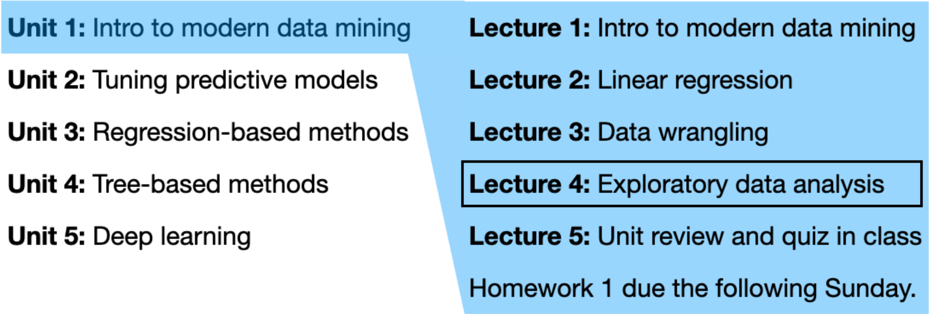
Unit 1 Lecture 4: Exploratory data analysis

September 9, 2021

Welcome back to STAT 471! We are now in Unit 1 Lecture 4:



Unit 1: Intro to modern data mining

Unit 2: Tuning predictive models

Unit 3: Regression-based methods

Unit 4: Tree-based methods

Unit 5: Deep learning

Lecture 1: Intro to modern data mining

Lecture 2: Linear regression

Lecture 3: Data wrangling

Lecture 4: Exploratory data analysis

Lecture 5: Unit review and quiz in class

Homework 1 due the following Sunday.

This lecture is about *exploratory data analysis*, which involves data transformation and visualization. It draws on Chapters 3, 5, and 7 from the excellent R for Data Science book (direct quotations are presented using block quotes).

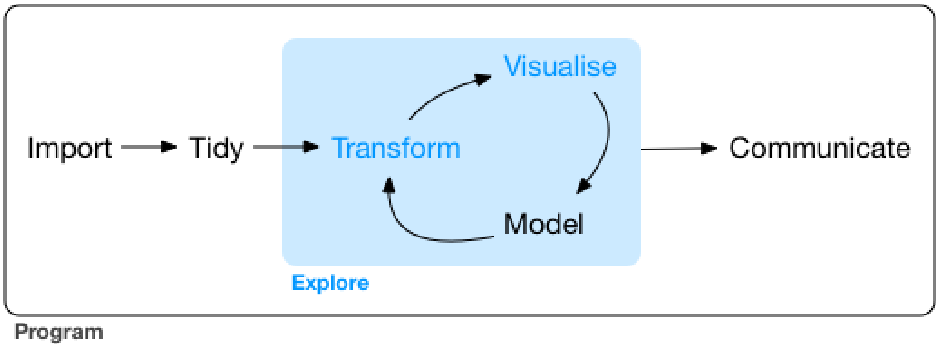


Figure 1: Image source: R4DS Chapter 2.

As usual, let's load the `tidyverse`:

```
library(tidyverse)
```

1 Data visualization

> R has several systems for making graphs, but `ggplot2` [one of the core members of the `tidyverse`] is one of the most elegant and most versatile. `ggplot2` implements the grammar of graphics, a coherent system for describing and building graphs. With `ggplot2`, you can do more faster by learning one system and applying it in many places.

1.1 `ggplot` basics

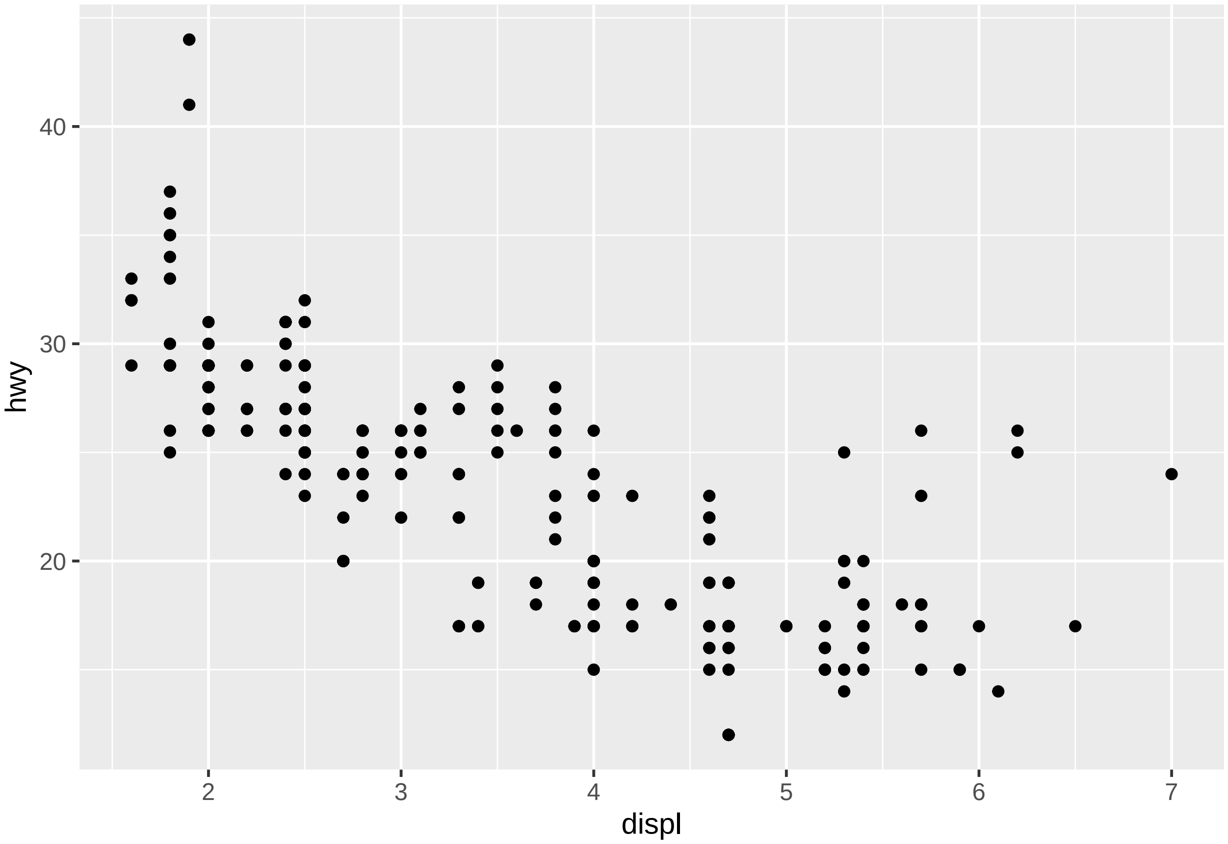
Let's recall the `mpg` data frame from last lecture:

```
mpg
```

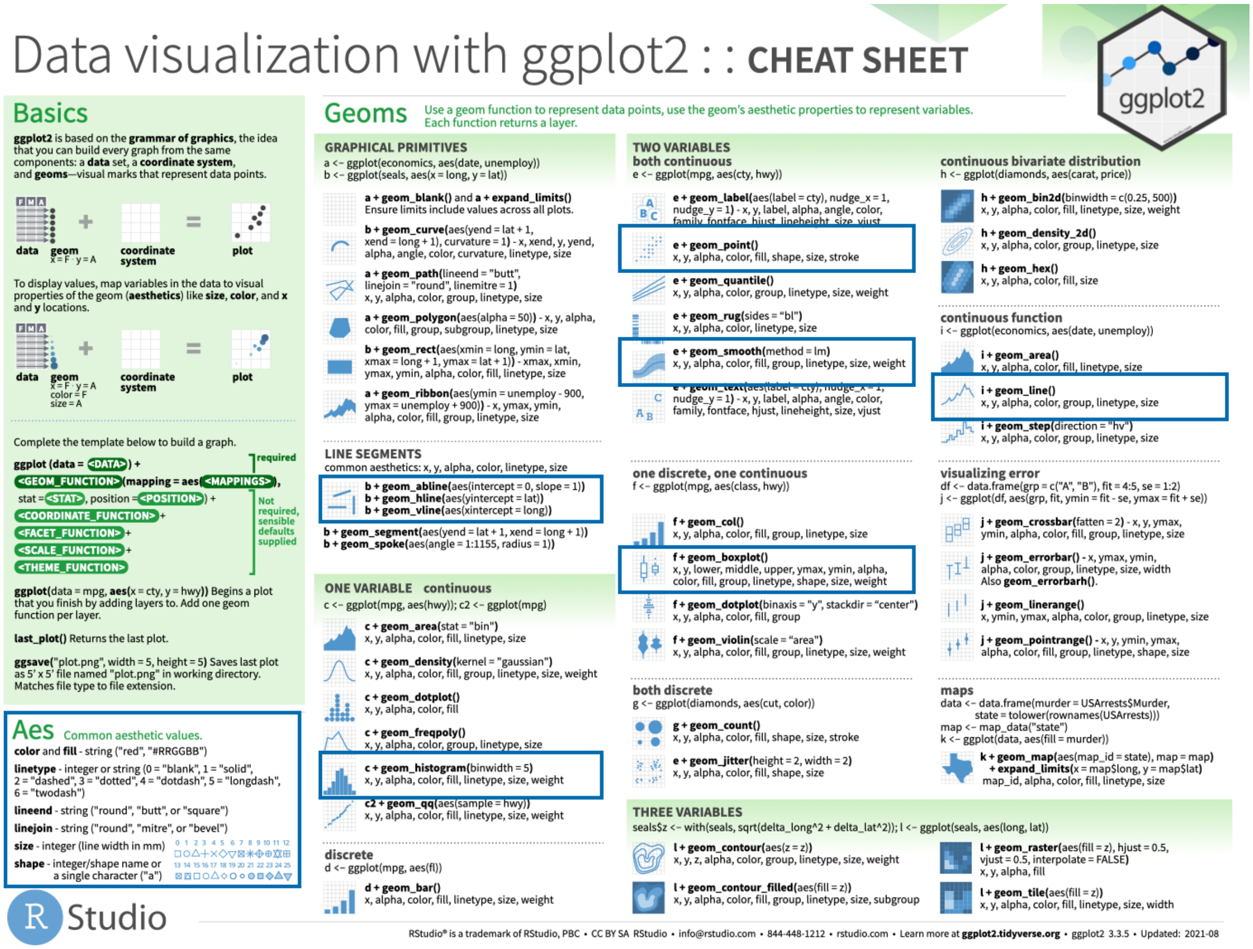
```
## # A tibble: 234 x 11
##    manufacturer model      displ  year   cyl trans drv     cty   hwy fl    class
##    <chr>        <chr>      <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
##  1 audi         a4           1.8  1999     4 auto~ f        18    29 p     comp~
##  2 audi         a4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

Let's plot the relationship between `displ` (a car's engine size in liters) and `hwy` (a car's fuel efficiency on the highway, in miles per gallon).

```
mpg %>%                           # pipe in the data
  ggplot() +                      # create an empty ggplot
  geom_point(mapping =            # add scatter plot
               aes(x = displ,     # x axis location of points
                   y = hwy))      # y axis location of points
```



An *aesthetic* is a visual property of the objects in your plot. We create a plot by *mapping* variables in our tibble to aesthetics of the plot. In the above case, we `displ` is mapped to `x` (the horizontal axis position) and `hwy` is mapped to `y` (the vertical axis position). A *geom* function adds a specific representation of the data to the plot (scatter plot, box plot, etc). In the above case, we used `geom_point` to create a scatter plot. A plot can have multiple geoms and multiple aesthetics. The plot above contains only one *panel*, but multi-panel plots can be created using *faceting*.



Data visualization with ggplot2 : : CHEAT SHEET

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.

F M A
+
=
data
geom
x = F · y = A
coordinate system
plot

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>, position = <POSITION>) +                   Not
<COORDINATE_FUNCTION> +                                    required,
<FACET_FUNCTION> +                                         sensible
<SCALE_FUNCTION> +                                         defaults
<THEME_FUNCTION>                                           supplied
```

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

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.

Aes Common aesthetic values.

color and **fill** - string ("red", "#RRGGBB")

linetype - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash")

lineend - string ("round", "butt", or "square")

linejoin - string ("round", "mitre", or "bevel")

size - integer (line width in mm)

shape - integer/shape name or a single character ("a")

0 1 2 3 4 5 6 7 8 9 10 11 12
□ ○ △ + × ◇ ▽ ⊠ ✳ ⊕ ⊕ ✡ ⊞
13 14 15 16 17 18 19 20 21 22 23 24 25
⊠ ◪ ■ ● ▲ ◆ ● • ● ■ ◆ ▲ ▼

RStudio

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() and **a + expand_limits()**
Ensure limits include values across all plots.

b + geom_curve(aes(yend = lat + 1, xend = long + 1), curvature = 1**)** - 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(alpha = 50)**)** - x, y, alpha, color, fill, group, subgroup, 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

both continuous

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

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1**)** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

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**)** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

one discrete, one continuous

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

both discrete

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

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

e + geom_jitter(height = 2, width = 2**)**
x, y, alpha, color, fill, shape, size

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_density_2d()
x, y, alpha, color, group, linetype, size

h + geom_hex()
x, y, alpha, color, 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, color, group, linetype, size, weight

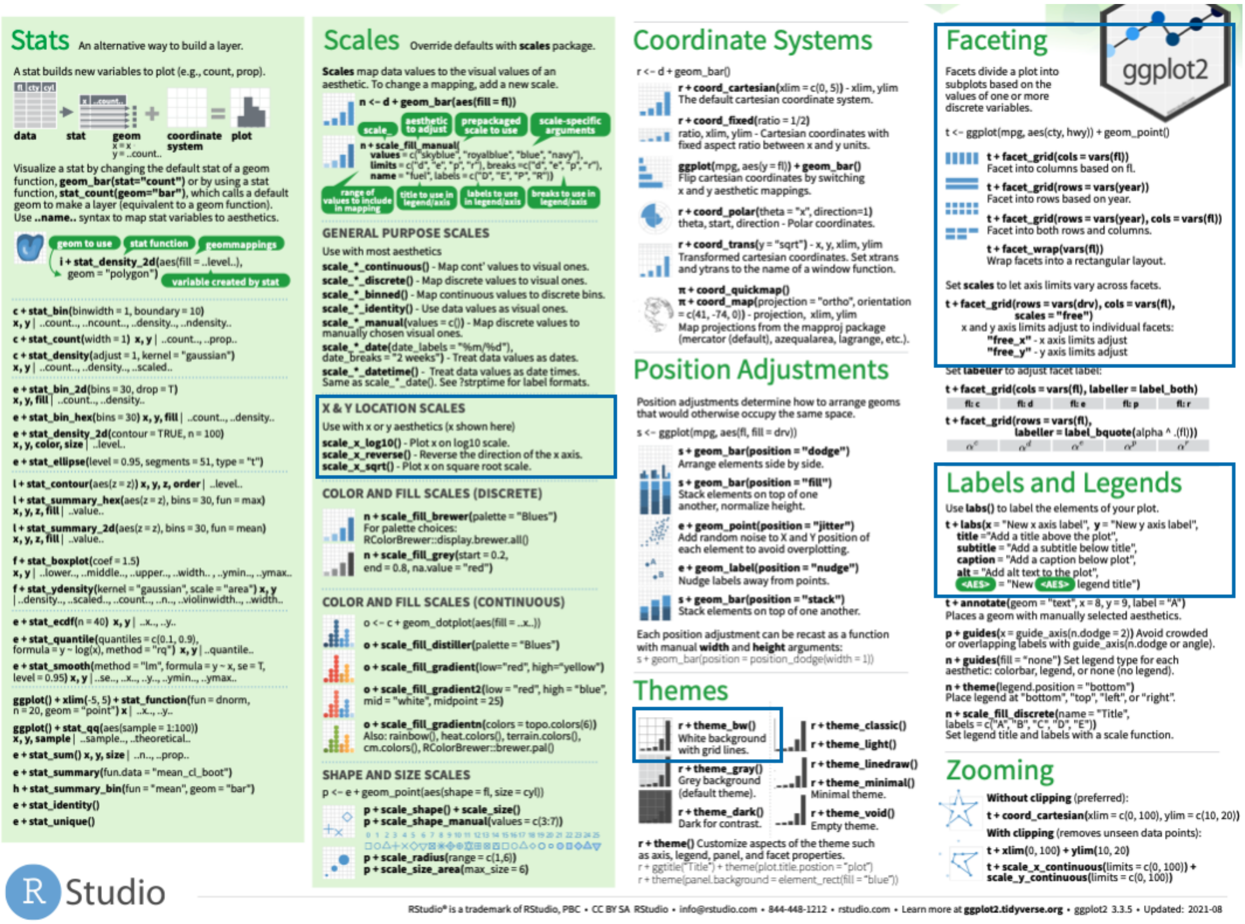
l + geom_contour_filled(aes(fill = z)**)**
x, y, alpha, color, fill, group, linetype, size, subgroup

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

RStudio® is a trademark of RStudio, PBC • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **ggplot2.tidyverse.org** • ggplot2 3.3.5 • Updated: 2021-08

Figure 2: Image source: https://www.rstudio.com/resources/cheatsheets/



Stats An alternative way to build a layer.

A stat builds new variables to plot (e.g., count, prop).

fl cty cyl
x ..count..
data
stat
geom
x = x
y = ..count..
coordinate system
plot

Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.

geom to use
stat function
geommappings
variable created by stat

i + stat_density_2d(aes(fill = ..level..), geom = "polygon")

c + stat_bin(binwidth = 1, boundary = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1) **x, y** | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian")
x, y | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins = 30) **x, y, fill** | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) **x, y, z, order** | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5)
x, y | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = "area") **x, y**
| ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40) **x, y** | ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq") **x, y** | ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se = T, level = 0.95) **x, y** | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + xlim(-5, 5) **+ stat_function**(fun = dnorm, n = 20, geom = "point") **x** | ..x.., ..y..

ggplot() + stat_qq(aes(sample = 1:100))
x, y, sample | ..sample.., ..theoretical..

e + stat_sum() x, y, size | ..n.., ..prop..

e + stat_summary(fun.data = "mean_cl_boot")

h + stat_summary_bin(fun = "mean", geom = "bar")

e + stat_identity()

e + stat_unique()

RStudio

Scales Override defaults with **scales** package.

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

n <- d + geom_bar(aes(fill = fl))

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping
title to use in legend/axis
labels to use in legend/axis
breaks to use in legend/axis

GENERAL PURPOSE SCALES

Use with most aesthetics

scale_*_continuous() - Map cont' values to visual ones.
scale_*_discrete() - Map discrete values to visual ones.
scale_*_binned() - Map continuous values to discrete bins.
scale_*_identity() - Use data values as visual ones.
scale_*_manual(values = c()) - Map discrete values to manually chosen visual ones.
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks") - Treat data values as dates.
scale_*_datetime() - Treat data values as date times. Same as scale_*_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale.
scale_x_reverse() - Reverse the direction of the x axis.
scale_x_sqrt() - Plot x on square root scale.

COLOR AND FILL SCALES (DISCRETE)

n + scale_fill_brewer(palette = "Blues")
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

COLOR AND FILL SCALES (CONTINUOUS)

o <- c + geom_dotplot(aes(fill = ..x..))

o + scale_fill_distiller(palette = "Blues")

o + scale_fill_gradient(low="red", high="yellow")

o + scale_fill_gradient2(low = "red", high = "blue", mid = "white", midpoint = 25)

o + scale_fill_gradientn(colors = topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES

p <- e + geom_point(aes(shape = fl, size = cyl))

p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7))

p + scale_radius(range = c(1,6))
p + scale_size_area(max_size = 6)

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5)) - xlim, ylim
The default cartesian coordinate system.

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim - Cartesian coordinates with fixed aspect ratio between x and y units.

ggplot(mpg, aes(y = fl)) **+ geom_bar()**
Flip cartesian coordinates by switching x and y aesthetic mappings.

r + coord_polar(theta = "x", direction=1)
theta, start, direction - Polar coordinates.

r + coord_trans(y = "sqrt") - x, y, xlim, ylim
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation = c(41, -74, 0)) - projection, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.).

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Arrange elements side by side.

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height.

e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting.

e + geom_label(position = "nudge")
Nudge labels away from points.

s + geom_bar(position = "stack")
Stack elements on top of one another.

Each position adjustment can be recast as a function with manual **width** and **height** arguments:
s + geom_bar(position = position_dodge(width = 1))

Themes

r + theme_bw()
White background with grid lines.

r + theme_gray()
Grey background (default theme).

r + theme_dark()
Dark for contrast.

r + theme_classic()

r + theme_light()

r + theme_linedraw()

r + theme_minimal()
Minimal theme.

r + theme_void()
Empty theme.

r + theme() Customize aspects of the theme such as axis, legend, panel, and facet properties.
r + ggtitle("Title") + theme(plot.title.postion = "plot")
r + theme(panel.background = element_rect(fill = "blue"))

Faceting

ggplot2

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(cols = vars(fl))
Facet into columns based on fl.

t + facet_grid(rows = vars(year))
Facet into rows based on year.

t + facet_grid(rows = vars(year), cols = vars(fl))
Facet into both rows and columns.

t + facet_wrap(vars(fl))
Wrap facets into a rectangular layout.

Set **scales** to let axis limits vary across facets.

t + facet_grid(rows = vars(drv), cols = vars(fl),
scales = "free")
x and y axis limits adjust to individual facets:
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet label:

t + facet_grid(cols = vars(fl), labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(rows = vars(fl),
labeller = label_bquote(alpha ^ .(fl)))

α^c	α^d	α^e	α^p	α^r

Labels and Legends

Use **labs()** to label the elements of your plot.

t + labs(x = "New x axis label", **y** = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
alt = "Add alt text to the plot",
<AES> = "New <AES> legend title")

t + annotate(geom = "text", x = 8, y = 9, label = "A")
Places a geom with manually selected aesthetics.

p + guides(x = guide_axis(n.dodge = 2)) Avoid crowded or overlapping labels with guide_axis(n.dodge or angle).

n + guides(fill = "none") Set legend type for each aesthetic: colorbar, legend, or none (no legend).

n + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right".

n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred):

t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points):

t + xlim(0, 100) **+ ylim**(10, 20)

t + scale_x_continuous(limits = c(0, 100)) **+ scale_y_continuous**(limits = c(0, 100))

RStudio® is a trademark of RStudio, PBC • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **ggplot2.tidyverse.org** • ggplot2 3.3.5 • Updated: 2021-08

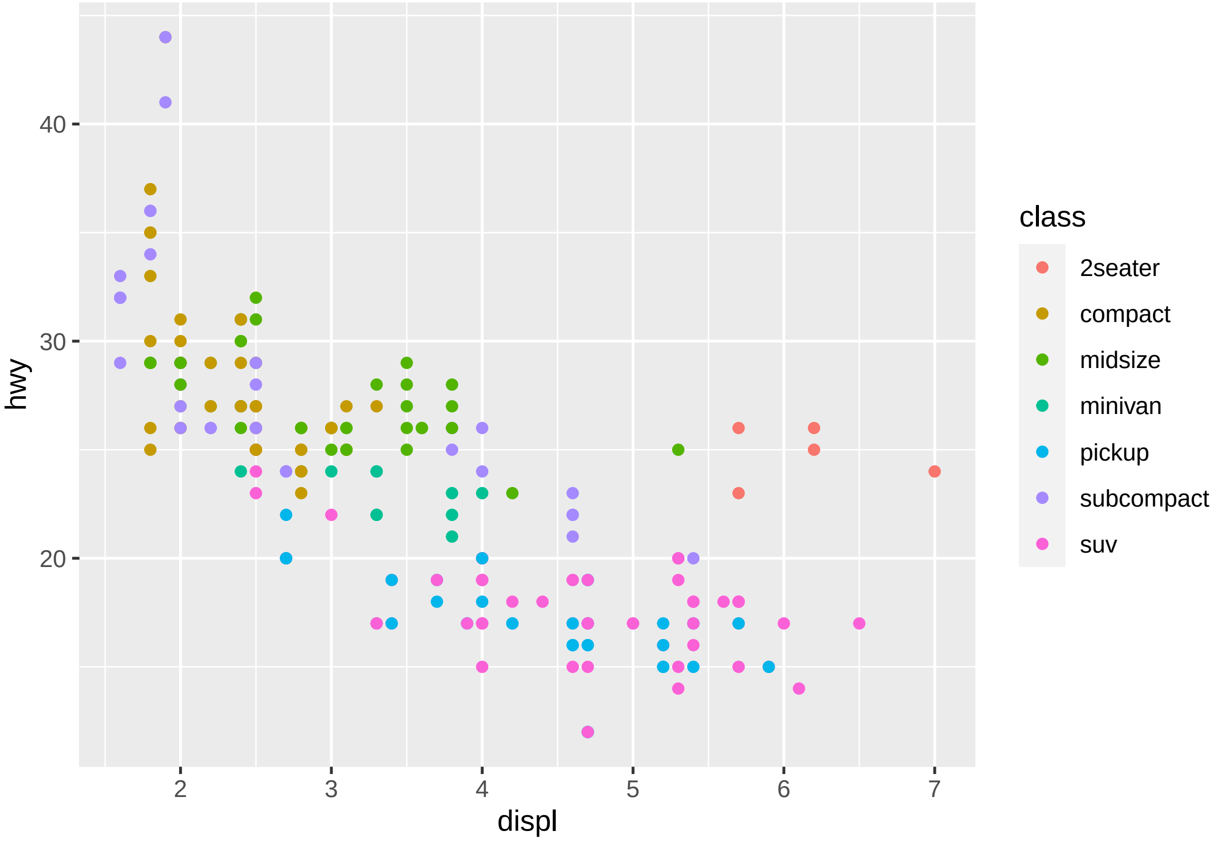
Figure 3: Image source: https://www.rstudio.com/resources/cheatsheets/

1.2 Aesthetics

Let's see some examples of different aesthetics we can add to the above scatter plot.

Adding a color aesthetic:

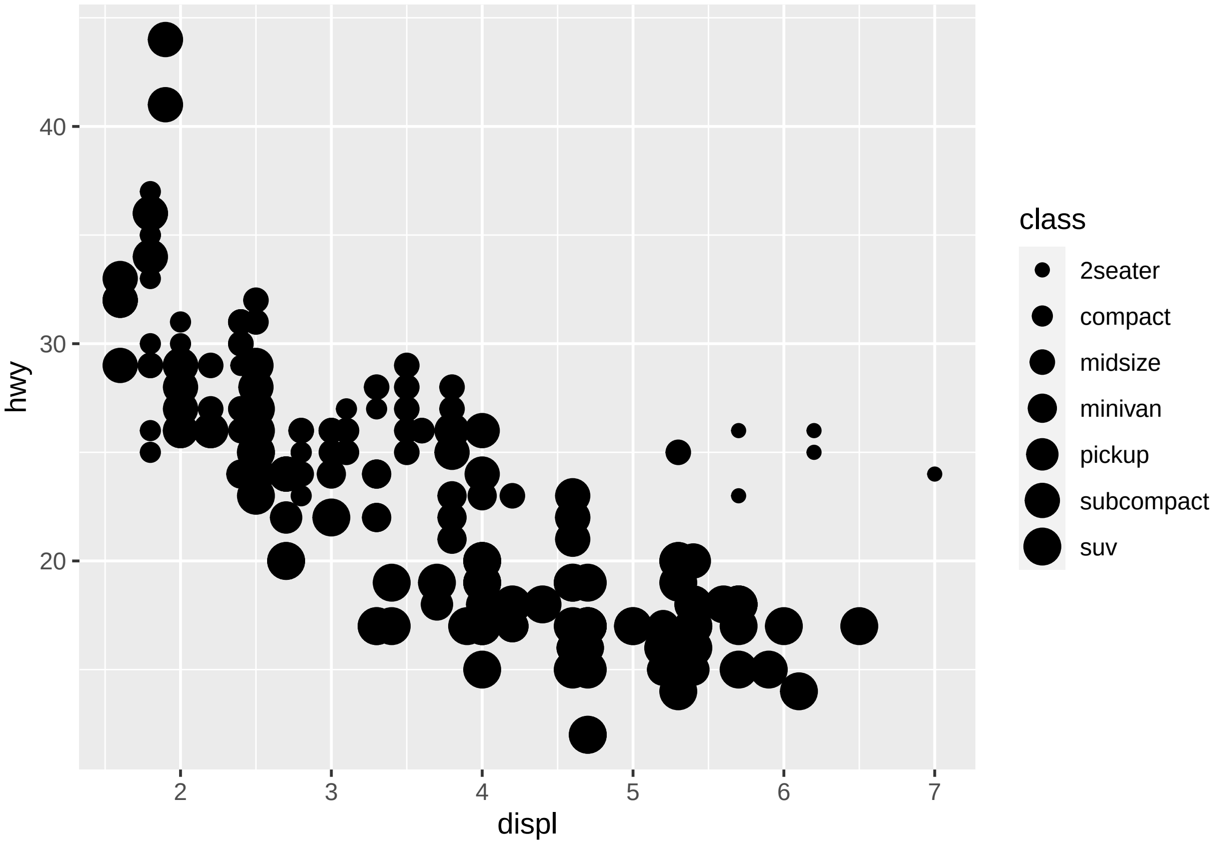
```
mpg %>%                            # pipe in the data
  ggplot() +                       # create an empty ggplot
  geom_point(mapping =             # add scatter plot
               aes(x = displ,      # x axis location of points
                   y = hwy,        # y axis location of points
                   color = class)) # color of points
```



Adding a size aesthetic:

```
mpg %>%                            # pipe in the data
  ggplot() +                       # create an empty ggplot
  geom_point(mapping =             # add scatter plot
               aes(x = displ,      # x axis location of points
                   y = hwy,        # y axis location of points
                   size = class))  # size of points
```

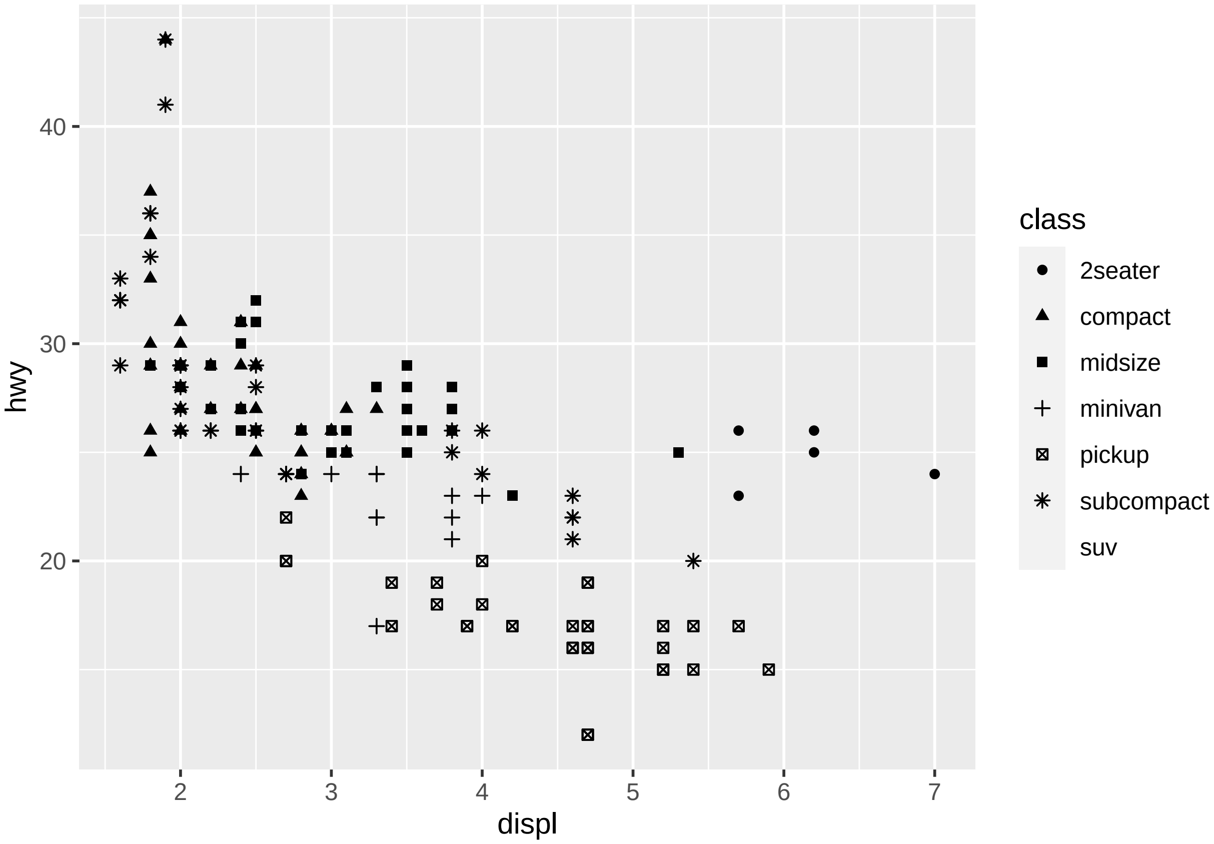
```
## Warning: Using size for a discrete variable is not advised.
```



Adding a shape aesthetic:

```
mpg %>%                            # pipe in the data
  ggplot() +                       # create an empty ggplot
  geom_point(mapping =             # add scatter plot
             aes(x = displ,        # x axis location of points
                 y = hwy,          # y axis location of points
                 shape = class)) # size of points
```

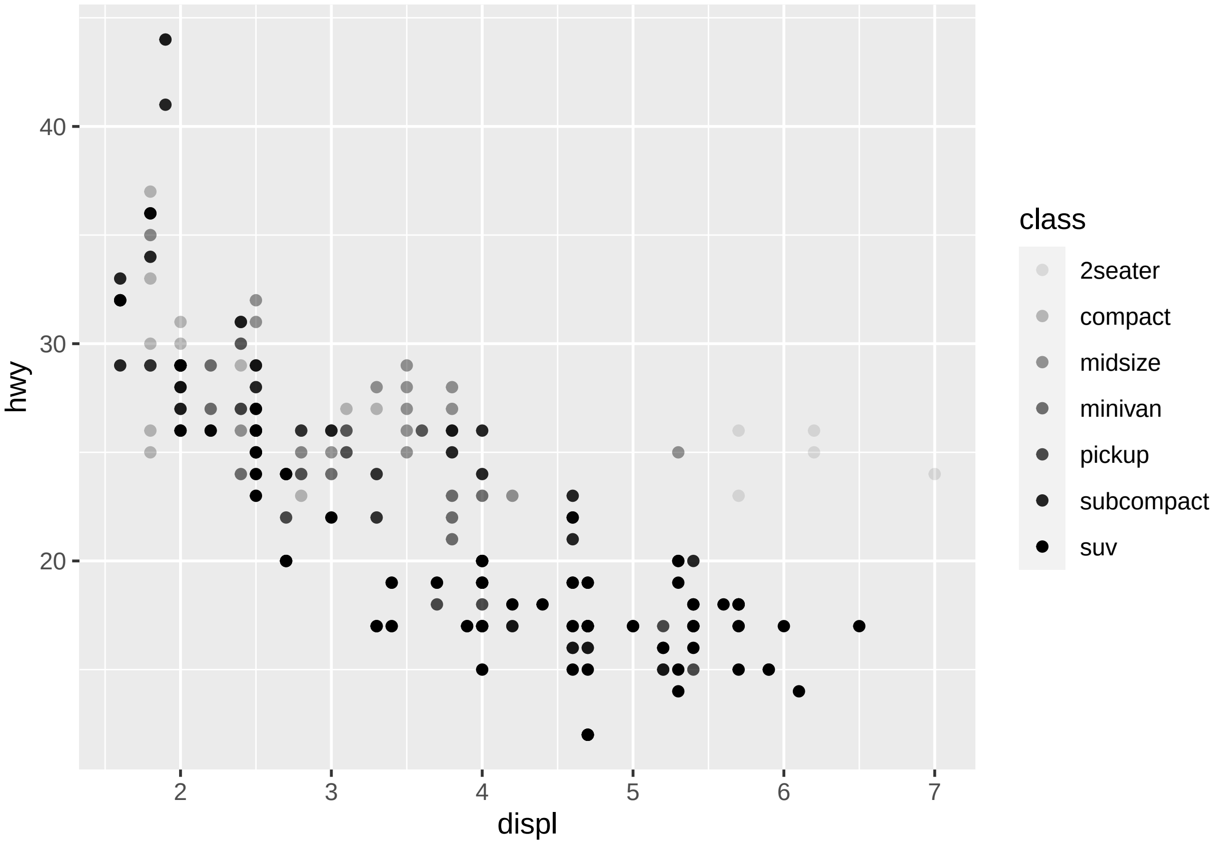
```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 7. Consider
## specifying shapes manually if you must have them.

## Warning: Removed 62 rows containing missing values (geom_point).
```



Adding a transparency aesthetic:

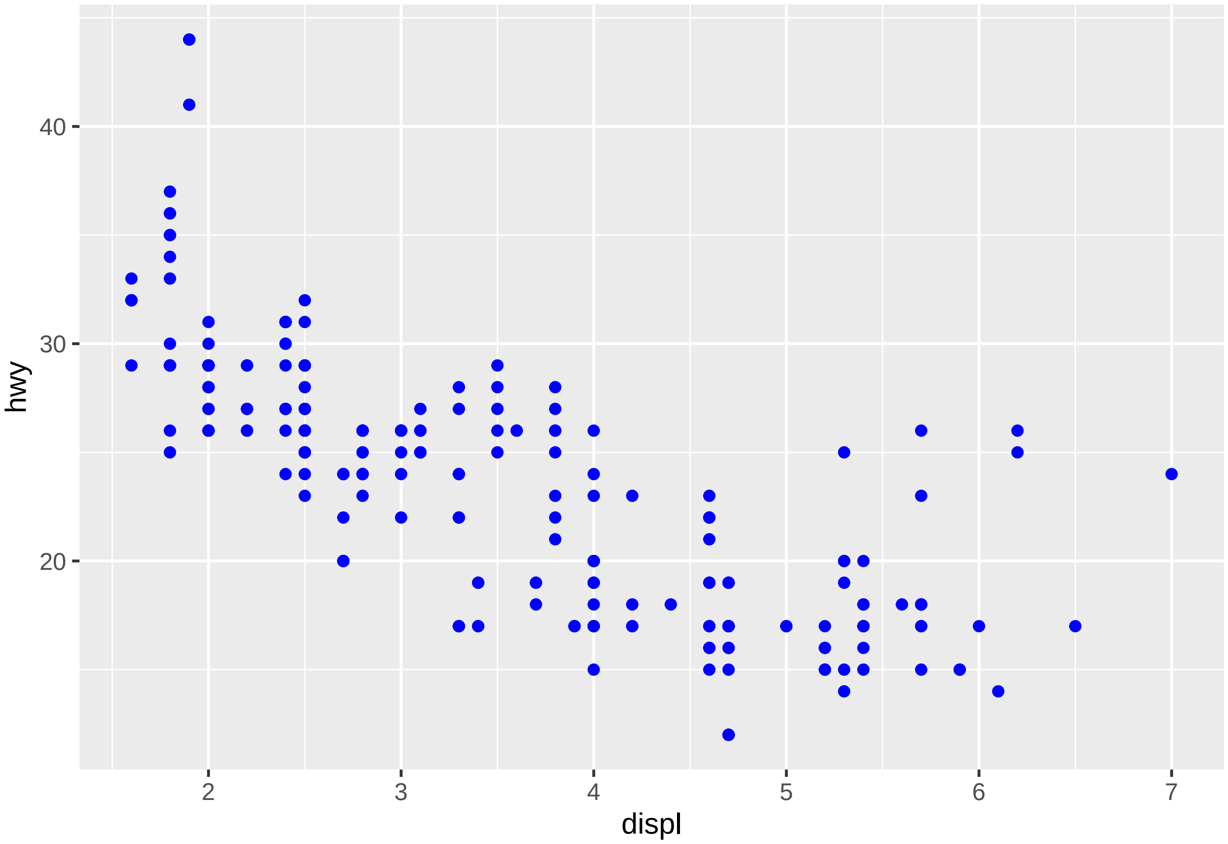
```
mpg %>%                          # pipe in the data
  ggplot() +                     # create an empty ggplot
  geom_point(mapping =           # add scatter plot
             aes(x = displ,      # x axis location of points
                 y = hwy,        # y axis location of points
                 alpha = class)) # transparency of points
```

```
## Warning: Using alpha for a discrete variable is not advised.
```



Specifying an aesthetic manually, instead of mapping from a variable:

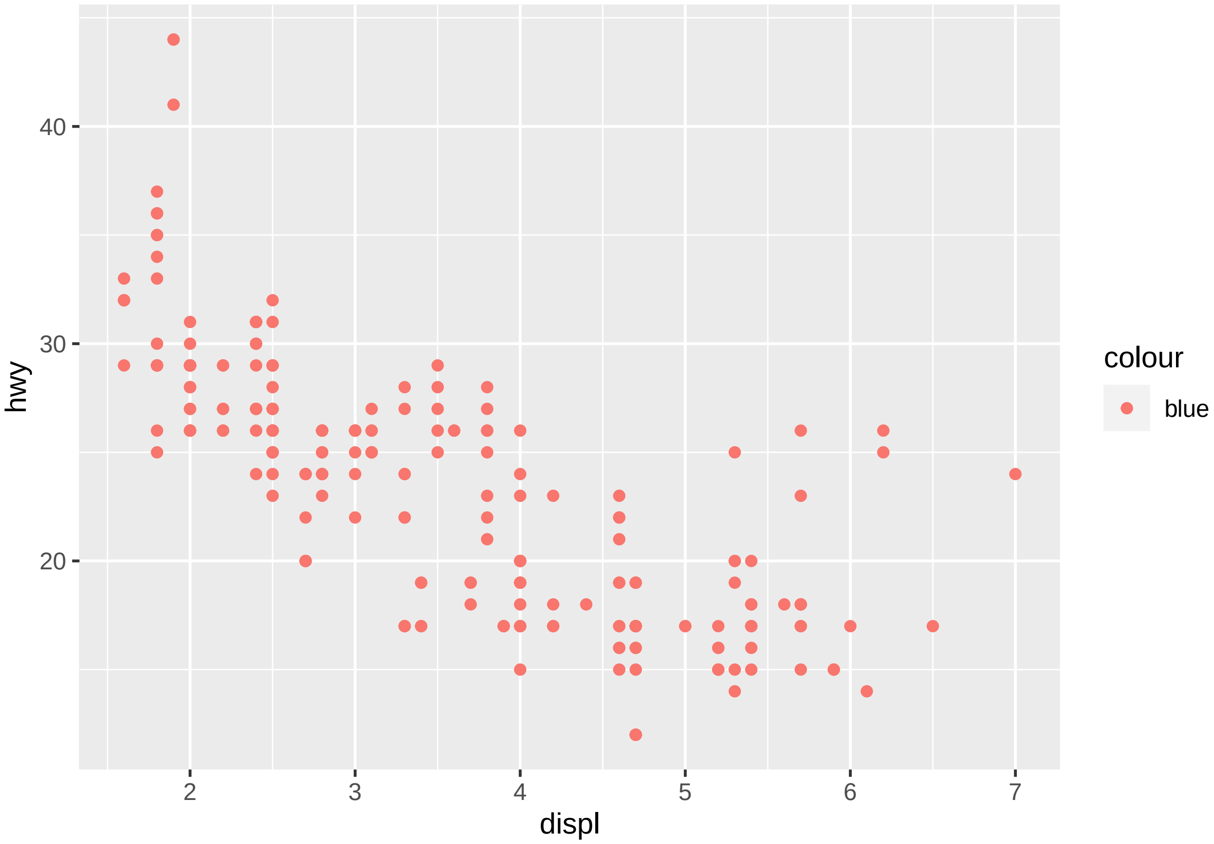
```
mpg %>%                                # pipe in the data
  ggplot() +                           # create an empty ggplot
  geom_point(mapping =                 # add scatter plot
             aes(x = displ,            # x axis location of points
                 y = hwy),             # y axis location of points
             color = "blue")           # color of points (outside of aes)
```



1.3 Exercises

1. What's gone wrong with this code? Why are the points not blue?

```
mpg %>%
  ggplot() +
  geom_point(mapping = aes(x = displ, y = hwy, color = "blue"))
```



2. Map a continuous variable to `color`, `size`, and `shape`. How do these aesthetics behave differently for categorical vs. continuous variables?

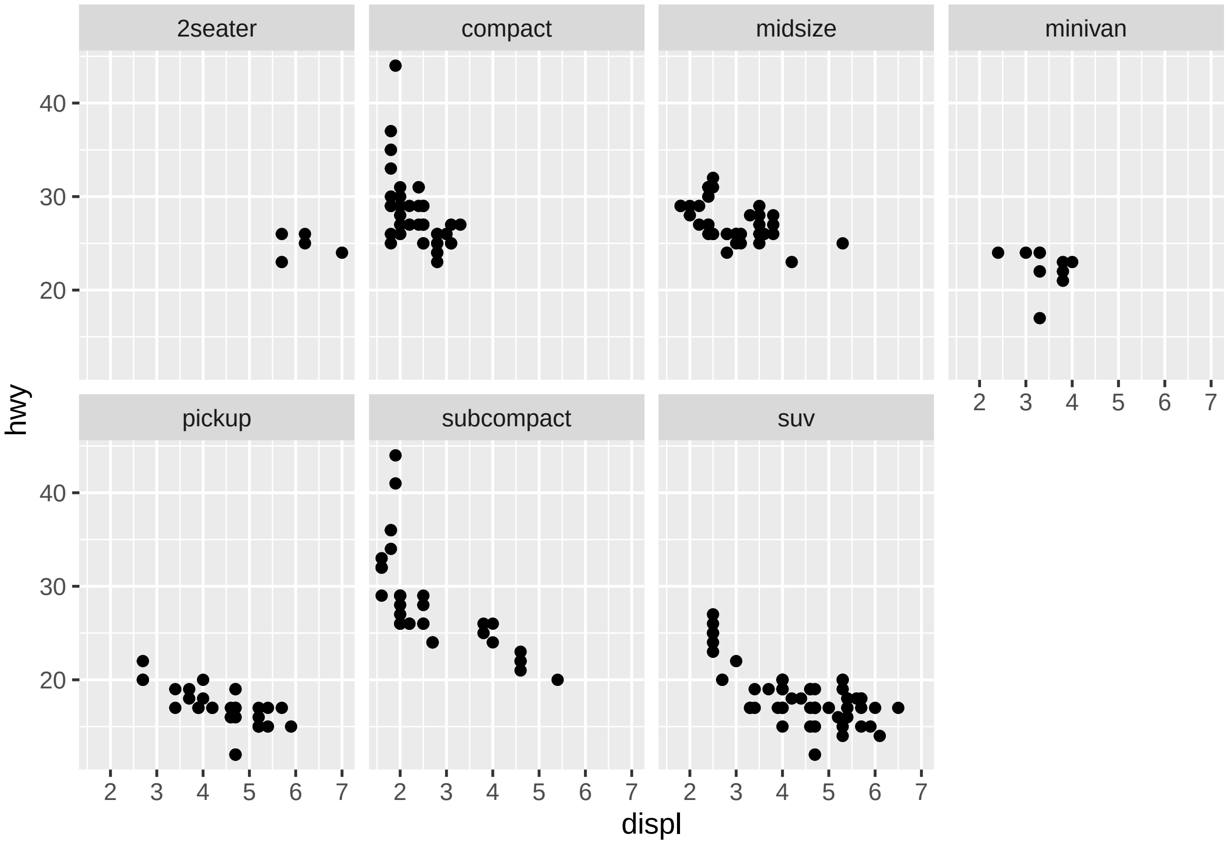
3. What happens if you map the same variable to multiple aesthetics?

4. What happens if you map an aesthetic to something other than a variable name, like `aes(colour = displ < 5)`? Note, you'll also need to specify `x` and `y`.

1.4 Facets

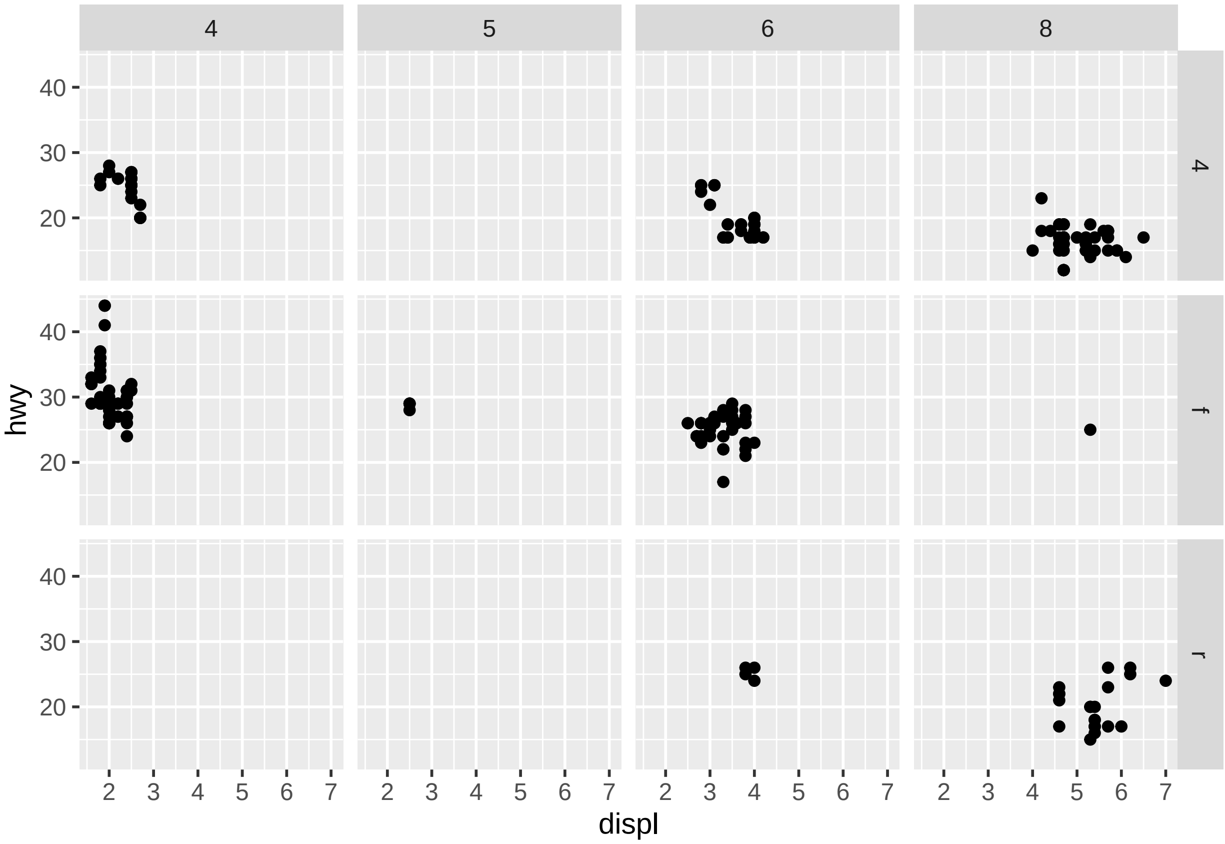
To facet your plot by a single variable, use `facet_wrap()`. The first argument of `facet_wrap()` should be a formula, which you create with `~` followed by a variable name (here "formula" is the name of a data structure in R, not a synonym for "equation"). The variable that you pass to `facet_wrap()` should be discrete.

```
mpg %>%                          # pipe in the data
  ggplot() +                     # create empty ggplot
  geom_point(mapping =           # create scatter plot
             aes(x = displ,      # map displ to x axis location
                 y = hwy)) +     # map hwy to y axis location
  facet_wrap(~ class,            # split into facets based on class
          nrow = 2)              # have two rows of facets
```



To facet your plot on the combination of two variables, add `facet_grid()` to your plot call. The first argument of `facet_grid()` is also a formula. This time the formula should contain two variable names separated by a `~`.

```
mpg %>%
  ggplot() +
  geom_point(mapping =
               aes(x = displ, y = hwy)) +
  facet_grid(drv ~ cyl)                     # facet on drv and cyl
```



If you prefer to not facet in the rows or columns dimension, use a `.` instead of a variable name, e.g. `+ facet_grid(. ~ cyl)`.

1.5 Exercises

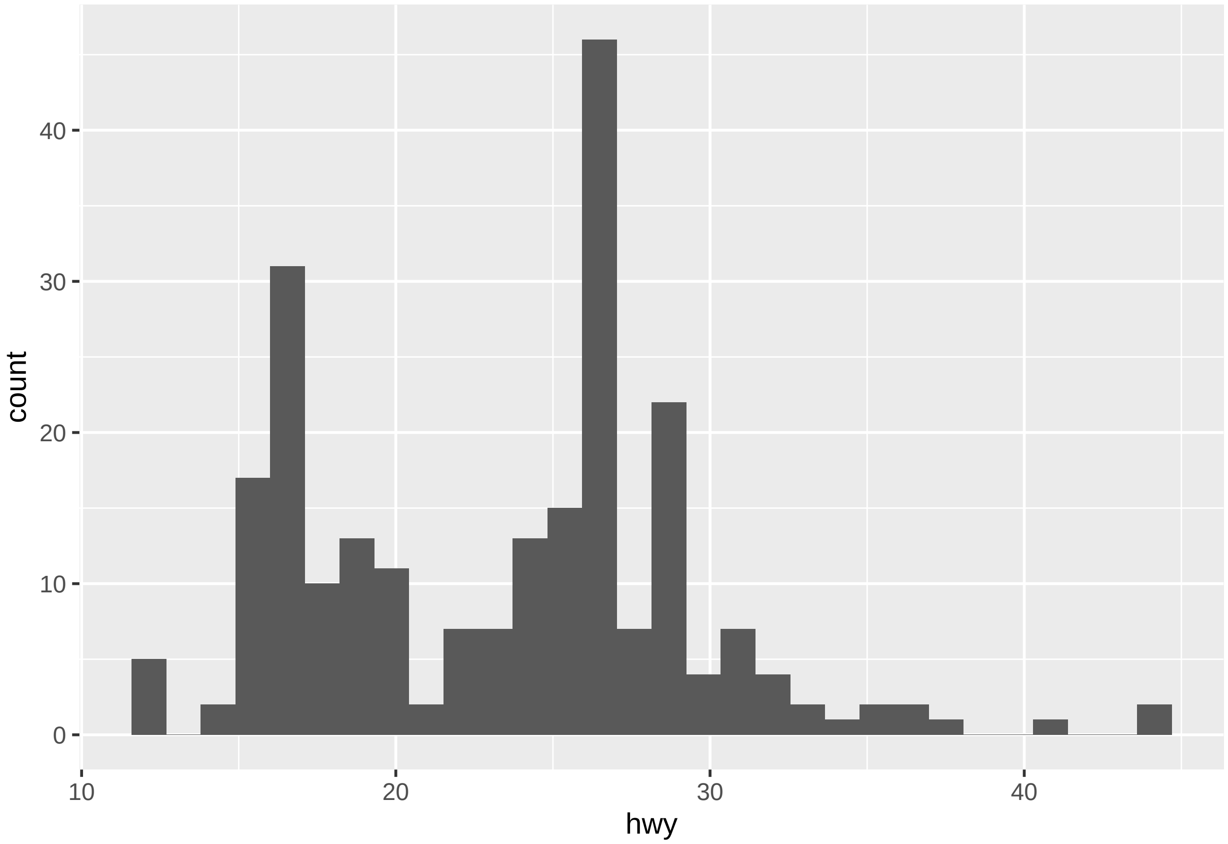
1. What happens if you facet on a continuous variable?
2. Why are there empty facets in the plot with `facet_grid(drv ~ cyl)`?
3. Read `?facet_wrap`. What does `nrow` do? What does `ncol` do? Why doesn't `facet_grid()` have `nrow` and `ncol` arguments?

1.6 geoms

To visualize the distribution of a quantitative variable:

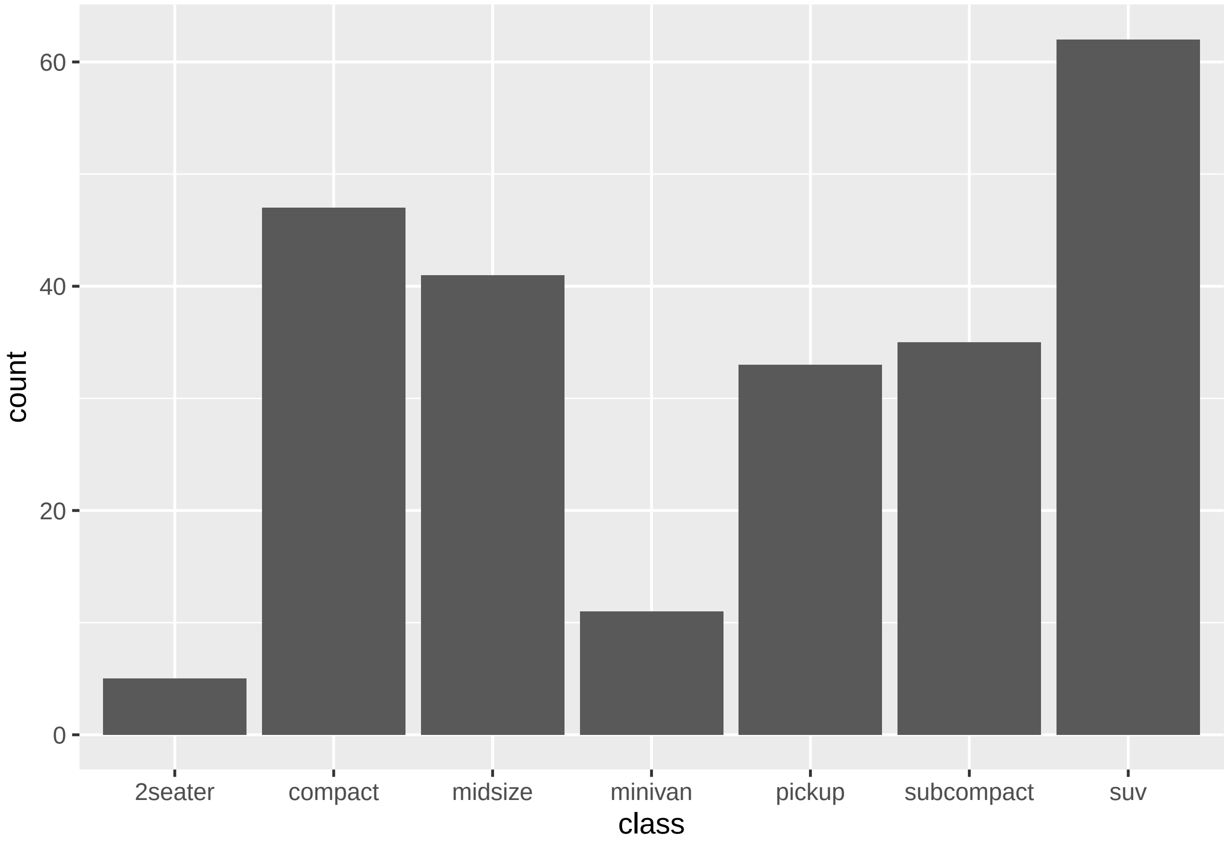
```
mpg %>%
  ggplot() +
  geom_histogram(aes(x = hwy)) # we usually drop "mapping ="
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

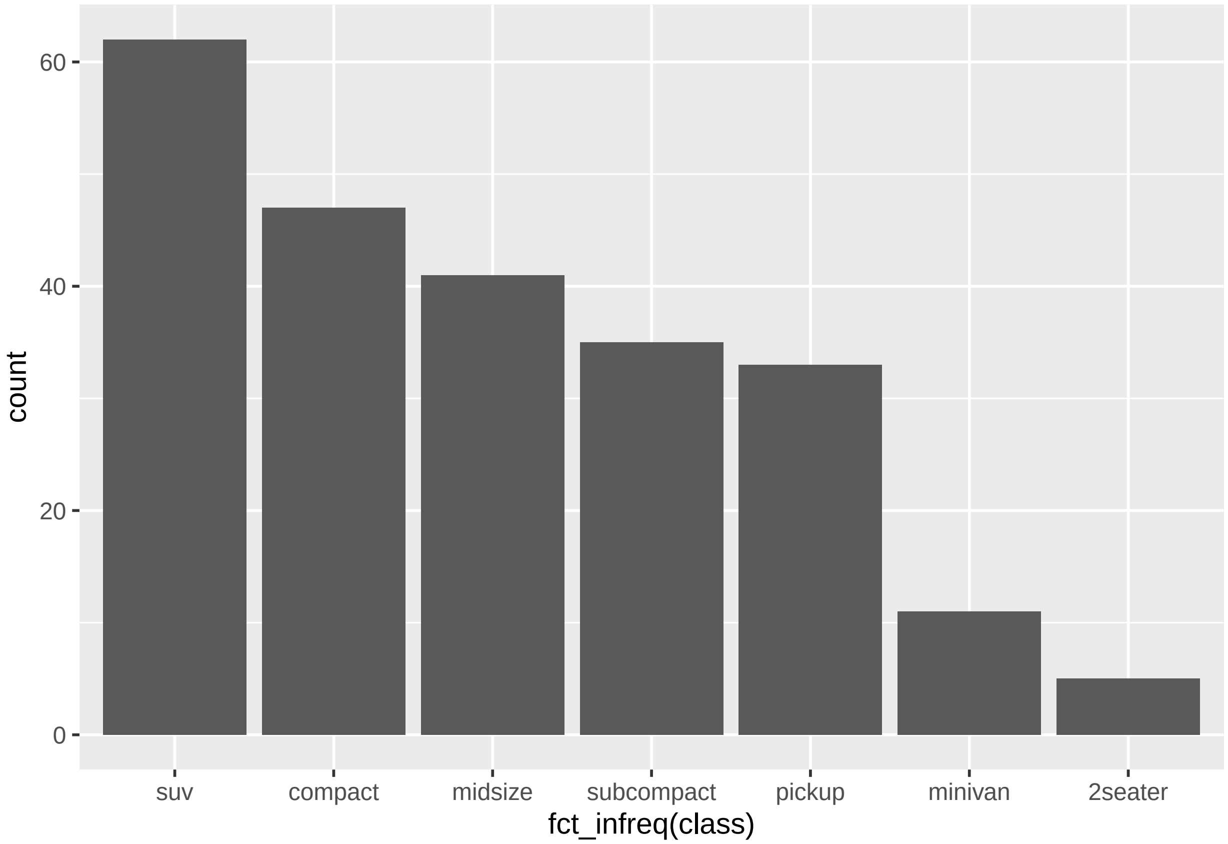


To visualize the distribution of a categorical variable:

```
# bar plot
mpg %>%
  ggplot() +
  geom_bar(aes(x = class))
```

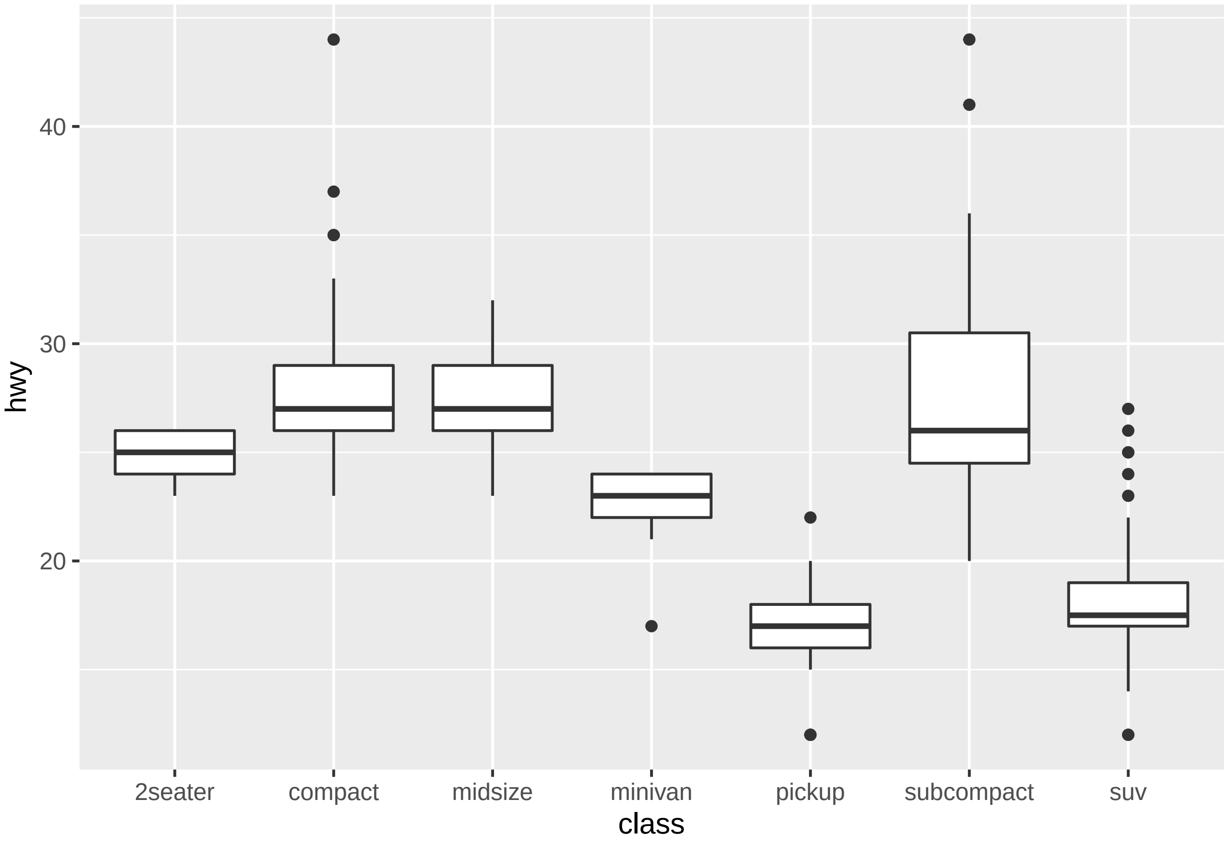


```
# better to reorder class variable
mpg %>%
  ggplot() +
  geom_bar(aes(x = fct_infreq(class)))
```

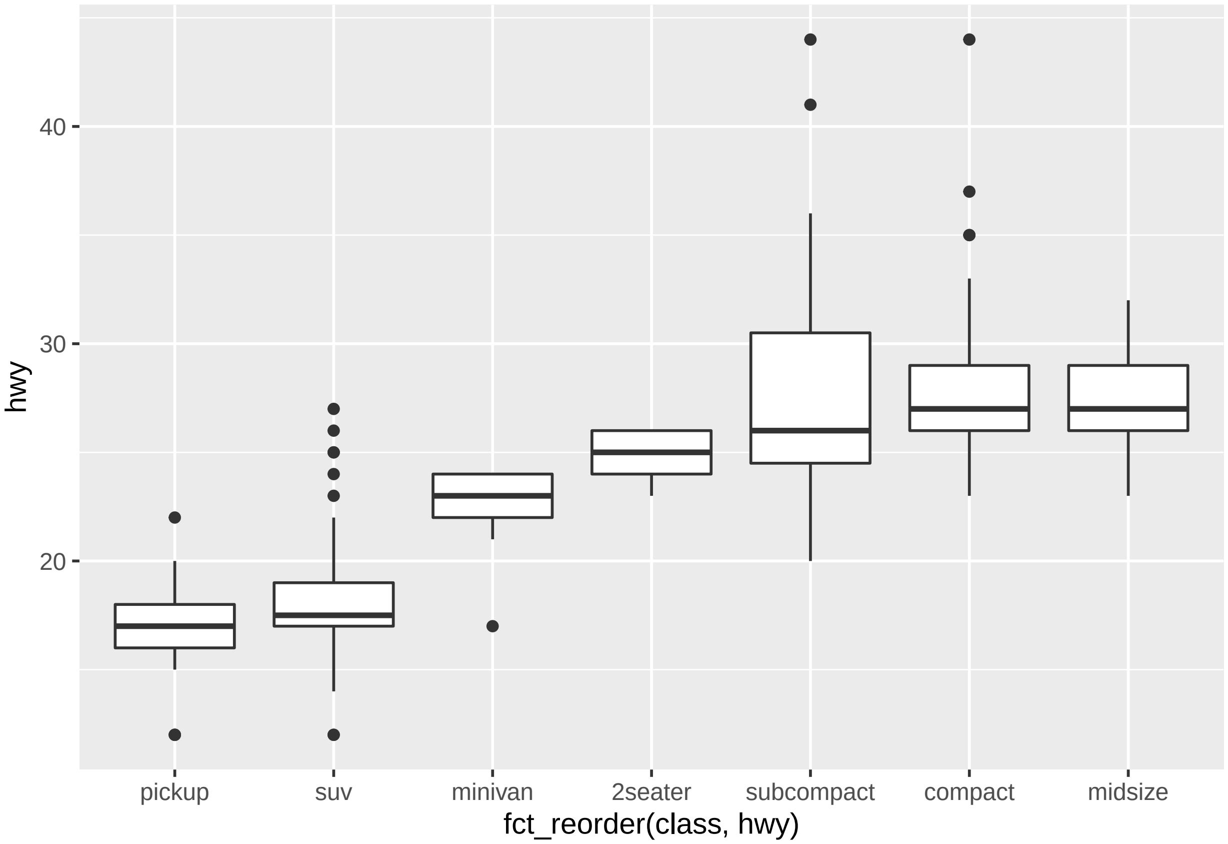


To visualize the relationship between a quantitative and categorical variable:

```
# boxplot
mpg %>%
  ggplot() +
  geom_boxplot(aes(x = class, y = hwy))
```



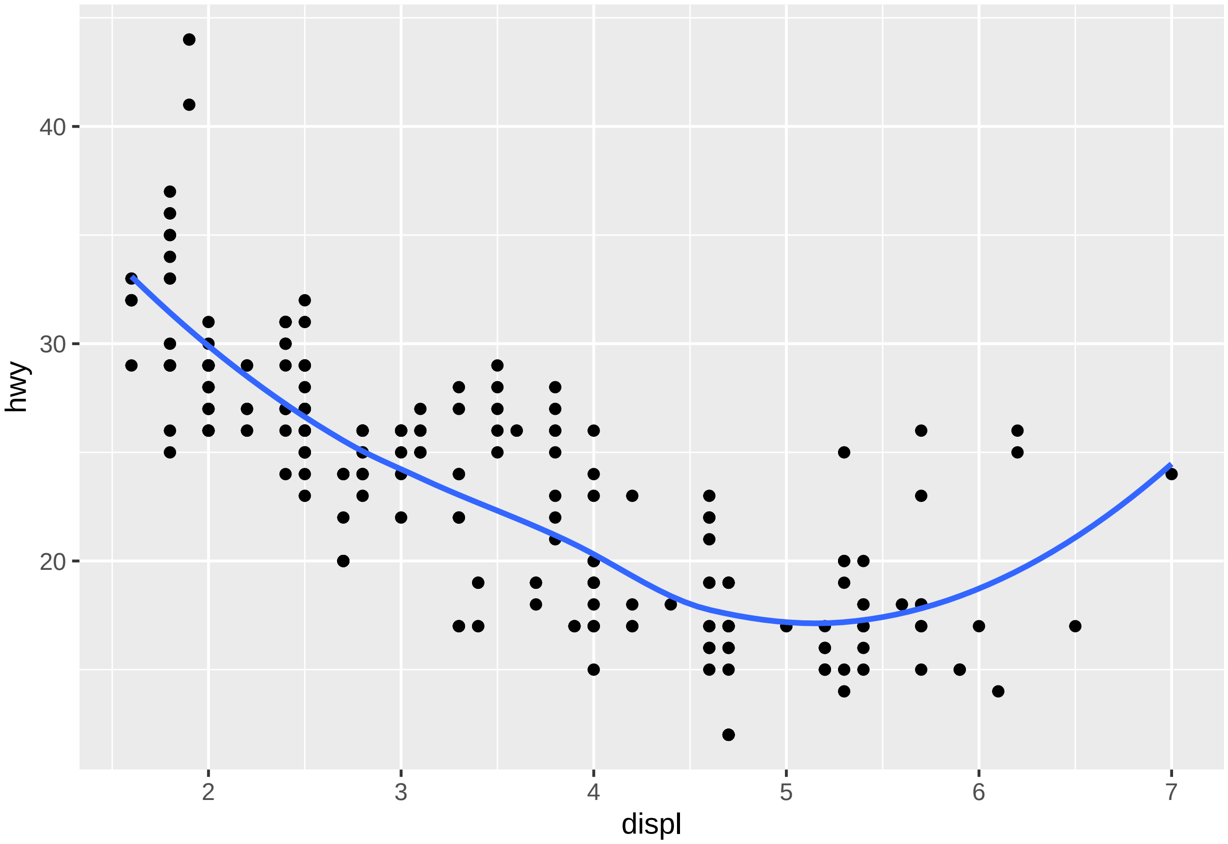
```
# better to reorder factors based on hwy
mpg %>%
  ggplot() +
  geom_boxplot(aes(x = fct_reorder(class, hwy), y = hwy))
```



To add a smooth curve:

```
mpg %>%
  ggplot() +
  geom_point(aes(x = displ, y = hwy)) +                # create scatter plot
  geom_smooth(aes(x = displ, y = hwy), se = FALSE)    # add smooth curve
```

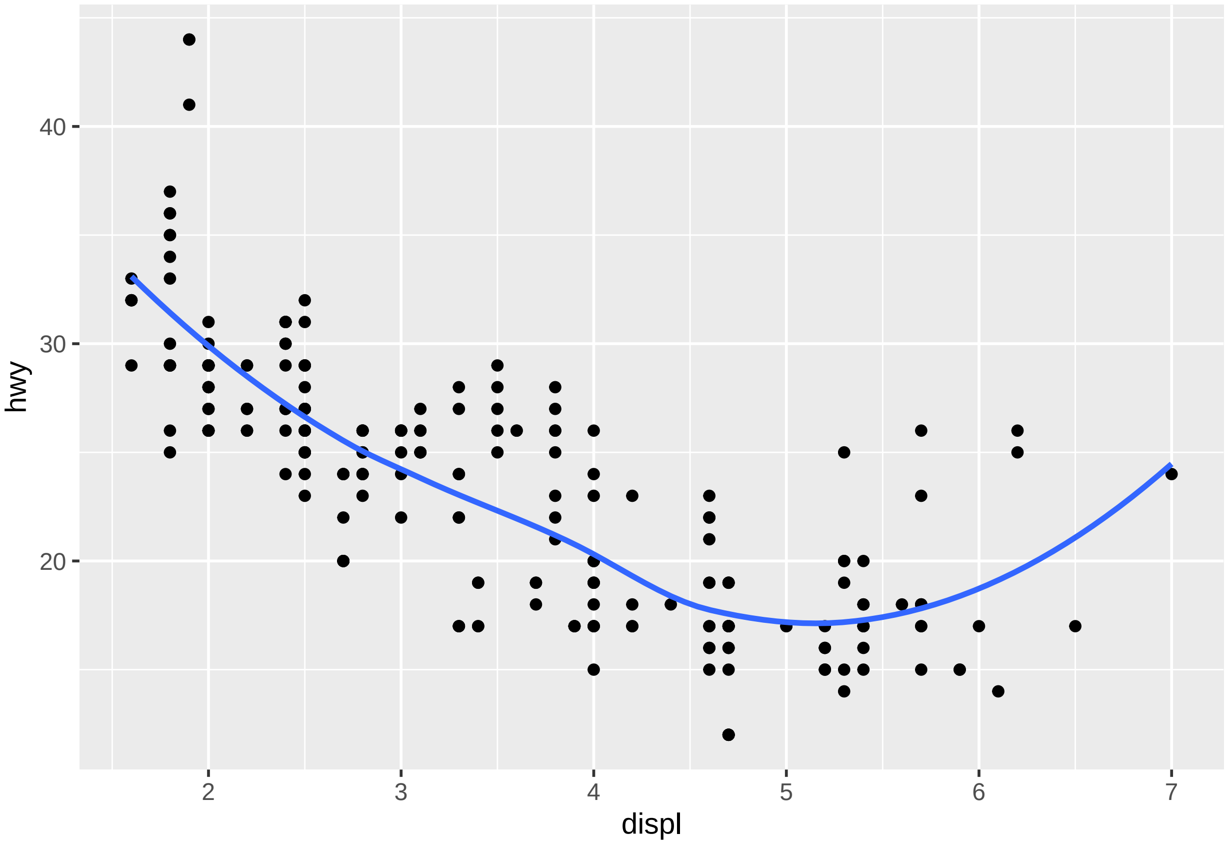
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Can instead set aesthetic mapping *globally*:

```
mpg %>%
  ggplot(aes(x = displ, y = hwy)) + # set aesthetic mapping globally
  geom_point() +                    # add scatter plot
  geom_smooth(se = FALSE)           # add smooth curve
```

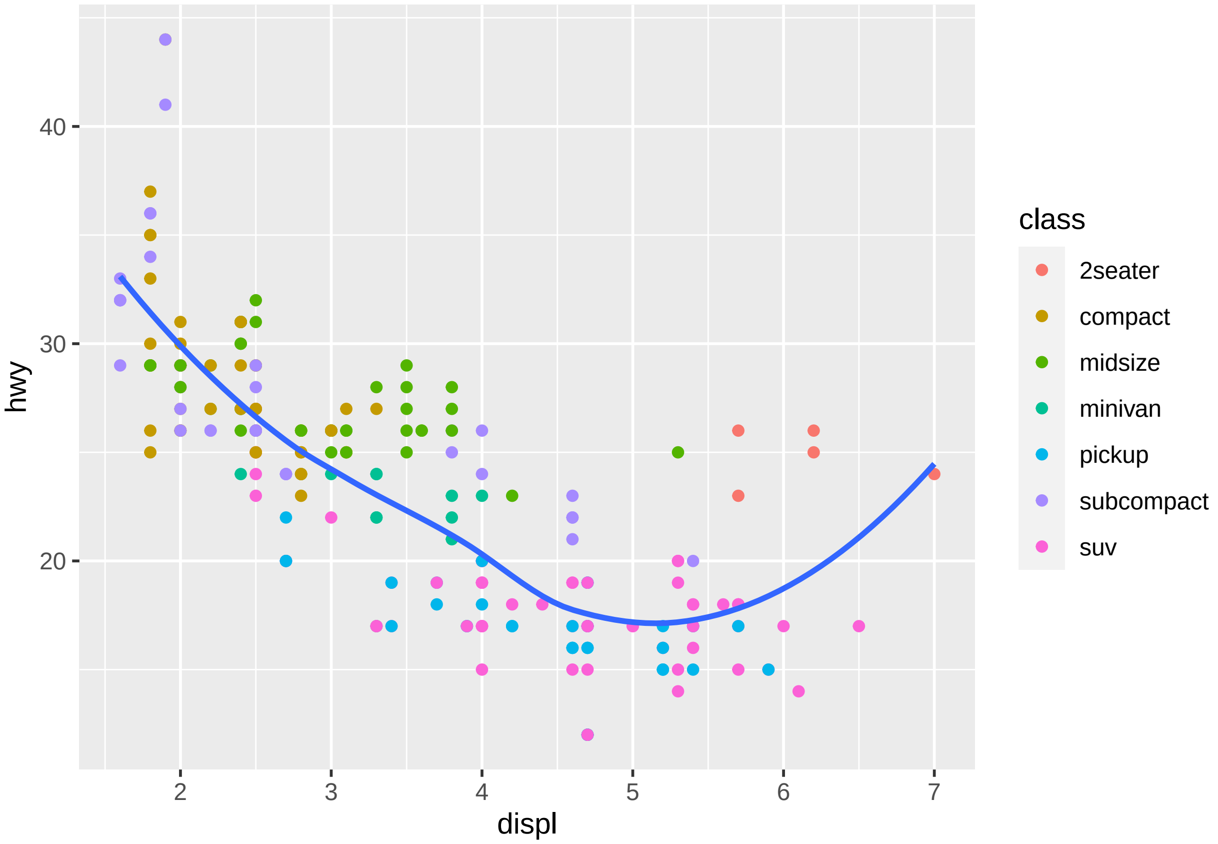
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Can set some aesthetic mappings globally and some locally:

```
mpg %>%
  ggplot(aes(x = displ, y = hwy)) + # set x and y aesthetic mappings globally
  geom_point(aes(color = class)) +  # set color aesthetic mapping locally
  geom_smooth(se = FALSE)           # add smooth curve
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



If we had set `color` aesthetic globally, ggplot would separate smooth curves by `class`:

```
mpg %>%
  ggplot(aes(x = displ, y = hwy, color = class)) + # set x, y, color aesthetic
  geom_point() +                                   #  mappings globally
  geom_smooth(se = FALSE)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : span too small. fewer data values than degrees of freedom.
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : pseudoinverse used at 5.6935
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : neighborhood radius 0.5065
```

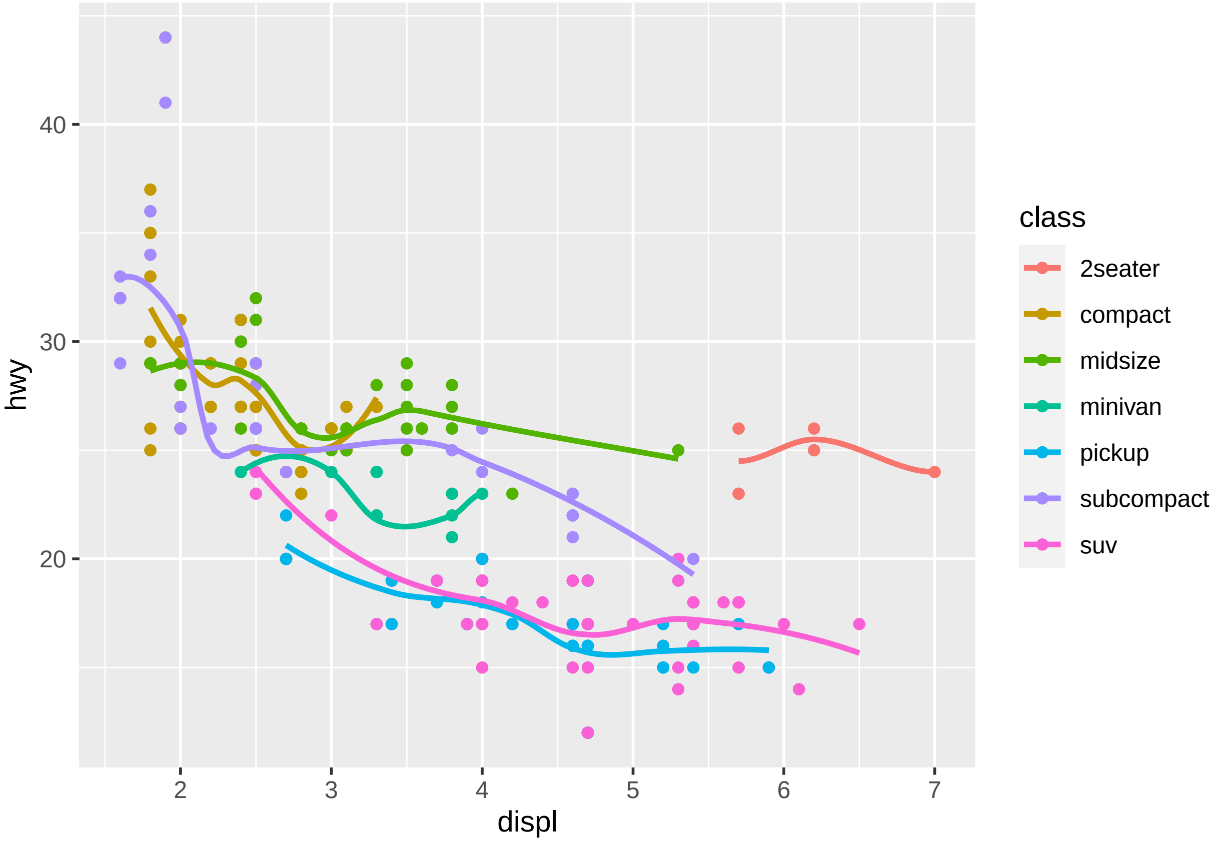
```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : reciprocal condition number 0
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : There are other near singularities as well. 0.65044
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : pseudoinverse used at 4.008
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : neighborhood radius 0.708
```

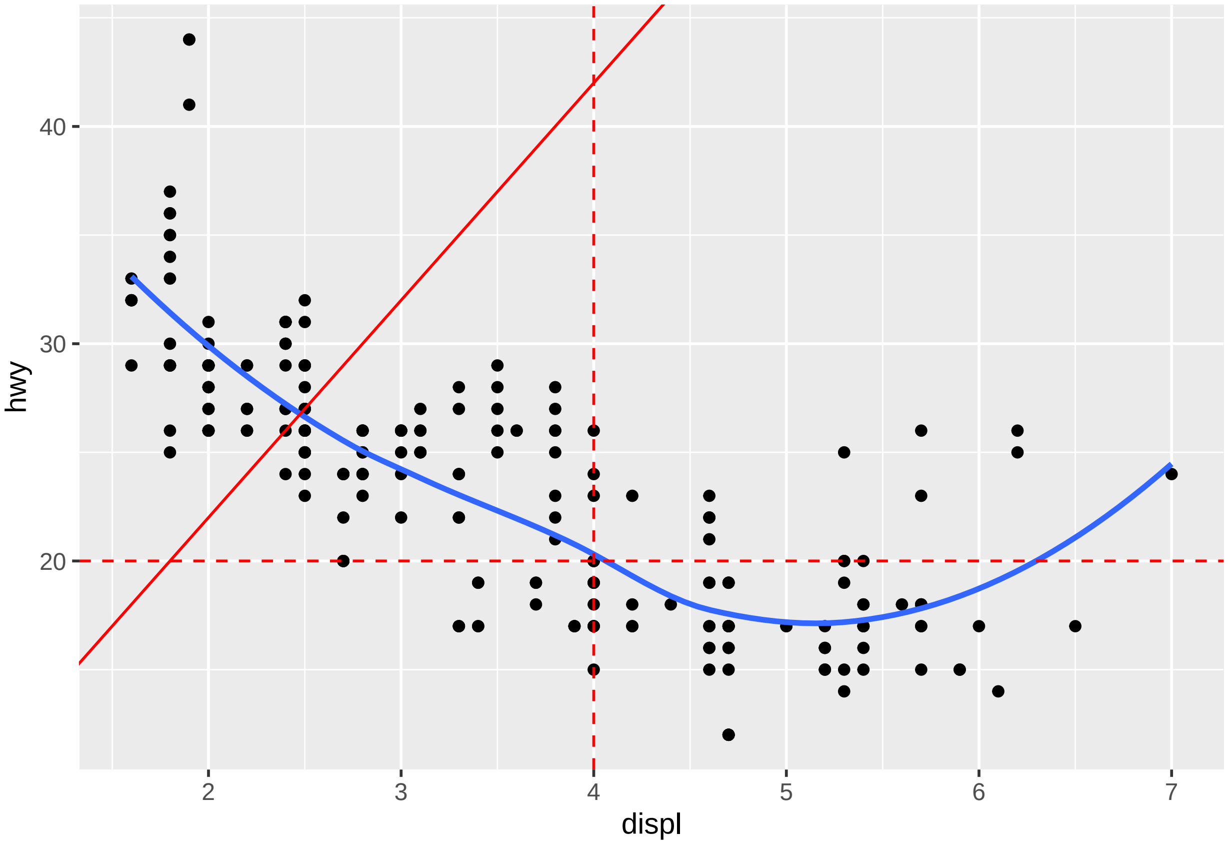
```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : reciprocal condition number 0
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : There are other near singularities as well. 0.25
```



Add horizontal, vertical, and/or oblique lines:

```
mpg %>%
  ggplot(aes(x = displ, y = hwy)) + # set aesthetic mapping globally
  geom_point() +                    # add scatter plot
  geom_smooth(se = FALSE) +         # add smooth curve
  geom_vline(xintercept = 4,        # add vertical line with x-intercept 4
             linetype = "dashed",
             color = "red") +
  geom_hline(yintercept = 20,       # add horizontal line with y-intercept 20
             linetype = "dashed",
             color = "red") +
  geom_abline(slope = 10,           # add oblique line with slope 10
              intercept = 2,        #  and y-intercept 2
              linetype = "solid",
              color = "red")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

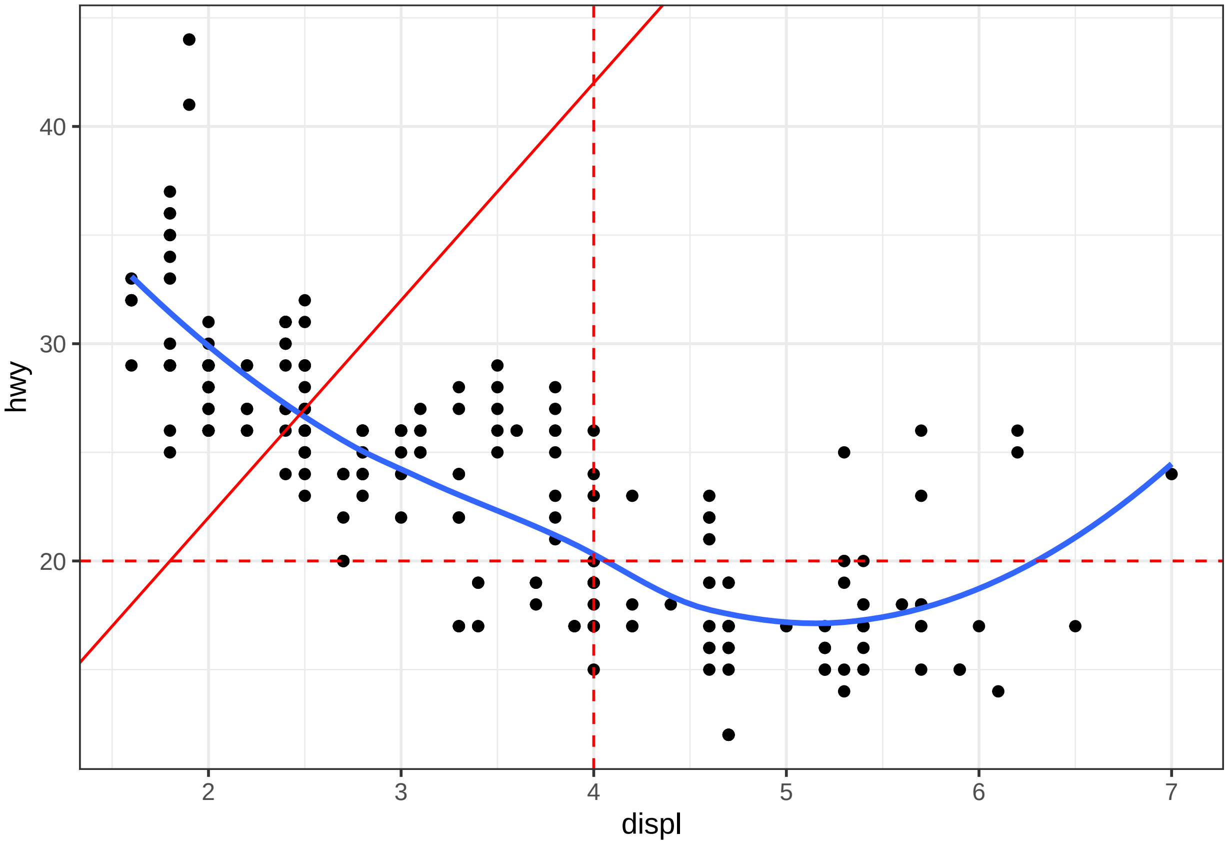


1.7 Themes

Different pre-set "themes" give plots different appearances. A theme I prefer to use is `theme_bw()`:

```
mpg %>%
  ggplot(aes(x = displ, y = hwy)) + # set aesthetic mapping globally
  geom_point() +                    # add scatter plot
  geom_smooth(se = FALSE) +         # add smooth curve
  geom_vline(xintercept = 4,        # add vertical line with x-intercept 4
             linetype = "dashed",
             color = "red") +
  geom_hline(yintercept = 20,       # add horizontal line with y-intercept 20
             linetype = "dashed",
             color = "red") +
  geom_abline(slope = 10,           # add oblique line with slope 10
              intercept = 2,        #  and y-intercept 2
              linetype = "solid",
              color = "red") +
  theme_bw()                        # add classy bw theme
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



1.8 High-quality figures for communication

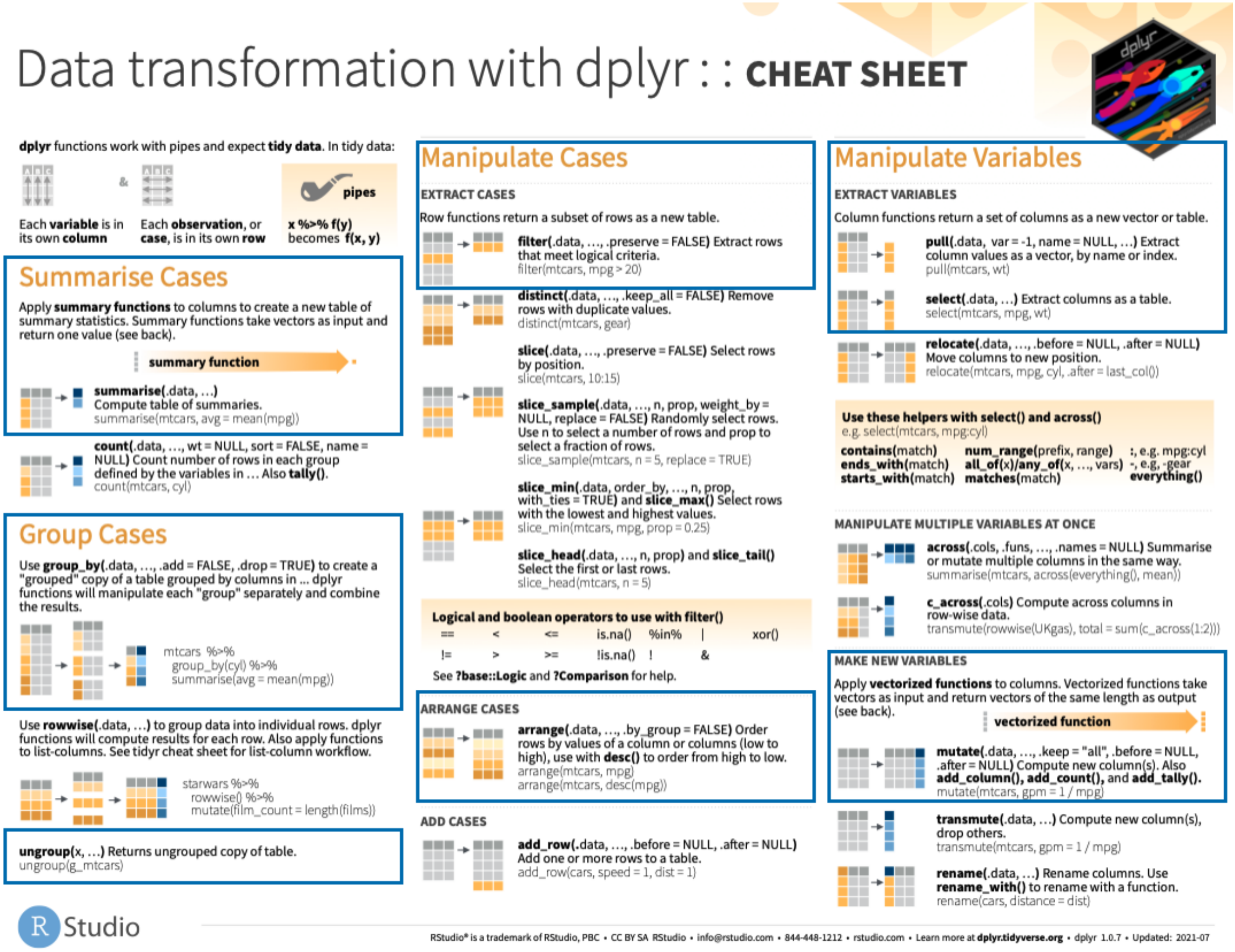
See preparing-reports.pdf. Part of your homework will be graded on presentation quality.

2 Data transformation

The `dplyr` package (another core member of the `tidyverse`) facilitates manipulation of data. This includes key operations:

- Pick observations by their values (`filter()`).
- Reorder the rows (`arrange()`).
- Pick variables by their names (`select()`).
- Create new variables with functions of existing variables (`mutate()`).
- Collapse many values down to a single summary (`summarise()`).

These can all be used in conjunction with `group_by()` which changes the scope of each function from operating on the entire dataset to operating on it group-by-group. These six functions provide the verbs for a language of data manipulation.


&
pipes
summary function
vectorized function
dplyr

Data transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

x %>% f(y) becomes **f(x, y)**

Summarise Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summarise(.data, …**)** Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(.data, …, wt = NULL, sort = FALSE, name = NULL**)** Count number of rows in each group defined by the variables in … Also **tally()**.
count(mtcars, cyl)

Group Cases

Use **group_by(**.data, …, .add = FALSE, .drop = TRUE**)** to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

```
mtcars %>%
  group_by(cyl) %>%
  summarise(avg = mean(mpg))
```

Use **rowwise(**.data, …**)** to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyr cheat sheet for list-column workflow.

```
starwars %>%
  rowwise() %>%
  mutate(film_count = length(films))
```

ungroup(x, …**)** Returns ungrouped copy of table.
ungroup(g_mtcars)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …, .preserve = FALSE**)** Extract rows that meet logical criteria.
filter(mtcars, mpg > 20)

distinct(.data, …, .keep_all = FALSE**)** Remove rows with duplicate values.
distinct(mtcars, gear)

slice(.data, …, .preserve = FALSE**)** Select rows by position.
slice(mtcars, 10:15)

slice_sample(.data, …, n, prop, weight_by = NULL, replace = FALSE**)** Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
slice_sample(mtcars, n = 5, replace = TRUE)

slice_min(.data, order_by, …, n, prop, with_ties = TRUE**)** and **slice_max()** Select rows with the lowest and highest values.
slice_min(mtcars, mpg, prop = 0.25)

slice_head(.data, …, n, prop**)** and **slice_tail()** Select the first or last rows.
slice_head(mtcars, n = 5)

Logical and boolean operators to use with filter()

==	<	<=	is.na()	%in%	|	xor()
!=	>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …, .by_group = FALSE**)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, …, .before = NULL, .after = NULL**)** Add one or more rows to a table.
add_row(cars, speed = 1, dist = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1, name = NULL, …**)** Extract column values as a vector, by name or index.
pull(mtcars, wt)

select(.data, …**)** Extract columns as a table.
select(mtcars, mpg, wt)

relocate(.data, …, .before = NULL, .after = NULL**)** Move columns to new position.
relocate(mtcars, mpg, cyl, .after = last_col())

Use these helpers with select() and across()
e.g. select(mtcars, mpg:cyl)

contains(match**)**	**num_range(**prefix, range**)**	:, e.g. mpg:cyl
ends_with(match**)**	**all_of(**x**)/any_of(**x, …, vars**)**	-, e.g, -gear
starts_with(match**)**	**matches(**match**)**	**everything()**

MANIPULATE MULTIPLE VARIABLES AT ONCE

across(.cols, .funs, …, .names = NULL**)** Summarise or mutate multiple columns in the same way.
summarise(mtcars, across(everything(), mean))

c_across(.cols**)** Compute across columns in row-wise data.
transmute(rowwise(UKgas), total = sum(c_across(1:2)))

MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

mutate(.data, …, .keep = "all", .before = NULL, .after = NULL**)** Compute new column(s). Also **add_column(), add_count(),** and **add_tally().**
mutate(mtcars, gpm = 1 / mpg)

transmute(.data, …**)** Compute new column(s), drop others.
transmute(mtcars, gpm = 1 / mpg)

rename(.data, …**)** Rename columns. Use **rename_with()** to rename with a function.
rename(cars, distance = dist)

RStudio

RStudio® is a trademark of RStudio, PBC • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **dplyr.tidyverse.org** • dplyr 1.0.7 • Updated: 2021-07

Figure 4: Image source: https://www.rstudio.com/resources/cheatsheets/

2.1 Filter rows with filter()

```
mpg %>%
  filter(class == "compact")
```

```
## # A tibble: 47 x 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~ f        18    29 p     comp~
##  2 audi         a4           1.8  1999     4 manu~ f        21    29 p     comp~
##  3 audi         a4           2    2008     4 manu~ f        20    31 p     comp~
##  4 audi         a4           2    2008     4 auto~ f        21    30 p     comp~
##  5 audi         a4           2.8  1999     6 auto~ f        16    26 p     comp~
##  6 audi         a4           2.8  1999     6 manu~ f        18    26 p     comp~
##  7 audi         a4           3.1  2008     6 auto~ f        18    27 p     comp~
##  8 audi         a4 quattro   1.8  1999     4 manu~ 4        18    26 p     comp~
##  9 audi         a4 quattro   1.8  1999     4 auto~ 4        16    25 p     comp~
## 10 audi         a4 quattro   2    2008     4 manu~ 4        20    28 p     comp~
## # ... with 37 more rows
```

```
mpg %>%
  filter(class %in% c("compact", "2seater"))
```

```
## # A tibble: 52 x 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~ f        18    29 p     comp~
##  2 audi         a4           1.8  1999     4 manu~ f        21    29 p     comp~
##  3 audi         a4           2    2008     4 manu~ f        20    31 p     comp~
##  4 audi         a4           2    2008     4 auto~ f        21    30 p     comp~
##  5 audi         a4           2.8  1999     6 auto~ f        16    26 p     comp~
##  6 audi         a4           2.8  1999     6 manu~ f        18    26 p     comp~
##  7 audi         a4           3.1  2008     6 auto~ f        18    27 p     comp~
##  8 audi         a4 quattro   1.8  1999     4 manu~ 4        18    26 p     comp~
##  9 audi         a4 quattro   1.8  1999     4 auto~ 4        16    25 p     comp~
## 10 audi         a4 quattro   2    2008     4 manu~ 4        20    28 p     comp~
## # ... with 42 more rows
```

```
mpg %>%
  filter(class %in% c("compact", "2seater") & year > 2000) # & means "and"
```

```
## # A tibble: 25 x 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           2    2008     4 manu~ f        20    31 p     comp~
##  2 audi         a4           2    2008     4 auto~ f        21    30 p     comp~
##  3 audi         a4           3.1  2008     6 auto~ f        18    27 p     comp~
##  4 audi         a4 quattro   2    2008     4 manu~ 4        20    28 p     comp~
##  5 audi         a4 quattro   2    2008     4 auto~ 4        19    27 p     comp~
##  6 audi         a4 quattro   3.1  2008     6 auto~ 4        17    25 p     comp~
##  7 audi         a4 quattro   3.1  2008     6 manu~ 4        15    25 p     comp~
##  8 chevrolet    corvette     6.2  2008     8 manu~ r        16    26 p     2sea~
##  9 chevrolet    corvette     6.2  2008     8 auto~ r        15    25 p     2sea~
## 10 chevrolet    corvette     7    2008     8 manu~ r        15    24 p     2sea~
## # ... with 15 more rows
```

2.2 Exercises

1. Find all cars manufactured by a Japanese company (`Honda`, `Toyota`, `Nissan`, `Subaru`). How many such cars are there in these data?
2. Find all cars whose highway fuel efficiency (`hwy`) exceeded their city fuel efficiency (`cty`) by at least a factor of 1.5. How many such cars are there in these data?

```
mpg %>%
  filter(class %in% c("compact", "2seater") | year > 2000) # | means "or"
```

```
## # A tibble: 144 x 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~ f        18    29 p     comp~
##  2 audi         a4           1.8  1999     4 manu~ f        21    29 p     comp~
##  3 audi         a4           2    2008     4 manu~ f        20    31 p     comp~
##  4 audi         a4           2    2008     4 auto~ f        21    30 p     comp~
##  5 audi         a4           2.8  1999     6 auto~ f        16    26 p     comp~
##  6 audi         a4           2.8  1999     6 manu~ f        18    26 p     comp~
##  7 audi         a4           3.1  2008     6 auto~ f        18    27 p     comp~
##  8 audi         a4 quattro   1.8  1999     4 manu~ 4        18    26 p     comp~
##  9 audi         a4 quattro   1.8  1999     4 auto~ 4        16    25 p     comp~
## 10 audi         a4 quattro   2    2008     4 manu~ 4        20    28 p     comp~
## # ... with 134 more rows
```

2.3 Arrange rows with `arrange()`

You can sort the rows of a tibble according to the values of a certain variable:

```
mpg %>% arrange(hwy)
```

```
## # A tibble: 234 x 11
##    manufacturer model     displ  year   cyl trans  drv     cty   hwy fl    class
##    <chr>        <chr>     <dbl> <int> <int> <chr>  <chr> <int> <int> <chr> <chr>
##  1 dodge        dakota p~   4.7  2008     8 auto(~ 4         9    12 e     pick~
##  2 dodge        durango ~   4.7  2008     8 auto(~ 4         9    12 e     suv
##  3 dodge        ram 1500~   4.7  2008     8 auto(~ 4         9    12 e     pick~
##  4 dodge        ram 1500~   4.7  2008     8 manua~ 4         9    12 e     pick~
##  5 jeep         grand ch~   4.7  2008     8 auto(~ 4         9    12 e     suv
##  6 chevrolet    k1500 ta~   5.3  2008     8 auto(~ 4        11    14 e     suv
##  7 jeep         grand ch~   6.1  2008     8 auto(~ 4        11    14 p     suv
##  8 chevrolet    c1500 su~   5.3  2008     8 auto(~ r        11    15 e     suv
##  9 chevrolet    k1500 ta~   5.7  1999     8 auto(~ 4        11    15 r     suv
## 10 dodge        dakota p~   5.2  1999     8 auto(~ 4        11    15 r     pick~
## # ... with 224 more rows
```

Or in descending order:

```
mpg %>% arrange(desc(hwy))
```

```
## # A tibble: 234 x 11
##    manufacturer model      displ  year   cyl trans drv     cty   hwy fl    class
##    <chr>        <chr>      <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
##  1 volkswagen   jetta        1.9  1999     4 manu~ f        33    44 d     comp~
##  2 volkswagen   new beetle   1.9  1999     4 manu~ f        35    44 d     subc~
##  3 volkswagen   new beetle   1.9  1999     4 auto~ f        29    41 d     subc~
##  4 toyota       corolla      1.8  2008     4 manu~ f        28    37 r     comp~
##   5 honda        civic        1.8  2008     4 auto~ f        25    36 r     subc~
##   6 honda        civic        1.8  2008     4 auto~ f        24    36 c     subc~
##   7 toyota       corolla      1.8  1999     4 manu~ f        26    35 r     comp~
##   8 toyota       corolla      1.8  2008     4 auto~ f        26    35 r     comp~
##   9 honda        civic        1.8  2008     4 manu~ f        26    34 r     subc~
##  10 honda        civic        1.6  1999     4 manu~ f        28    33 r     subc~
## # ... with 224 more rows
```

Which car had the best highway fuel efficiency?

2.4 Select columns with `select()`

Select columns:

```
mpg %>% select(manufacturer, model, year)
```

```
## # A tibble: 234 x 3
##    manufacturer model       year
##    <chr>        <chr>      <int>
##  1 audi         a4          1999
##  2 audi         a4          1999
##  3 audi         a4          2008
##  4 audi         a4          2008
##  5 audi         a4          1999
##  6 audi         a4          1999
##  7 audi         a4          2008
##  8 audi         a4 quattro  1999
##  9 audi         a4 quattro  1999
## 10 audi         a4 quattro  2008
## # ... with 224 more rows
```

De-select columns:

```
mpg %>% select(-manufacturer, -model, -year)
```

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

2.5 Add new variables with `mutate()`

```
mpg_small = mpg %>% select(manufacturer, model, year, cty, hwy)
mpg_small %>%
  mutate(hwy_boost = hwy - cty,
         japanese = manufacturer %in% c("Honda", "Toyota", "Nissan", "Subaru"))
```

```
## # A tibble: 234 x 7
##    manufacturer model       year   cty   hwy hwy_boost japanese
##    <chr>        <chr>      <int> <int> <int>     <int> <lgl>
##  1 audi         a4          1999    18    29        11 FALSE
##  2 audi         a4          1999    21    29         8 FALSE
##  3 audi         a4          2008    20    31        11 FALSE
##  4 audi         a4          2008    21    30         9 FALSE
##  5 audi         a4          1999    16    26        10 FALSE
##  6 audi         a4          1999    18    26         8 FALSE
##  7 audi         a4          2008    18    27         9 FALSE
##  8 audi         a4 quattro  1999    18    26         8 FALSE
##  9 audi         a4 quattro  1999    16    25         9 FALSE
## 10 audi         a4 quattro  2008    20    28         8 FALSE
## # ... with 224 more rows
```

2.6 Grouped summaries with summarise()

Extract mean fuel economy for cities and highways:

```
mpg %>%
  summarise(mean_cty = mean(cty),
            mean_hwy = mean(hwy))
```

```
## # A tibble: 1 x 2
##   mean_cty mean_hwy
##      <dbl>    <dbl>
## 1     16.9     23.4
```

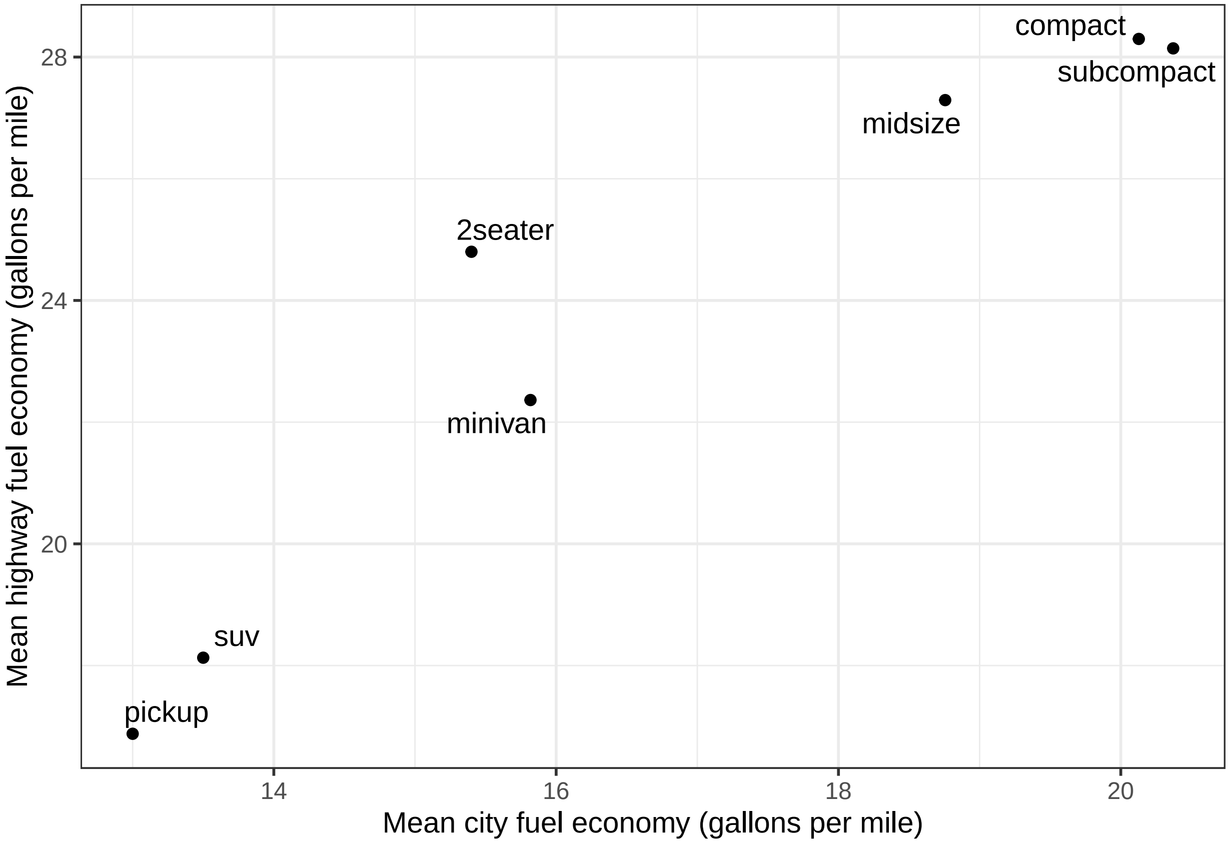
Extract mean fuel economy for cities and highways, by car class:

```
mpg %>%
  group_by(class) %>%
  summarise(mean_cty = mean(cty),
            mean_hwy = mean(hwy))
```

```
## # A tibble: 7 x 3
##   class      mean_cty mean_hwy
##   <chr>         <dbl>    <dbl>
## 1 2seater        15.4     24.8
## 2 compact        20.1     28.3
## 3 midsize        18.8     27.3
## 4 minivan        15.8     22.4
## 5 pickup         13       16.9
## 6 subcompact     20.4     28.1
## 7 suv            13.5     18.1
```

Note that we strung together two operations using the pipe. We can string together arbitrarily many operations using the pipe, including plotting:

```
mpg %>%
  group_by(class) %>%
  summarise(mean_cty = mean(cty),
            mean_hwy = mean(hwy)) %>%
  ggplot(aes(x = mean_cty, y = mean_hwy, label = class)) +
  geom_point() +
  ggrepel::geom_text_repel() +
  labs(x = "Mean city fuel economy (gallons per mile)",
       y = "Mean highway fuel economy (gallons per mile)") +
  theme_bw()
```



Common functions used with `summarise()`: `mean`, `median`, `sum`, `min`, `max`, `n`, `sd`, …

3 Exploratory data analysis

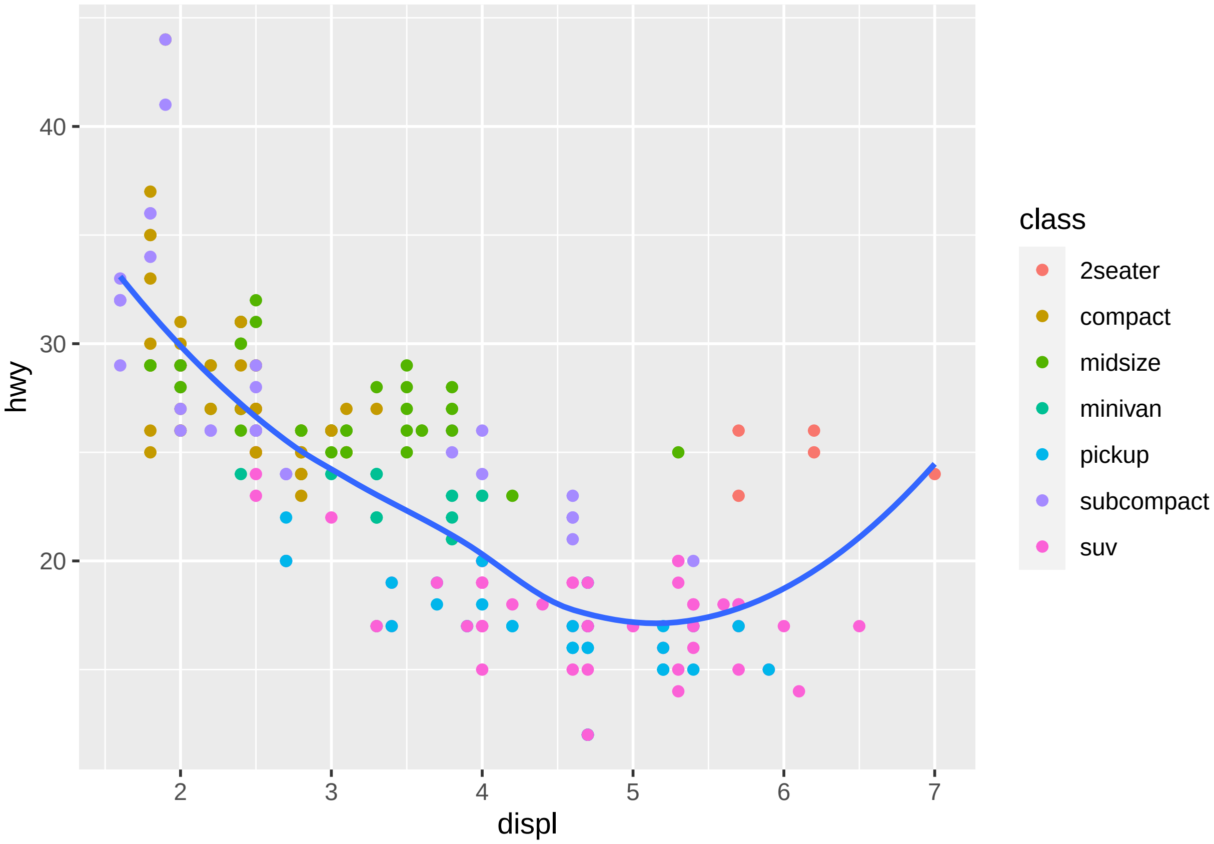
All these visualizations and transformations can help us to explore the data and find interesting patterns.

3.1 Relationship between engine size and fuel economy

What is the relationship between `displ` (a car's engine size in liters) and `hwy` (a car's fuel efficiency on the highway, in miles per gallon)? We created the plot below already:

```
mpg %>%
  ggplot(aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Note that the smooth curve is first decreasing but then increasing. What would we expect the relationship to be between `hwy` and `displ`? What color points seem to be "pulling up" the smooth curve fit? Why might this be the case?

3.2 Fuel economy in cities versus on highways

What is the relationship between fuel economy in cities and on highways?

3.3 Comparing manufacturers based on fuel economy

Which manufacturers had the best and worst highway fuel economy in the year 1999, on average over models? What about the year 2008?