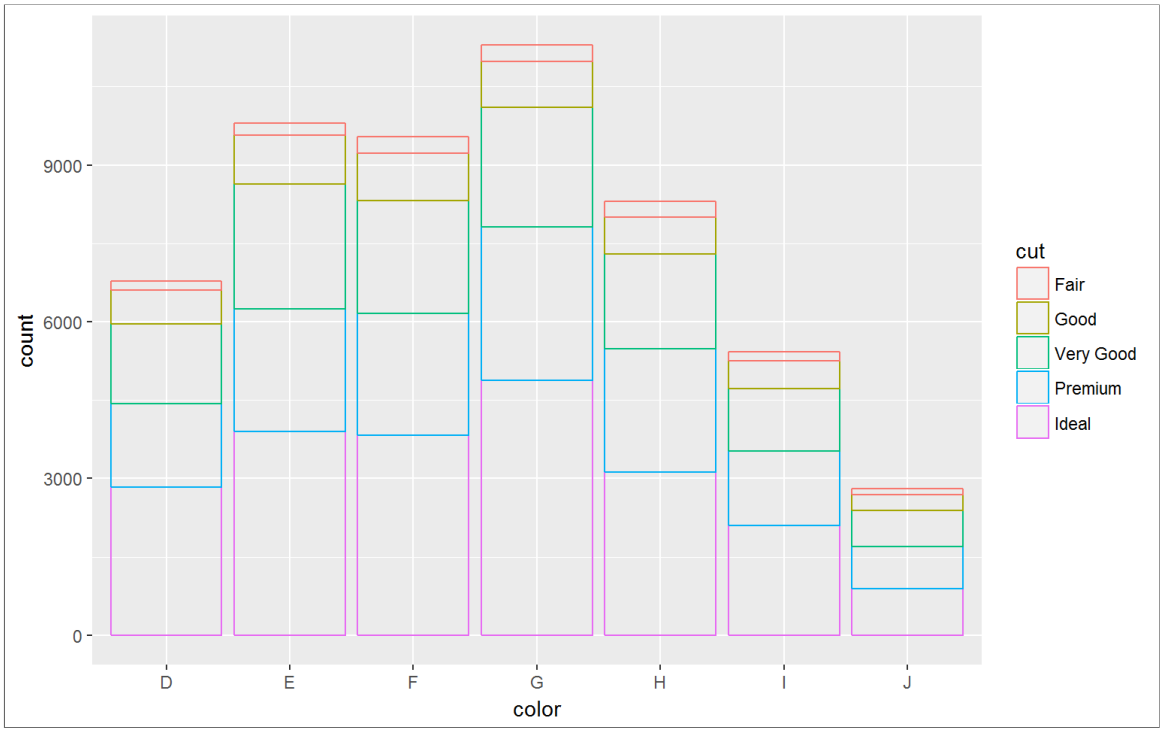
```
ggplot(diamonds) + geom_bar(aes(x = color, fill = cut), position = "dodge")
```




```
ggplot(diamonds) + geom_bar(aes(x = color, fill = cut), position = "fill")
```



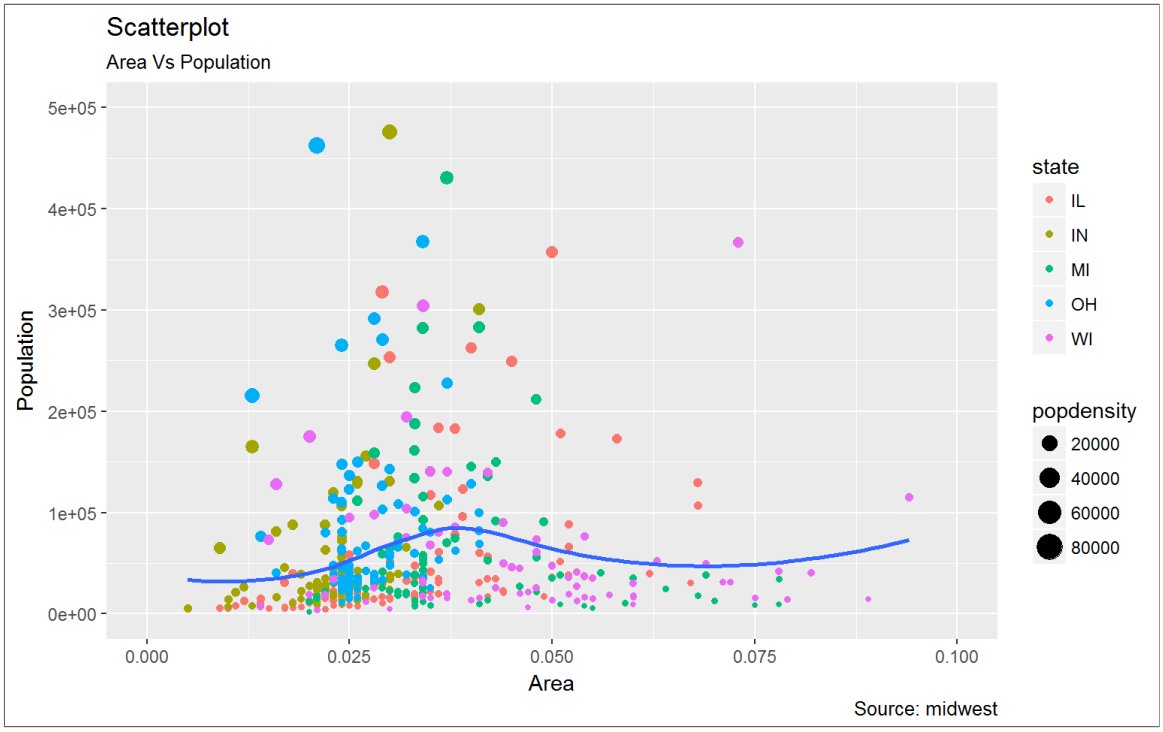

```r
ggplot(diamonds) + geom_bar(aes(x = color, color = cut), position = "stack", fill = NA)
```



EXAMPLE PLOTS

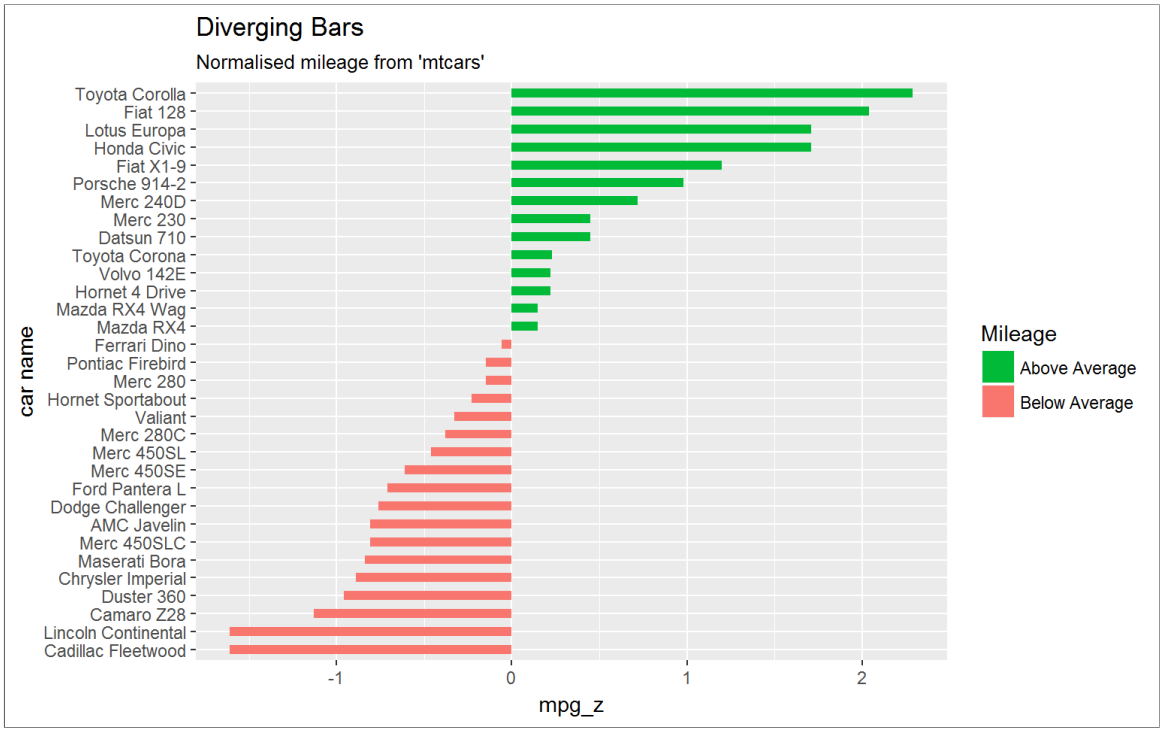
plots from **http://r-statistics.co/Top50-Ggplot2-Visualizations-MasterList-R-Code.html**

```
ggplot(midwest, aes(x=area, y=poptotal)) +
  geom_point(aes(col=state, size=popdensity)) +
  geom_smooth(method="loess", se=F) + xlim(c(0, 0.1)) + ylim(c(0, 500000)) +
  labs(subtitle="Area Vs Population", y="Population", x="Area", title="Scatterplot", caption
```



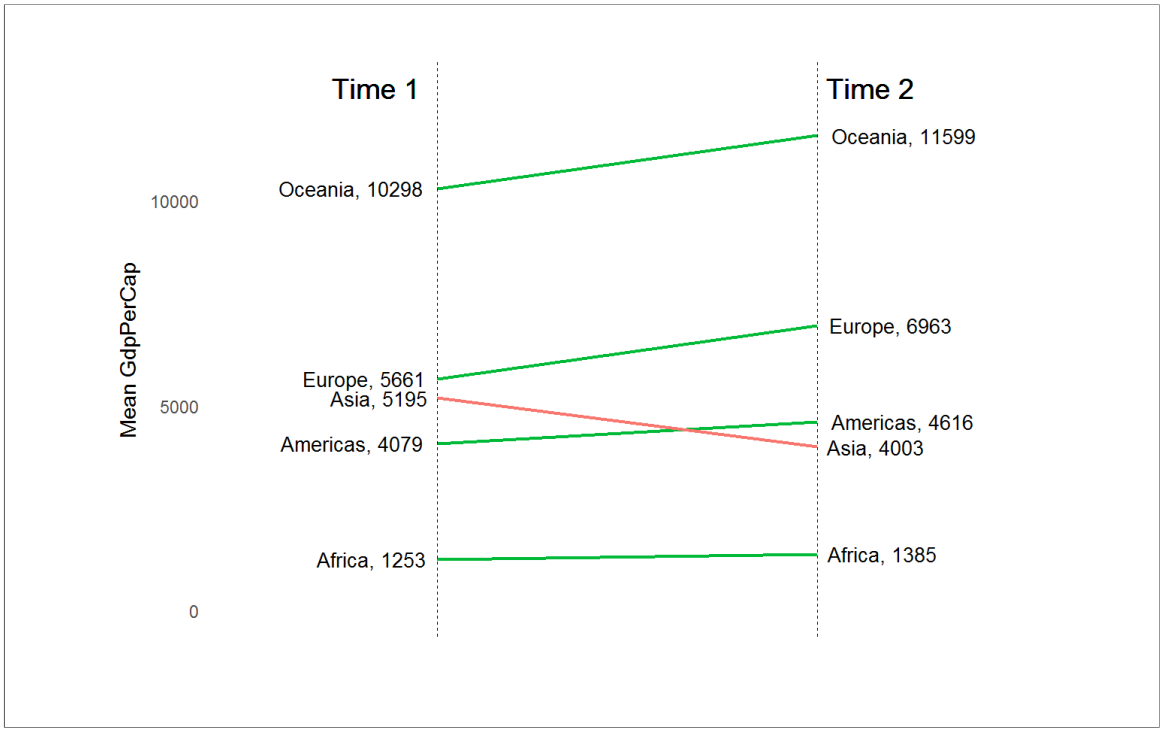

```
mtcars$`car name` <- rownames(mtcars)  # create new column for car names
mtcars$mpg_z <- round((mtcars$mpg - mean(mtcars$mpg))/sd(mtcars$mpg), 2)  # compute normaliz
mtcars$mpg_type <- ifelse(mtcars$mpg_z < 0, "below", "above")  # above / below avg flag
mtcars <- mtcars[order(mtcars$mpg_z), ]  # sort
mtcars$`car name` <- factor(mtcars$`car name`, levels = mtcars$`car name`)  # convert to fac

ggplot(mtcars, aes(x=`car name`, y=mpg_z, label=mpg_z)) +
  geom_bar(stat='identity', aes(fill=mpg_type), width=.5)  +
  scale_fill_manual(name="Mileage",
                    labels = c("Above Average", "Below Average"),
                    values = c("above"="#00ba38", "below"="#f8766d")) +
  labs(subtitle="Normalised mileage from 'mtcars'", title= "Diverging Bars") +
  coord_flip()
```

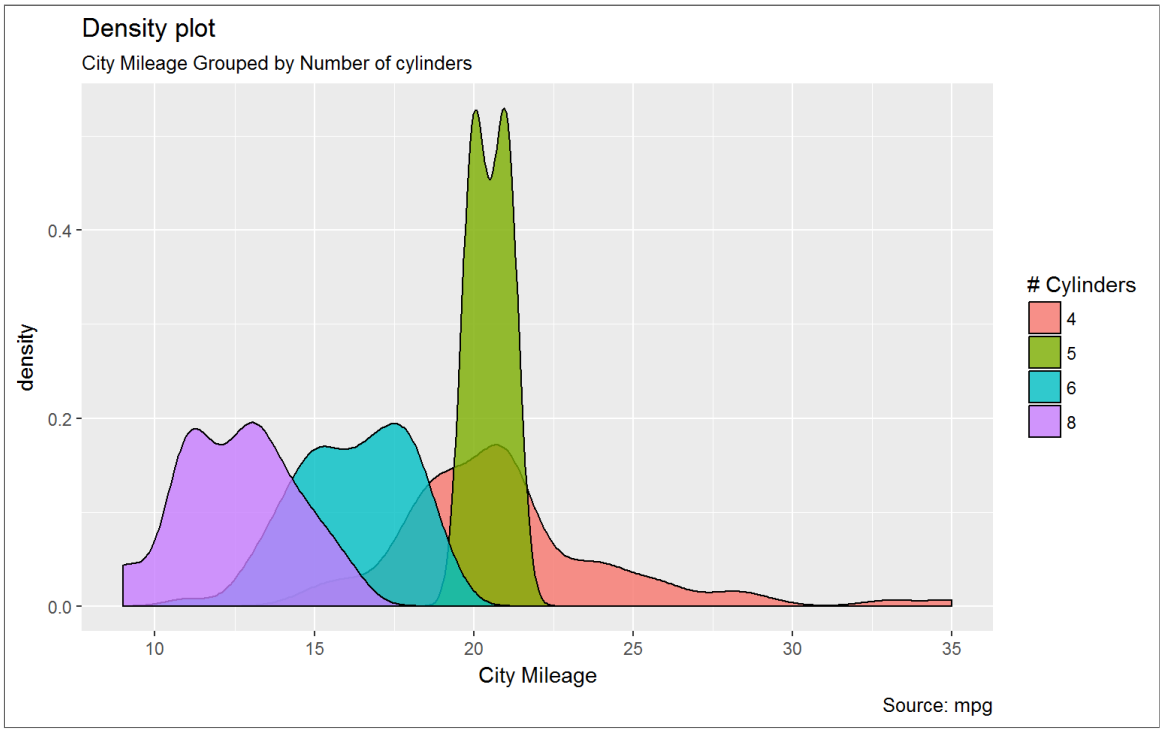



```r
# prep data
df <- read.csv("https://raw.githubusercontent.com/selva86/datasets/master/gdppercap.csv")
colnames(df) <- c("continent", "1952", "1957")
left_label <- paste(df$continent, round(df$`1952`),sep=", ")
right_label <- paste(df$continent, round(df$`1957`),sep=", ")
df$class <- ifelse((df$`1957` - df$`1952`) < 0, "red", "green")

ggplot(df) + geom_segment(aes(x=1, xend=2, y=`1952`, yend=`1957`, col=class), size=.75, show
  geom_vline(xintercept=1, linetype="dashed", size=.1) +
  geom_vline(xintercept=2, linetype="dashed", size=.1) +
  scale_color_manual(labels = c("Up", "Down"),
                     values = c("green"="#00ba38", "red"="#f8766d")) +  # color of lines
  labs(x="", y="Mean GdpPerCap") +  # Axis labels
  xlim(.5, 2.5) + ylim(0,(1.1*(max(df$`1952`, df$`1957`)))) + # X and Y axis limits
  geom_text(label=left_label, y=df$`1952`, x=rep(1, NROW(df)), hjust=1.1, size=3.5) +
  geom_text(label=right_label, y=df$`1957`, x=rep(2, NROW(df)), hjust=-0.1, size=3.5) +
  geom_text(label="Time 1", x=1, y=1.1*(max(df$`1952`, df$`1957`)), hjust=1.2, size=5) +  #
  geom_text(label="Time 2", x=2, y=1.1*(max(df$`1952`, df$`1957`)), hjust=-0.1, size=5) + #
  theme(panel.background = element_blank(),panel.grid = element_blank(),axis.ticks = element
        axis.text.x = element_blank(),panel.border = element_blank(),plot.margin = unit(c(
```

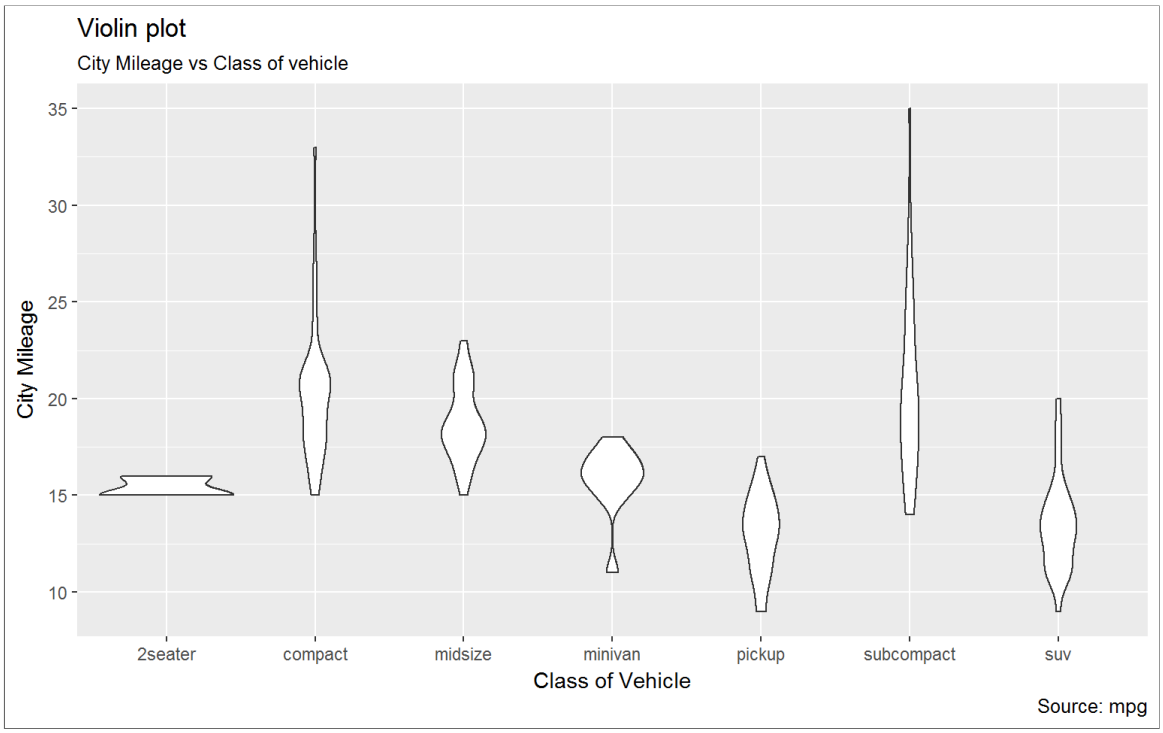



```
ggplot(mpg, aes(cty)) +
  geom_density(aes(fill=factor(cyl)), alpha=0.8) +
  labs(title="Density plot",
       subtitle="City Mileage Grouped by Number of cylinders",
       caption="Source: mpg",
       x="City Mileage",
       fill="# Cylinders")
```

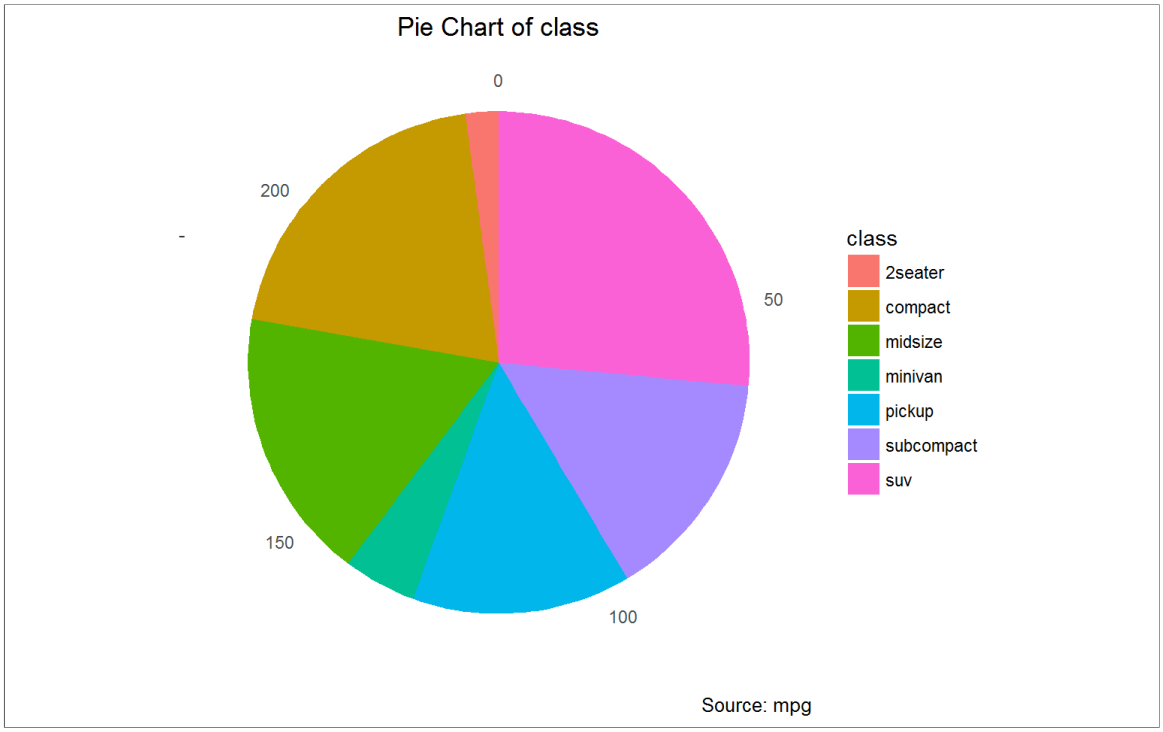



```
ggplot(mpg, aes(class, cty)) +
  geom_violin() +
```

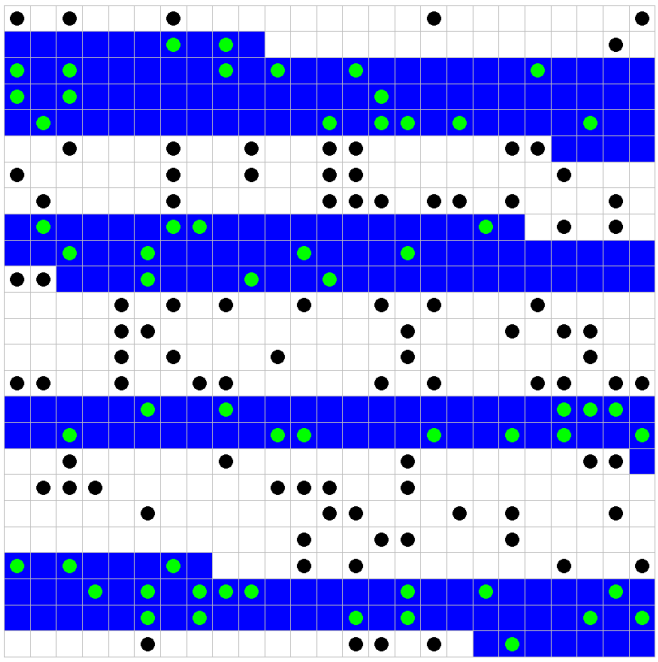
```
  labs(title="Violin plot",
       subtitle="City Mileage vs Class of vehicle",
       caption="Source: mpg",
       x="Class of Vehicle",
       y="City Mileage")
```



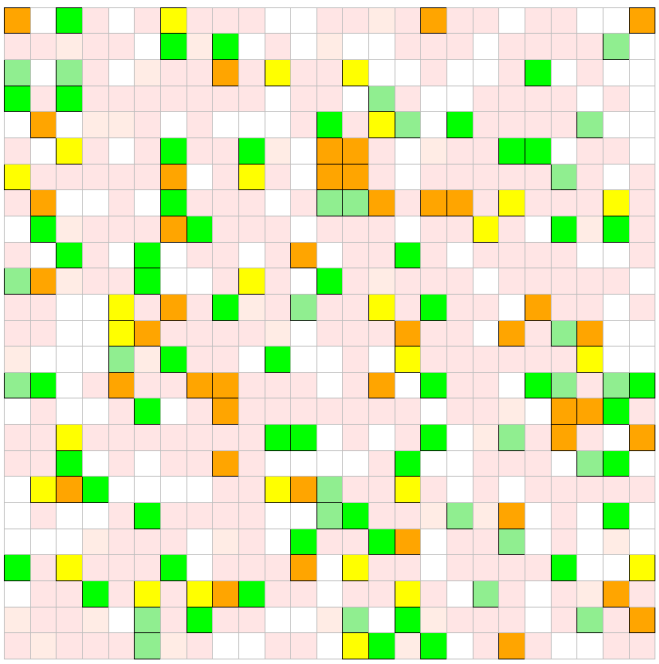

```
theme_set(theme_classic())
df <- as.data.frame(table(mpg$class))
colnames(df) <- c("class", "freq")
ggplot(df, aes(x = "", y=freq, fill = factor(class))) +
  geom_bar(width = 1, stat = "identity") +
  theme(axis.line = element_blank(),
        plot.title = element_text(hjust=0.5)) +
  labs(fill="class",
       x=NULL,
       y=NULL,
       title="Pie Chart of class",
       caption="Source: mpg") +
  coord_polar(theta = "y", start=0)
```




```
EncSz  <-  25
SynPermCon  <-  0.5
PtPrcnt <- 0.75
SPSmpSz <- round(EncSz^2*PtPrcnt)
ENC <- rep(.3,EncSz^2)
ENC[c(19:83,200:250,353:420,497:585)] <- 1
SPEncBoxes <- tibble(x = rep(c(1:EncSz),EncSz), y = sort(rep(c(1:EncSz),EncSz)))
j <- rep(NA,EncSz^2)
j[sample(EncSz^2,SPSmpSz)] <- rnorm(SPSmpSz,mean=.9*SynPermCon,sd=SynPermCon/5)
j2 <- rep(NA,EncSz^2)
j2[j>0.5] <- 1
j2[j>0.5 & ENC ==1] <- 2
j2[is.na(j)] <- NA
EncAct <- rep(0.1,EncSz^2)
EncAct[j>SynPermCon] <- 1
j <- cut(j,breaks = c(-Inf,seq(0.4,0.6,0.025),Inf))
BlnkGrph = theme(axis.line=element_blank(), axis.text.x=element_blank(), axis.text.y=element
                 axis.title.y=element_blank(), legend.position="none", panel.background=elem
                 panel.grid.minor=element_blank(), plot.background=element_blank(),plot.marg
SPEncBoxes %>% ggplot(aes(x,y,fill = factor(round(ENC)))) +
  geom_tile(color = "gray",show.legend=FALSE) + BlnkGrph + coord_fixed() +
  geom_point(aes(x,y, color = factor(j2)),shape = 16,na.rm=TRUE, size = 3) +
  scale_fill_manual(values = c("white","blue")) + scale_shape_identity() +
  scale_color_manual(values = c("black","green"))
```



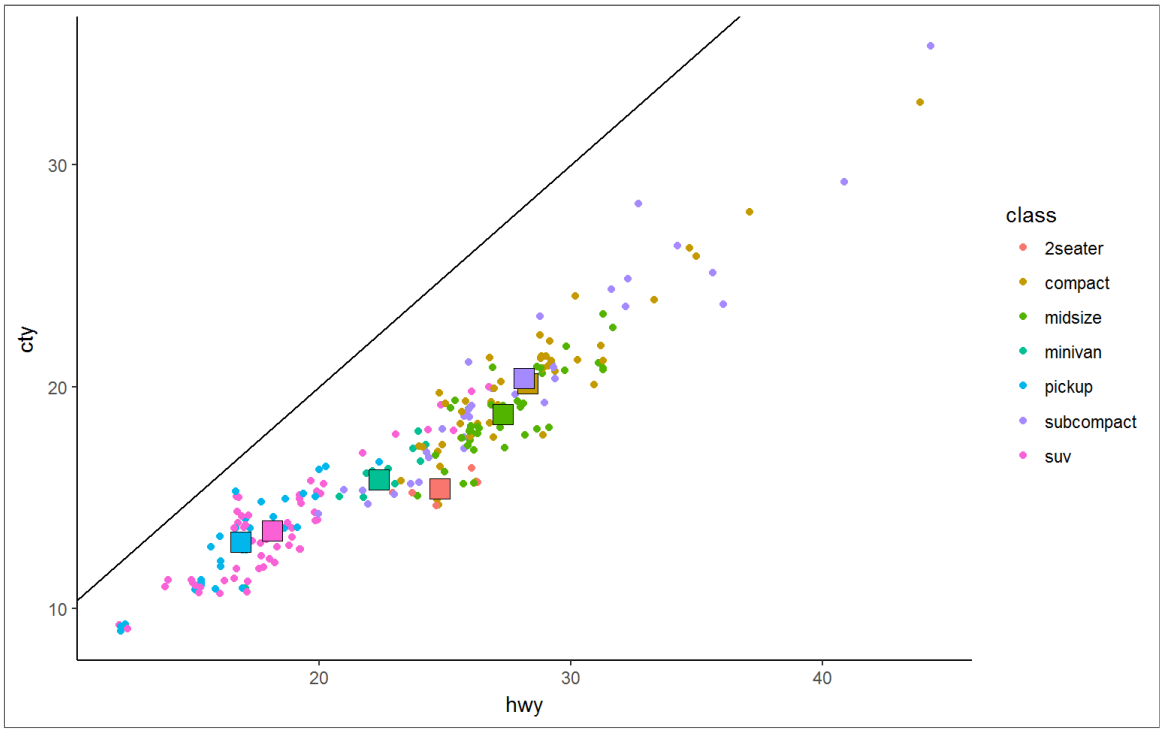
```
SPEncBoxes %>% ggplot(aes(x,y,fill = j, color = EncAct)) +
  geom_tile(show.legend=FALSE, size = 0.2,alpha=EncAct) + BlnkGrph + coord_fixed() +
  scale_color_gradient(low="gray",high ="black") +
  scale_fill_manual(values = c("red","red","red","red","orangered", "orange","yellow","light
```



SAVING PLOTS

- `ggsave(filename)` allows you to save plots
- It guesses the filetype based on the extension used in the `filename` parameter
    - You can manually set the extension using the `device` parameter
    - options are: 'eps', 'ps', 'tex', 'pdf', 'jpeg/jpg', 'tiff', 'png', 'bmp', 'svg', 'wmf' (windows only)
- By default it saves the last plot displayed unless you change the `plot` parameter

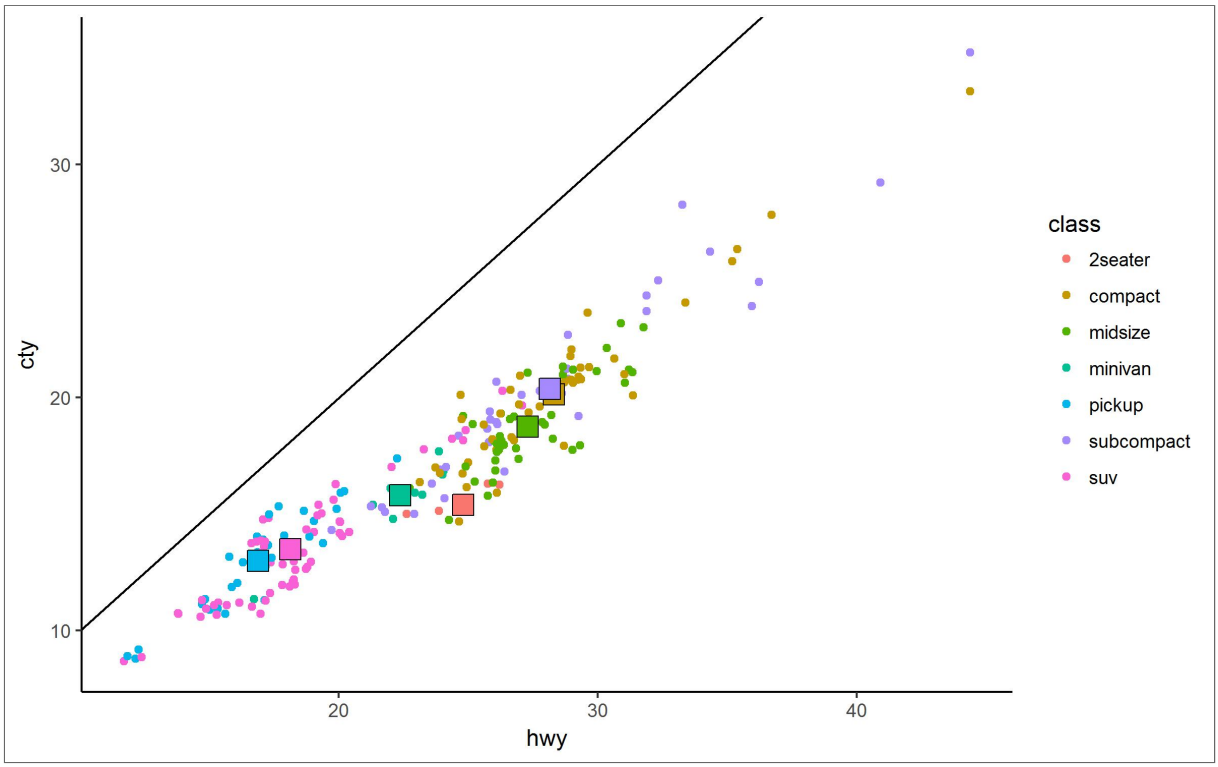
```
ggplot(mpg,aes(x=hwy,y=cty)) +
  geom_blank() +
  geom_abline() +
  geom_jitter(aes(color=class)) +
  geom_point(data=mpg_class,aes(x=hwy_mean,y=cty_mean,fill=class),color="black",size=5,shape
```




```
ggsave("./images/cars.jpg")
```

```
## Saving 8 x 5 in image
```

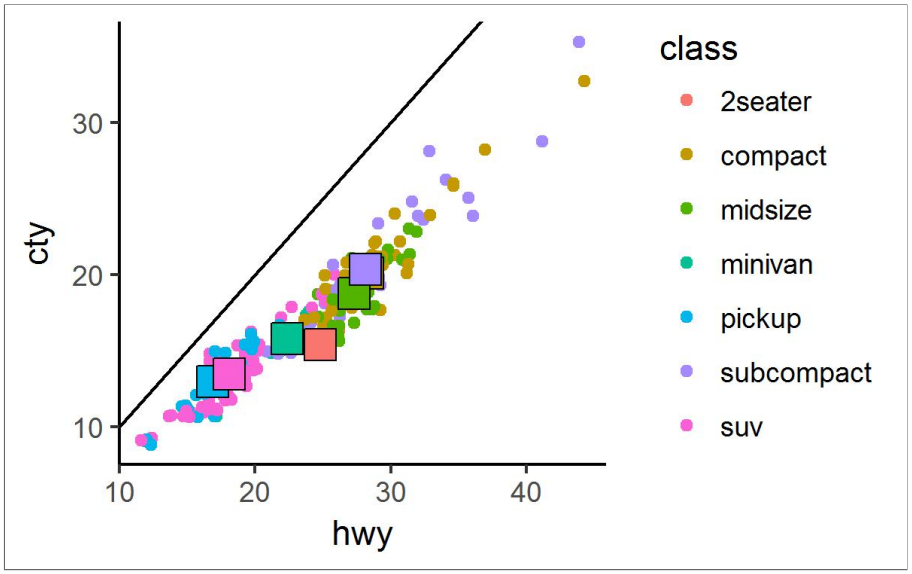
```
knitr::include_graphics("./images/cars.jpg")
```



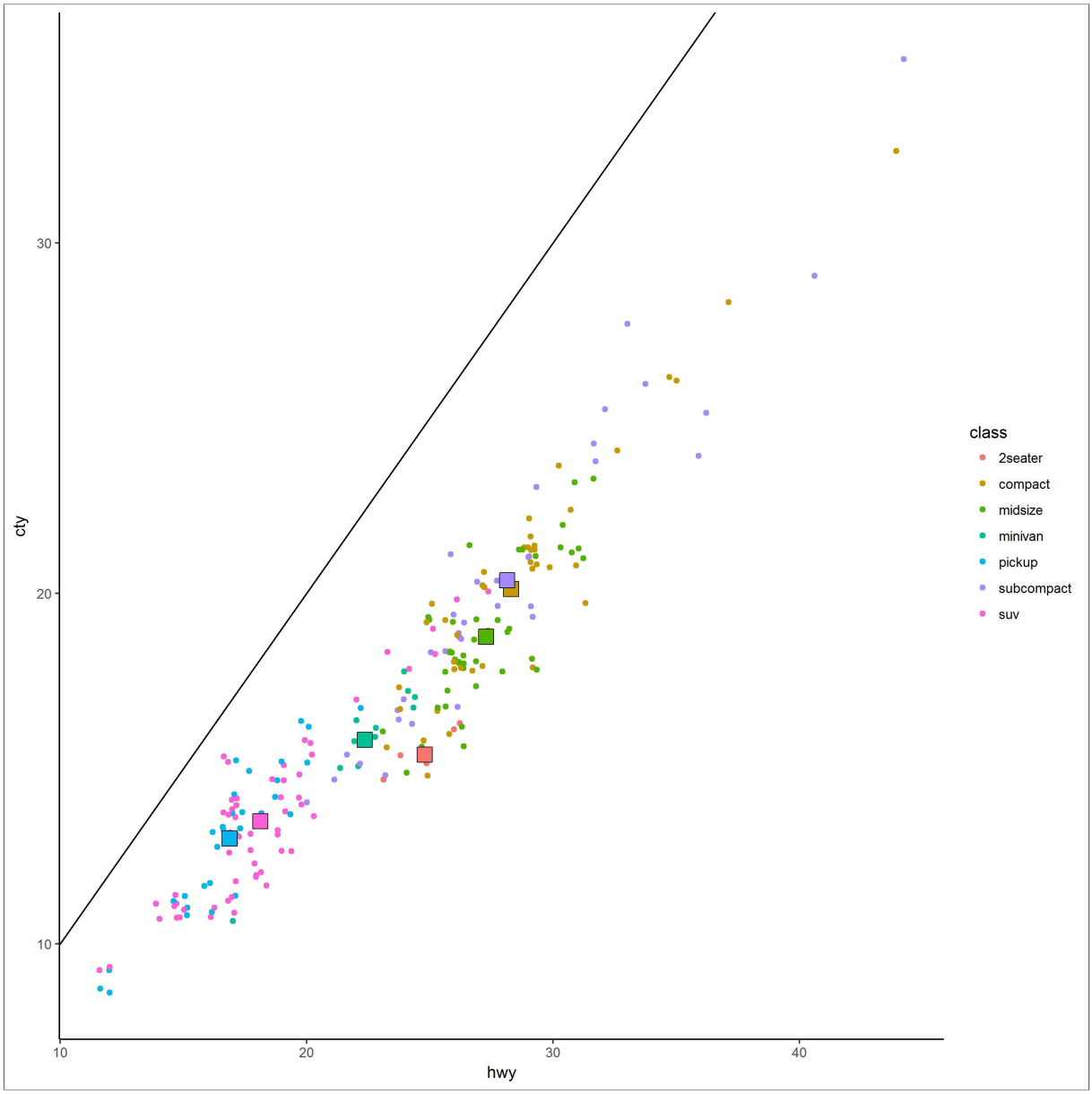

```
ggsave("./images/cars_halfscale.jpg",scale=0.5)
```

```
## Saving 4 x 2.5 in image
```

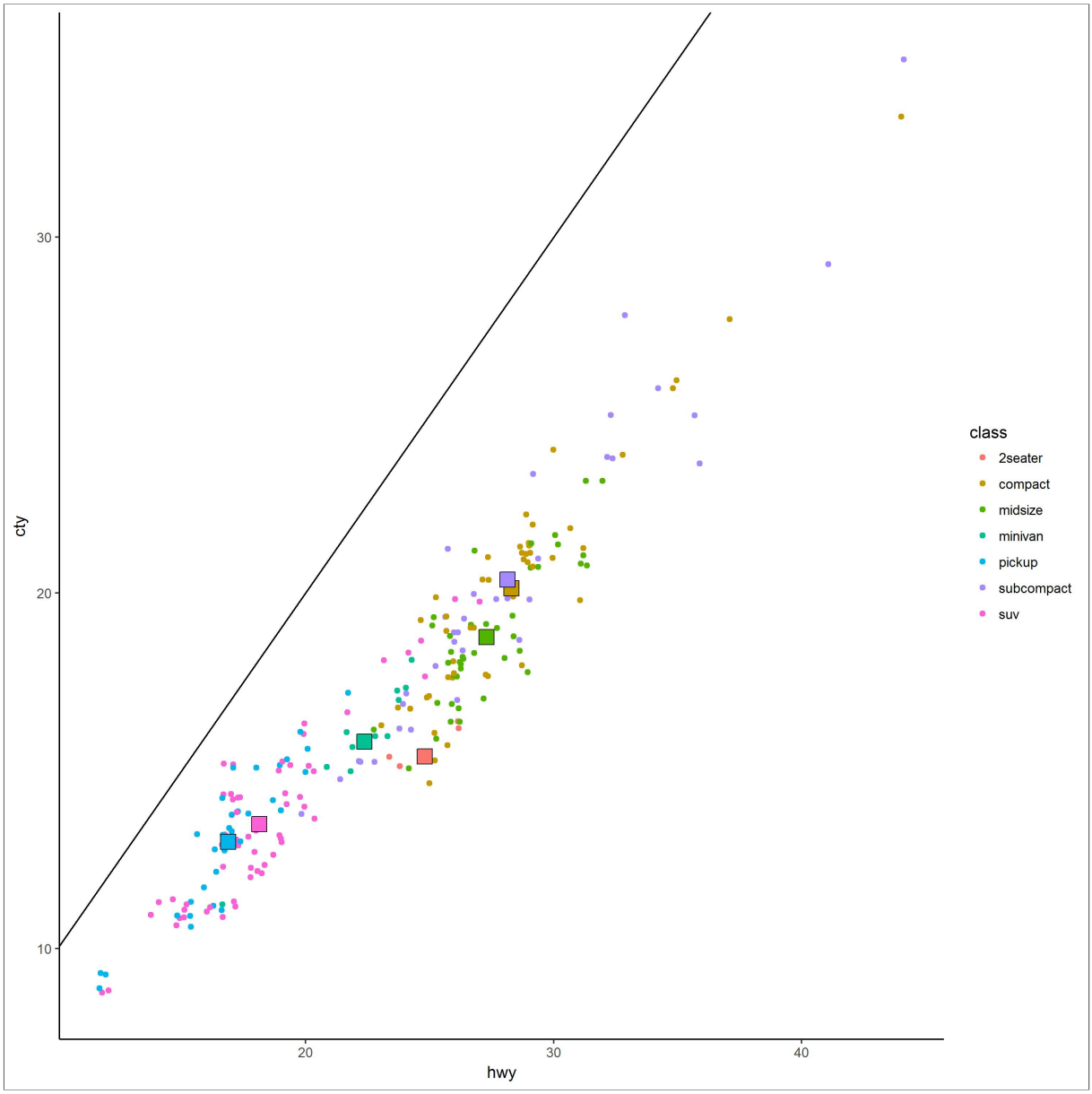
```
knitr::include_graphics("./images/cars_halfscale.jpg")
```



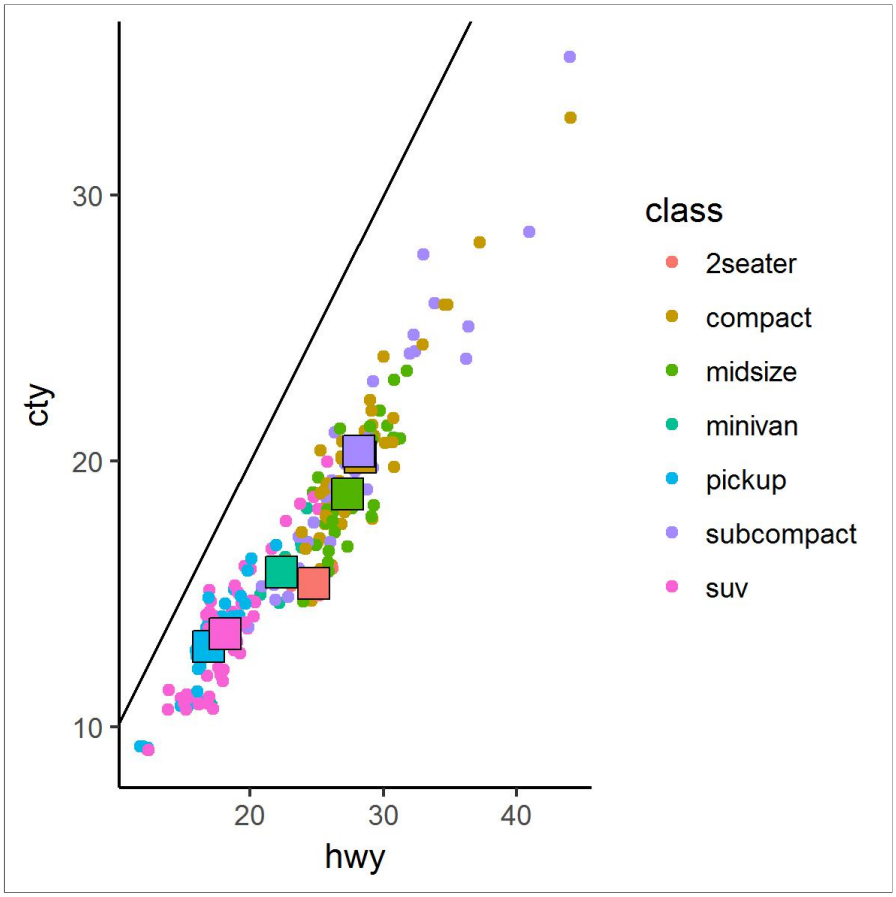

```
ggsave("./images/cars_w10h10.jpg",width=10,height=10)
knitr::include_graphics("./images/cars_w10h10.jpg")
```




```
ggsave("./images/cars_w10h10in.jpg",width=10,height=10,units="in")
knitr::include_graphics("./images/cars_w10h10in.jpg")
```




```
ggsave("./images/cars_w10h10cm.jpg",width=10,height=10,units="cm")
knitr::include_graphics("./images/cars_w10h10cm.jpg")
```


cty
hwy
30
20
10
20
30
40
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

SPEND SOME TIME PLAYING WITH GGPLOT

- See **http://ggplot2.tidyverse.org/reference/** for a full list of functions in ggplot
- Here are a list of the datasets built into ggplot (use ?"dataset name" to find out more about the individual datasets)
    - diamonds
        - Prices of 50,000 round cut diamonds
    - economics
        - US economic time series
    - faithfuld
        - 2d density estimate of Old Faithful data
    - midwest
        - Midwest demographics
    - mpg
        - Fuel economy data from 1999 and 2008 for 38 popular models of car
    - msleep
        - An updated and expanded version of the mammals sleep dataset
    - presidential
        - Terms of 11 presidents from Eisenhower to Obama
    - seals
        - Vector field of seal movements

- txhousing
  - Housing sales in TX
- luv_colours
  - colors() in Luv space

SECTION 3 - DATA MANIPULATION

MATHEMATICAL AND BOOLEAN OPERATORS

```
-sqrt(25) + (5 + 3)/4 * 7 - 2^2
```

```
## [1] 5
```

```
5%/%3  # Integer Division
```

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

```
5%%3  # Modulo (remainder after division)
```

```
## [1] 2
```

```
5 == 6
```

```
## [1] FALSE
```

```
5 != 6
```

```
## [1] TRUE
```

```
83 > (25 >= 23)
```

```
## [1] TRUE
```

```r
5 > 3 & 3 < 2
```

```
## [1] FALSE
```

```r
5 > 3 | 3 < 2
```

```
## [1] TRUE
```

VECTORS AND SEQUENCES

```
1:4
```

```
## [1] 1 2 3 4
```

```
c(5, 3, 2, 1)  # Creates a vector via concatenation (hence the c)
```

```
## [1] 5 3 2 1
```

```
c(12, 1:4, 6)
```

```
## [1] 12  1  2  3  4  6
```

```
seq(from = 1, t = 10, by = 2)  # Creates a vector with the given paramters
```

```
## [1] 1 3 5 7 9
```

```
seq(1, 10, 2)  # creates the same vector without naming the paramters
```

```
## [1] 1 3 5 7 9
```

```
seq(1, 10)  # R uses the default values for any empty parameters
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
seq(to = 10, by = 2)
```

```
## [1] 1 3 5 7 9
```

```r
seq(by = 2, to = 10)
```

```
## [1] 1 3 5 7 9
```

```r
c(seq(1, 10, 2), 25, 10)
```

```
## [1]  1  3  5  7  9 25 10
```

```r
c(seq(1, 10, 2), 25, 10) > 12
```

```
## [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
```

```r
c(seq(1, 10, 2), 25, 10) * 2
```

```
## [1]  2  6 10 14 18 50 20
```

VARIABLES AND ASSIGNMENT

```
x = 5 + 3
(x = 5 + 3)
```

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

```
x <- 5 + 3
(x <- 5 + 3)
```

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

- Terminology
  - The term on the left hand side is referred to as an object
  - The term on the right hand side is the object's value
  - The <- is the assignment operator

```
y <- x
y
```

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

```
x <- 5 + 3 > 2
x
```

```
## [1] TRUE
```

```
x <- seq(172, 23, -13)
x
```

```
##  [1] 172 159 146 133 120 107  94  81  68  55  42  29
```

INDEXING

- Indexing is locating values by their index/indices within an N dimensional object
- Unlike some other programming languages R starts at 1
- Note that indexing uses [], while functions, like `seq`, use ()

```
x <- seq(172, 23, -13)
x[1]
```

```
## [1] 172
```

```
x[c(1, 3)]
```

```
## [1] 172 146
```

```
x[2:4]
```

```
## [1] 159 146 133
```

```
x[4:2]
```

```
## [1] 133 146 159
```

```
x[]
```

```
##  [1] 172 159 146 133 120 107  94  81  68  55  42  29
```

```
x[-1]
```

```
##  [1] 159 146 133 120 107  94  81  68  55  42  29
```

```
x[-c(1, 3)]
```

```
##  [1] 159 133 120 107  94  81  68  55  42  29
```

```
x[x%%2 == 0]
```

```
## [1] 172 146 120  94  68  42
```

```
y <- x[x%%2 == 0]
y[9] <- 10
y
```

```
## [1] 172 146 120  94  68  42  NA  NA  10
```

BUILT IN FUNCTIONS

```
x <- 1:20
mean(x)
```

```
## [1] 10.5
```

```
max(x)
```

```
## [1] 20
```

```
min(x)
```

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

```
length(x)
```

```
## [1] 20
```

```
range(x)
```

```
## [1]  1 20
```

```
prod(x)
```

```
## [1] 2.432902e+18
```

```
var(x)
```

```
## [1] 35
```

```
log(x)
```

```
##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101 2.0794415 2.19
## [12] 2.4849066 2.5649494 2.6390573 2.7080502 2.7725887 2.8332133 2.8903718 2.9444390 2.99
```

```
sqrt(x)
```

```
##  [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427 3.000000 3.1
## [13] 3.605551 3.741657 3.872983 4.000000 4.123106 4.242641 4.358899 4.472136
```

Note that many functions in R have default values for some of their parameters and you should always try to be aware of them even if you don't want to change them

```
rnorm(10)
```

```
##  [1]  0.86403310  0.36151193 -0.99675278 -0.96825363  0.76196346  1.33340652 -0.02976676
## [10]  0.16586029
```

`rnorm` randomly generates values from a normal distribution, but a normal distribution requires a mean and a standard deviation. If you type `?rnorm` you will see the full documentation but for our purposes the important part is

rnorm(n, mean = 0, sd = 1)

By default, the `rnorm` function assumes a mean of 0 and a standard deviation of 1. You can change those values easily, but only if you are aware of them.

EXERCISES

1. Create a vector of 2 through 8 squared:
    - 4, 9, 16, 25, 36, 49, 64
2. Create a vector of the square roots of the sum of sqaures of every pair of digits of 1 to 100:
    - sqrt(1^2 + 2^2), sqrt(3^2 + 4^2), sqrt(5^2 + 6^2), … , sqrt(99^2 + 100^2)
3. Create a vector of the numbers 1 to 100 not divisible by 3 or 5:
    - 1, 2, 4, 7, 8, 11, 13, 14, 16, 17, … , 97, 98

DATA FRAMES

When you have data consisting of multiple observations of multiple variables, i.e., a data set, this is most conveniently stored as a dataframe

```
iris  # Famous iris data set which gives the measurements for 50 flowers from each of 3 spec
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
## 14           4.3         3.0          1.1         0.1     setosa
## 15           5.8         4.0          1.2         0.2     setosa
## 16           5.7         4.4          1.5         0.4     setosa
## 17           5.4         3.9          1.3         0.4     setosa
## 18           5.1         3.5          1.4         0.3     setosa
## 19           5.7         3.8          1.7         0.3     setosa
## 20           5.1         3.8          1.5         0.3     setosa
## 21           5.4         3.4          1.7         0.2     setosa
## 22           5.1         3.7          1.5         0.4     setosa
## 23           4.6         3.6          1.0         0.2     setosa
## 24           5.1         3.3          1.7         0.5     setosa
```

```
summary(iris)  # Very useful function, which gives summaries of each variable
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

MATCHING

- You can call a single variable fo a dataframe using any of the following formats
  - `dataset$variablename`
  - `dataset['variablename']`
  - `dataset[variable column position]`

```
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
iris$Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1 5.7 5.1 5.4
##  [29] 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3
##  [57] 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8
##  [85] 5.4 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5
## [113] 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9 6.4
## [141] 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

```
iris["Sepal.Length"]
```

```
##     Sepal.Length
## 1            5.1
## 2            4.9
## 3            4.7
## 4            4.6
## 5            5.0
## 6            5.4
## 7            4.6
## 8            5.0
## 9            4.4
## 10           4.9
## 11           5.4
## 12           4.8
## 13           4.8
## 14           4.3
## 15           5.8
```

```
## 16           5.7
## 17           5.4
## 18           5.1
## 19           5.7
## 20           5.1
## 21           5.4
## 22           5.1
## 23           4.6
```

```
iris[1]
```

```
##     Sepal.Length
## 1            5.1
## 2            4.9
## 3            4.7
## 4            4.6
## 5            5.0
## 6            5.4
## 7            4.6
## 8            5.0
## 9            4.4
## 10           4.9
## 11           5.4
## 12           4.8
## 13           4.8
## 14           4.3
## 15           5.8
## 16           5.7
## 17           5.4
## 18           5.1
## 19           5.7
## 20           5.1
## 21           5.4
## 22           5.1
## 23           4.6
## 24           5.1
```

INDEXING

- To call specific elements from the data set you can use either
  - Variable indexing using the $ identifier, then using brackets after the variable name to indicate specific position(s)
  - Two element indexing, where the first element is the row and the second element is the column (name or position)

```
iris$Sepal.Length[25:30]
```

```
## [1] 4.8 5.0 5.0 5.2 5.2 4.7
```

```
iris[25:30, "Sepal.Length"]
```

```
## [1] 4.8 5.0 5.0 5.2 5.2 4.7
```

```
iris[25:30, 1]
```

```
## [1] 4.8 5.0 5.0 5.2 5.2 4.7
```

```
iris[c(25:30, 17, 1), c(1, 4)]
```

```
##    Sepal.Length Petal.Width
## 25          4.8         0.2
## 26          5.0         0.2
## 27          5.0         0.4
## 28          5.2         0.2
## 29          5.2         0.2
## 30          4.7         0.2
## 17          5.4         0.4
## 1           5.1         0.2
```

DIMENSIONS

- In general programming when referring to the dimensions of any matrix, the first number is the number of rows and the second is the number of columns
- Use length when referring to one-dimensional vectors, but if used on two or higher dimensional arrays it will return the number of columns
    - This is because vectors are actually stored as arrays with 1 row and X number of columns, where X is the number of elements in the vector

```
dim(iris)
```

```
## [1] 150   5
```

```
nrow(iris)
```

```
## [1] 150
```

```
ncol(iris)
```

```
## [1] 5
```

TROUBLE WITH TIBBLES

TIBBLES

- While the dataframes built into R are very useful, they are lacking in certain features
- Tibbles are a type of data frame that are lazy (they don't change variable names or types) and surly (e.g., they complain when a variable does not exist)
- Tibbles differ from traditional data frames in two key ways, printing and subsetting

```
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
## 14           4.3         3.0          1.1         0.1     setosa
## 15           5.8         4.0          1.2         0.2     setosa
## 16           5.7         4.4          1.5         0.4     setosa
## 17           5.4         3.9          1.3         0.4     setosa
## 18           5.1         3.5          1.4         0.3     setosa
## 19           5.7         3.8          1.7         0.3     setosa
## 20           5.1         3.8          1.5         0.3     setosa
## 21           5.4         3.4          1.7         0.2     setosa
## 22           5.1         3.7          1.5         0.4     setosa
## 23           4.6         3.6          1.0         0.2     setosa
## 24           5.1         3.3          1.7         0.5     setosa
```

```
as_tibble(iris)  # shows only a few rows as well as the type of data in each row
```

```
## # A tibble: 150 × 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl>  <fctr>
## 1           5.1         3.5          1.4         0.2  setosa
## 2           4.9         3.0          1.4         0.2  setosa
## 3           4.7         3.2          1.3         0.2  setosa
## 4           4.6         3.1          1.5         0.2  setosa
## 5           5.0         3.6          1.4         0.2  setosa
## 6           5.4         3.9          1.7         0.4  setosa
```

```
## 7           4.6         3.4          1.4         0.3  setosa
## 8           5.0         3.4          1.5         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa
## # ... with 140 more rows
```

TIBBLES - PARTIAL MATCHING

```
iris$Spec
```

```
##   [1] setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa
##  [11] setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa
##  [21] setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa
##  [31] setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa
##  [41] setosa     setosa     setosa     setosa     setosa     setosa     setosa     setosa
##  [51] versicolor versicolor versicolor versicolor versicolor versicolor versicolor versic
##  [61] versicolor versicolor versicolor versicolor versicolor versicolor versicolor versic
##  [71] versicolor versicolor versicolor versicolor versicolor versicolor versicolor versic
##  [81] versicolor versicolor versicolor versicolor versicolor versicolor versicolor versic
##  [91] versicolor versicolor versicolor versicolor versicolor versicolor versicolor versic
## [101] virginica  virginica  virginica  virginica  virginica  virginica  virginica  virgin
## [111] virginica  virginica  virginica  virginica  virginica  virginica  virginica  virgin
## [121] virginica  virginica  virginica  virginica  virginica  virginica  virginica  virgin
## [131] virginica  virginica  virginica  virginica  virginica  virginica  virginica  virgin
## [141] virginica  virginica  virginica  virginica  virginica  virginica  virginica  virgin
## Levels: setosa versicolor virginica
```

```
as_tibble(iris)$Spec
```

```
## Warning: Unknown or uninitialised column: 'Spec'.
```

```
## NULL
```

TIBBLES - SUBSETTING

```
iris[1]
```

```
##     Sepal.Length
## 1            5.1
## 2            4.9
## 3            4.7
## 4            4.6
## 5            5.0
## 6            5.4
## 7            4.6
## 8            5.0
## 9            4.4
## 10           4.9
## 11           5.4
## 12           4.8
## 13           4.8
## 14           4.3
## 15           5.8
## 16           5.7
## 17           5.4
## 18           5.1
## 19           5.7
## 20           5.1
## 21           5.4
## 22           5.1
## 23           4.6
## 24           5.1
```

```
iris[, 1]
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1 5.7 5.1 5.4
##  [29] 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3
##  [57] 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8
##  [85] 5.4 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5
## [113] 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9 6.4
## [141] 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

```
as_tibble(iris)[1]
```

```
## # A tibble: 150 × 1
##    Sepal.Length
##           <dbl>
## 1           5.1
## 2           4.9
## 3           4.7
## 4           4.6
## 5           5.0
## 6           5.4
## 7           4.6
```

```
## 8           5.0
## 9           4.4
## 10          4.9
## # ... with 140 more rows
```

```
as_tibble(iris)[, 1]
```

```
## # A tibble: 150 × 1
##    Sepal.Length
##           <dbl>
## 1           5.1
## 2           4.9
## 3           4.7
## 4           4.6
## 5           5.0
## 6           5.4
## 7           4.6
## 8           5.0
## 9           4.4
## 10          4.9
## # ... with 140 more rows
```

```
as_tibble(iris)[[1]]
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1 5.7 5.1 5.4
##  [29] 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3
##  [57] 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8
##  [85] 5.4 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5
## [113] 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9 6.4
## [141] 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

DPLYR AND THE 5 + 1 VERBS OF DATA MANIPULATION

dplyr is one of the packages in tidyverse which provides a consistent set of data manipulation verbs.

1. filter
    - Select based on values
2. arrange
    - reorder
3. select & rename
    - select based on names
4. mutate & transmute
    - add new variables that are functions of existing variables
5. summarise
    - condense multiple values to a single value
6. group by
    - perform any operation by group

DPLYR SYNTAX

- The structure of the verbs is the same regardless of which one you use
  - verb(data frame, variable1/argument1, variable2/argument2, …)
  - The result is a new data frame

```
filter(iris, Sepal.Length > 4, Petal.Width == 0.1)  # Note that the variable names do not ha
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          4.9         3.1          1.5         0.1  setosa
## 2          4.8         3.0          1.4         0.1  setosa
## 3          4.3         3.0          1.1         0.1  setosa
## 4          5.2         4.1          1.5         0.1  setosa
## 5          4.9         3.6          1.4         0.1  setosa
```

FLIGHTS DATASET

- On-time data for all flights that departed NYC (i.e. JFK, LGA or EWR) in 2013
- Install the nycflights13 package, `install.packages("nycflights13")`, and load its library, `library(nycflights13)`.

```
library(nycflights13)
flights
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
## 1   2013     1     1      517            515         2      830            819        11
## 2   2013     1     1      533            529         4      850            830        20
## 3   2013     1     1      542            540         2      923            850        33
## 4   2013     1     1      544            545        -1     1004           1022       -18
## 5   2013     1     1      554            600        -6      812            837       -25
## 6   2013     1     1      554            558        -4      740            728        12
## 7   2013     1     1      555            600        -5      913            854        19
## 8   2013     1     1      557            600        -3      709            723       -14
## 9   2013     1     1      557            600        -3      838            846        -8
## 10  2013     1     1      558            600        -2      753            745         8
## # ... with 336,766 more rows, and 6 more variables: dest <chr>, air_time <dbl>, distance
## #   minute <dbl>, time_hour <dttm>
```

```
summary(flights)
```

```
##       year          month             day           dep_time    sched_dep_time   dep_dela
##  Min.   :2013   Min.   : 1.000   Min.   : 1.00   Min.   :   1   Min.   : 106   Min.   : -
##  1st Qu.:2013   1st Qu.: 4.000   1st Qu.: 8.00   1st Qu.: 907   1st Qu.: 906   1st Qu.:
##  Median :2013   Median : 7.000   Median :16.00   Median :1401   Median :1359   Median :
##  Mean   :2013   Mean   : 6.549   Mean   :15.71   Mean   :1349   Mean   :1344   Mean   :
##  3rd Qu.:2013   3rd Qu.:10.000   3rd Qu.:23.00   3rd Qu.:1744   3rd Qu.:1729   3rd Qu.:
##  Max.   :2013   Max.   :12.000   Max.   :31.00   Max.   :2400   Max.   :2359   Max.   :13
##                                                  NA's   :8255                  NA's   :82
##  sched_arr_time   arr_delay          carrier              flight       tailnum
##  Min.   :   1   Min.   : -86.000   Length:336776      Min.   :   1   Length:336776      L
##  1st Qu.:1124   1st Qu.: -17.000   Class :character   1st Qu.: 553   Class :character   C
##  Median :1556   Median :  -5.000   Mode  :character   Median :1496   Mode  :character   M
##  Mean   :1536   Mean   :   6.895                      Mean   :1972
##  3rd Qu.:1945   3rd Qu.:  14.000                      3rd Qu.:3465
##  Max.   :2359   Max.   :1272.000                      Max.   :8500
##                 NA's   :9430
##      dest              air_time        distance         hour           minute        time
##  Length:336776      Min.   : 20.0   Min.   :  17   Min.   : 1.00   Min.   : 0.00   Min.
##  Class :character   1st Qu.: 82.0   1st Qu.: 502   1st Qu.: 9.00   1st Qu.: 8.00   1st Qu
##  Mode  :character   Median :129.0   Median : 872   Median :13.00   Median :29.00   Mediar
##                     Mean   :150.7   Mean   :1040   Mean   :13.18   Mean   :26.23   Mean
```

```
##                  3rd Qu.:192.0   3rd Qu.:1389   3rd Qu.:17.00   3rd Qu.:44.00   3rd Qu
##                  Max.   :695.0   Max.   :4983   Max.   :23.00   Max.   :59.00   Max.
##                  NA's   :9430
```

DATA CLASSES

- `int` integers
- `dbl` doubles or real numbers
- `chr` character vectors (strings)
- `dttm` date-time
- `date` date
- `lgl` logical (TRUE or FALSE)
- `fctr` factors (catgeorical variables with fixed possible values, e.g., dropdown list)
- `list` like a vector but can contain different types of elements
    - You can determine a variable's class by running `class(variable_name)`

```
flights[c("dep_time", "tailnum", "air_time", "time_hour")]
```

```
## # A tibble: 336,776 × 4
##    dep_time tailnum air_time           time_hour
##       <int>   <chr>    <dbl>              <dttm>
## 1       517  N14228      227 2013-01-01 05:00:00
## 2       533  N24211      227 2013-01-01 05:00:00
## 3       542  N619AA      160 2013-01-01 05:00:00
## 4       544  N804JB      183 2013-01-01 05:00:00
## 5       554  N668DN      116 2013-01-01 06:00:00
## 6       554  N39463      150 2013-01-01 05:00:00
## 7       555  N516JB      158 2013-01-01 06:00:00
## 8       557  N829AS       53 2013-01-01 06:00:00
## 9       557  N593JB      140 2013-01-01 06:00:00
## 10      558  N3ALAA      138 2013-01-01 06:00:00
## # ... with 336,766 more rows
```

FILTER

use `filter()` to find rows/cases where conditions are true

```
# Find all flights which went from JFK to Fort Lauderdale in the first week of January
filter(flights, origin == "JFK", dest == "FLL", month == 1, day <= 7)
```

```
## # A tibble: 106 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
## 1   2013     1     1      659            700        -1     1008           1007         1
## 2   2013     1     1      712            715        -3     1023           1035       -12
## 3   2013     1     1      805            800         5     1118           1106        12
## 4   2013     1     1      933            904        29     1252           1210        42
## 5   2013     1     1     1153           1123        30     1454           1425        29
## 6   2013     1     1     1251           1252        -1     1611           1555        16
## 7   2013     1     1     1452           1457        -5     1753           1811       -18
## 8   2013     1     1     1527           1530        -3     1841           1855       -14
## 9   2013     1     1     1610           1615        -5     1913           1948       -35
## 10  2013     1     1     1713           1700        13     2006           2014        -8
## # ... with 96 more rows, and 6 more variables: dest <chr>, air_time <dbl>, distance <dbl>
## #   time_hour <dttm>
```

```
filter(flights, (origin == "JFK" | dest == "FLL"), month == 1, day <= 7)  # , is the same as
```

```
## # A tibble: 2,340 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
## 1   2013     1     1      542            540         2      923            850        33
## 2   2013     1     1      544            545        -1     1004           1022       -18
## 3   2013     1     1      555            600        -5      913            854        19
## 4   2013     1     1      557            600        -3      838            846        -8
## 5   2013     1     1      558            600        -2      849            851        -2
## 6   2013     1     1      558            600        -2      853            856        -3
## 7   2013     1     1      558            600        -2      924            917         7
## 8   2013     1     1      559            559         0      702            706        -4
## 9   2013     1     1      600            600         0      851            858        -7
## 10  2013     1     1      606            610        -4      837            845        -8
## # ... with 2,330 more rows, and 6 more variables: dest <chr>, air_time <dbl>, distance <d
## #   time_hour <dttm>
```

```
# Find all flights going to Fort Lauderdale, Atlanta or O'Hare
filter(flights, dest == "FLL" | dest == "ATL" | dest == "ORD")
```

```
## # A tibble: 46,553 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
```

```
## 1   2013     1     1      554            600        -6      812            837       -25
## 2   2013     1     1      554            558        -4      740            728        12
## 3   2013     1     1      555            600        -5      913            854        19
## 4   2013     1     1      558            600        -2      753            745         8
## 5   2013     1     1      600            600         0      851            858        -7
## 6   2013     1     1      600            600         0      837            825        12
## 7   2013     1     1      606            610        -4      837            845        -8
## 8   2013     1     1      608            600         8      807            735        32
## 9   2013     1     1      615            615         0      833            842        -9
## 10  2013     1     1      629            630        -1      824            810        14
## # ... with 46,543 more rows, and 6 more variables: dest <chr>, air_time <dbl>, distance <
## #   minute <dbl>, time_hour <dttm>
```

```
filter(flights, dest %in% c("FLL", "ATL", "ORD"))  # use %in% to search for multiple values
```

```
## # A tibble: 46,553 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
## 1   2013     1     1      554            600        -6      812            837       -25
## 2   2013     1     1      554            558        -4      740            728        12
## 3   2013     1     1      555            600        -5      913            854        19
## 4   2013     1     1      558            600        -2      753            745         8
## 5   2013     1     1      600            600         0      851            858        -7
## 6   2013     1     1      600            600         0      837            825        12
## 7   2013     1     1      606            610        -4      837            845        -8
## 8   2013     1     1      608            600         8      807            735        32
## 9   2013     1     1      615            615         0      833            842        -9
## 10  2013     1     1      629            630        -1      824            810        14
## # ... with 46,543 more rows, and 6 more variables: dest <chr>, air_time <dbl>, distance <
## #   minute <dbl>, time_hour <dttm>
```

```
# Find all flights that have values for their departure delays
filter(flights, is.na(dep_delay))  # show all the rows with NA (missing values)
```

```
## # A tibble: 8,255 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
## 1   2013     1     1       NA           1630        NA       NA           1815        NA
## 2   2013     1     1       NA           1935        NA       NA           2240        NA
## 3   2013     1     1       NA           1500        NA       NA           1825        NA
## 4   2013     1     1       NA            600        NA       NA            901        NA
## 5   2013     1     2       NA           1540        NA       NA           1747        NA
## 6   2013     1     2       NA           1620        NA       NA           1746        NA
## 7   2013     1     2       NA           1355        NA       NA           1459        NA
## 8   2013     1     2       NA           1420        NA       NA           1644        NA
## 9   2013     1     2       NA           1321        NA       NA           1536        NA
## 10  2013     1     2       NA           1545        NA       NA           1910        NA
## # ... with 8,245 more rows, and 6 more variables: dest <chr>, air_time <dbl>, distance <d
## #   time_hour <dttm>
```

```
filter(flights, !is.na(dep_delay))  # show only the rows without missing values
```

```
## # A tibble: 328,521 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
## 1   2013     1     1      517            515         2      830            819        11
## 2   2013     1     1      533            529         4      850            830        20
## 3   2013     1     1      542            540         2      923            850        33
## 4   2013     1     1      544            545        -1     1004           1022       -18
## 5   2013     1     1      554            600        -6      812            837       -25
## 6   2013     1     1      554            558        -4      740            728        12
## 7   2013     1     1      555            600        -5      913            854        19
## 8   2013     1     1      557            600        -3      709            723       -14
## 9   2013     1     1      557            600        -3      838            846        -8
## 10  2013     1     1      558            600        -2      753            745         8
## # ... with 328,511 more rows, and 6 more variables: dest <chr>, air_time <dbl>, distance
## #   minute <dbl>, time_hour <dttm>
```

ARRANGE

use `arrange()` to sort the data based on one or more variables

```
# Sort the flights based on their scheduled departure time
arrange(flights, sched_dep_time)
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
## 1   2013     7    27       NA            106        NA       NA            245        NA
## 2   2013     1     2      458            500        -2      703            650        13
## 3   2013     1     3      458            500        -2      650            650         0
## 4   2013     1     4      456            500        -4      631            650       -19
## 5   2013     1     5      458            500        -2      640            650       -10
## 6   2013     1     6      458            500        -2      718            650        28
## 7   2013     1     7      454            500        -6      637            648       -11
## 8   2013     1     8      454            500        -6      625            648       -23
## 9   2013     1     9      457            500        -3      647            648        -1
## 10  2013     1    10      450            500       -10      634            648       -14
## # ... with 336,766 more rows, and 6 more variables: dest <chr>, air_time <dbl>, distance
## #   minute <dbl>, time_hour <dttm>
```

```
# Sort the flights based on their scheduled departure time, and break ties using their actua
arrange(flights, sched_dep_time, dep_time)
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
## 1   2013     7    27       NA            106        NA       NA            245        NA
## 2   2013     5     8      445            500       -15      620            640       -20
## 3   2013     5     5      446            500       -14      636            640        -4
## 4   2013     9     4      446            500       -14      618            648       -30
## 5   2013    10     1      447            500       -13      614            648       -34
## 6   2013     9    19      447            500       -13      620            648       -28
## 7   2013     1    29      448            500       -12      635            648       -13
## 8   2013    12    27      448            500       -12      648            651        -3
## 9   2013     5     7      448            500       -12      624            640       -16
## 10  2013    10     2      449            500       -11      620            648       -28
## # ... with 336,766 more rows, and 6 more variables: dest <chr>, air_time <dbl>, distance
## #   minute <dbl>, time_hour <dttm>
```

```
# Sort the flights by those scheduled to depart latest, and break ties in that group by thos
arrange(flights, desc(sched_dep_time), dep_time)
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
```

```
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
## 1   2013    11    13        1           2359         2      442            440         2
## 2   2013    12    16        1           2359         2      447            437        10
## 3   2013    12    20        1           2359         2      430            440       -10
## 4   2013    12    26        1           2359         2      437            440        -3
## 5   2013    12    30        1           2359         2      441            437         4
## 6   2013     4     5        1           2359         2      410            339        31
## 7   2013     5    25        1           2359         2      336            341        -5
## 8   2013     6    20        1           2359         2      340            350       -10
## 9   2013     7    27        1           2359         2      345            340         5
## 10  2013     7    28        1           2359         2      423            350        33
## # ... with 336,766 more rows, and 6 more variables: dest <chr>, air_time <dbl>, distance
## #   minute <dbl>, time_hour <dttm>
```

- The problems is I forgot about flights which were delayed so left the next morning
- Since dep_time is day agnostic, this does not give me the data I am looking for
- I can try and figure out which flights left that day and which left the next or just use a different variable as my metric of the late evening flights which left earliest.

```
arrange(flights, desc(sched_dep_time), dep_delay)
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
## 1   2013    10     2     2341           2359       -18      324            350       -26
## 2   2013     9    23     2342           2359       -17      331            350       -19
## 3   2013    10    22     2343           2359       -16      347            350        -3
## 4   2013     3     4     2343           2359       -16      418            438       -20
## 5   2013     1    20     2344           2359       -15      428            437        -9
## 6   2013     4    16     2344           2359       -15      313            343       -30
## 7   2013     1    27     2345           2359       -14      424            444       -20
## 8   2013     3     3     2345           2359       -14      441            438         3
## 9   2013     3     5     2345           2359       -14      439            438         1
## 10  2013    10     6     2346           2359       -13      333            350       -17
## # ... with 336,766 more rows, and 6 more variables: dest <chr>, air_time <dbl>, distance
## #   minute <dbl>, time_hour <dttm>
```

SELECT & RENAME

use `select()` and `rename()` to pick variables based on their names

```
# Select the year, month, day, dep_times, and sched_dep_time columns
select(flights, year, month, day, dep_time, sched_dep_time)
```

```
## # A tibble: 336,776 × 5
##     year month   day dep_time sched_dep_time
##    <int> <int> <int>    <int>          <int>
## 1   2013     1     1      517            515
## 2   2013     1     1      533            529
## 3   2013     1     1      542            540
## 4   2013     1     1      544            545
## 5   2013     1     1      554            600
## 6   2013     1     1      554            558
## 7   2013     1     1      555            600
## 8   2013     1     1      557            600
## 9   2013     1     1      557            600
## 10  2013     1     1      558            600
## # ... with 336,766 more rows
```

```
select(flights, year:sched_dep_time)
```

```
## # A tibble: 336,776 × 5
##     year month   day dep_time sched_dep_time
##    <int> <int> <int>    <int>          <int>
## 1   2013     1     1      517            515
## 2   2013     1     1      533            529
## 3   2013     1     1      542            540
## 4   2013     1     1      544            545
## 5   2013     1     1      554            600
## 6   2013     1     1      554            558
## 7   2013     1     1      555            600
## 8   2013     1     1      557            600
## 9   2013     1     1      557            600
## 10  2013     1     1      558            600
## # ... with 336,766 more rows
```

```
select(flights, 1:5)
```

```
## # A tibble: 336,776 × 5
##     year month   day dep_time sched_dep_time
##    <int> <int> <int>    <int>          <int>
## 1   2013     1     1      517            515
## 2   2013     1     1      533            529
## 3   2013     1     1      542            540
## 4   2013     1     1      544            545
```

```
## 5   2013     1     1      554            600
## 6   2013     1     1      554            558
## 7   2013     1     1      555            600
## 8   2013     1     1      557            600
## 9   2013     1     1      557            600
## 10  2013     1     1      558            600
## # ... with 336,766 more rows
```

```
select(flights, -(dep_delay:time_hour))  # more useful when removing only a few columns
```

```
## # A tibble: 336,776 × 5
##     year month   day dep_time sched_dep_time
##    <int> <int> <int>    <int>          <int>
## 1   2013     1     1      517            515
## 2   2013     1     1      533            529
## 3   2013     1     1      542            540
## 4   2013     1     1      544            545
## 5   2013     1     1      554            600
## 6   2013     1     1      554            558
## 7   2013     1     1      555            600
## 8   2013     1     1      557            600
## 9   2013     1     1      557            600
## 10  2013     1     1      558            600
## # ... with 336,766 more rows
```

`rename` lets you change the name of a variable while still keeping the full data set

```
rename(flights, sun_cycles = year)
```

```
## # A tibble: 336,776 × 19
##    sun_cycles month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_d
##         <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <
## 1        2013     1     1      517            515         2      830            819
## 2        2013     1     1      533            529         4      850            830
## 3        2013     1     1      542            540         2      923            850
## 4        2013     1     1      544            545        -1     1004           1022
## 5        2013     1     1      554            600        -6      812            837
## 6        2013     1     1      554            558        -4      740            728
## 7        2013     1     1      555            600        -5      913            854
## 8        2013     1     1      557            600        -3      709            723
## 9        2013     1     1      557            600        -3      838            846
## 10       2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 7 more variables: origin <chr>, dest <chr>, air_time <d
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
flights  # Note that we are not assigning any of these outputs, so if you call the original
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
```

```
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
## 1   2013     1     1      517            515         2      830            819        11
## 2   2013     1     1      533            529         4      850            830        20
## 3   2013     1     1      542            540         2      923            850        33
## 4   2013     1     1      544            545        -1     1004           1022       -18
## 5   2013     1     1      554            600        -6      812            837       -25
## 6   2013     1     1      554            558        -4      740            728        12
## 7   2013     1     1      555            600        -5      913            854        19
## 8   2013     1     1      557            600        -3      709            723       -14
## 9   2013     1     1      557            600        -3      838            846        -8
## 10  2013     1     1      558            600        -2      753            745         8
## # ... with 336,766 more rows, and 6 more variables: dest <chr>, air_time <dbl>, distance
## #   minute <dbl>, time_hour <dttm>
```

The `everything()` helper lets you use `select` to rearrange the order of the variables

```
select(flights, distance, air_time, everything())
```

```
## # A tibble: 336,776 × 19
##    distance air_time  year month   day dep_time sched_dep_time dep_delay arr_time sched_a
##       <dbl>    <dbl> <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1      1400      227  2013     1     1      517            515         2      830
## 2      1416      227  2013     1     1      533            529         4      850
## 3      1089      160  2013     1     1      542            540         2      923
## 4      1576      183  2013     1     1      544            545        -1     1004
## 5       762      116  2013     1     1      554            600        -6      812
## 6       719      150  2013     1     1      554            558        -4      740
## 7      1065      158  2013     1     1      555            600        -5      913
## 8       229       53  2013     1     1      557            600        -3      709
## 9       944      140  2013     1     1      557            600        -3      838
## 10      733      138  2013     1     1      558            600        -2      753
## # ... with 336,766 more rows, and 7 more variables: flight <int>, tailnum <chr>, origin <
## #   minute <dbl>, time_hour <dttm>
```

MUTATE & TRANSMUTE

`mutate` adds new variables, while `transmute` drops existing variables

```
# create a subset of the full dataset so that you can see new variables being added
flights_sml <- select(flights, dep_time, arr_time, air_time, distance)
flights_sml
```

```
## # A tibble: 336,776 × 4
##    dep_time arr_time air_time distance
##       <int>    <int>    <dbl>    <dbl>
## 1       517      830      227     1400
## 2       533      850      227     1416
## 3       542      923      160     1089
## 4       544     1004      183     1576
## 5       554      812      116      762
## 6       554      740      150      719
## 7       555      913      158     1065
## 8       557      709       53      229
## 9       557      838      140      944
## 10      558      753      138      733
## # ... with 336,766 more rows
```

```
mutate(flights_sml, avg_speed = distance/air_time, dep_hr = dep_time%/%100, dep_min = dep_ti
```

```
## # A tibble: 336,776 × 7
##    dep_time arr_time air_time distance avg_speed dep_hr dep_min
##       <int>    <int>    <dbl>    <dbl>     <dbl>  <dbl>   <dbl>
## 1       517      830      227     1400  6.167401      5      17
## 2       533      850      227     1416  6.237885      5      33
## 3       542      923      160     1089  6.806250      5      42
## 4       544     1004      183     1576  8.612022      5      44
## 5       554      812      116      762  6.568966      5      54
## 6       554      740      150      719  4.793333      5      54
## 7       555      913      158     1065  6.740506      5      55
## 8       557      709       53      229  4.320755      5      57
## 9       557      838      140      944  6.742857      5      57
## 10      558      753      138      733  5.311594      5      58
## # ... with 336,766 more rows
```

```
transmute(flights_sml, avg_speed = distance/air_time, dep_hr = dep_time%/%100, dep_min = dep
```

```
## # A tibble: 336,776 × 3
##    avg_speed dep_hr dep_min
##        <dbl>  <dbl>   <dbl>
## 1   6.167401      5      17
## 2   6.237885      5      33
## 3   6.806250      5      42
```

```
## 4   8.612022      5      44
## 5   6.568966      5      54
## 6   4.793333      5      54
## 7   6.740506      5      55
## 8   4.320755      5      57
## 9   6.742857      5      57
## 10  5.311594      5      58
## # ... with 336,766 more rows
```

- There are lots of useful functions which can be applied within mutate/transmute, use `?mutate` for the full suggested list
  - `lead()` and `lag()` find the next and previous values in a vector, respectively.
  - `cumsum()`, `cummean()`, and others (see help doc) take running sums, means and other properties

```
# How much time is there between each flight and the next?
mutate(flights_sml, dep_time_offset = lag(dep_time), dep_time_lag = dep_time - lag(dep_time)
```

```
## # A tibble: 336,776 × 6
##    dep_time arr_time air_time distance dep_time_offset dep_time_lag
##       <int>    <int>    <dbl>    <dbl>           <int>        <int>
## 1       517      830      227     1400              NA           NA
## 2       533      850      227     1416             517           16
## 3       542      923      160     1089             533            9
## 4       544     1004      183     1576             542            2
## 5       554      812      116      762             544           10
## 6       554      740      150      719             554            0
## 7       555      913      158     1065             554            1
## 8       557      709       53      229             555            2
## 9       557      838      140      944             557            0
## 10      558      753      138      733             557            1
## # ... with 336,766 more rows
```

```
mutate(flights_sml, dep_time_offset = lead(dep_time), dep_time_lead = lead(dep_time) - dep_t
```

```
## # A tibble: 336,776 × 6
##    dep_time arr_time air_time distance dep_time_offset dep_time_lead
##       <int>    <int>    <dbl>    <dbl>           <int>         <int>
## 1       517      830      227     1400             533            16
## 2       533      850      227     1416             542             9
## 3       542      923      160     1089             544             2
## 4       544     1004      183     1576             554            10
## 5       554      812      116      762             554             0
## 6       554      740      150      719             555             1
## 7       555      913      158     1065             557             2
## 8       557      709       53      229             557             0
## 9       557      838      140      944             558             1
## 10      558      753      138      733             558             0
## # ... with 336,766 more rows
```

```
mutate(flights_sml, total_dist = cumsum(distance))
```

```
## # A tibble: 336,776 × 5
##    dep_time arr_time air_time distance total_dist
##       <int>    <int>    <dbl>    <dbl>      <dbl>
## 1       517      830      227     1400       1400
## 2       533      850      227     1416       2816
## 3       542      923      160     1089       3905
## 4       544     1004      183     1576       5481
## 5       554      812      116      762       6243
## 6       554      740      150      719       6962
## 7       555      913      158     1065       8027
## 8       557      709       53      229       8256
## 9       557      838      140      944       9200
## 10      558      753      138      733       9933
## # ... with 336,766 more rows
```

SUMMARISE & GROUP BY

Summarise reduces multiple values to a single summary metric

```
summarise(flights, delay = mean(dep_delay))
```

```
## # A tibble: 1 × 1
##   delay
##   <dbl>
## 1    NA
```

- Why does this give us `NA`?
- In R, missing values are represented by the symbol `NA` (not available)
    - This is not to be confused with `NaN` (not a number), which refers to impossible values, e.g., dividing by zero
- When you try to do any operation that includes `NA` values, the output will always be `NA`
    - Think of `NA`'s as being any possible value, as a result any summary metric will result in an unknown quantity as the uknown `NA` value could have significantly impacted the results
- To solve this problem most R functions have the option to ignore `NA` value
    - Usually it is of the form `na.rm=TRUE`

```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 × 1
##      delay
##      <dbl>
## 1 12.63907
```

- By itself `summarise` isn't that useful as we rarely want to reduce all our data down to a single metric
- `summarise` is much more useful when combined with the other expressions
    - For example, determining the average departure delay of all flights in January

```
jan_delay <- filter(flights, month == 1)
summarise(jan_delay, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 × 1
##      delay
##      <dbl>
## 1 10.03667
```

- However, the real power of `summarise` is seen when coupled with `group_by`
- `group_by` takes an existing tibble and converts it into a grouped tibble where operations are performed "by group"
    - By itself `group_by` does not change how the data looks, instead it changes how it interacts with the other verbs, most notably `summarise`

```
flights_month <- group_by(flights, month)
flights_month
```

```
## Source: local data frame [336,776 x 19]
## Groups: month [12]
##
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
## 1   2013     1     1      517            515         2      830            819        11
## 2   2013     1     1      533            529         4      850            830        20
## 3   2013     1     1      542            540         2      923            850        33
## 4   2013     1     1      544            545        -1     1004           1022       -18
## 5   2013     1     1      554            600        -6      812            837       -25
## 6   2013     1     1      554            558        -4      740            728        12
## 7   2013     1     1      555            600        -5      913            854        19
## 8   2013     1     1      557            600        -3      709            723       -14
## 9   2013     1     1      557            600        -3      838            846        -8
## 10  2013     1     1      558            600        -2      753            745         8
```

```
## # ... with 336,766 more rows, and 6 more variables: dest <chr>, air_time <dbl>, distance
## #   minute <dbl>, time_hour <dttm>
```

```
summarise(flights_month, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 12 × 2
##    month     delay
##    <int>     <dbl>
## 1      1 10.036665
## 2      2 10.816843
## 3      3 13.227076
## 4      4 13.938038
## 5      5 12.986859
## 6      6 20.846332
## 7      7 21.727787
## 8      8 12.611040
## 9      9  6.722476
## 10    10  6.243988
## 11    11  5.435362
## 12    12 16.576688
```

Also it's good practice when grouping to add a counts column using `n()`

```
summarise(flights_month, delay = mean(dep_delay, na.rm = TRUE), count = n())
```

```
## # A tibble: 12 × 3
##    month     delay count
##    <int>     <dbl> <int>
## 1      1 10.036665 27004
## 2      2 10.816843 24951
## 3      3 13.227076 28834
## 4      4 13.938038 28330
## 5      5 12.986859 28796
## 6      6 20.846332 28243
## 7      7 21.727787 29425
## 8      8 12.611040 29327
## 9      9  6.722476 27574
## 10    10  6.243988 28889
## 11    11  5.435362 27268
## 12    12 16.576688 28135
```

PIPES

PIPES

Often you will need to string multiple actions together which can get somewhat messy

```
# On average which hour of the day has the most delayed american airline flights
flights_mut <- mutate(flights, hr = sched_dep_time%/%100)
flights_filt <- filter(flights_mut, carrier == "AA", complete.cases(flights_mut))
flights_sel <- select(flights_filt, dep_time, hr, sched_dep_time, dep_delay)
flights_sel  # print out to confirm that you are selecting what you intend
```

```
## # A tibble: 31,947 × 4
##    dep_time    hr sched_dep_time dep_delay
##       <int> <dbl>          <int>     <dbl>
## 1       542     5            540         2
## 2       558     6            600        -2
## 3       559     6            600        -1
## 4       606     6            610        -4
## 5       623     6            610        13
## 6       628     6            630        -2
## 7       629     6            630        -1
## 8       635     6            635         0
## 9       656     7            700        -4
## 10      656     6            659        -3
## # ... with 31,937 more rows
```

```
flights_gb <- group_by(flights_sel, hr)
flights_sum <- summarise(flights_gb, mean_delay = mean(dep_delay), count = n())
flights_arr <- arrange(flights_sum, desc(mean_delay))
print(flights_arr, n = 24)
```

```
## # A tibble: 17 × 3
##       hr mean_delay count
##    <dbl>      <dbl> <int>
## 1     19 21.2870418  1937
## 2     17 19.9348861  3993
## 3     21 18.2857143   406
## 4     18 14.3431713  1728
## 5     15 12.3666921  2618
## 6     20 12.2851240   484
## 7     16 11.9728287  2061
## 8     14  7.7936709  1580
## 9     13  7.3642305  1683
## 10    10  5.8927536  1380
## 11    11  5.1475167  1349
## 12    12  4.9592920  2260
## 13     9  2.6576763  2410
## 14     8  2.1545312  1909
## 15     5  0.6814404   361
## 16     7 -0.4511713  3287
## 17     6 -1.0215914  2501
```

The pipes %>% or CTRL+Shift+M (from the magrittr package which is included in tidyverse) allows you to do the same set of actions in a much simpler manner

```
flights %>%
  mutate(hr = sched_dep_time%/% 100) %>%
  filter(carrier == 'AA', complete.cases(flights_mut)) %>%
  select(dep_time, hr, sched_dep_time, dep_delay) %>%
  group_by(hr) %>%
  summarise(mean_delay = mean(dep_delay),count = n()) %>%
  arrange(desc(mean_delay))
```

```
## # A tibble: 17 × 3
##       hr mean_delay count
##    <dbl>      <dbl> <int>
## 1     19 21.2870418  1937
## 2     17 19.9348861  3993
## 3     21 18.2857143   406
## 4     18 14.3431713  1728
## 5     15 12.3666921  2618
## 6     20 12.2851240   484
## 7     16 11.9728287  2061
## 8     14  7.7936709  1580
## 9     13  7.3642305  1683
## 10    10  5.8927536  1380
## 11    11  5.1475167  1349
## 12    12  4.9592920  2260
## 13     9  2.6576763  2410
## 14     8  2.1545312  1909
## 15     5  0.6814404   361
## 16     7 -0.4511713  3287
## 17     6 -1.0215914  2501
```

- Pipes can be thought of as the phrase “and then”, so the above code would be read as:
    1. Take the input flights and then
    2. mutate it and then
    3. filter it and then
    4. select it and then …
- Pipes are useful when
    - There is only a single input and you don’t need to combine inputs
    - You only want a single ouptut and don’t care about the outputs from the intermediate steps
- But remember that without an assignment, the output is not saved to a variable

EXERCISES & PLAYTIME

- Using the babynames dataset from the babynames package:
    1. How popular was your name in the US in the year you were born, i.e. how many other babies were given the same name?
    2. Which are the overall most popular girl and boy baby names as measured by count? How many times more popular are they then the next most popular names?
    3. Since 1990 how many girls have been named “Michael”
    4. What was the 8th most popular year for the name “Michael” as measured by counts?
    5. Which girl’s name had the biggest increase in consecutive years, in what years was it, and why?

The `fivethirtyeight` package has lots of fun datasets. To see the motivation behind the package’s creation as well as a description of each of the different datasets use

```
vignette("fivethirtyeight", package = "fivethirtyeight")
```

COMBINING DATA MANIPULATION AND VISUALIZATION

1. Plot the popularity of the boy's name Brittany since 1950.
2. Using the 5 most popular girl's name from 2015 (based on counts), plot their counts over all the years recorded, using different color for each name. Now plot the same names using the prop value instead.
3. Plot the number of names which have a significant share of the total names, i.e., > 1%, over all years recorded

SECTION 4 - IMPORTING DATA

FOUND VS GENERATED DATA

- Most of the time in your research you will be using data that you or someone you know generated
    - This will often result in tidy and clean data or if not at least someone you can complain to about why the data is so messy
- But what about when you have to use 3rd party data, e.g., weather information
    - First you have to find the data (a topic we will not be covering)
    - Then you have to import the data
    - Then you have to clean and tidy the data (not covered today)
- Dataset formats
    - Columns are variables separated by a delimeter, such as a comma (.csv), semicolon (.csv2), tab (.tsv)
    - Rows are entries and their values form the index column of the data
        - Usually the values in the first row are the names of the columns

```
# Trip Duration,Start Time,Stop Time,Start Station ID,Start Station Name
# 1893,2017-03-01 00:00:32,2017-03-01 00:32:06,2009,Catherine St & Monroe St
# 223,2017-03-01 00:01:09,2017-03-01 00:04:53,127,Barrow St & Hudson St
# 1665,2017-03-01 00:01:27,2017-03-01 00:29:12,174,E 25 St & 1 Ave
# 100,2017-03-01 00:01:29,2017-03-01 00:03:10,316,Fulton St & William St
# 1229,2017-03-01 00:01:33,2017-03-01 00:22:02,536,1 Ave & E 30 St
# 613,2017-03-01 00:01:57,2017-03-01 00:12:11,259,South St & Whitehall St
# 157,2017-03-01 00:02:12,2017-03-01 00:04:49,3329,Degraw St & Smith St
```

READING CSV FILES

- R has a built in function for reading csv files `read.csv()` which reads the data into a dataframe, but instead we will be using the tidyverse package readr, which uses the slightly different `read_csv()`
    - Feel free to compare the documentation for the two functions if you want to know how they differ
    - Since the `read_csv` function is powerful and versatile it's good to look at its default options before using it

```
read_csv(file, col_names = TRUE, col_types = NULL,
  locale = default_locale(), na = c("", "NA"), quoted_na = TRUE,
  quote = "\"", comment = "", trim_ws = TRUE, skip = 0, n_max = Inf,
  guess_max = min(1000, n_max), progress = show_progress())
```

- file
    - use the full path ('E:\full\path\to\file.csv')
- Note that you can easily change the working directory but that is outside the scope of this bootcamp
- col_names
    - Either TRUE, FALSE or a character vector of column names.
        - If TRUE, the first row of the input will be used as the column names, and will not be included in the data frame
        - If FALSE, column names will be generated automatically: X1, X2, X3 etc.
        - If col_names is a character vector, the values will be used as the names of the columns, and the first row of the input will be read into the first row of the output data frame.

- col_types
  - NULL, a cols() specification, or a string
    - If NULL, all column types will be imputed from the first 1000 rows on the input
    - If a column specification created by cols(), it must contain one column specification for each column.
    - Alternatively, you can use a compact string representation where each character represents one column: c = character, i = integer, n = number, d = double, l = logical, D = date, T = date time, t = time, ? = guess

- na
  - Character vector of strings to use for missing values
- skip
  - Number of lines to skip before reading data.
- n_max
  - Maximum number of records to read.
- guess_max
  - Maximum number of records to use for guessing column types.

WRITING CSV FILES

- Similar to the `read_csv()` function there is a `write_csv()` function which writes csv to some designated path
- The parameters for this function are much simpler

```
write_csv(x, path, na = "NA", append = FALSE, col_names = !append)
```

- x
  - A data frame to write to disk
- path
  - Path or connection to write to.
- na
  - String used for missing values. Defaults to NA. Missing values will never be quoted; strings with the same value as na will always be quoted.
- append
  - If FALSE, will overwrite existing file. If TRUE, will append to existing file. In both cases, if file does not exist a new file is created.
- col_names
  - Write columns names at the top of the file?

```
write_csv(citibike, "./citibike_data/output_name.csv")
```

CITIBIKE DATASET

- Download the citibike csv files from the citibike_data directory where you downloaded the R file
    - The RBootcamp folder in **https://github.com/mseinstein/Presentations**
- This is a slightly modified version of the dataset which is publicly available online from citibike and contains every ride for an entire month

```
read_csv("./citibike_data/201701-citibike-tripdata.csv")
```

```
## Parsed with column specification:
## cols(
##   `Trip Duration` = col_integer(),
##   `Start Time` = col_datetime(format = ""),
##   `Stop Time` = col_datetime(format = ""),
##   `Start Station ID` = col_integer(),
##   `Start Station Name` = col_character(),
##   `End Station ID` = col_integer(),
##   `End Station Name` = col_character(),
##   `Bike ID` = col_integer(),
##   `User Type` = col_character(),
##   `Birth Year` = col_integer(),
##   Gender = col_integer()
## )
```

```
## # A tibble: 726,676 × 11
##    `Trip Duration`        `Start Time`         `Stop Time` `Start Station ID`        `Sta
##              <int>              <dttm>              <dttm>              <int>
## 1             680 2017-01-01 00:00:21 2017-01-01 00:11:41               3226 W 82 St & C
## 2            1282 2017-01-01 00:00:45 2017-01-01 00:22:08               3263      Cooper
## 3             648 2017-01-01 00:00:57 2017-01-01 00:11:46               3143
## 4             631 2017-01-01 00:01:10 2017-01-01 00:11:42               3143
## 5             621 2017-01-01 00:01:25 2017-01-01 00:11:47               3143
## 6             666 2017-01-01 00:01:51 2017-01-01 00:12:57               3163 Central Par
## 7             559 2017-01-01 00:05:00 2017-01-01 00:14:20                499          Br
## 8             826 2017-01-01 00:05:37 2017-01-01 00:19:24                362          Br
## 9             255 2017-01-01 00:05:47 2017-01-01 00:10:02                430
## 10            634 2017-01-01 00:07:34 2017-01-01 00:18:08               3165 Central Par
## # ... with 726,666 more rows, and 6 more variables: `End Station ID` <int>, `End Station
## #   `User Type` <chr>, `Birth Year` <int>, Gender <int>
```

You can see that R solved the problem of spaces in the column names by putting ` around them (note this is a backtick not a single quote), but

this makes it annoying to refer to these variables. Instead we can use a snake case version of the column names as an input into `read_csv`

```
column_names <- c("trip_duration", "start_time", "stop_time", "start_id", "start_name", "end
    "user_type", "birth_year", "gender")
read_csv("./citibike_data/201701-citibike-tripdata.csv", col_names = column_names)
```

```
## Parsed with column specification:
## cols(
##   trip_duration = col_character(),
##   start_time = col_character(),
##   stop_time = col_character(),
##   start_id = col_character(),
##   start_name = col_character(),
##   end_id = col_character(),
##   end_name = col_character(),
##   bike_id = col_character(),
##   user_type = col_character(),
##   birth_year = col_character(),
##   gender = col_character()
## )
```

```
## # A tibble: 726,677 × 11
##    trip_duration           start_time            stop_time         start_id
##            <chr>                <chr>                <chr>            <chr>
## 1  Trip Duration           Start Time            Stop Time Start Station ID
## 2            680 2017-01-01T00:00:21Z 2017-01-01T00:11:41Z             3226
## 3           1282 2017-01-01T00:00:45Z 2017-01-01T00:22:08Z             3263
## 4            648 2017-01-01T00:00:57Z 2017-01-01T00:11:46Z             3143
## 5            631 2017-01-01T00:01:10Z 2017-01-01T00:11:42Z             3143
## 6            621 2017-01-01T00:01:25Z 2017-01-01T00:11:47Z             3143
## 7            666 2017-01-01T00:01:51Z 2017-01-01T00:12:57Z             3163
## 8            559 2017-01-01T00:05:00Z 2017-01-01T00:14:20Z              499
## 9            826 2017-01-01T00:05:37Z 2017-01-01T00:19:24Z              362
## 10           255 2017-01-01T00:05:47Z 2017-01-01T00:10:02Z              430
## # ... with 726,667 more rows, and 5 more variables: end_name <chr>, bike_id <chr>, user_t
## #   gender <chr>
```

The problem is that when you provide the column names, `read_csv` treats the first row as a regular row of entries, so if we want to use our own column names we need to skip that row

```
citi_bike <- read_csv("./citibike_data/201701-citibike-tripdata.csv", col_names = column_nam
```

```
## Parsed with column specification:
## cols(
##   trip_duration = col_integer(),
##   start_time = col_datetime(format = ""),
##   stop_time = col_datetime(format = ""),
##   start_id = col_integer(),
##   start_name = col_character(),
##   end_id = col_integer(),
##   end_name = col_character(),
```

```
##   bike_id = col_integer(),
##   user_type = col_character(),
##   birth_year = col_integer(),
##   gender = col_integer()
## )
```

```
citi_bike
```

```
## # A tibble: 726,676 × 11
##    trip_duration          start_time           stop_time start_id                  start_
##            <int>              <dttm>              <dttm>    <int>                      <
## 1            680 2017-01-01 00:00:21 2017-01-01 00:11:41     3226 W 82 St & Central Park
## 2           1282 2017-01-01 00:00:45 2017-01-01 00:22:08     3263      Cooper Square & E
## 3            648 2017-01-01 00:00:57 2017-01-01 00:11:46     3143             5 Ave & E 7
## 4            631 2017-01-01 00:01:10 2017-01-01 00:11:42     3143             5 Ave & E 7
## 5            621 2017-01-01 00:01:25 2017-01-01 00:11:47     3143             5 Ave & E 7
## 6            666 2017-01-01 00:01:51 2017-01-01 00:12:57     3163 Central Park West & W 6
## 7            559 2017-01-01 00:05:00 2017-01-01 00:14:20      499          Broadway & W 6
## 8            826 2017-01-01 00:05:37 2017-01-01 00:19:24      362          Broadway & W 3
## 9            255 2017-01-01 00:05:47 2017-01-01 00:10:02      430            York St & Ja
## 10           634 2017-01-01 00:07:34 2017-01-01 00:18:08     3165 Central Park West & W 7
## # ... with 726,666 more rows, and 5 more variables: end_name <chr>, bike_id <int>, user_t
## #   gender <int>
```

PARSING

- Parsing , in reference to `read_csv` and similar functions, is the method of analyzing elements of a vector to determine the type of information within that vector
- By default, `read_csv` uses the first 1000 rows or the entire dataset, whichever is smaller, to parse each column
- Usually it does a very good job and most of the time you have to change data types because you want a specific data type, not because of an error within the parser

```
# Let's look again at our dataset and see if all our datatypes make sense
citi_bike
```

```
## # A tibble: 726,676 × 11
##    trip_duration          start_time           stop_time start_id                  start_
##            <int>              <dttm>              <dttm>    <int>                      <
## 1            680 2017-01-01 00:00:21 2017-01-01 00:11:41     3226 W 82 St & Central Park
## 2           1282 2017-01-01 00:00:45 2017-01-01 00:22:08     3263      Cooper Square & E
## 3            648 2017-01-01 00:00:57 2017-01-01 00:11:46     3143             5 Ave & E 7
## 4            631 2017-01-01 00:01:10 2017-01-01 00:11:42     3143             5 Ave & E 7
## 5            621 2017-01-01 00:01:25 2017-01-01 00:11:47     3143             5 Ave & E 7
## 6            666 2017-01-01 00:01:51 2017-01-01 00:12:57     3163 Central Park West & W 6
## 7            559 2017-01-01 00:05:00 2017-01-01 00:14:20      499          Broadway & W 6
## 8            826 2017-01-01 00:05:37 2017-01-01 00:19:24      362          Broadway & W 3
## 9            255 2017-01-01 00:05:47 2017-01-01 00:10:02      430            York St & Ja
## 10           634 2017-01-01 00:07:34 2017-01-01 00:18:08     3165 Central Park West & W 7
## # ... with 726,666 more rows
```

```
summary(citi_bike)
```

```
##  trip_duration       start_time                    stop_time                      start_i
##  Min.   :     61   Min.   :2017-01-01 00:00:21   Min.   :2017-01-01 00:10:02   Min.   :
##  1st Qu.:    331   1st Qu.:2017-01-11 08:40:46   1st Qu.:2017-01-11 08:52:47   1st Qu.: 3
##  Median :    526   Median :2017-01-18 09:25:48   Median :2017-01-18 09:37:23   Median : 4
##  Mean   :    778   Mean   :2017-01-17 16:36:15   Mean   :2017-01-17 16:49:13   Mean   :12
##  3rd Qu.:    860   3rd Qu.:2017-01-25 15:39:17   3rd Qu.:2017-01-25 15:52:34   3rd Qu.:30
##  Max.   :5325688   Max.   :2017-01-31 23:59:23   Max.   :2017-03-14 14:13:45   Max.   :34
##
##      end_id       end_name            bike_id       user_type           birth_year
##  Min.   :  72   Length:726676      Min.   :14529   Length:726676      Min.   :1885    Min
##  1st Qu.: 356   Class :character   1st Qu.:17859   Class :character   1st Qu.:1969    1st
##  Median : 479   Mode  :character   Median :21295   Mode  :character   Median :1979    Med
##  Mean   :1197                      Mean   :21713                      Mean   :1977    Mea
##  3rd Qu.:3078                      3rd Qu.:25803                      3rd Qu.:1987    3rd
```

```
##  Max.   :3447                    Max.   :27325                    Max.   :2000    Max
##                                                                   NA's   :29076
```

- Double and integers are very useful for continuous values, but when you have a limited number of discrete values (and where the orders of the valuesare typically irrelevant), the factor type is often more useful
    - Factors are useful when you have a categorical variable (think of it like a dropdown menu)
    - Factor values are always converted to characters
    - The `level()` function list all the possible options of the factor variable
- There are two ways to convert a csv tibble column into a factor
    - When you initially read the file you can specify the type for each column (useful when changing most of the columns) using the `col_factor()` function
    - After reading the file you can convert the specified columns to factors using the `as.factor()` function

```
# In the citi_bike case start_id, end_id, user_type and gender are all variables which shoul
citi_bike_fac <- citi_bike %>% mutate(start_id = as.factor(start_id), end_id = as.factor(end
    gender = as.factor(gender))
citi_bike_fac
```

```
## # A tibble: 726,676 × 11
##    trip_duration          start_time           stop_time start_id                  start_
##            <int>              <dttm>              <dttm>   <fctr>                      <
## 1            680 2017-01-01 00:00:21 2017-01-01 00:11:41     3226 W 82 St & Central Park
## 2           1282 2017-01-01 00:00:45 2017-01-01 00:22:08     3263      Cooper Square & E
## 3            648 2017-01-01 00:00:57 2017-01-01 00:11:46     3143             5 Ave & E 7
## 4            631 2017-01-01 00:01:10 2017-01-01 00:11:42     3143             5 Ave & E 7
## 5            621 2017-01-01 00:01:25 2017-01-01 00:11:47     3143             5 Ave & E 7
## 6            666 2017-01-01 00:01:51 2017-01-01 00:12:57     3163 Central Park West & W 6
## 7            559 2017-01-01 00:05:00 2017-01-01 00:14:20      499          Broadway & W 6
## 8            826 2017-01-01 00:05:37 2017-01-01 00:19:24      362          Broadway & W 3
## 9            255 2017-01-01 00:05:47 2017-01-01 00:10:02      430            York St & Ja
## 10           634 2017-01-01 00:07:34 2017-01-01 00:18:08     3165 Central Park West & W 7
## # ... with 726,666 more rows, and 5 more variables: end_name <chr>, bike_id <int>, user_t
## #   gender <fctr>
```

```
levels(citi_bike_fac$gender)
```

```
## [1] "0" "1" "2"
```

- Other data types have their own version of `col_factor()` and `as.factor()`
  - `col_logical()`, `col_integer()`, `col_double()`, `col_character()`
  - `as.logical()`, `as.integer()`, `as.double()`, `as.character()`