ggplot2 examples and exercises - part 1

This document contains all the code that is displayed during the workshop. The document is an RMarkdown document which means that it can be compiled, along with the code chunks thus executing and capturing the output of the code within the document. To read more about RMarkdown see the website for the package, as well as the Get Started guide.

Exercises

While it is encouraged to follow along in this document as the workshop progresses and execute the code to see the result, an important part is also to experiment and play, thus learning how the different settings affect the output.

The document will contain code chunks with the code examples discussed during the talk, but it will also contain chunks intended for completing small exercises. These will use the examples as a starting point and ask you to modify the code to achieve a given output. Completing these are optional, but highly recommended, either during or after the workshop.

Dependencies

This document comes with a list of required packages.

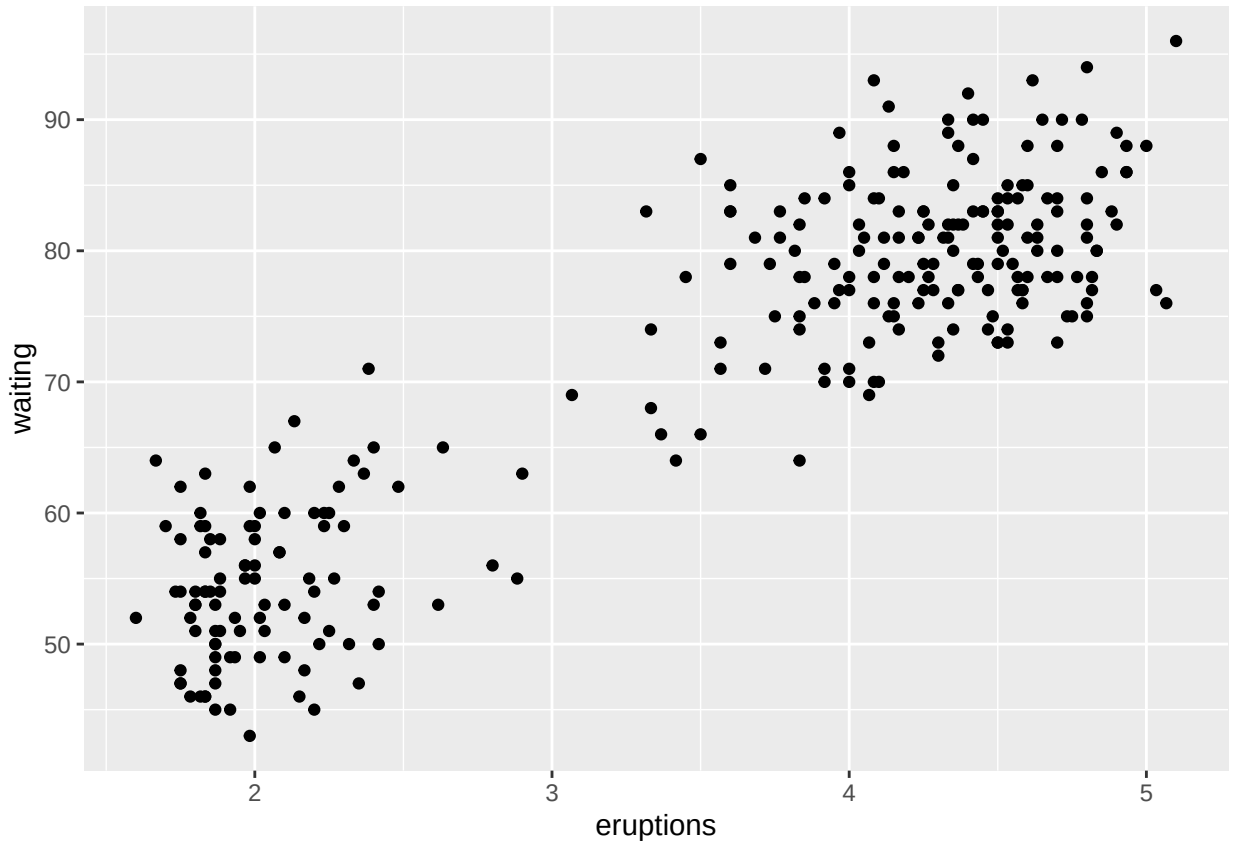
Datasets

We will use an assortment of datasets throughout the document. The purpose is mostly to showcase different plots, and less on getting some divine insight into the world. While not necessary we will call `data(<dataset>)` before using a new dataset to indicate the introduction of a new dataset.

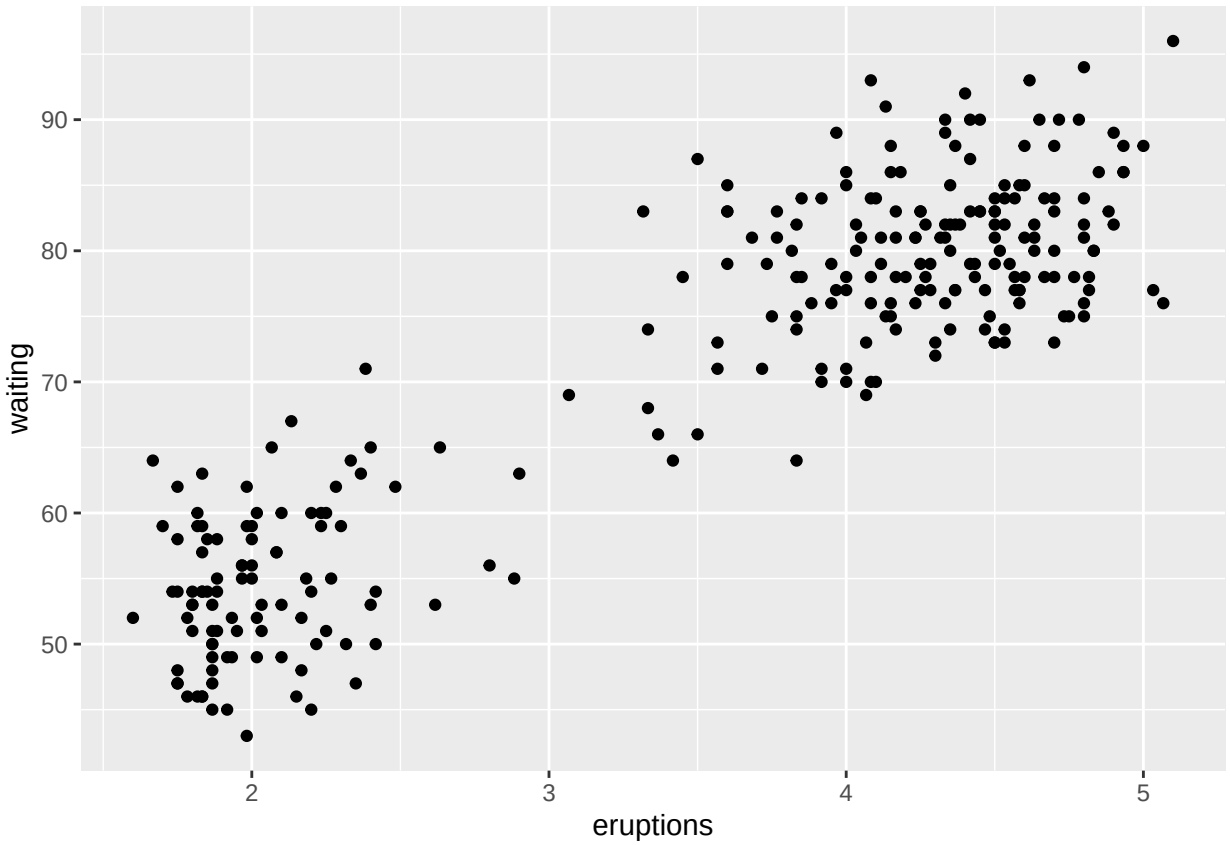
Introduction

We will look at the basic ggplot2 use using the faithful dataset, giving information on the eruption pattern of the Old Faithful geyser in Yellowstone National Park.

```
data("faithful")
# Basic scatterplot
ggplot(data = faithful,
       mapping = aes(x = eruptions, y = waiting)) +
       geom_point()
```

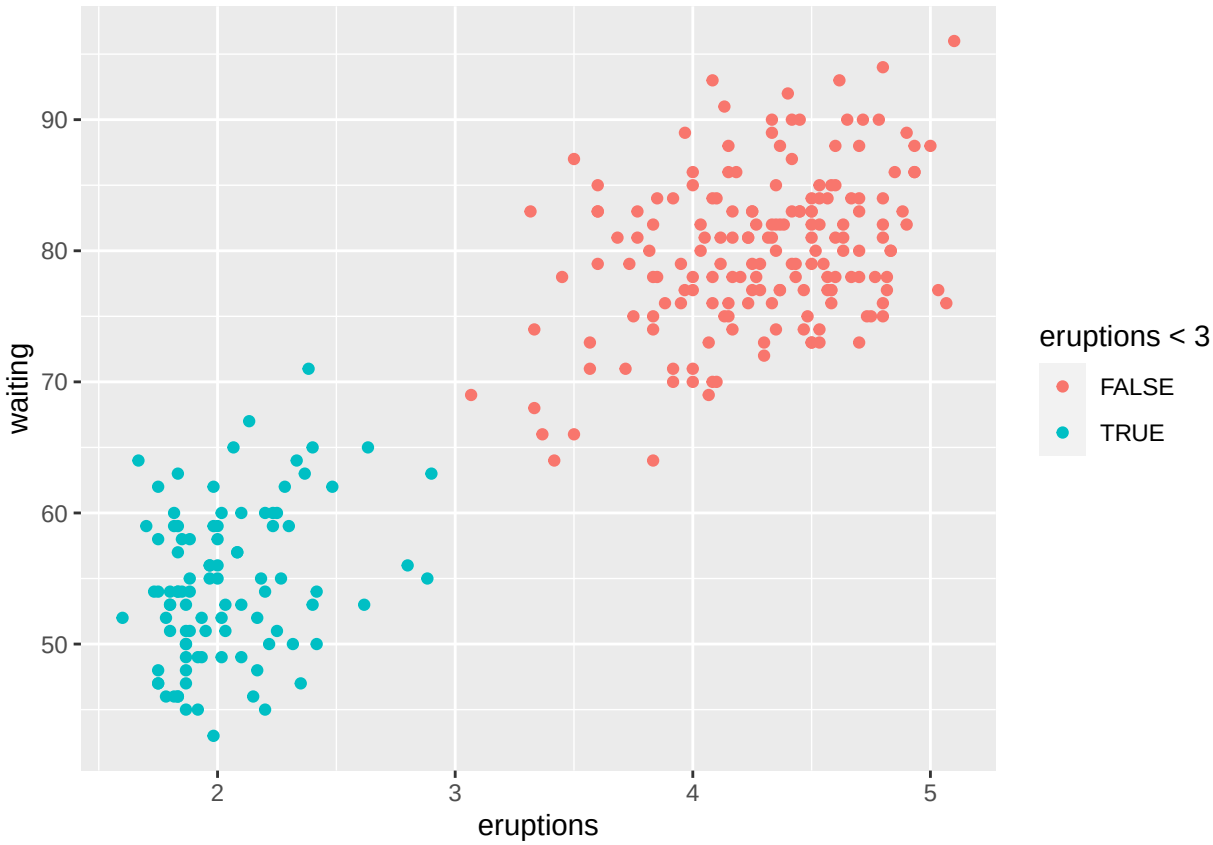



```
# Data and mapping can be given both as global (in ggplot()) or per layer
ggplot() +
  geom_point(mapping = aes(x = eruptions, y = waiting),
             data = faithful)
```



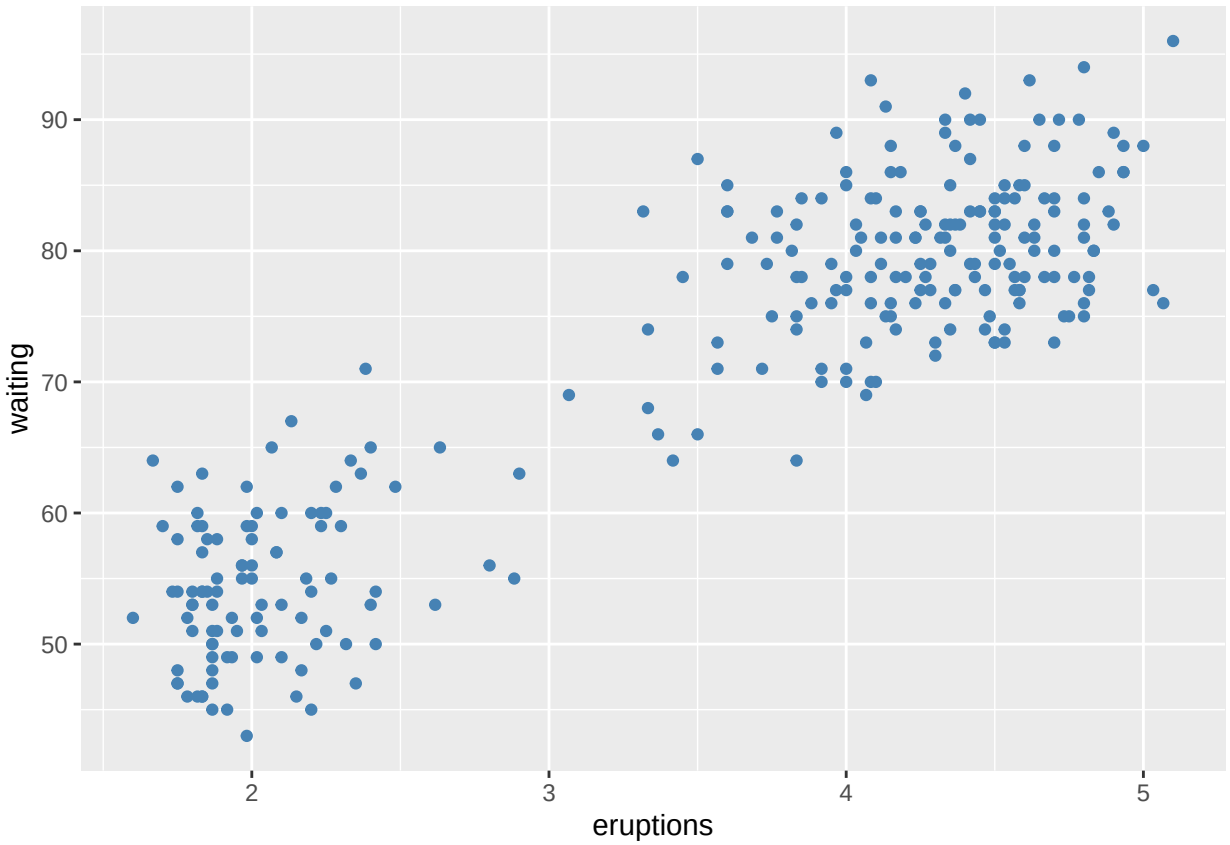
If an aesthetic (like colour) is linked to data it is put into `aes()`

```
ggplot(faithful) +
  geom_point(aes(x = eruptions, y = waiting, colour = eruptions < 3))  # colour aesthetic is put in aes
```



If you simply want to set it to a value, put it outside of `aes()`

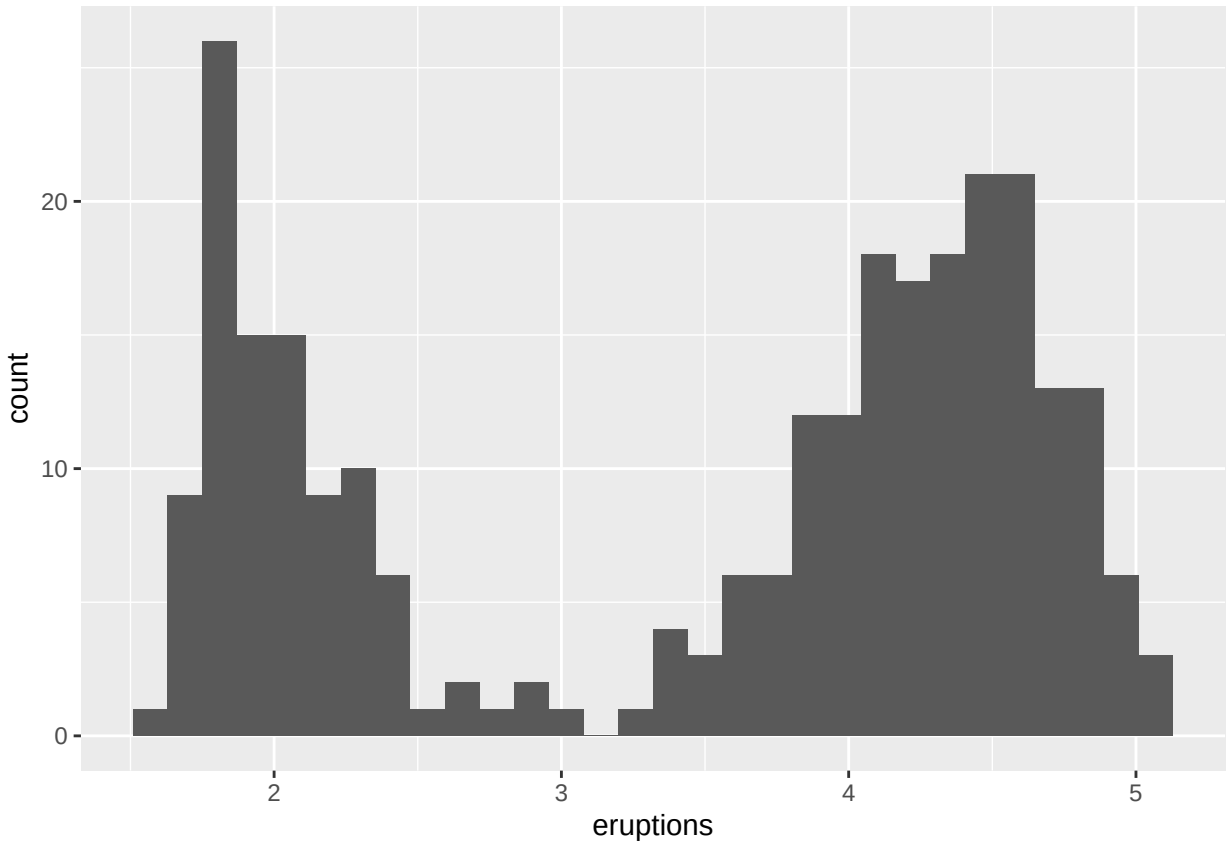
```
# Key point: if you want to take something from the data you put it inside aes(), but if you just want

ggplot(faithful) +
  geom_point(aes(x = eruptions, y = waiting),
             colour = 'steelblue')
```



Some geoms only need a single mapping and will calculate the rest for you

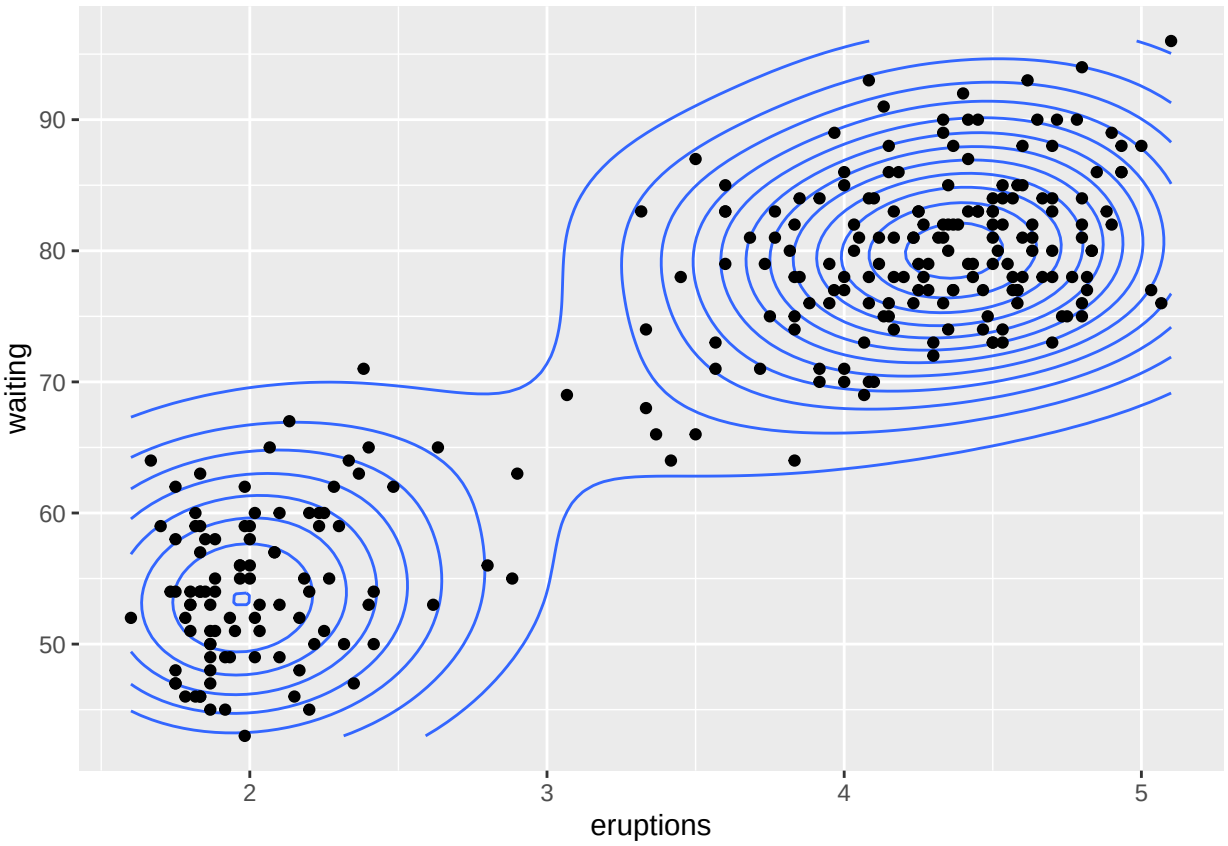
```
ggplot(faithful) +
  geom_histogram(aes(x = eruptions))
```

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



geoms are drawn in the order they are added. The point layer is thus drawn on top of the density contours in the example below.

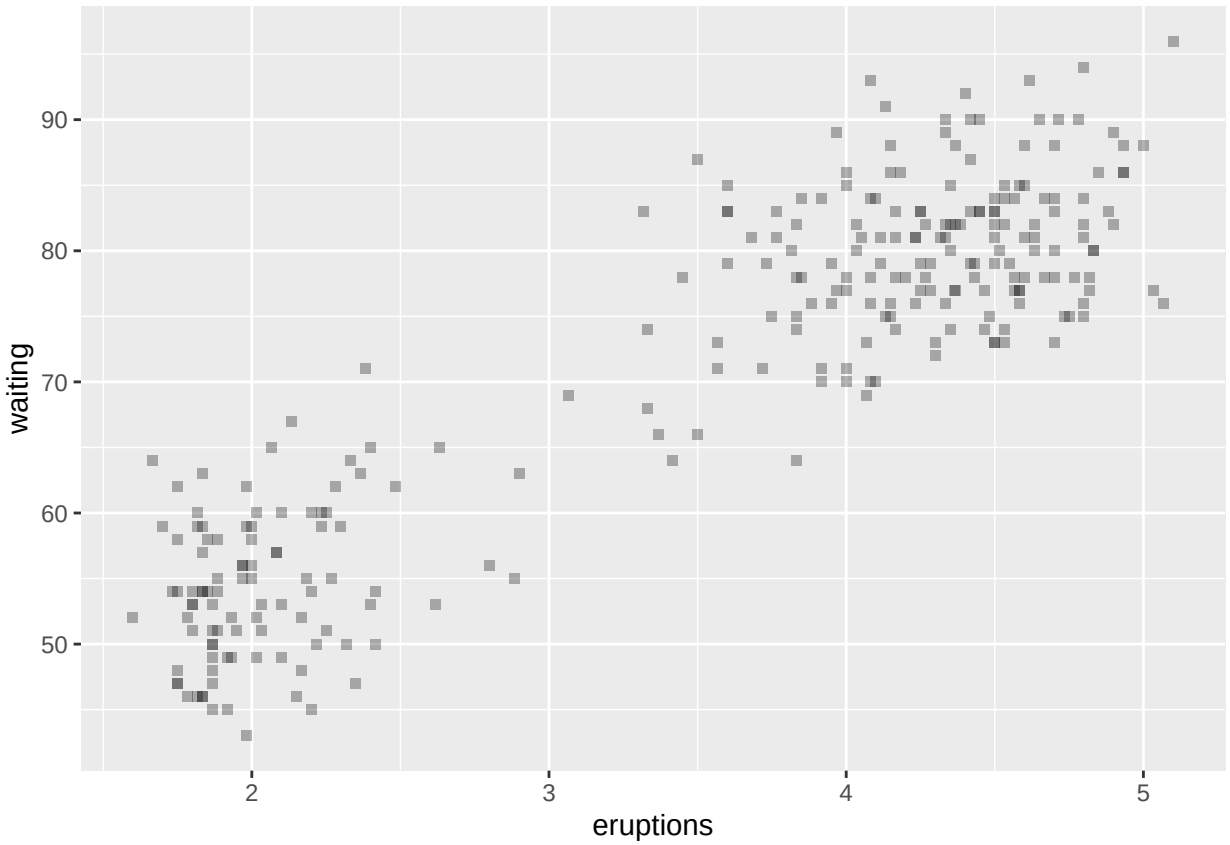
```
ggplot(faithful, aes(x = eruptions, y = waiting)) +
  geom_density_2d() +
  geom_point()
```



Exercise

Modify the code below to make the points larger squares and slightly transparent. See `?geom_point` for more information on the point layer.

```
ggplot(faithful) +
  geom_point(aes(x = eruptions, y = waiting), shape = 'square', alpha = 0.3)
```



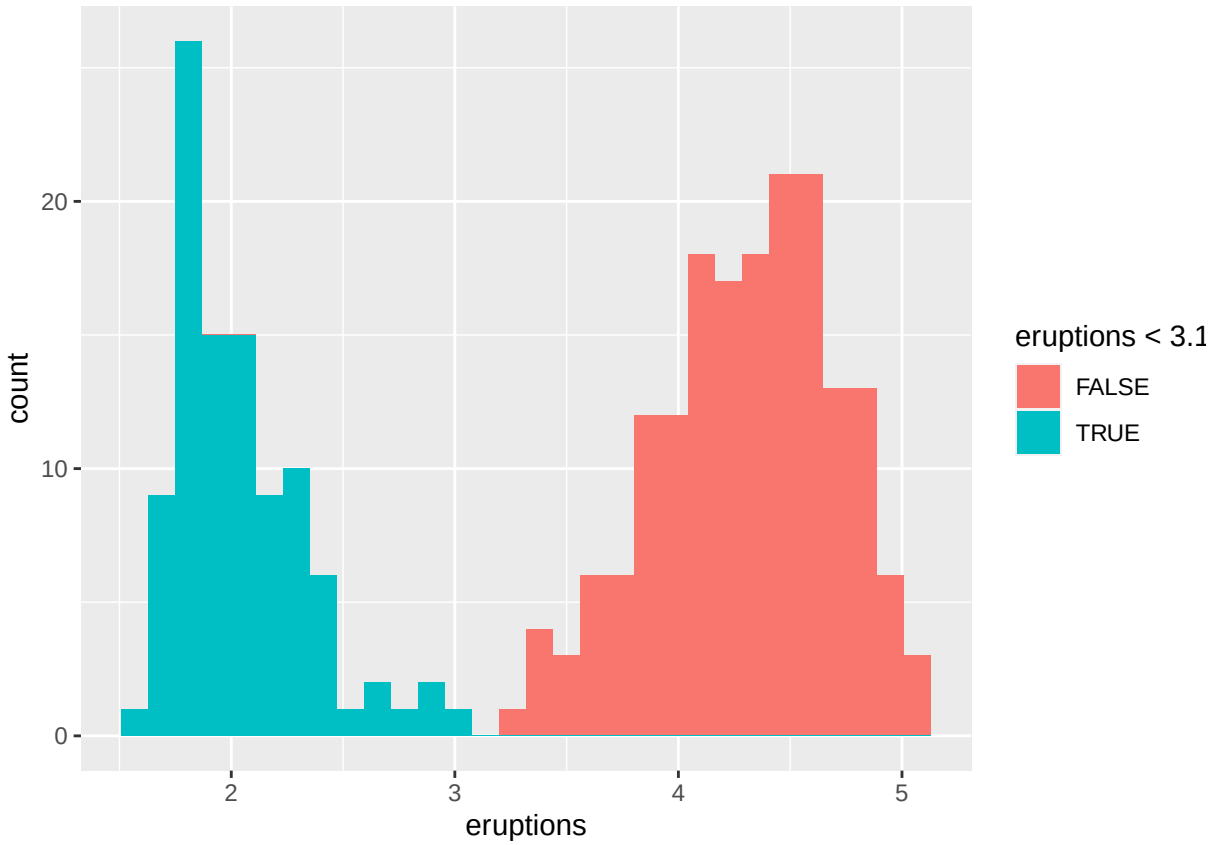
Hint 1: transparency is controlled with `alpha`, and shape with `shape` Hint 2: rememberthe difference between mapping and setting aesthetics

---

Colour the two distributions in the histogram with different colours

```
ggplot(faithful) +
  geom_histogram(aes(x = eruptions, fill = eruptions < 3.1))
```

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

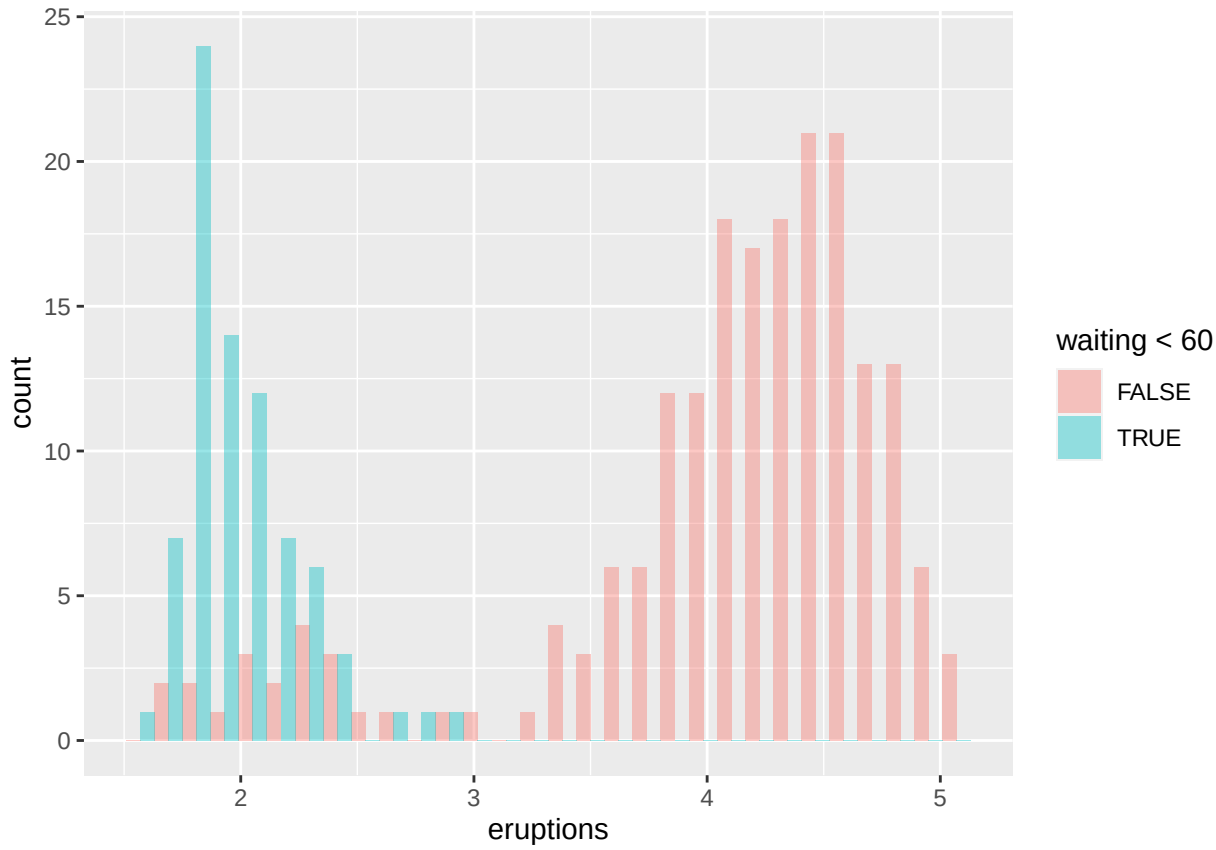


Hint 1: For polygons you can map two different colour-like aesthetics: `colour` (the colour of the stroke) and `fill` (the fill colour)

Colour the distributions in the histogram by whether `waiting` is above or below `60`. What happens?

```
ggplot(faithful) +
  geom_histogram(aes(x = eruptions, fill = waiting < 60), position = 'dodge', alpha = 0.4)
```

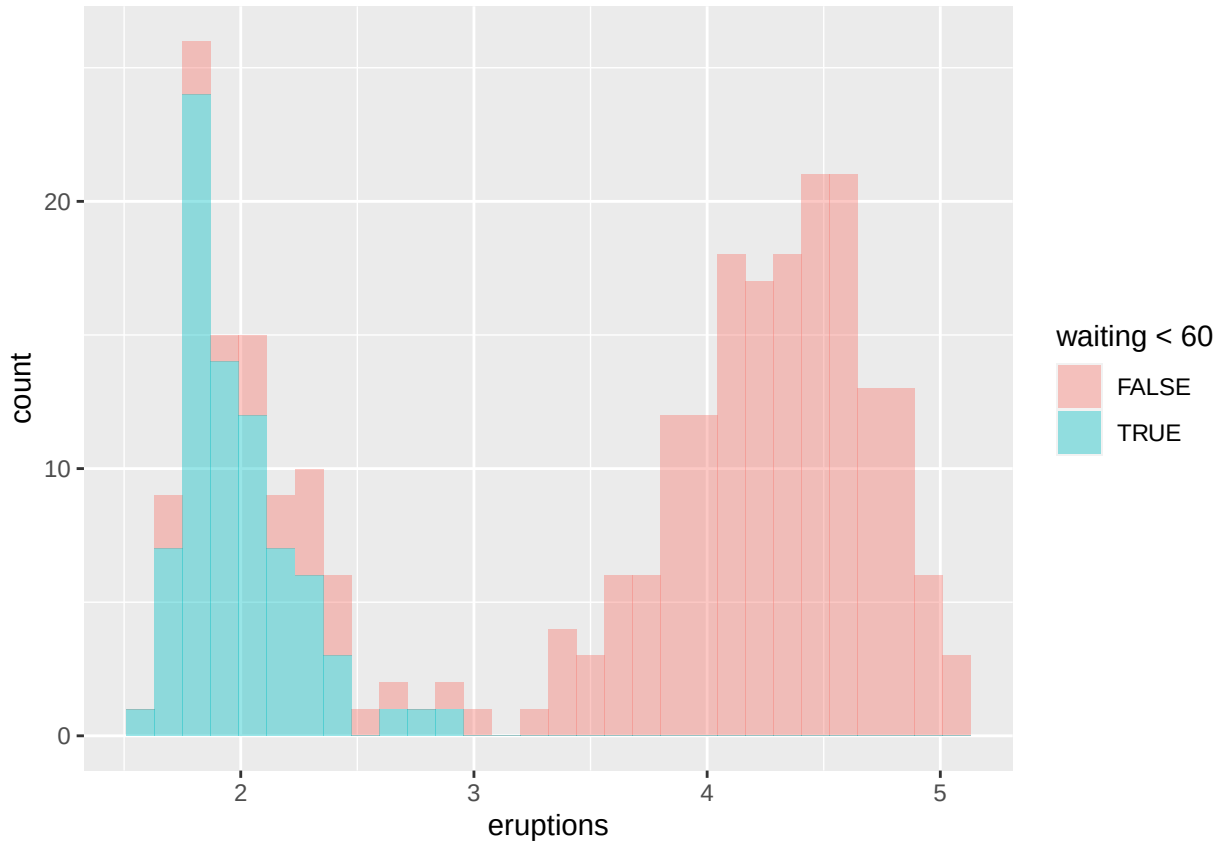
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# alternatives for position argument: 'stack' (default), 'identity'
```

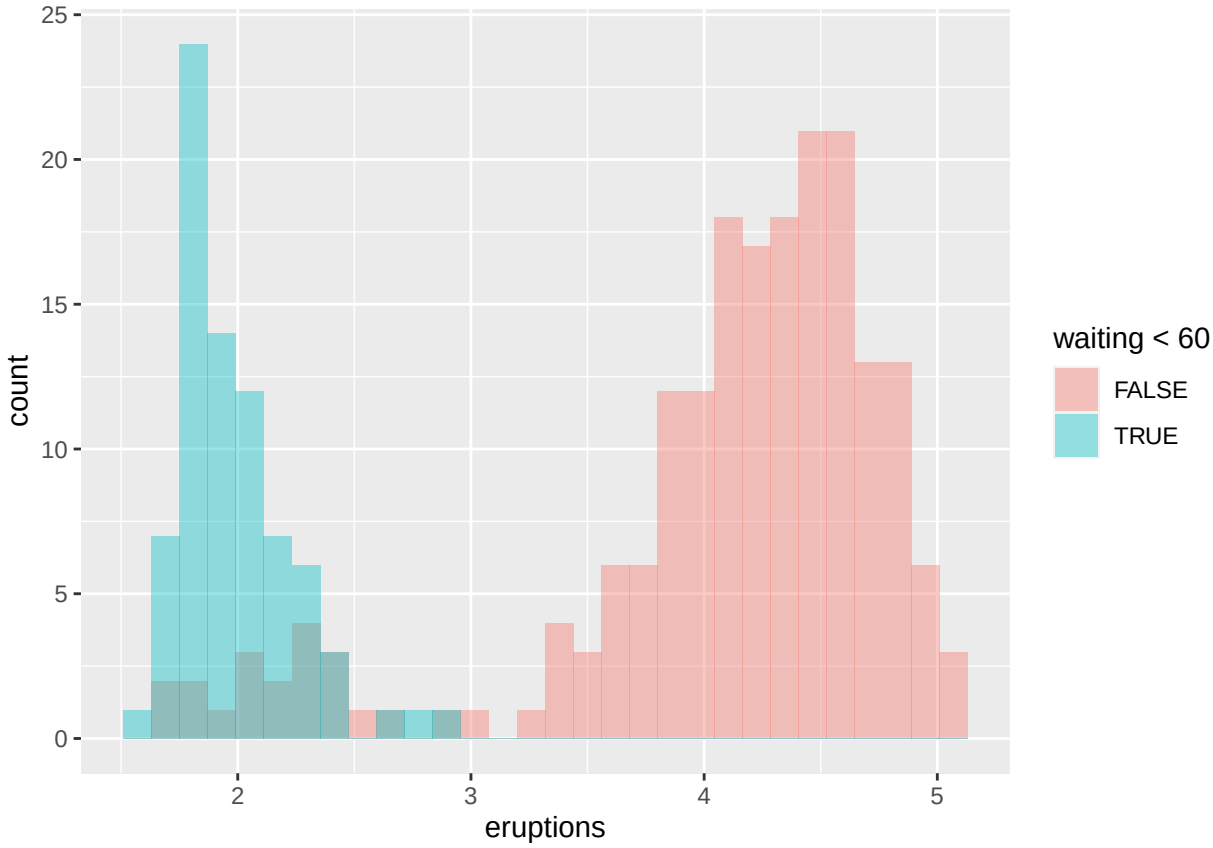
```
ggplot(faithful) +
  geom_histogram(aes(x = eruptions, fill = waiting < 60), position = 'stack', alpha = 0.4)
```

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




```
ggplot(faithful) +
  geom_histogram(aes(x = eruptions, fill = waiting < 60), position = 'identity', alpha = 0.4)
```

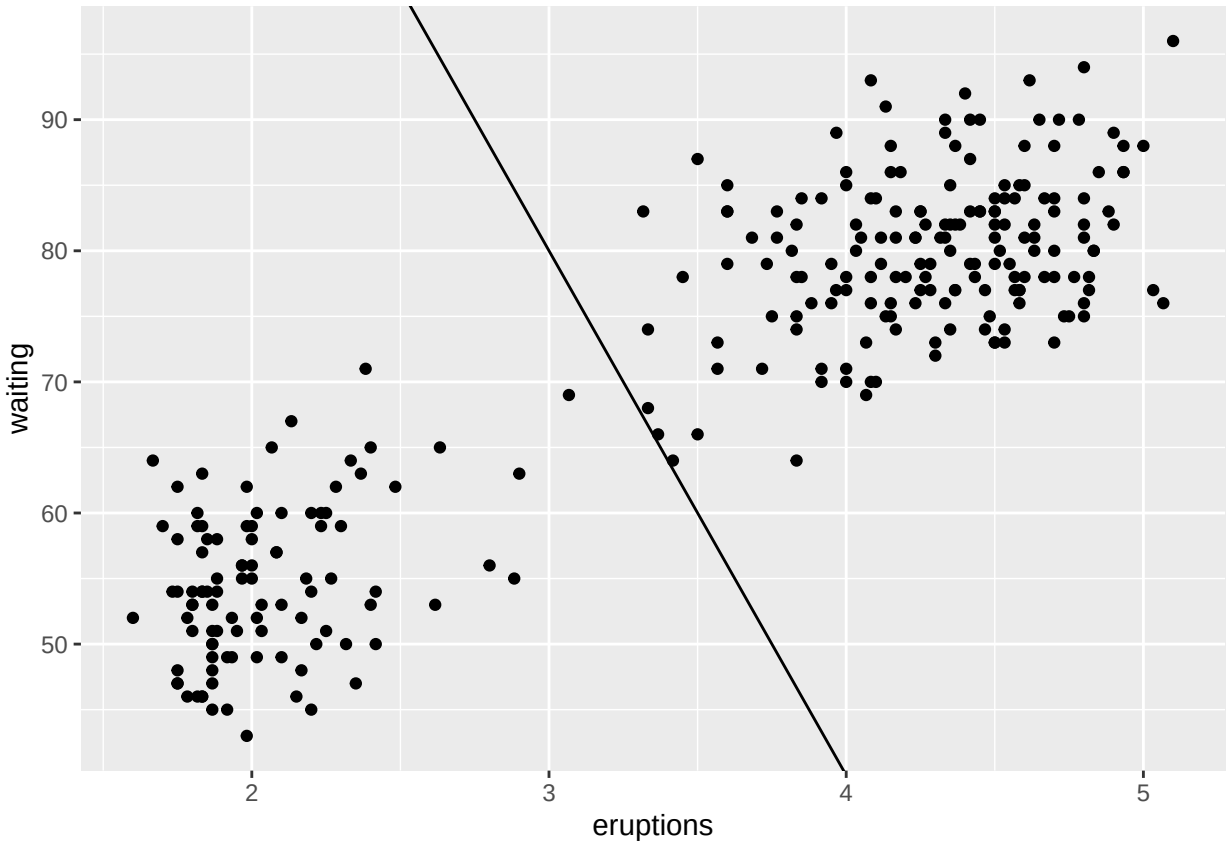
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Change the plot above by setting `position = 'dodge'` in `geom_histogram()` (while keeping the colouring by `waiting`). What do `position` control?

Add a line that separates the two point distributions. See `?geom_abline` for how to draw straight lines from a slope and intercept.

```
ggplot(faithful) +
  geom_point(aes(x = eruptions, y = waiting)) +
  geom_abline(slope = -40, intercept = 200)
```

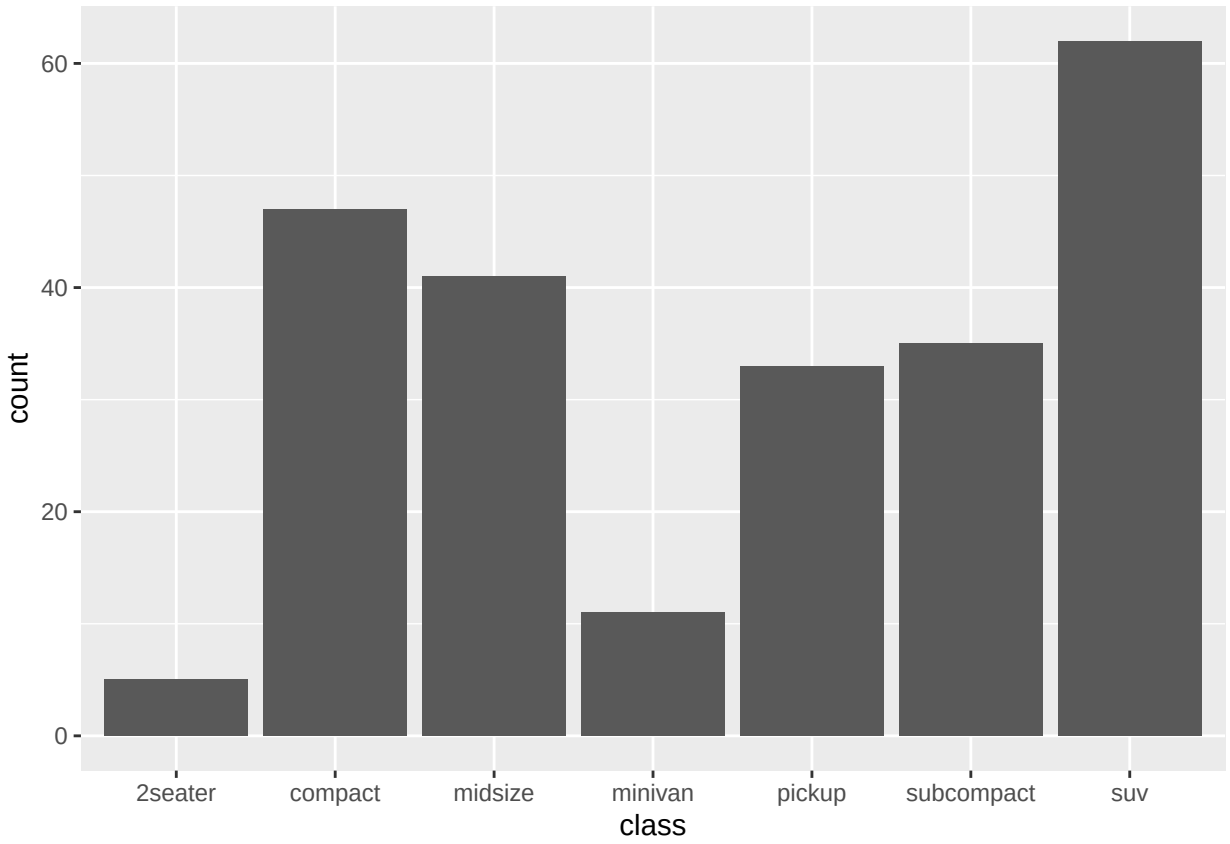


Stat

We will use the `mpg` dataset giving information about fuel economy on different car models.

Every geom has a stat. This is why new data (`count`) can appear when using `geom_bar()`.

```
data("mpg")
ggplot(mpg) +
  geom_bar(aes(x = class))
```



The stat can be overwritten. If we have precomputed count we don't want any additional computations to perform and we use the `identity` stat to leave the data alone

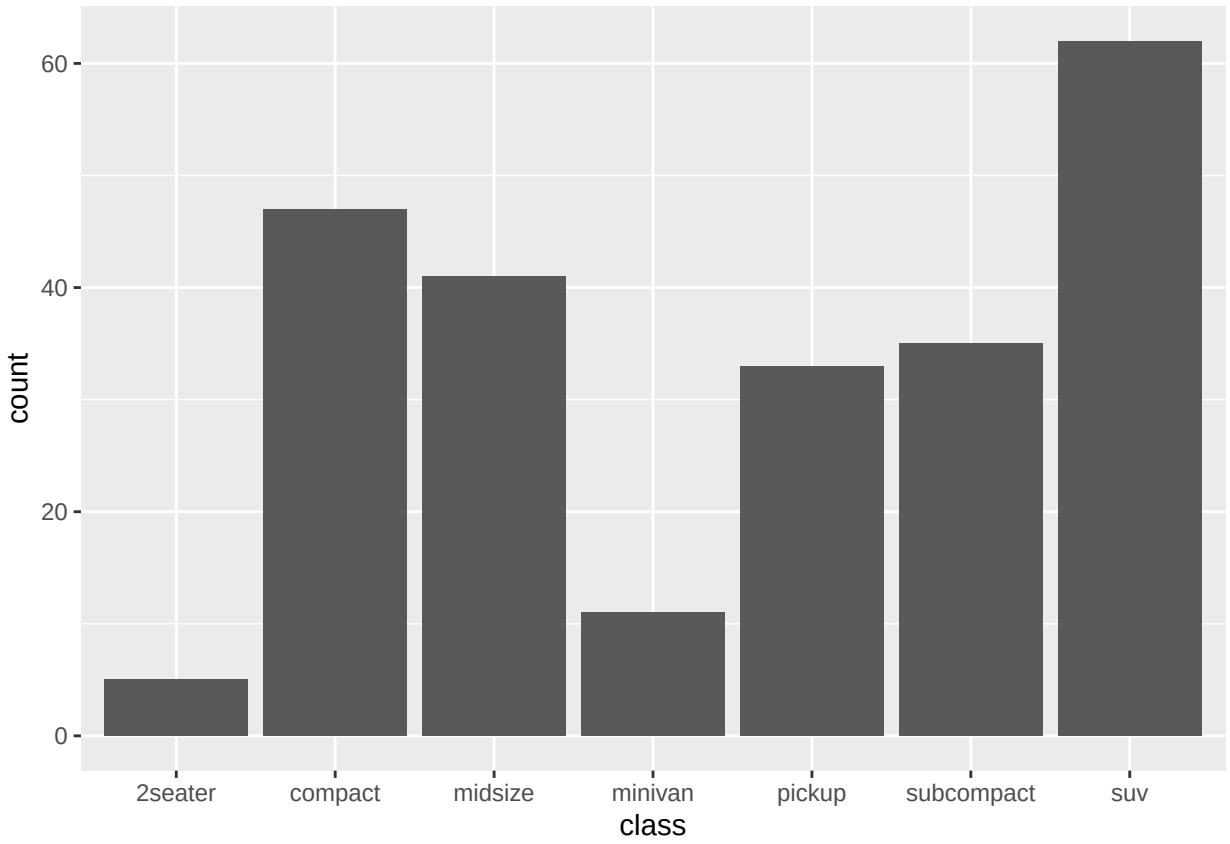
```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

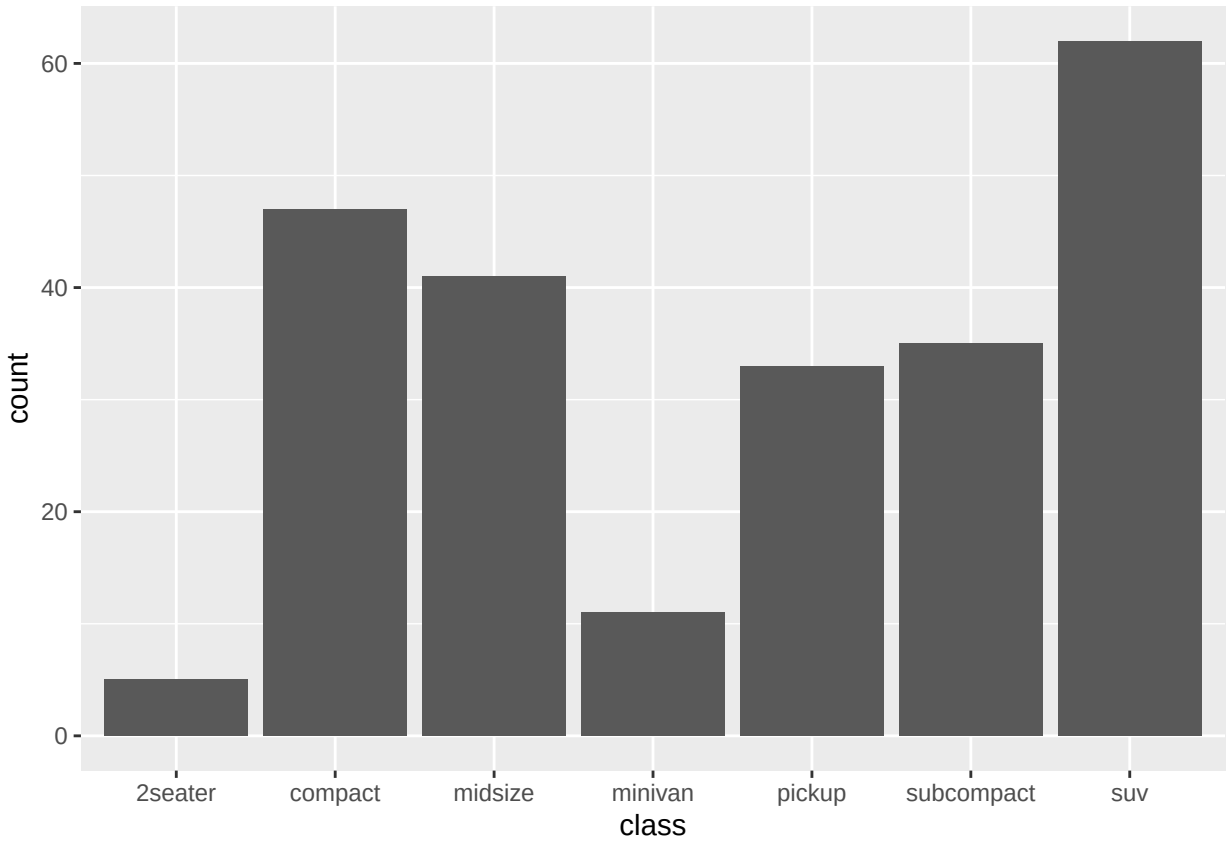
```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
mpg_counted <- mpg %>%
  count(class, name = 'count')

ggplot(mpg_counted) +
  geom_bar(aes(x = class, y = count), stat = 'identity')
```



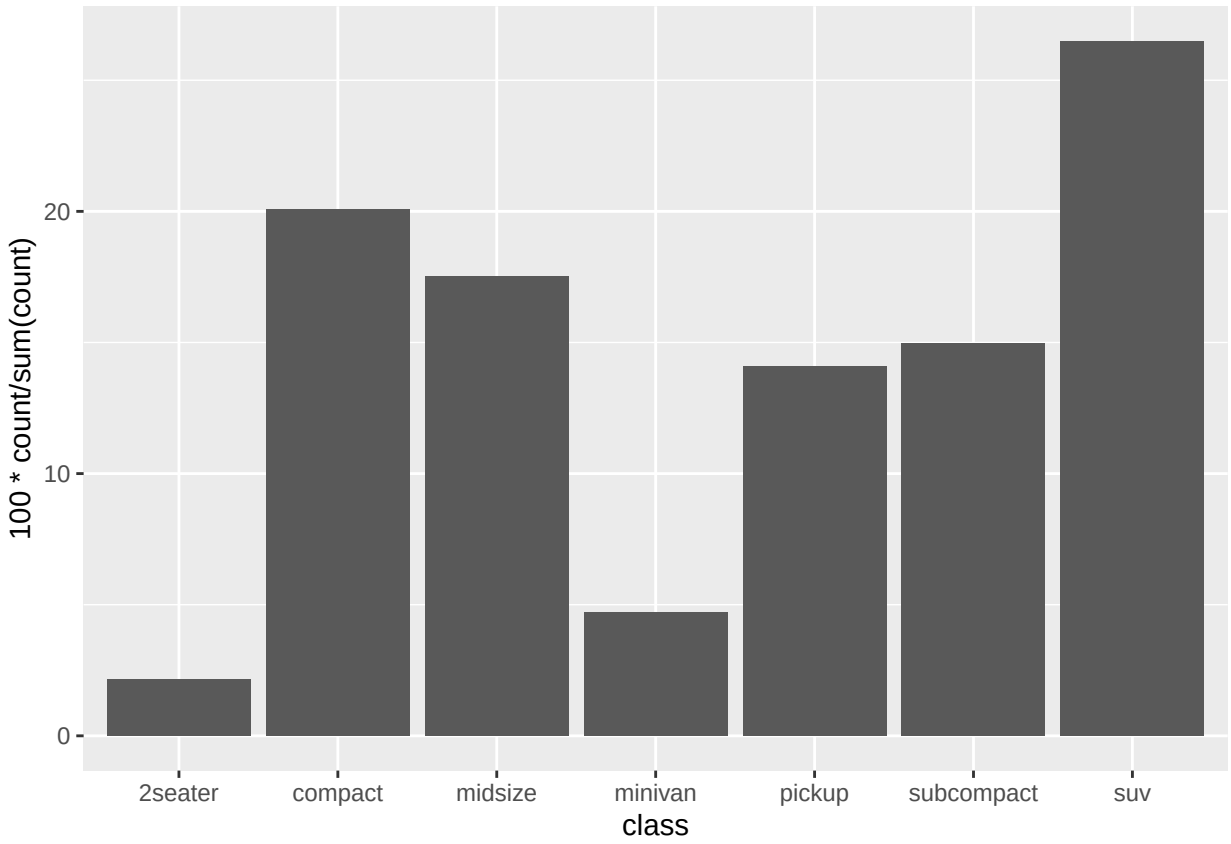
Most obvious geom+stat combinations have a dedicated geom constructor. The one above is available directly as `geom_col()`

```
ggplot(mpg_counted) +
  geom_col(aes(x = class, y = count))  # i.e. geom_col() has stat = identity, not count. If you don't h
```



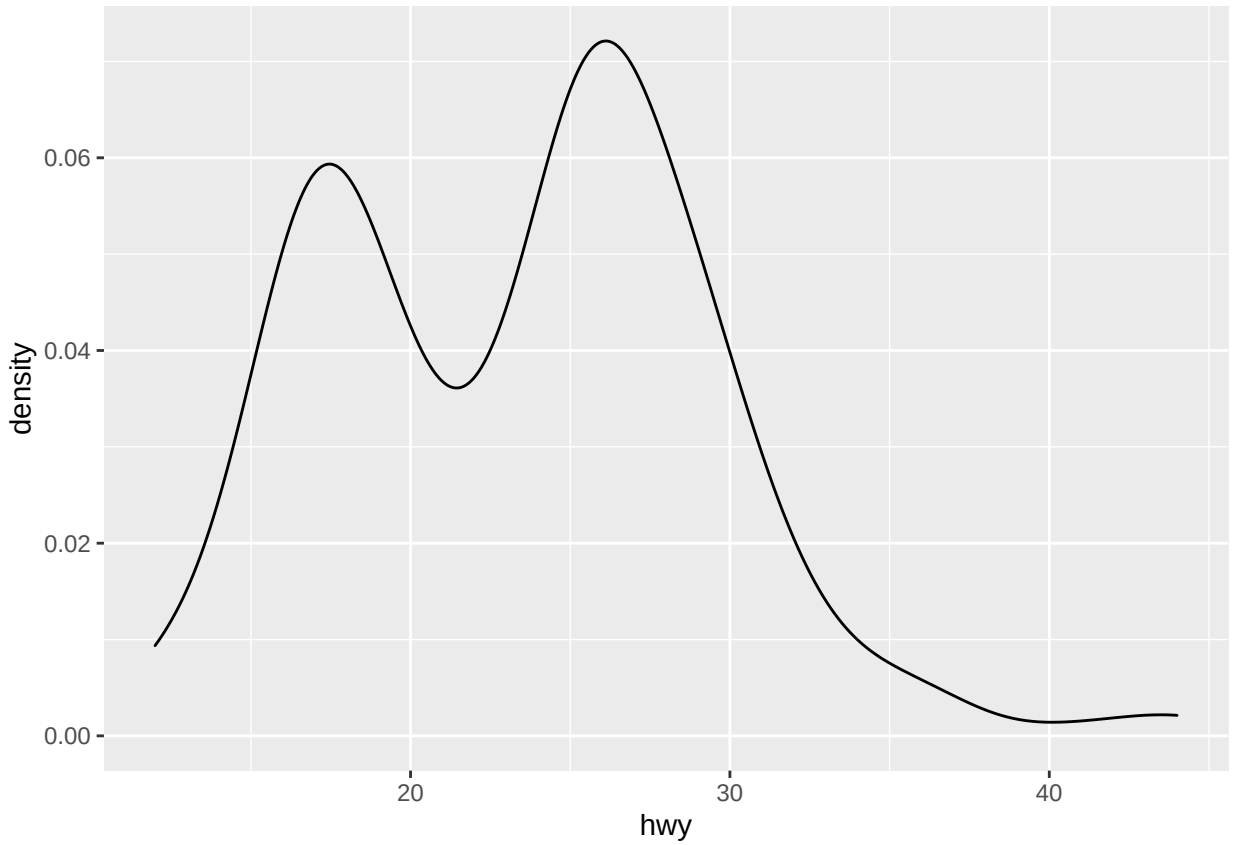
Values calculated by the stat is available with the `after_stat()` function inside `aes()`. You can do all sorts of computations inside that.

```
ggplot(mpg) +
  geom_bar(aes(x = class, y = after_stat(100 * count / sum(count))))
```



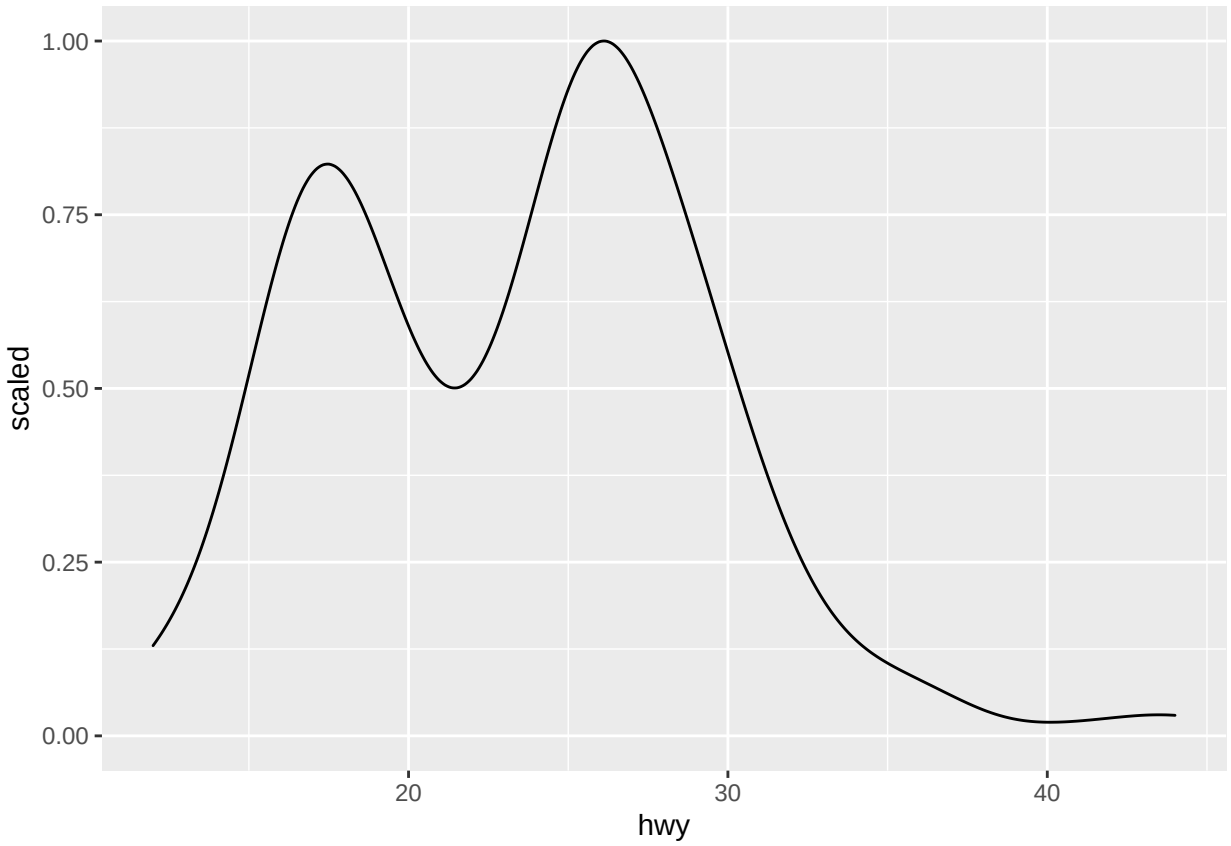
Many stats provide multiple variations of the same calculation, and provides a default (here, `density`)

```
ggplot(mpg) +
  geom_density(aes(x = hwy))
```



While the others must be used with the `after_stat()` function

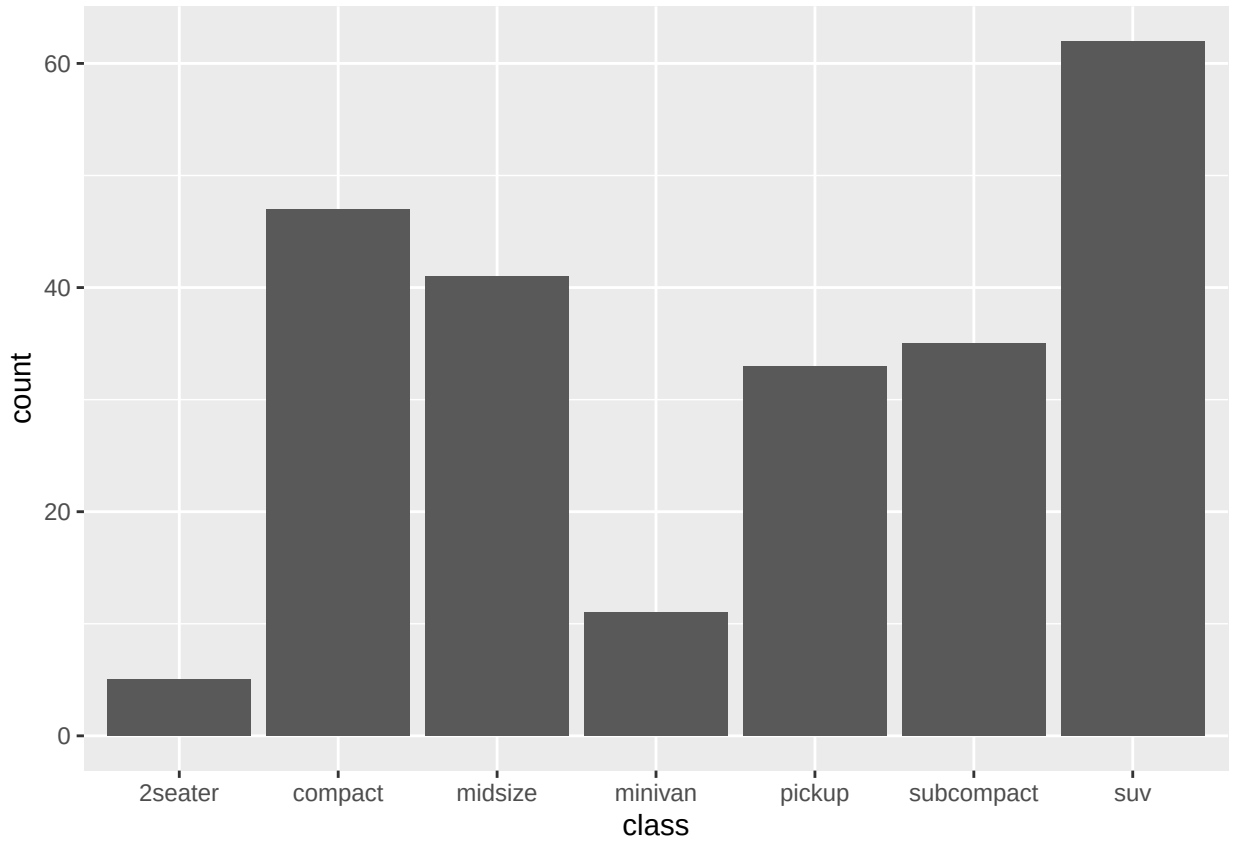
```
ggplot(mpg) +
  geom_density(aes(x = hwy, y = after_stat(scaled)))
```



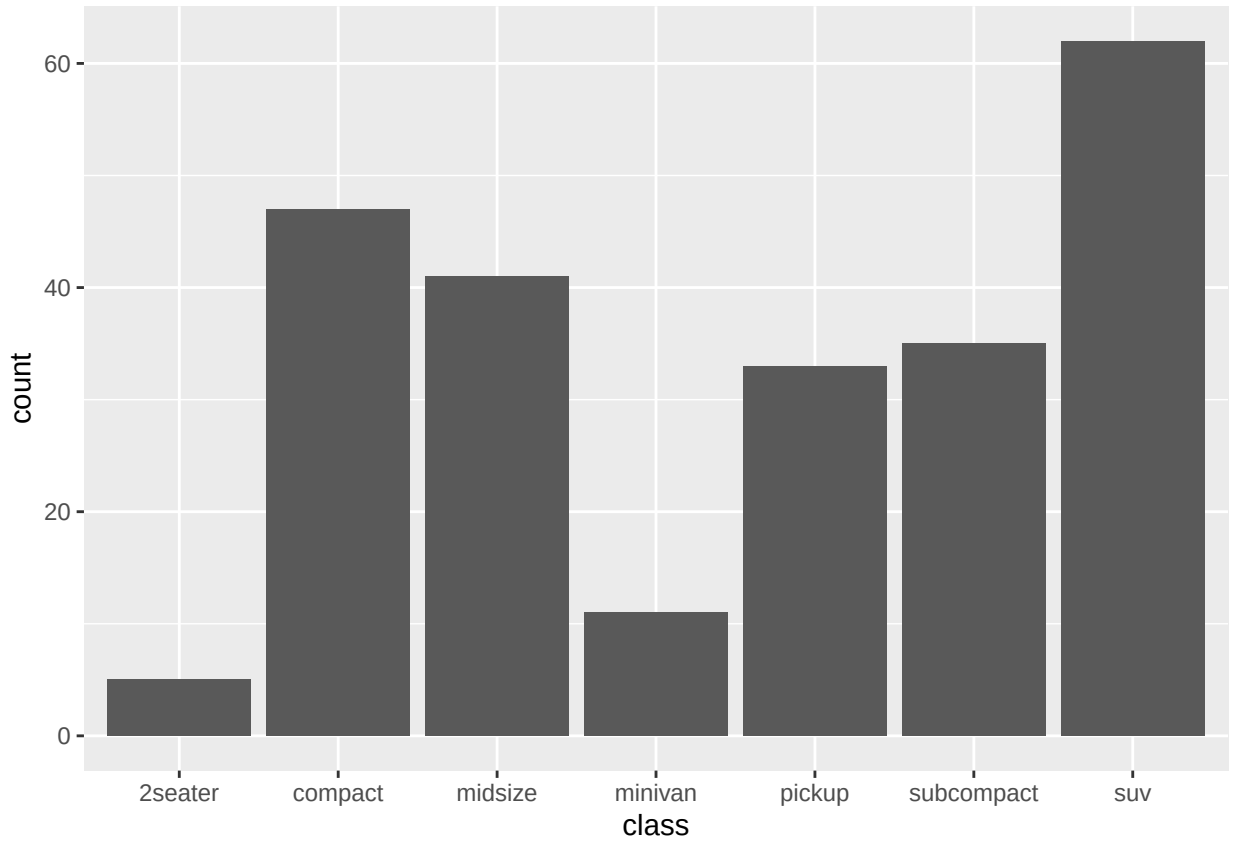
Exercises

While most people use `geom_*()` when adding layers, it is just as valid to add a `stat_*()` with an attached geom. Look at `geom_bar()` and figure out which stat it uses as default. Then modify the code to use the stat directly instead (i.e. adding `stat_*()` instead of `geom_bar()`)

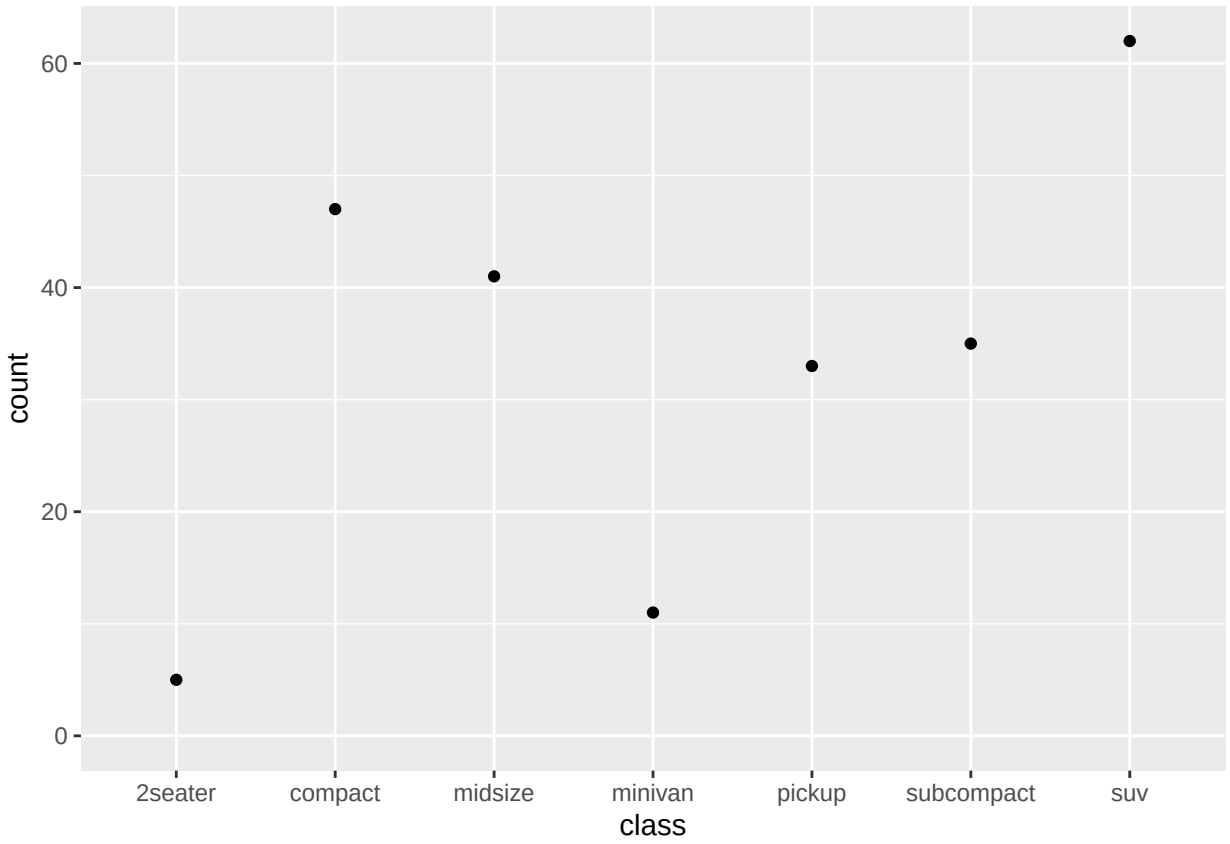
```
ggplot(mpg) +
  geom_bar(aes(x = class))
```




```
ggplot(mpg) +
  stat_count(aes(x = class))  # gives the same result as above
```

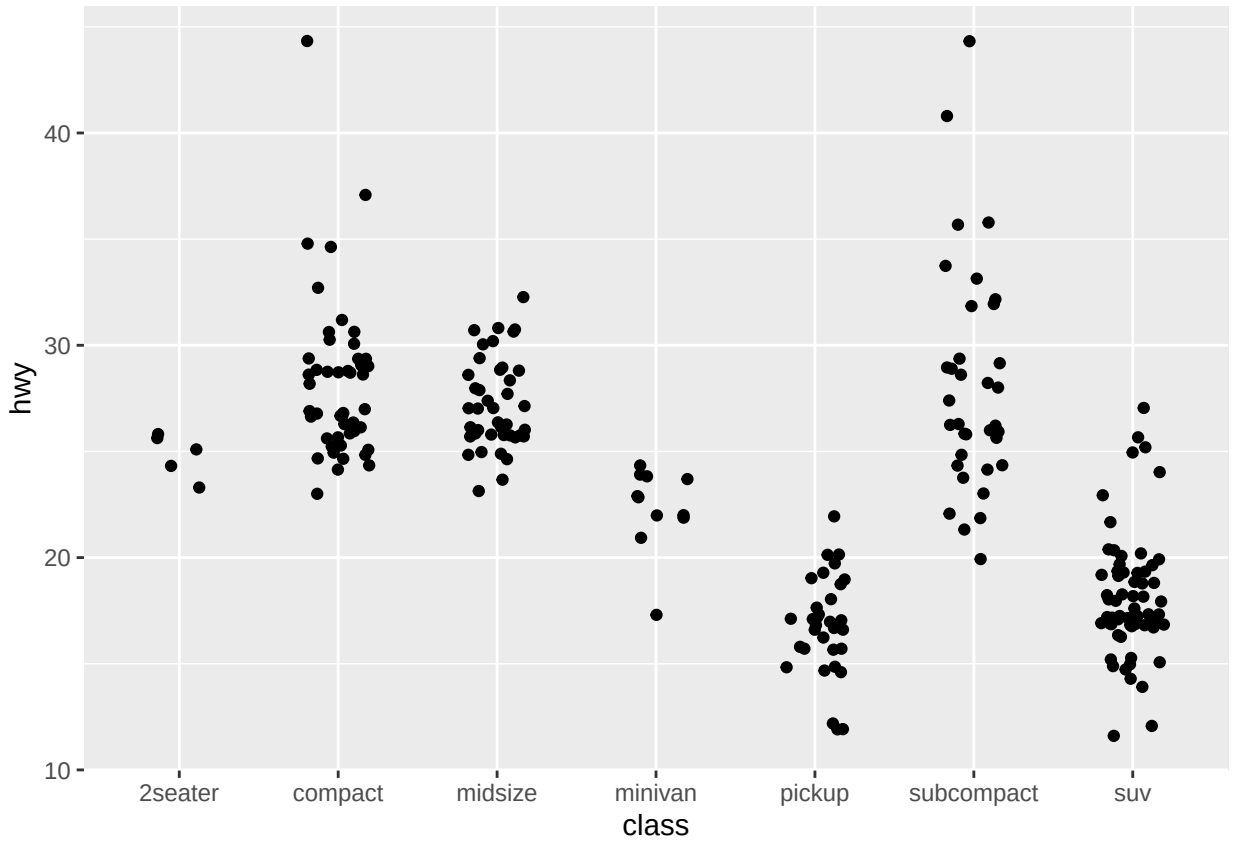



```
ggplot(mpg) +
  stat_count(aes(x = class), geom = 'point')  # changing the deafult geom of the stat
```



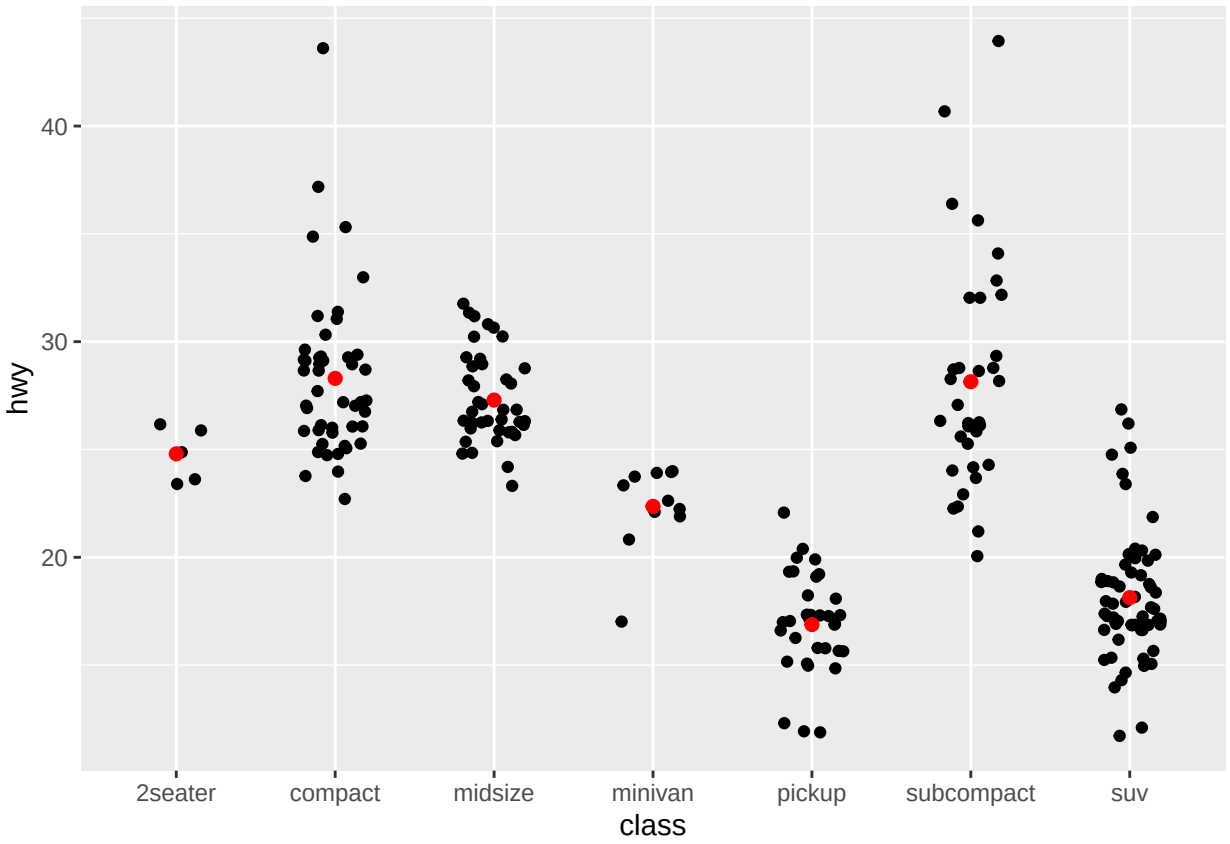
Use `stat_summary()` to add a red dot at the mean `hwy` for each group

```
ggplot(mpg) +
  geom_jitter(aes(x = class, y = hwy), width = 0.2)
```



Hint: You will need to change the default geom of `stat_summary()`

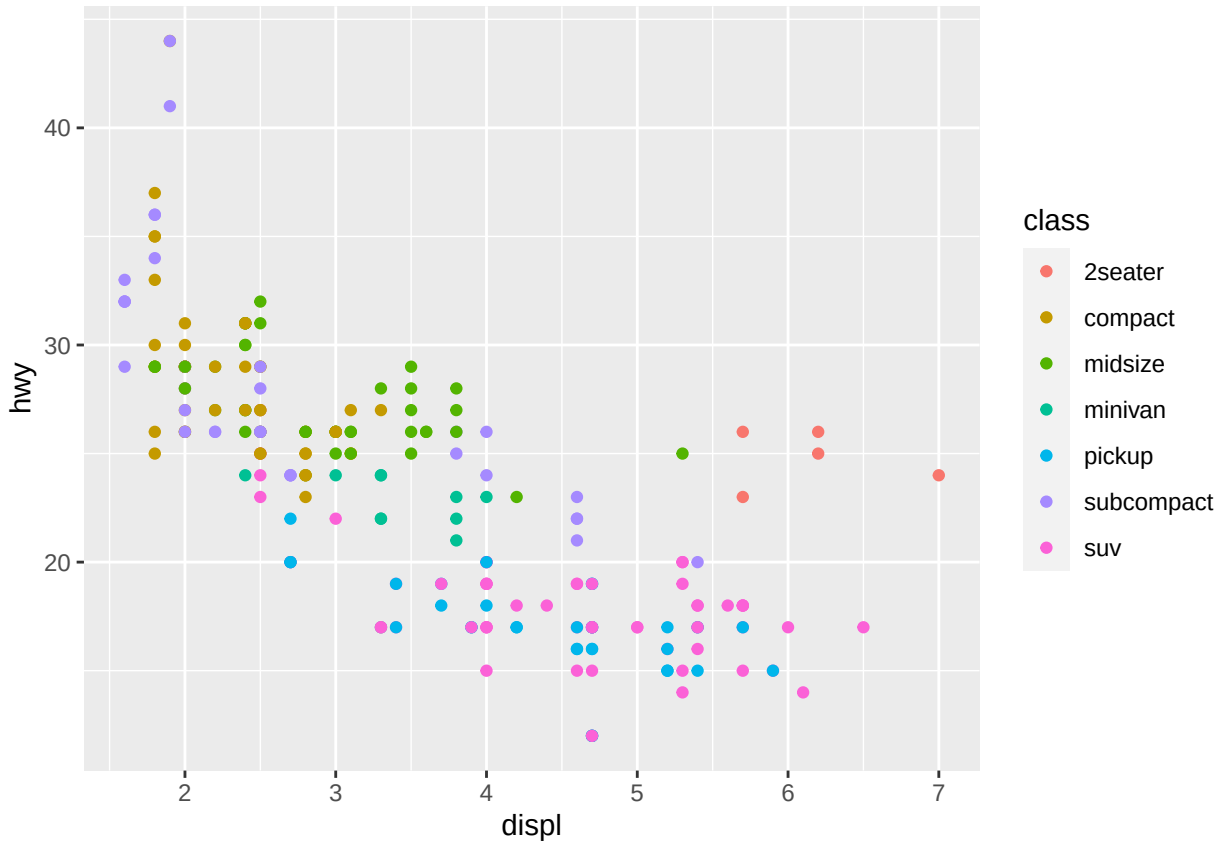
```
ggplot(mpg) +
  geom_jitter(aes(x = class, y = hwy), width = 0.2) +
  stat_summary(aes(x = class, y = hwy), fun = mean, geom = 'point', colour = 'red', size = 2)
```



Scales

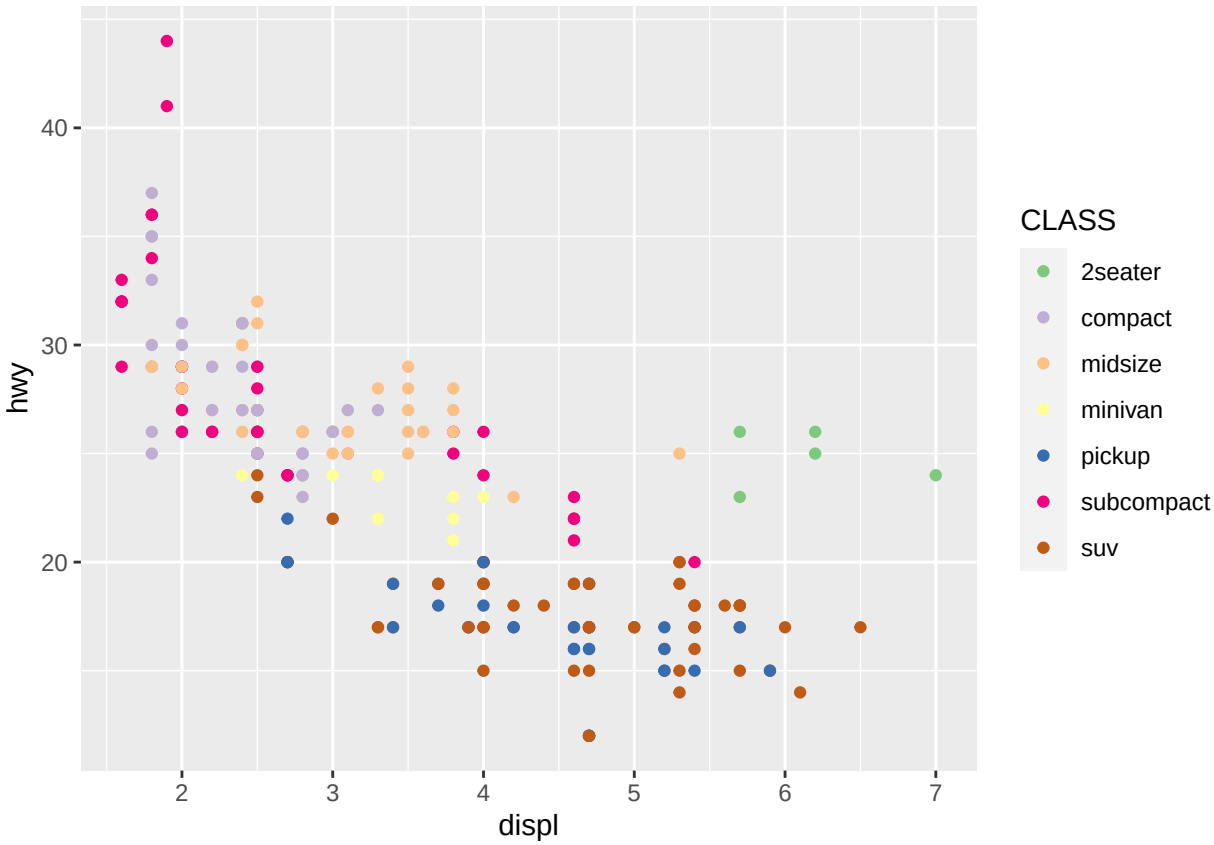
Scales define how the mapping you specify inside `aes()` should happen. All mappings have an associated scale even if not specified.

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy, colour = class))
```



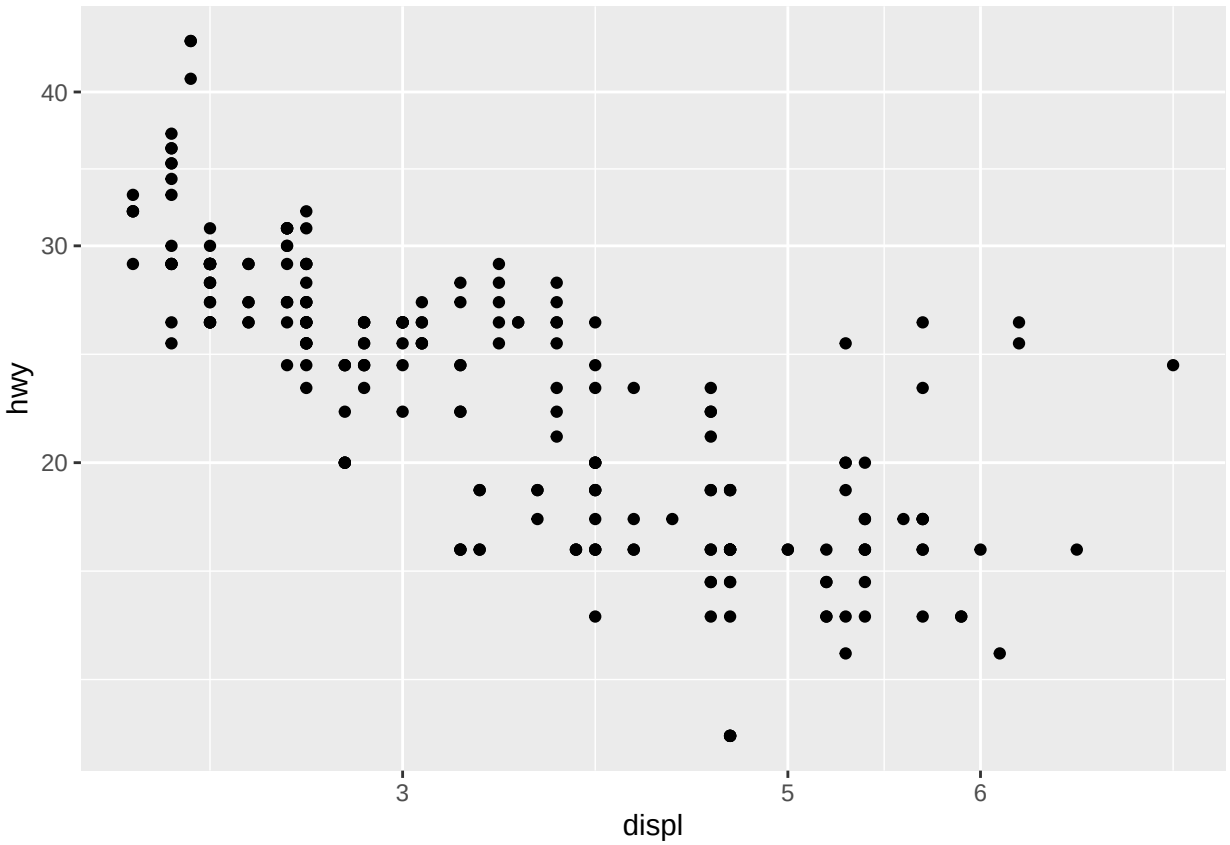
take control by adding one explicitly. All scales follow the same naming conventions.

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy, colour = class)) +
  scale_colour_brewer(name = "CLASS", type = 'qual')  # note how the variety of colours has now changed
```



Positional mappings (x and y) also have associated scales.

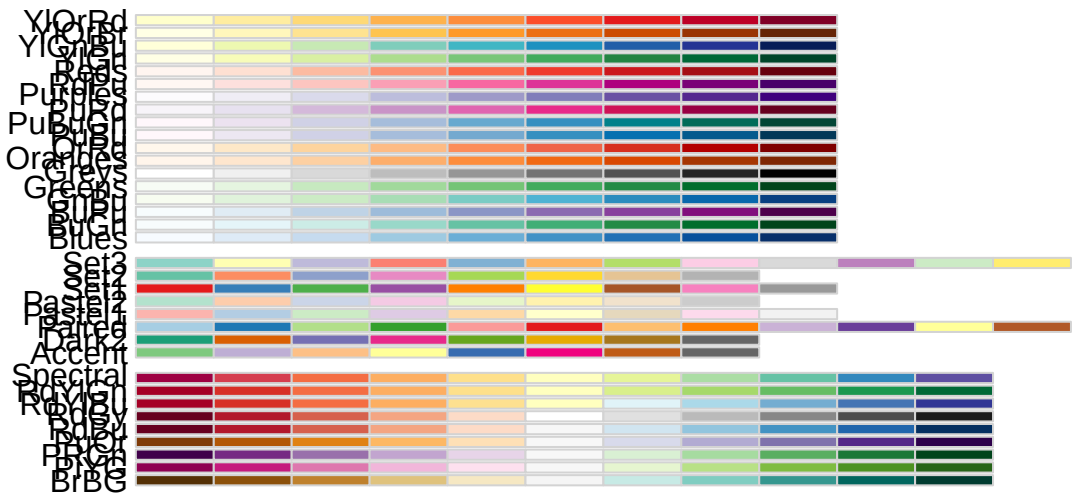
```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  scale_x_continuous(breaks = c(3, 5, 6)) +
  scale_y_continuous(trans = 'log10')   # trans = transformation
```



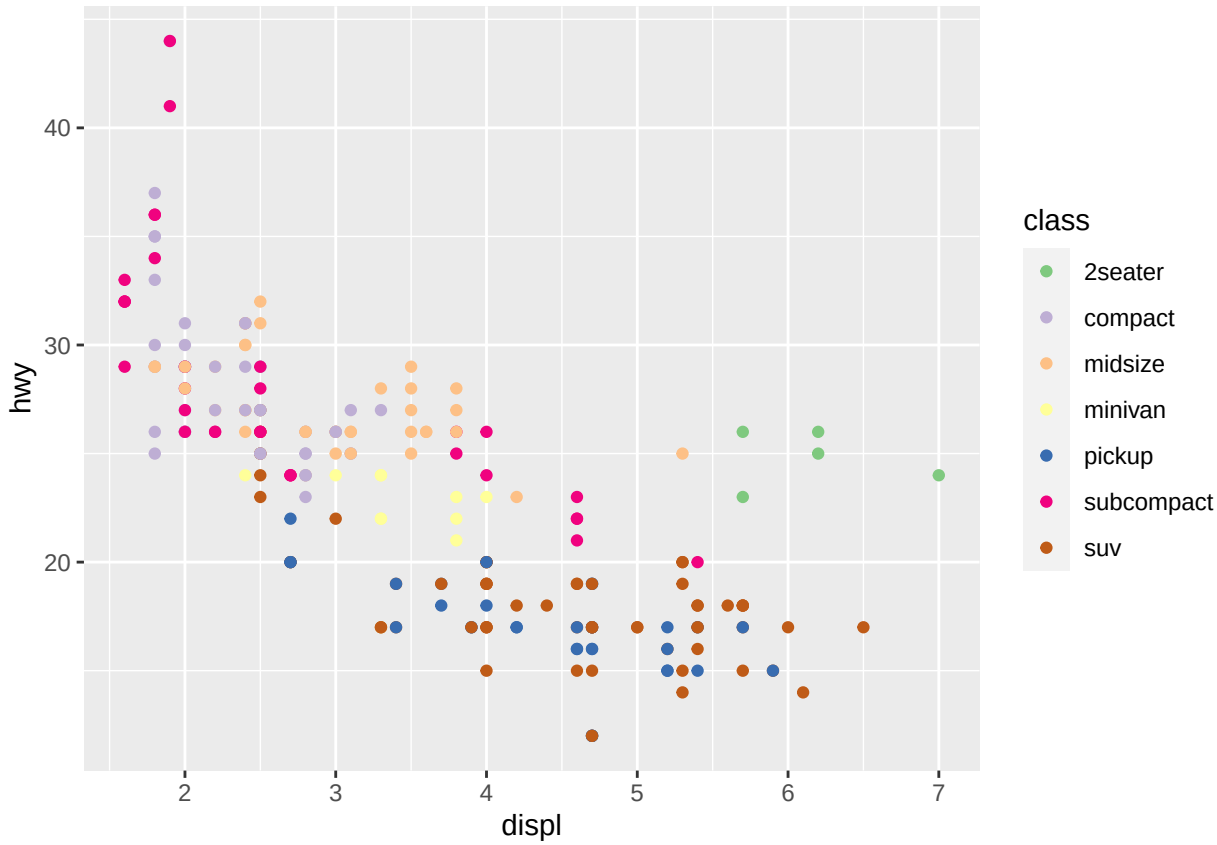
Exercises

Use `RColorBrewer::display.brewer.all()` to see all the different palettes from Color Brewer and pick your favourite. Modify the code below to use it

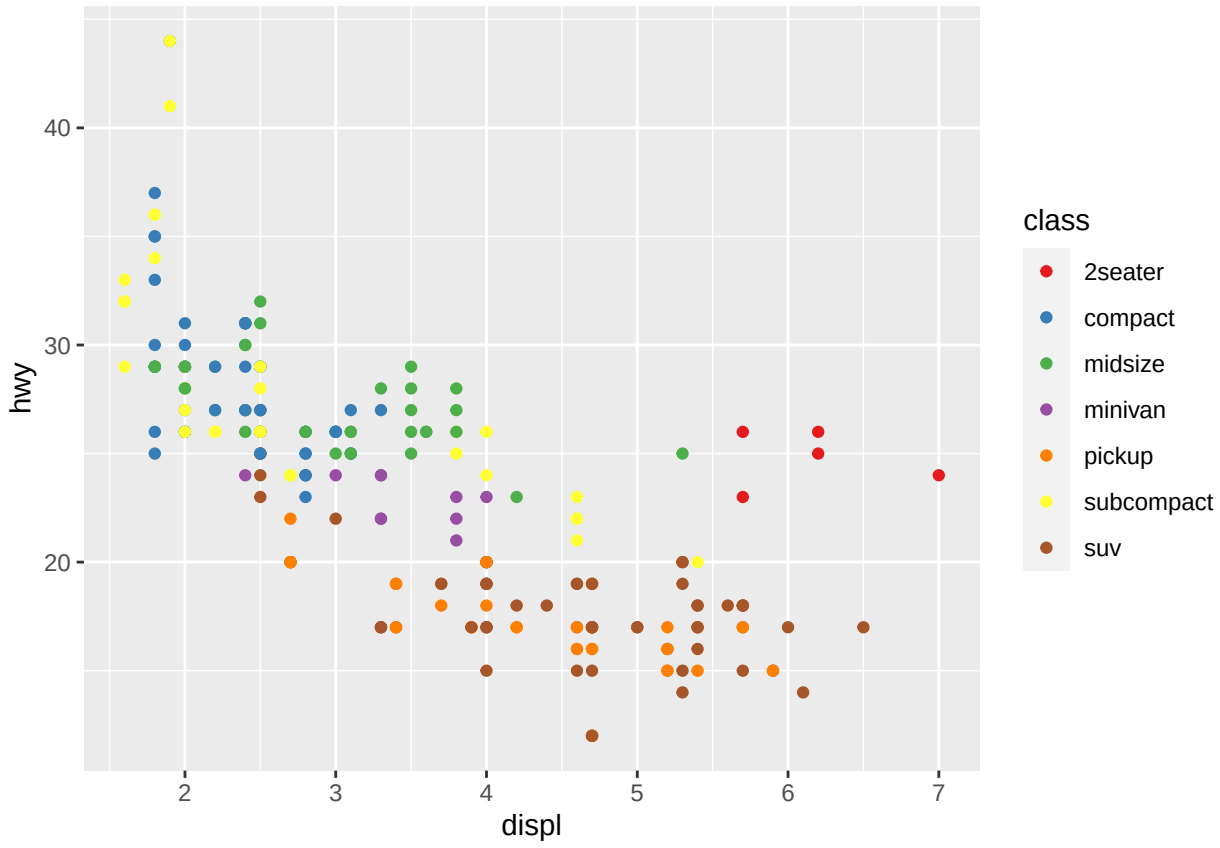
```
RColorBrewer::display.brewer.all()
```




```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy, colour = class)) +
  scale_colour_brewer(type = 'qual')
```

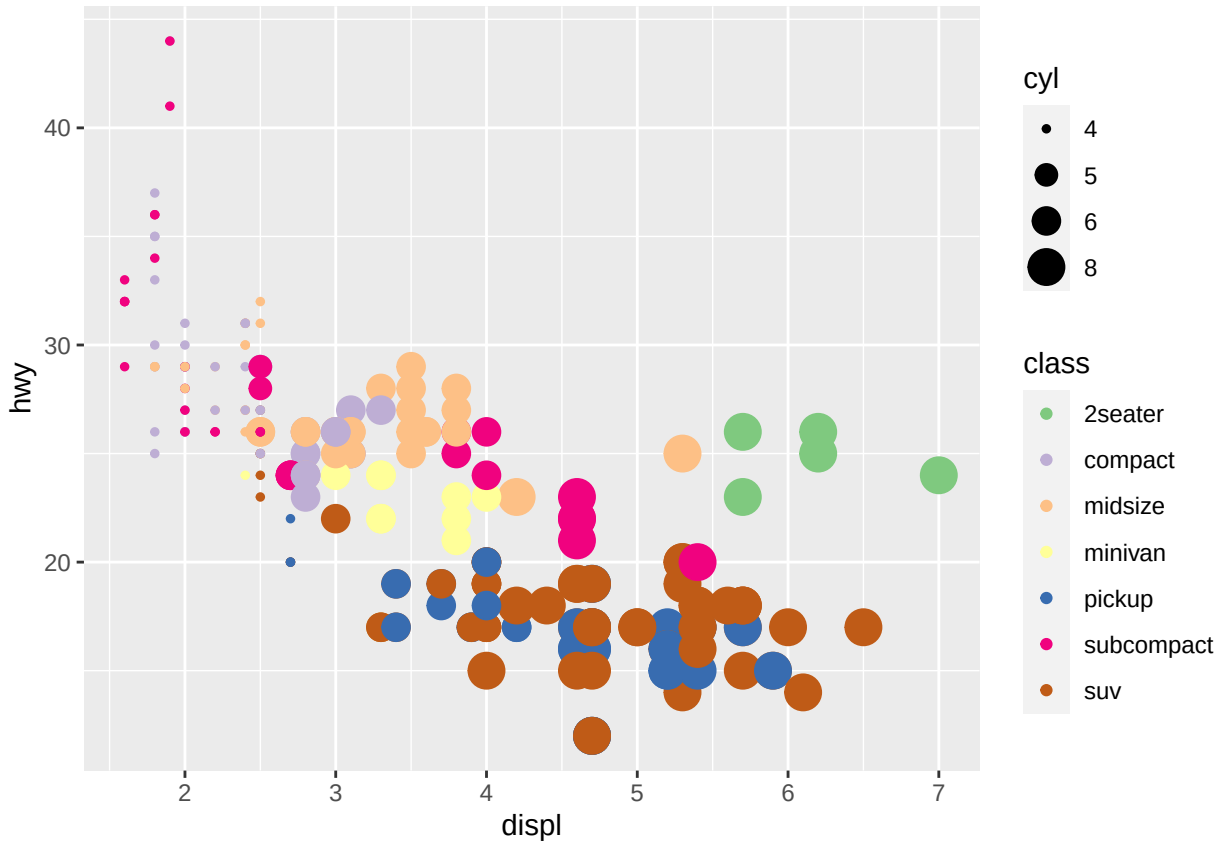



```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy, colour = class)) +
  scale_colour_brewer(palette = 'Set1')  # one of the qualitative colour sets above
```

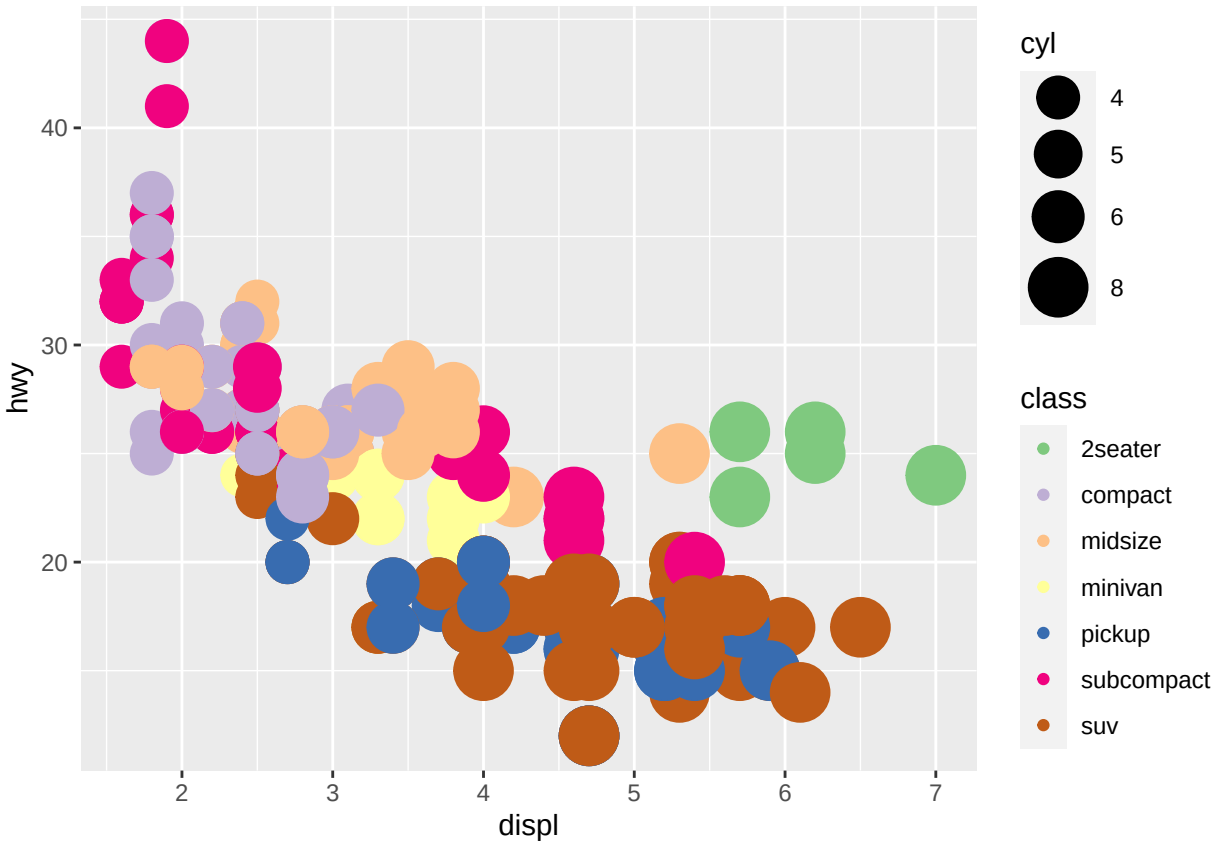


Modify the code below to create a bubble chart (scatterplot with size mapped to a continuous variable) showing `cyl` with size. Make sure that only the present amount of cylinders (4, 5, 6, and 8) are present in the legend.

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy, colour = class, size = cyl)) +
  scale_colour_brewer(type = 'qual') +
  scale_size(breaks = c(4, 5, 6, 8))  # gets rid of the non-existent cyl = 7
```




```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy, colour = class, size = cyl)) +
  scale_colour_brewer(type = 'qual') +
  scale_size_area(breaks = c(4, 5, 6, 8), max_size = 10)  # makes cyl=4 bigger than before when it was
```

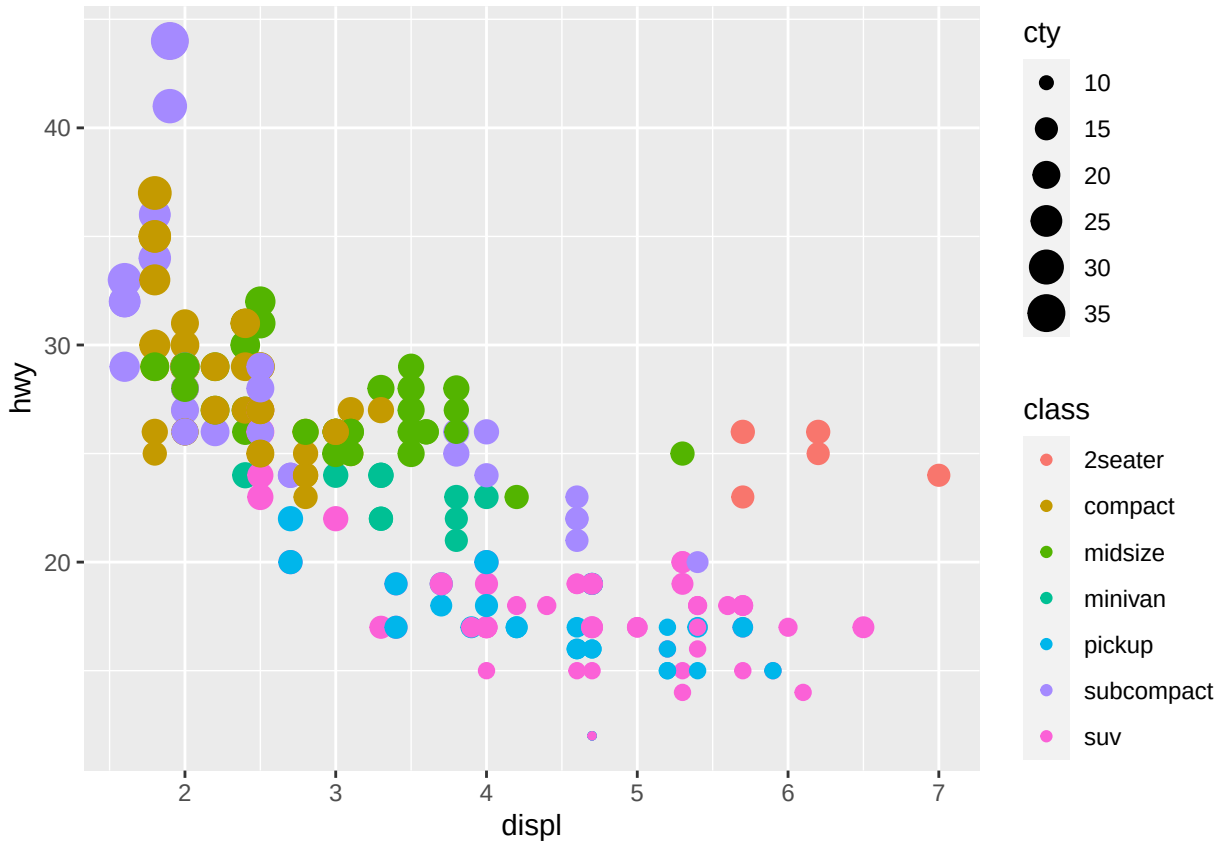


Hint: The `breaks` argument in the scale is used to control which values are present in the legend.

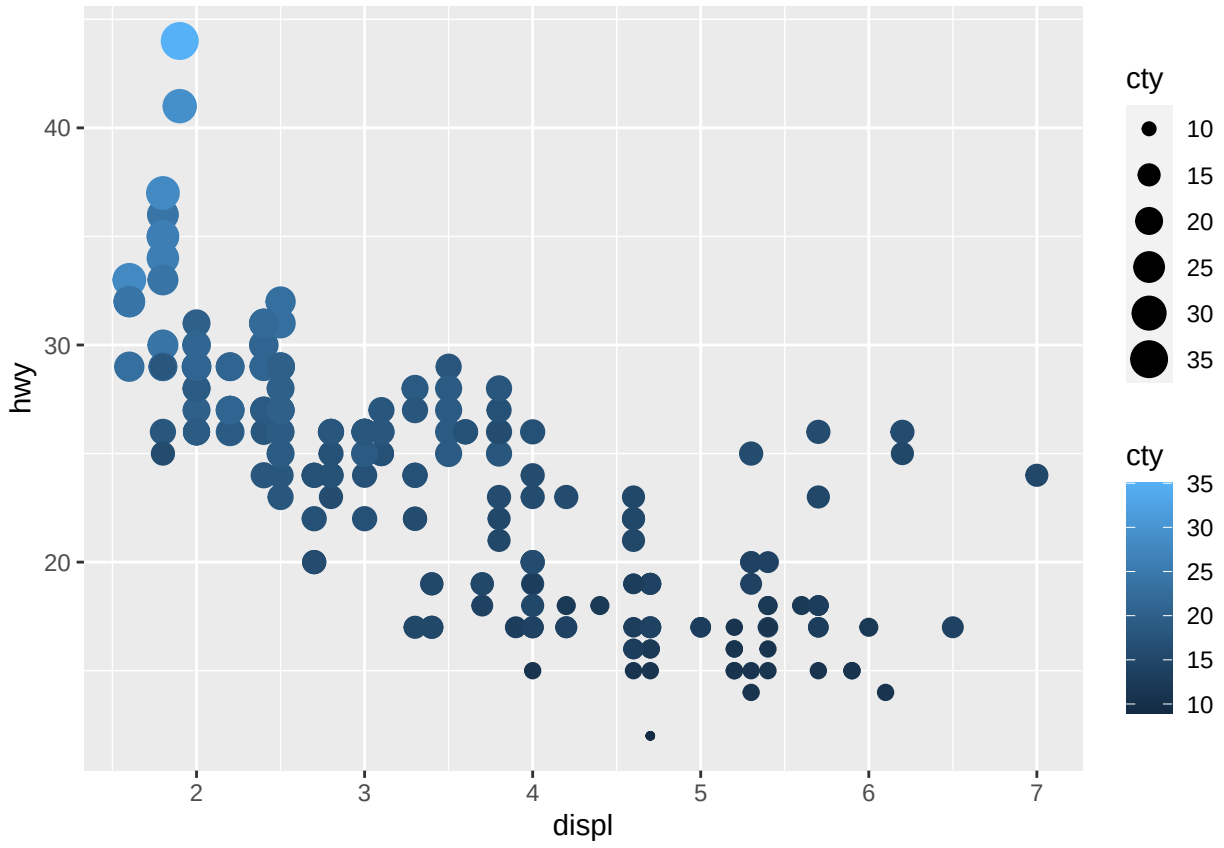
Explore the different types of size scales available in ggplot2. Is the default the most appropriate here?

Modify the code below so that colour is no longer mapped to the discrete `class` variable, but to the continuous `cty` variable. What happens to the guide?

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy, colour = class, size = cty))
```

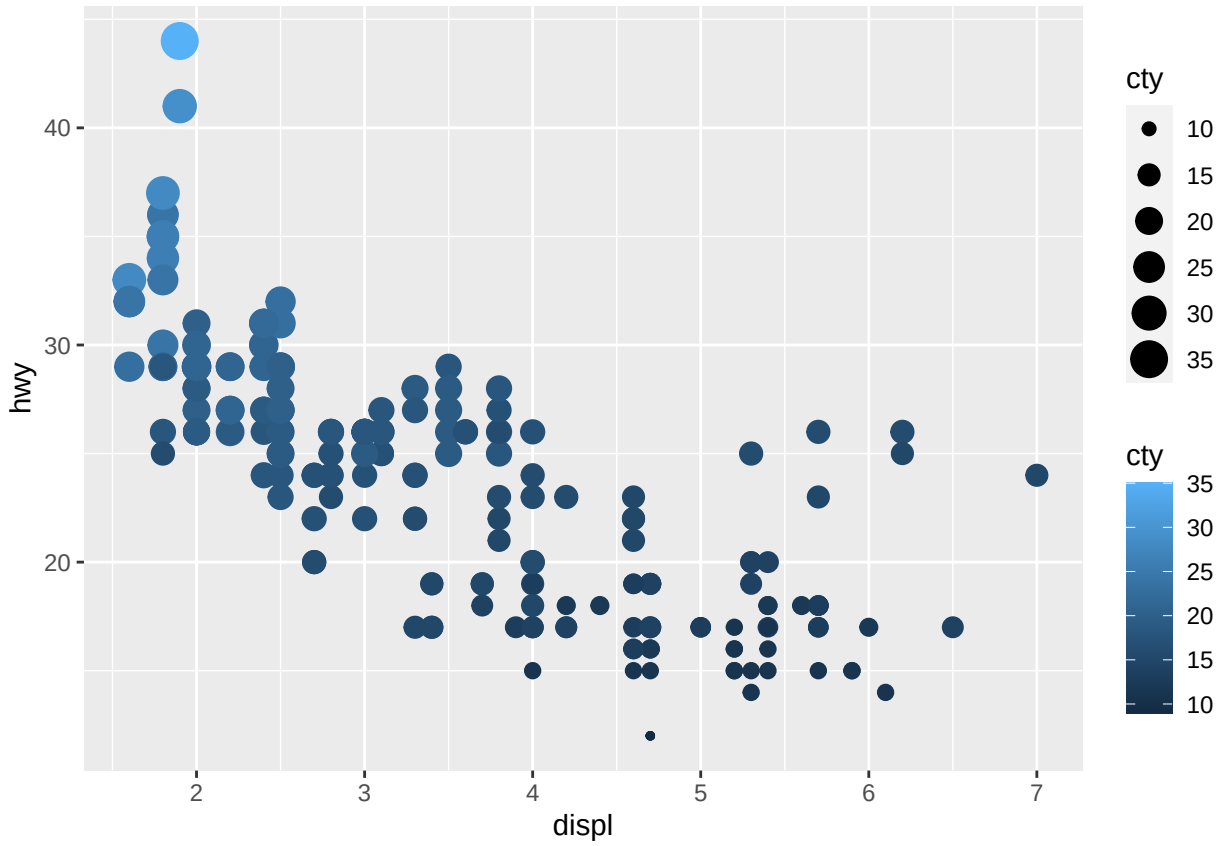



```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy, colour = cty, size = cty))
```

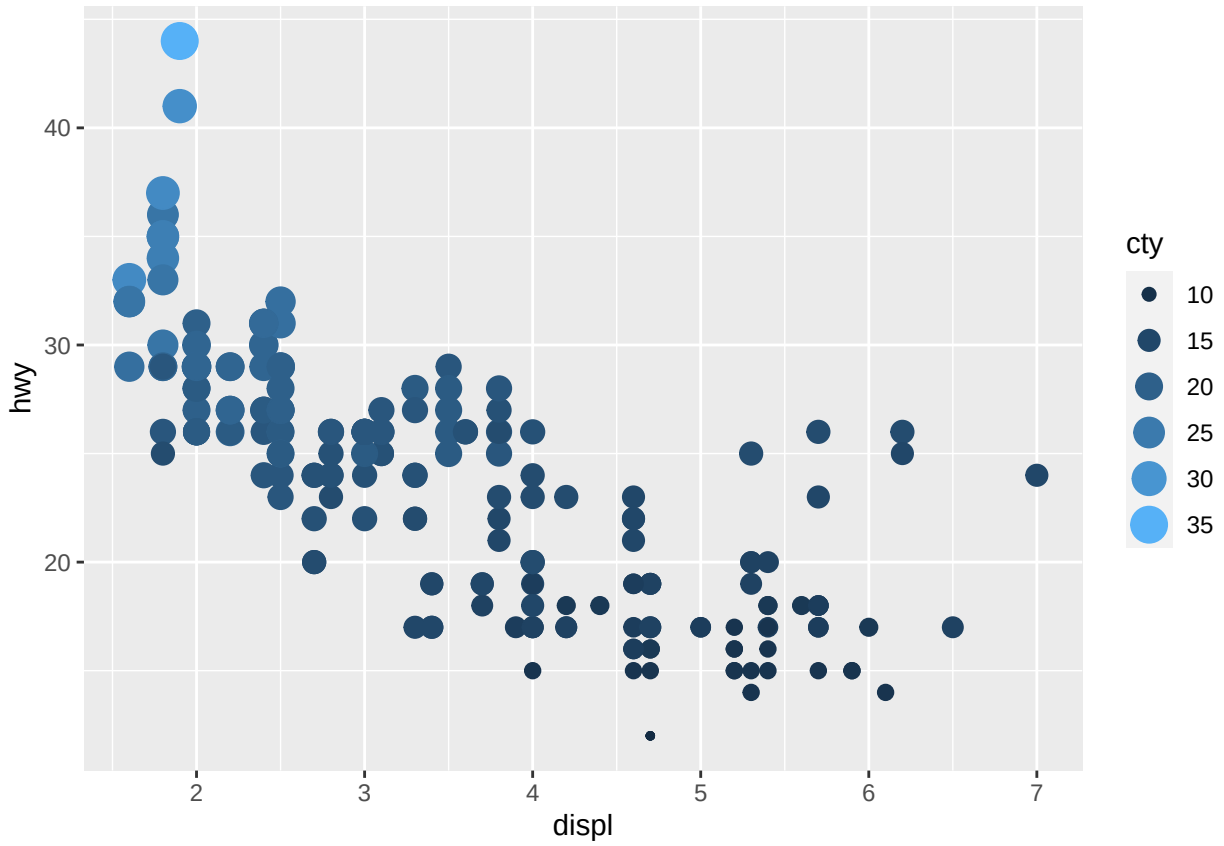


The type of guide can be controlled with the `guide` argument in the scale, or with the `guides()` function. Continuous colours have a gradient colour bar by default, but setting it to `legend` will turn it back to the standard look. What happens when multiple aesthetics are mapped to the same variable and uses the guide type?

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy, colour = cty, size = cty))
```



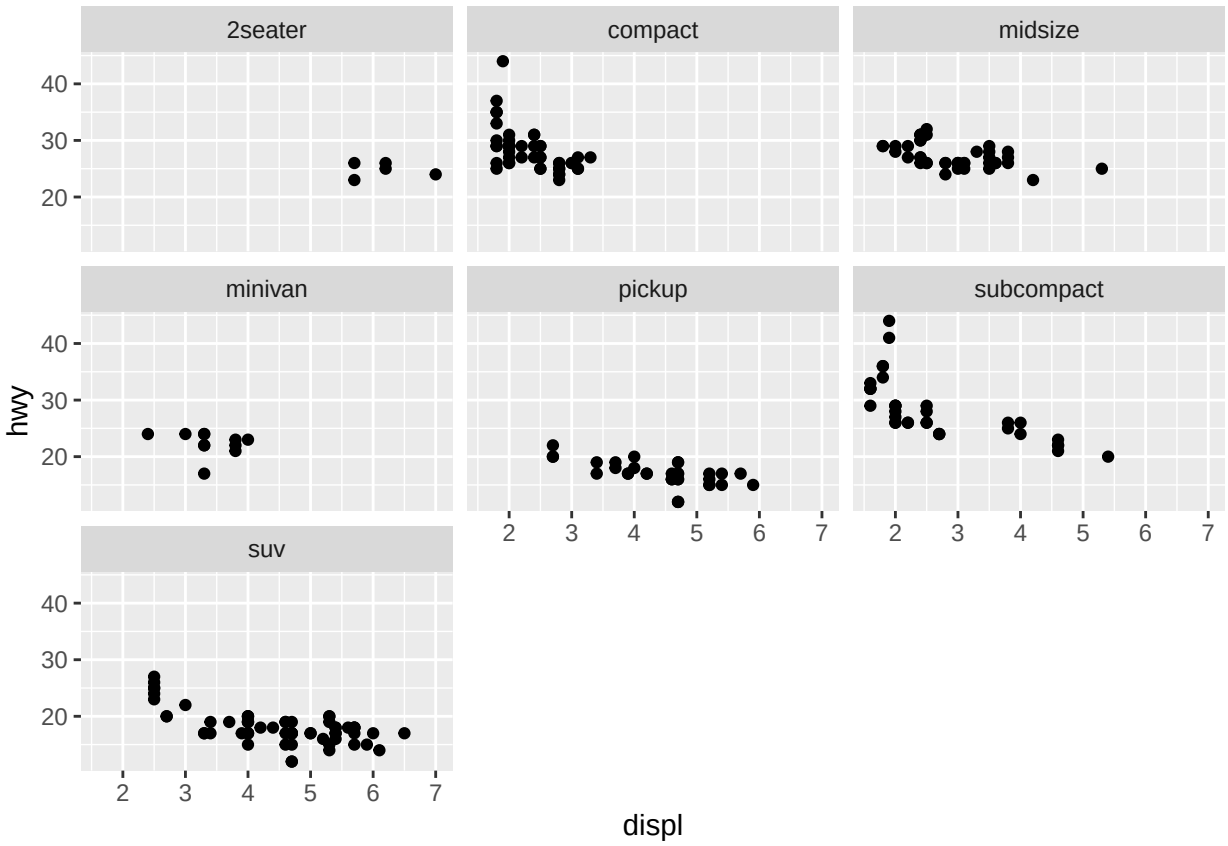

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy, colour = cty, size = cty)) +
  guides(colour = 'legend')  # integrates the two legends into one!
```



Facets

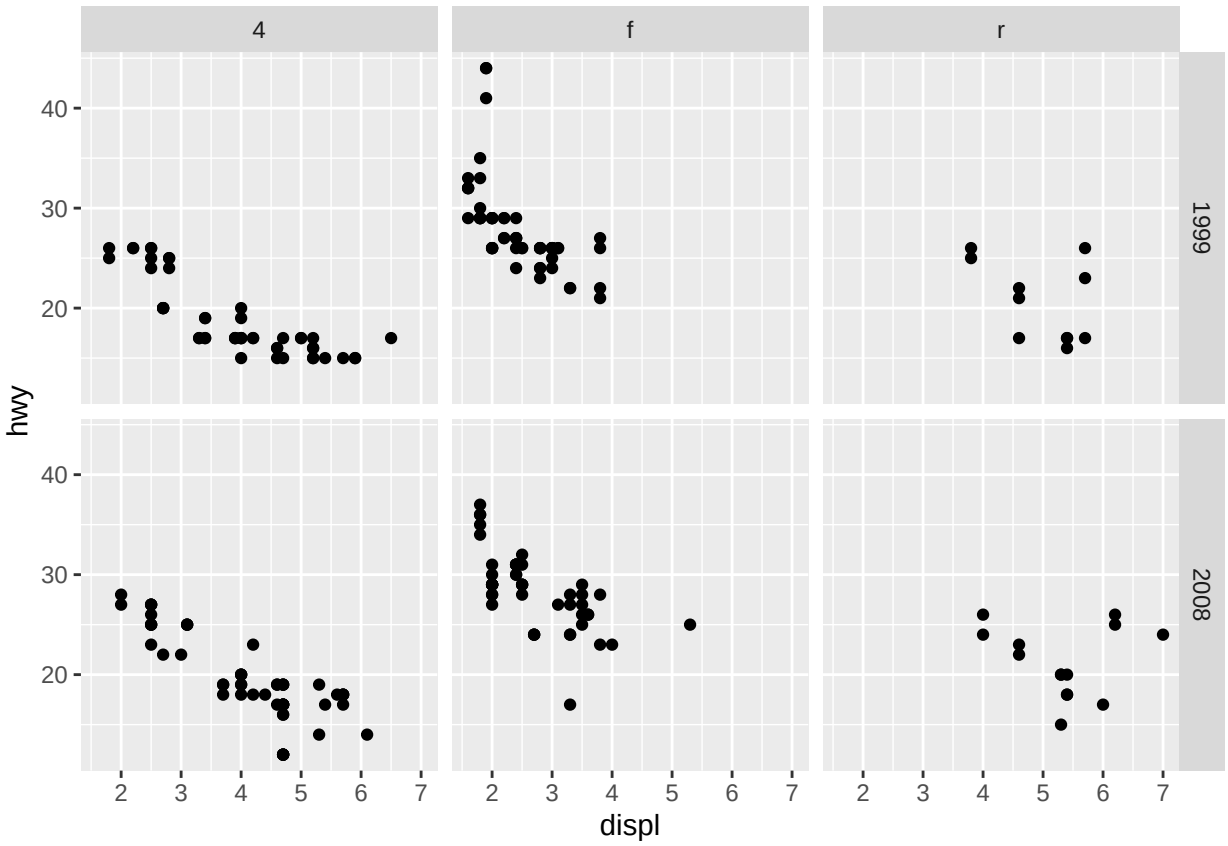
The facet defines how data is split among panels. The default facet (`facet_null()`) puts all the data in a single panel, while `facet_wrap()` and `facet_grid()` allows you to specify different types of small multiples

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_wrap(~ class)
```



And with `facet_grid()`:

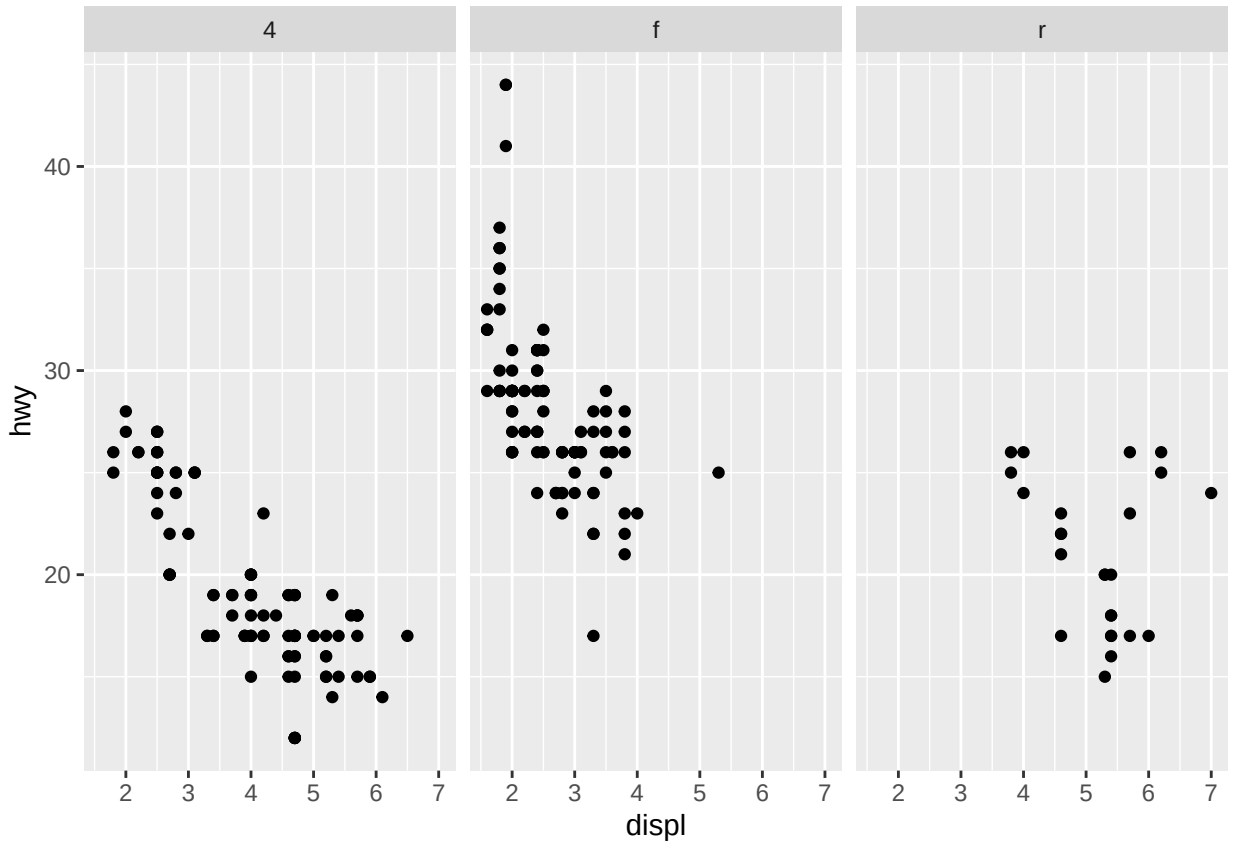
```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_grid(year ~ drv)
```



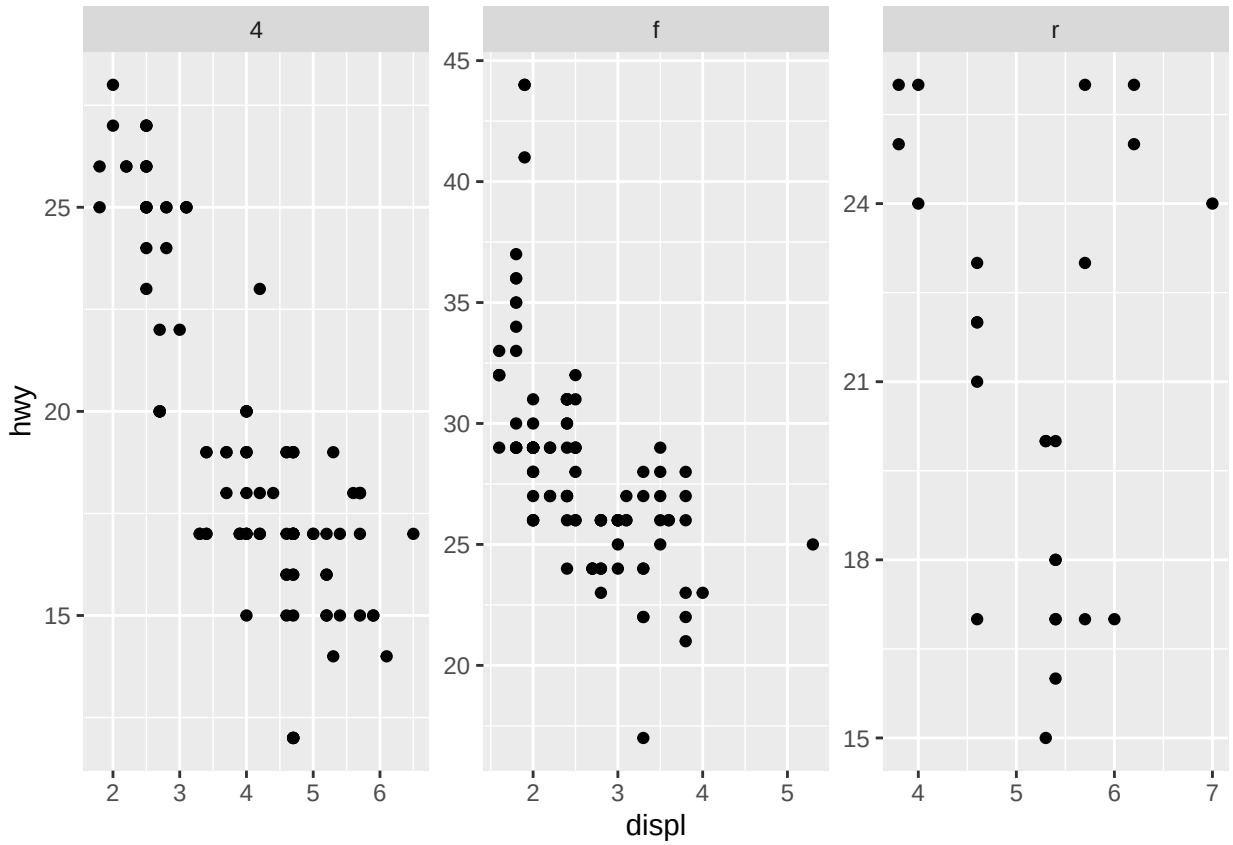
Exercises

One of the great things about facets is that they share the axes between the different panels. Sometimes this is undiserable though, and the behaviour can be changed with the `scales` argument. Experiment with the different possible settings in the plot below:

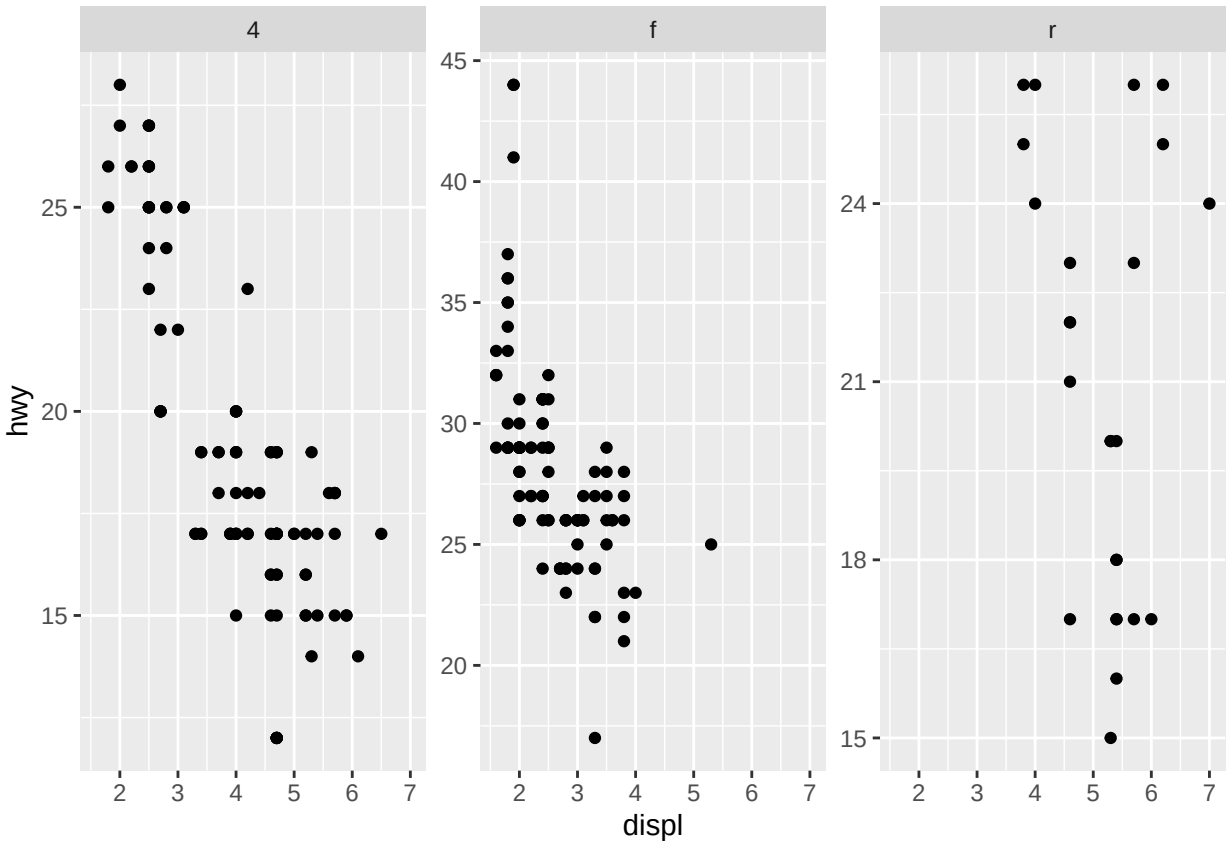
```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_wrap(~ drv)
```




```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_wrap(~ drv, scales = 'free')  # each plot gets its own x- and y-axis
```

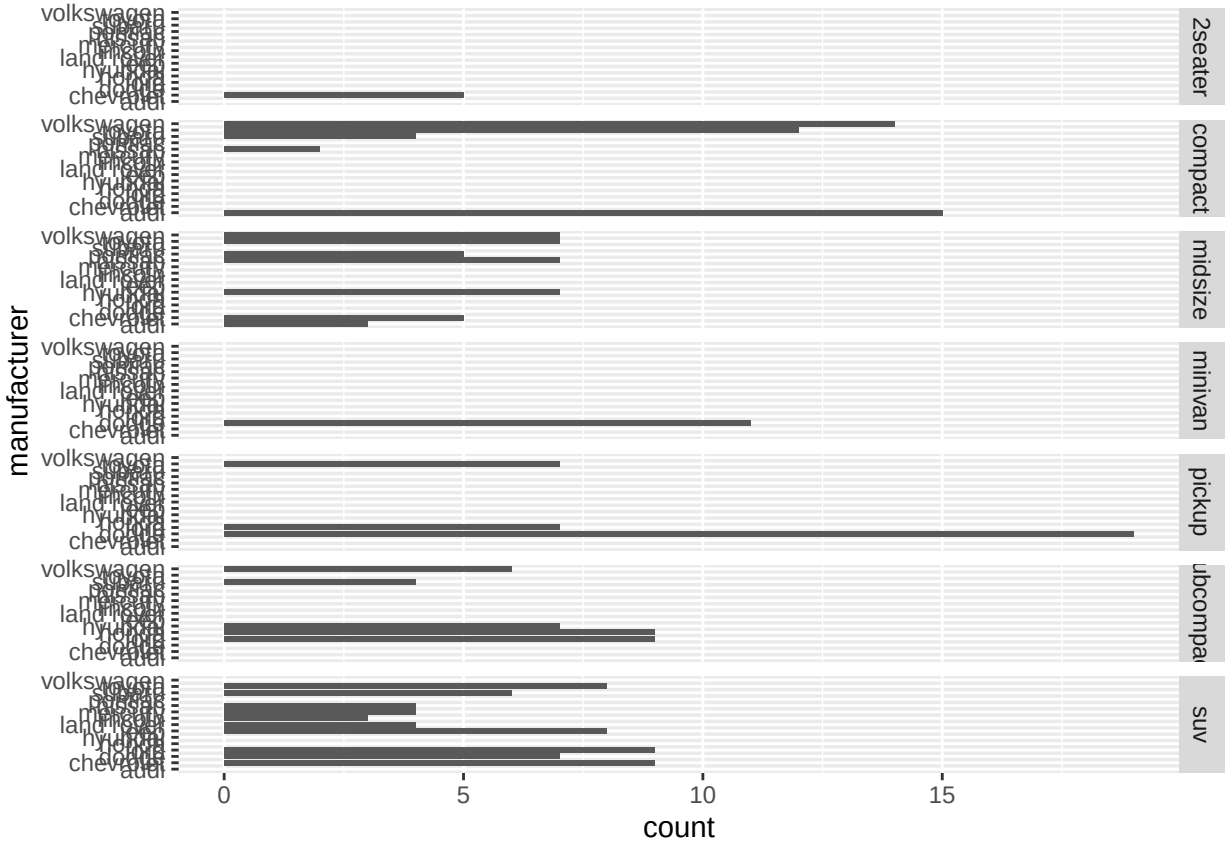



```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_wrap(~ drv, scales = 'free_y') # only y-axis is free, x-axis is fixed
```



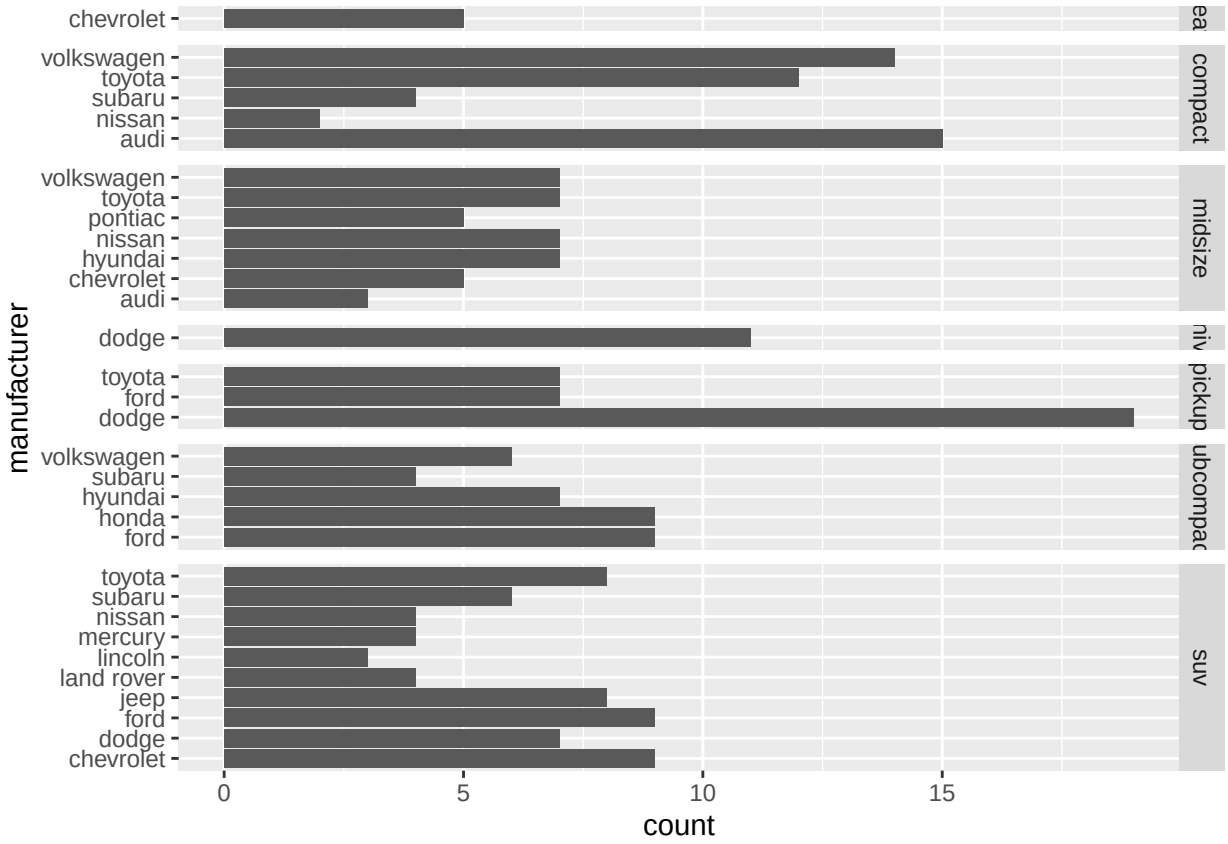
Usually the space occupied by each panel is equal. This can create problems when different scales are used. Modify the code below so that the y scale differs between the panels in the plot. What happens?

```
# Look how by using the same scale for each panel creates an ugly graph with too much whitespace:

ggplot(mpg) +
  geom_bar(aes(y = manufacturer)) +
  facet_grid(class ~ .)
```



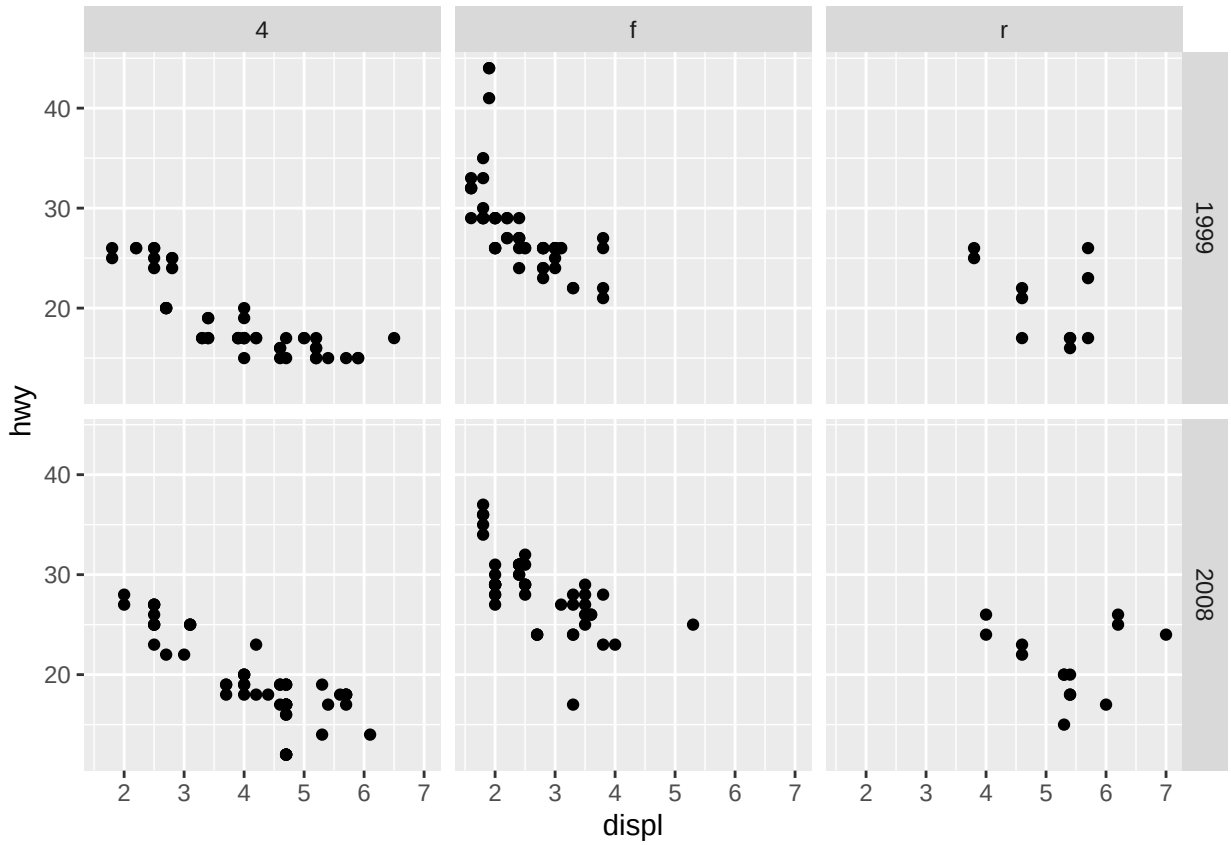
Use the `space` argument in `facet_grid()` to change the plot above so each bar has the same width again.

```
ggplot(mpg) +
  geom_bar(aes(y = manufacturer)) +
  facet_grid(class ~ ., space = 'free_y', scales = 'free_y')  # the space argument lets us reomove all
```

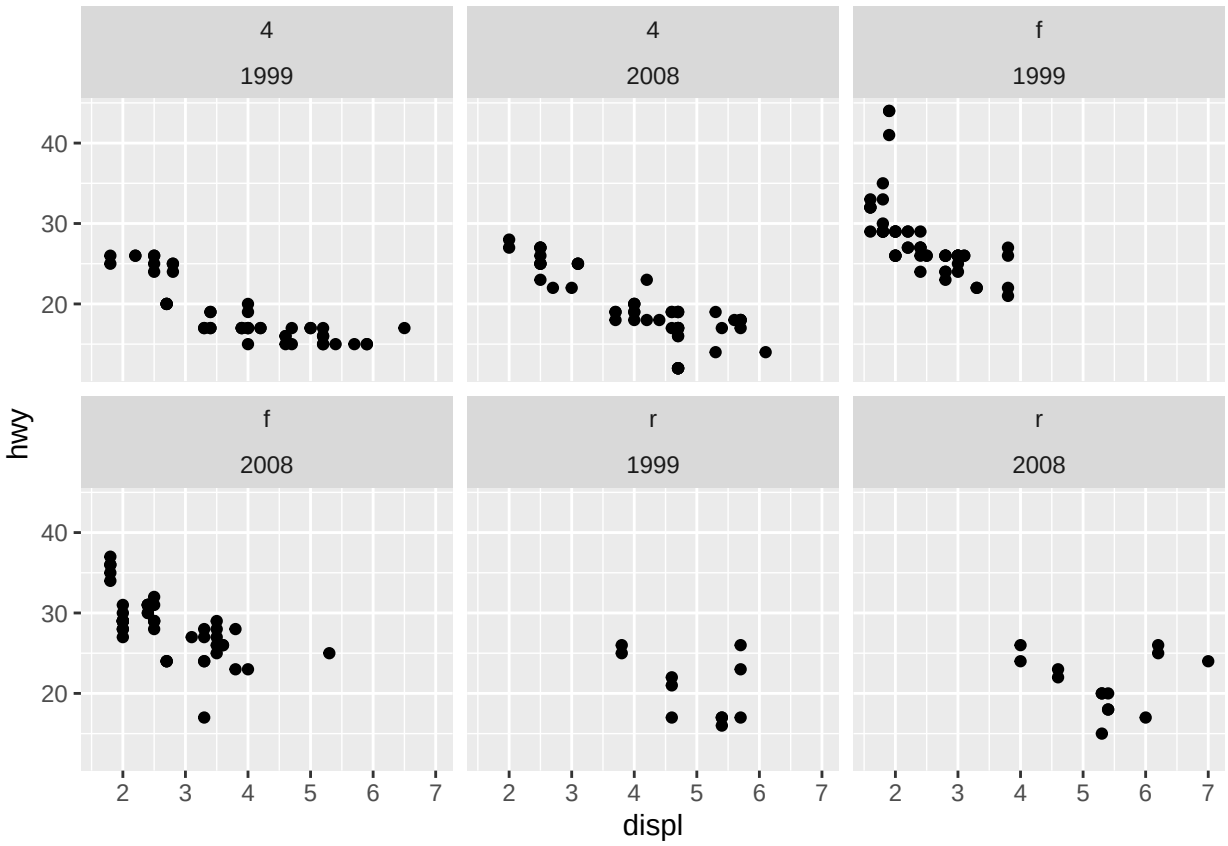


Facets can be based on multiple variables by adding them together. Try to recreate the same panels present in the plot below by using `facet_wrap()`

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_grid(year ~ drv)
```



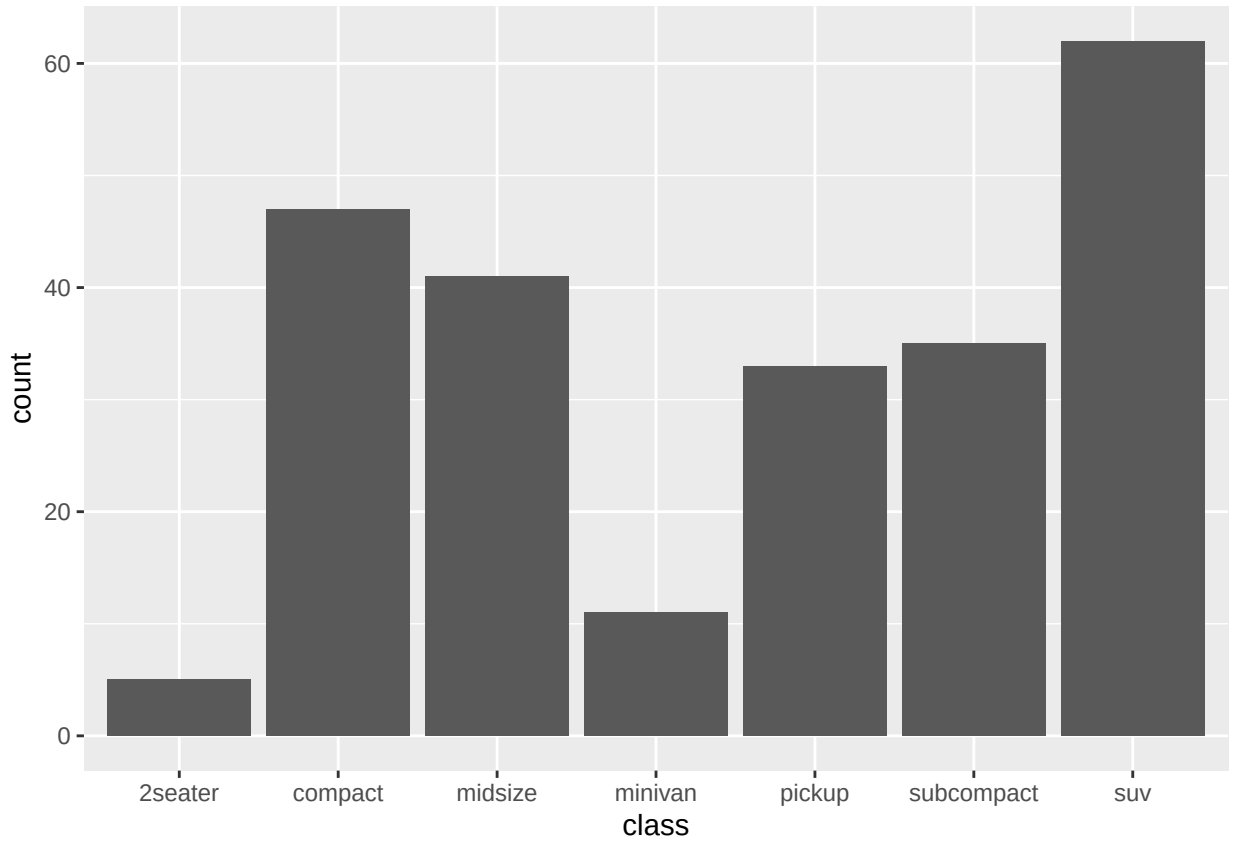

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_wrap(~ drv + year)  # Gives same panels but arranged a different, less intuitive way
```



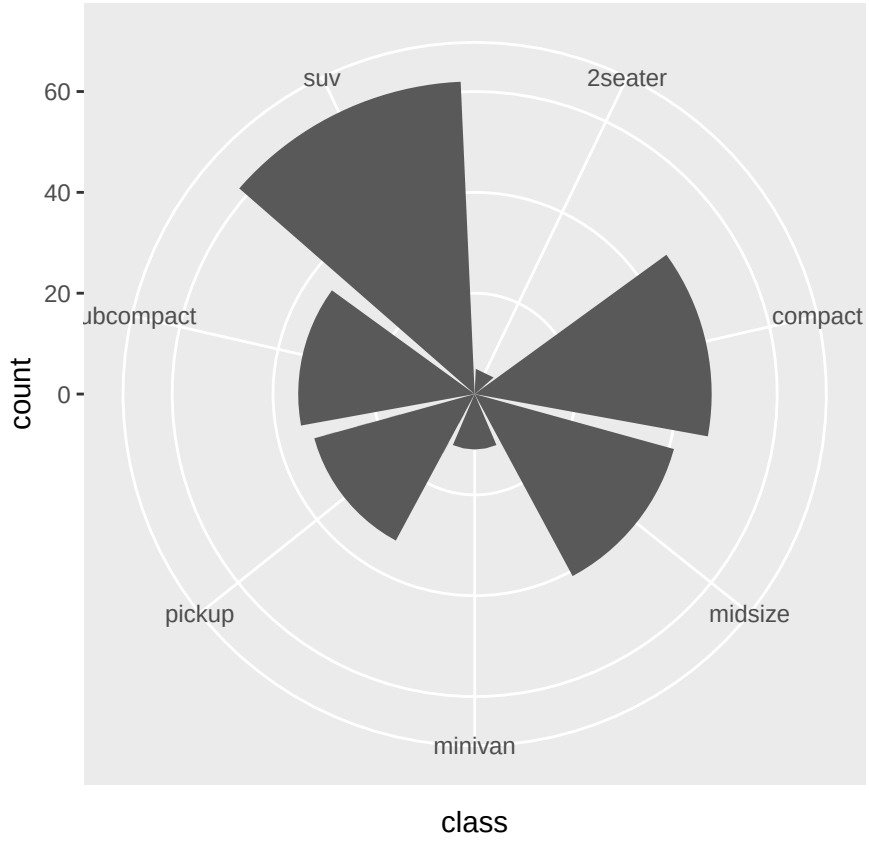
Coordinates

The coordinate system is the fabric you draw your layers on in the end. The default 'coord_cartesion provides the standard rectangular x-y coordinate system. Changing the coordinate system can have dramatic effects

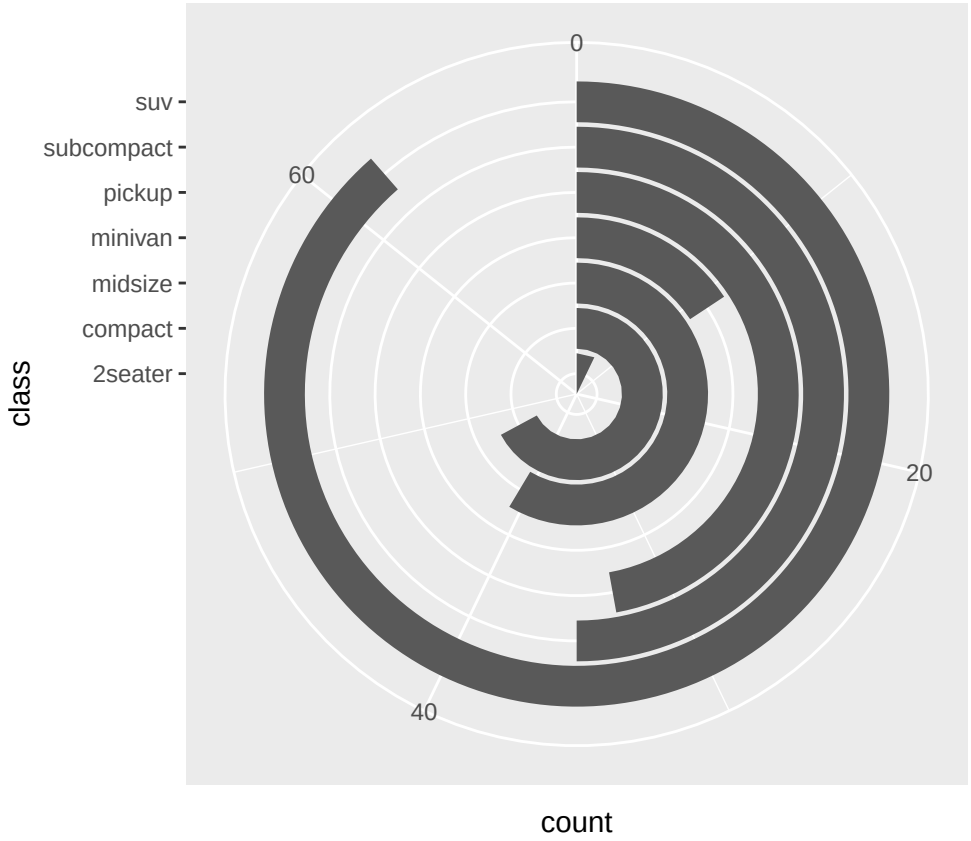
```
ggplot(mpg) +
  geom_bar(aes(x = class))
```




```
# Same but with polar coordinates - interprets x and y as radius and angle:
ggplot(mpg) +
  geom_bar(aes(x = class)) +
  coord_polar()
```



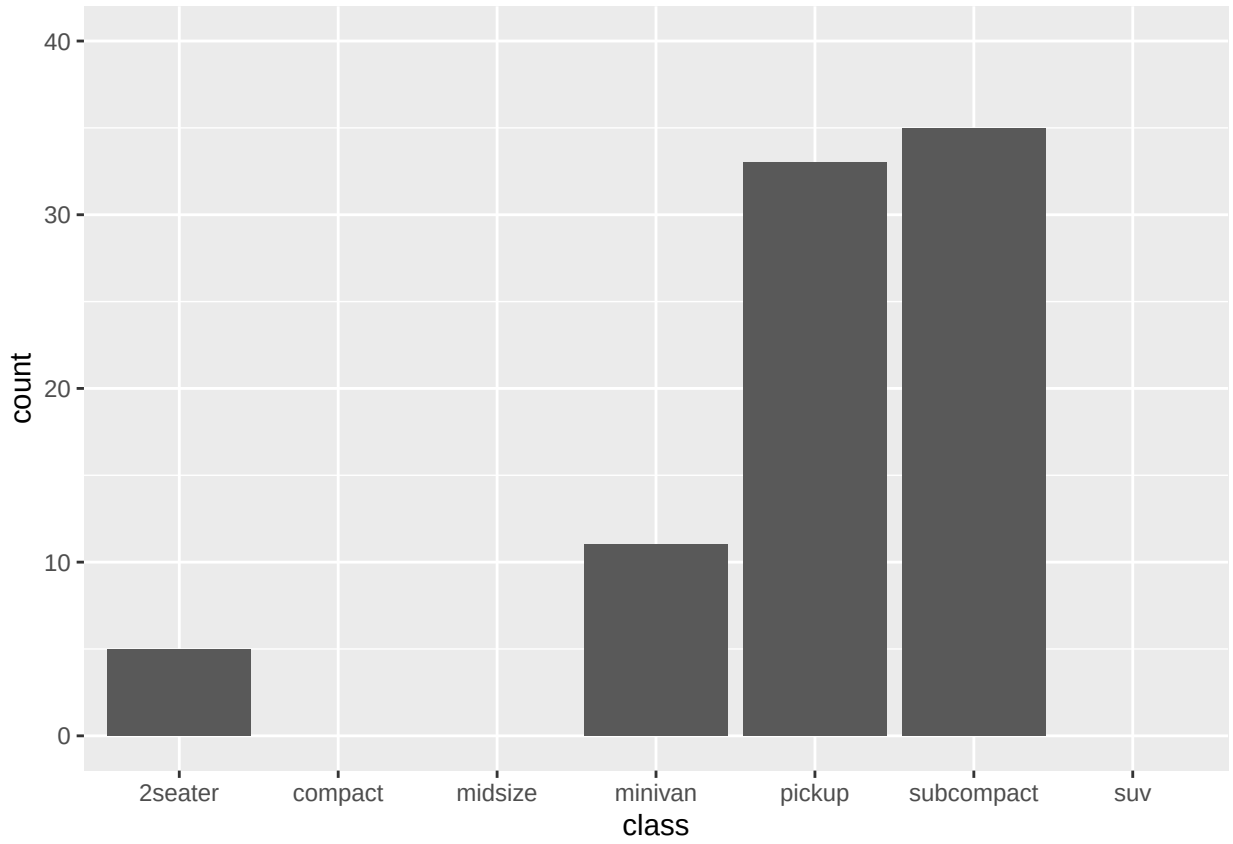

```
ggplot(mpg) +
  geom_bar(aes(x = class)) +
  coord_polar(theta = 'y') +
  expand_limits(y = 70)
```



You can zoom both on the scale...

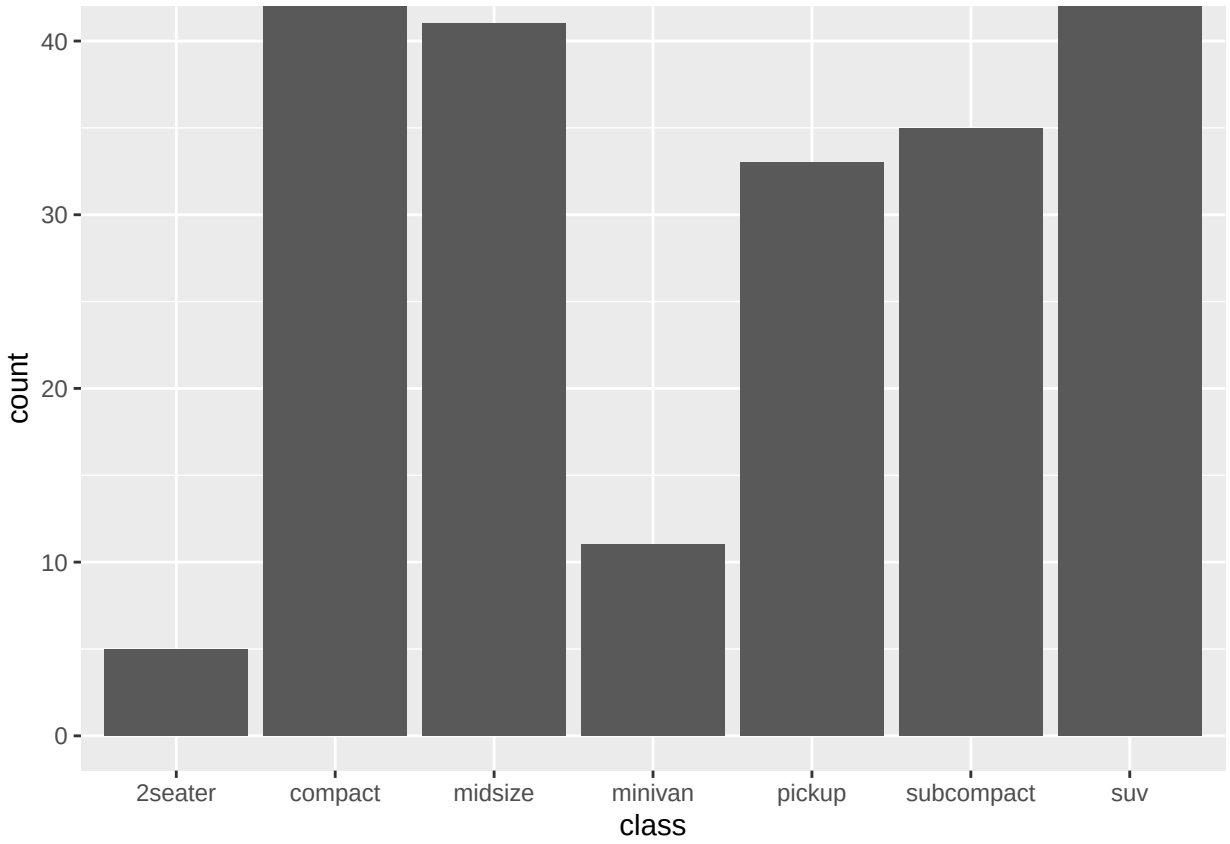
```
ggplot(mpg) +
  geom_bar(aes(x = class)) +
  scale_y_continuous(limits = c(0, 40))  #  removes data outside the limits!
```

```
## Warning: Removed 3 rows containing missing values (geom_bar).
```



and in the coord. You usually want the latter as it avoids changing the plotted data

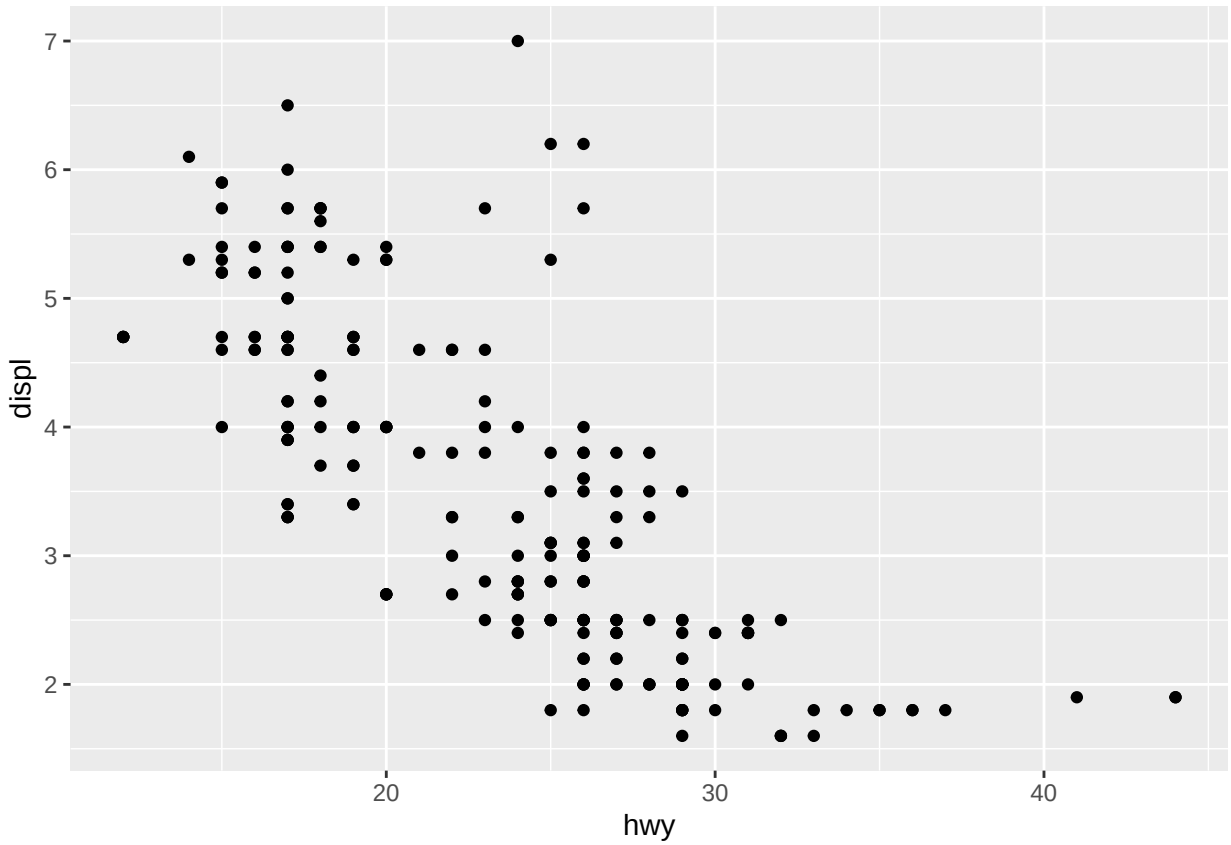
```
ggplot(mpg) +
  geom_bar(aes(x = class)) +
  coord_cartesian(ylim = c(0, 40))
```



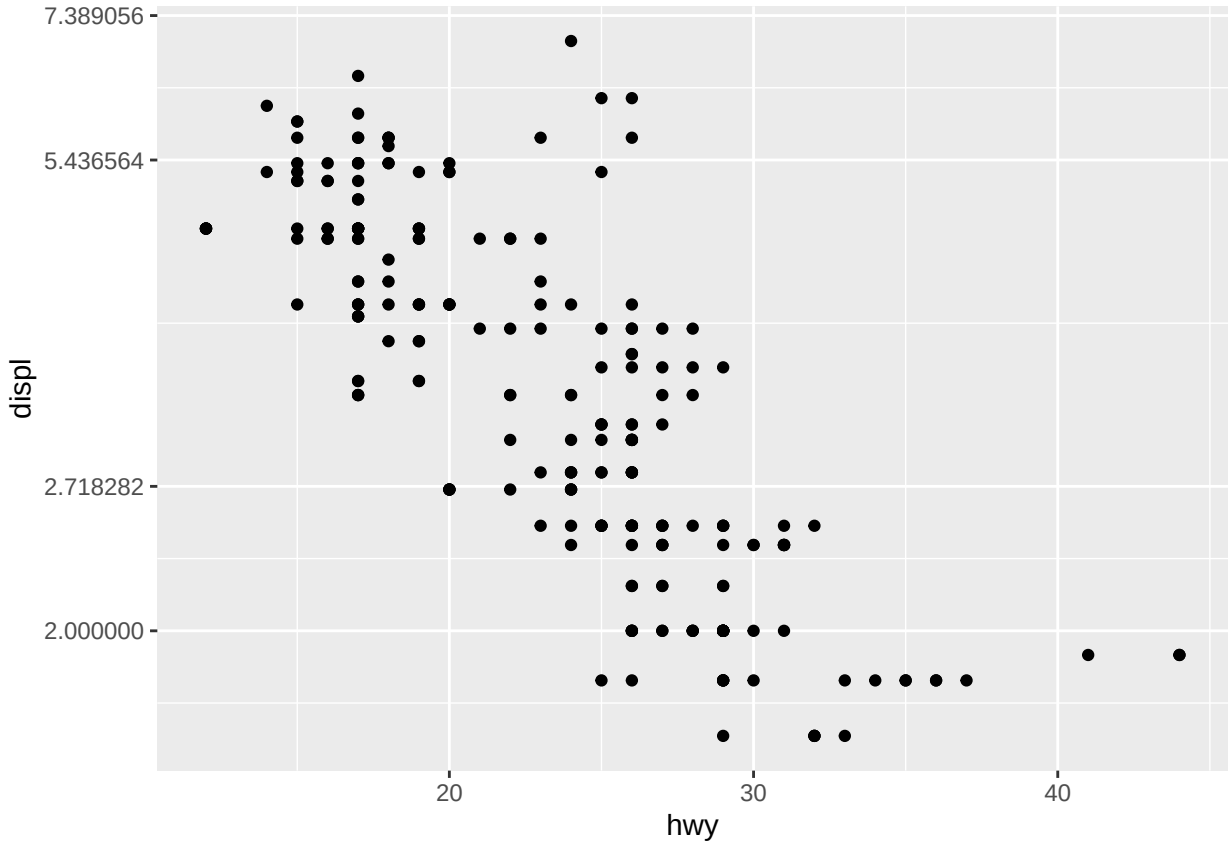
Exercises

In the same way as limits can be set in both the positional scale and the coord, so can transformations, using `coord_trans()`. Modify the code below to apply a log transformation to the y axis; first using `scale_y_continuous()`, and then using `coord_trans()`. Compare the results — how do they differ?

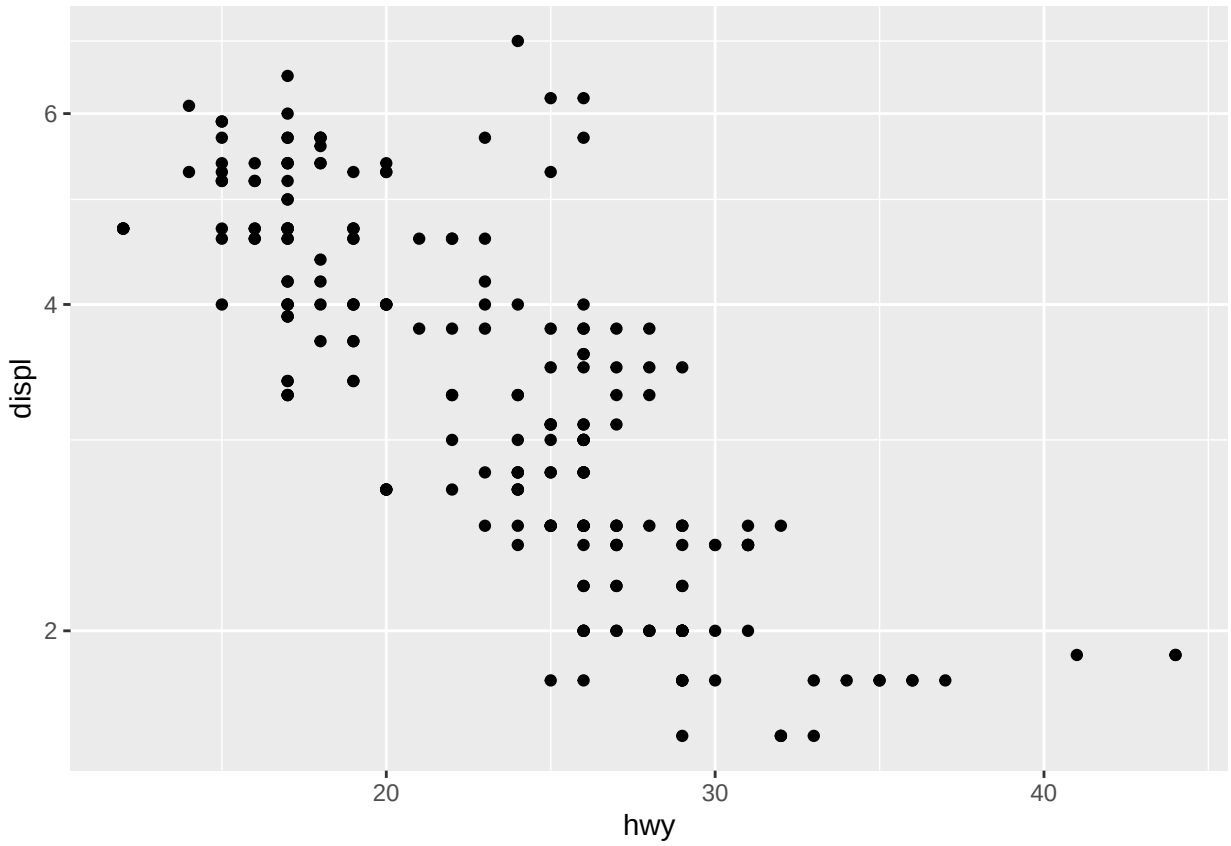
```
ggplot(mpg) +
  geom_point(aes(x = hwy, y = displ))
```




```
ggplot(mpg) +
  geom_point(aes(x = hwy, y = displ)) +
  scale_y_continuous(trans = 'log')  # gives horrible y-axis values
```




```
ggplot(mpg) +
  geom_point(aes(x = hwy, y = displ)) +
  coord_trans(y = 'log')  # much better
```



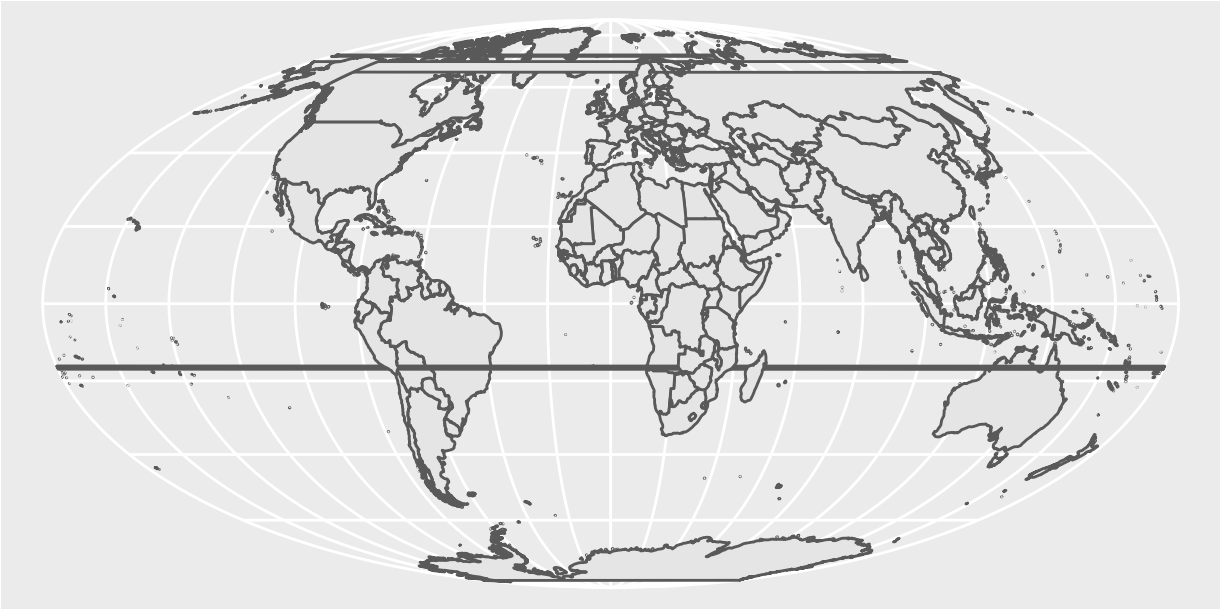
Coordinate systems are particularly important in cartography. While we will not spend a lot of time with it in this workshop, spatial plotting is well supported in ggplot2 with `geom_sf()` and `coord_sf()` (which interfaces with the sf package). The code below produces a world map. Try changing the `crs` argument in `coord_sf()` to be `'+proj=robin'` (This means using the Robinson projection).

```
# Get the borders of all countries
world <- sf::st_as_sf(maps::map('world', plot = FALSE, fill = TRUE))
world <- sf::st_wrap_dateline(world,
                              options = c("WRAPDATELINE=YES", "DATELINEOFFSET=180"),
                              quiet = TRUE)
```

```
## Warning in CPL_wrap_dateline(st_geometry(x), options, quiet): GDAL Error 6: GEOS
## support not enabled.

## Warning in CPL_wrap_dateline(st_geometry(x), options, quiet): GDAL Error 6: GEOS
## support not enabled.

## Warning in CPL_wrap_dateline(st_geometry(x), options, quiet): GDAL Error 6: GEOS
## support not enabled.

## Warning in CPL_wrap_dateline(st_geometry(x), options, quiet): GDAL Error 6: GEOS
## support not enabled.

## Warning in CPL_wrap_dateline(st_geometry(x), options, quiet): GDAL Error 6: GEOS
## support not enabled.
```

```
## Warning in CPL_wrap_dateline(st_geometry(x), options, quiet): GDAL Error 6: GEOS
## support not enabled.
```

```
# Plot code
ggplot(world) +
  geom_sf() +
  coord_sf(crs = "+proj=moll")
```

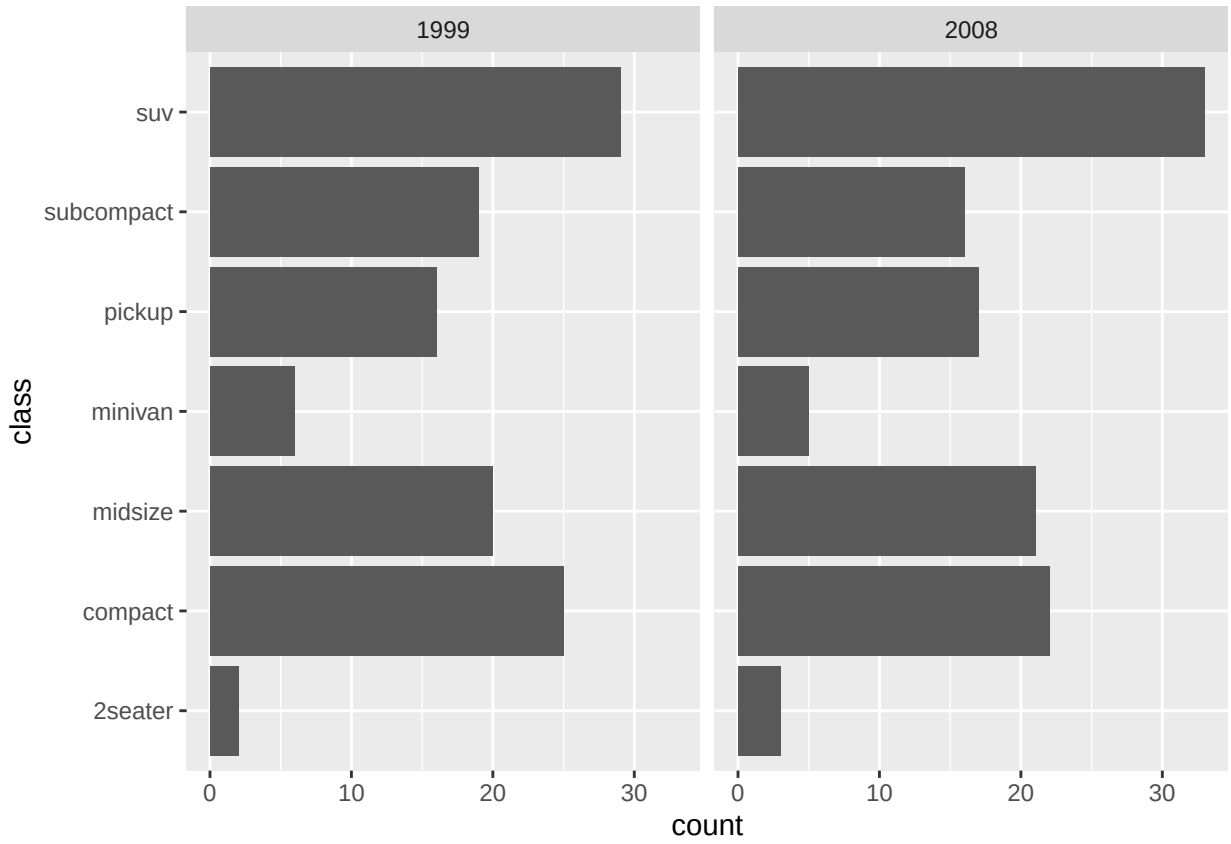


Maps are a huge area in data visualisation and simply too big to cover in this workshop. If you want to explore further I advice you to explore the r-spatial wbsite as well as the website for the sf package

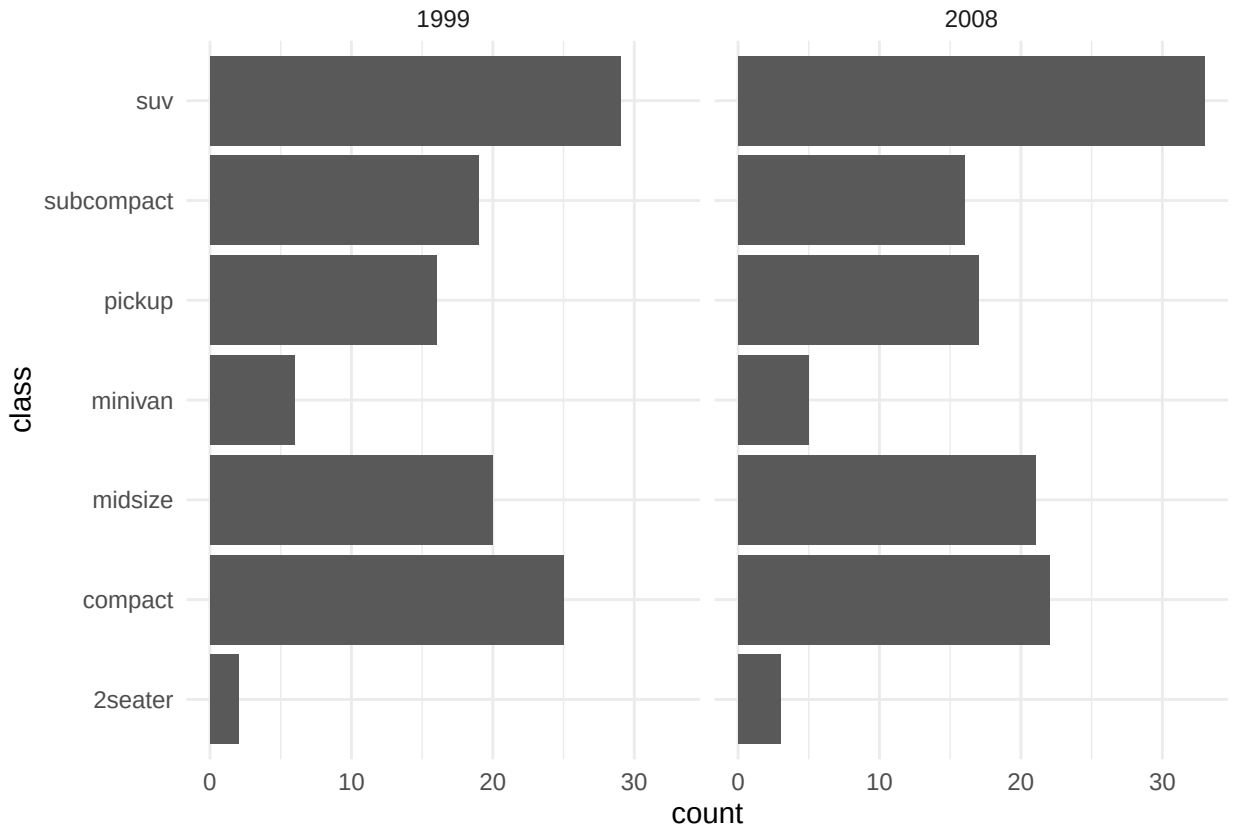
Theme

Theming defines the feel and look of your final visualisation and is something you will normally defer to the final polishing of the plot. It is very easy to change looks with a prebuild theme

```
ggplot(mpg) +
  geom_bar(aes(y = class)) +
  facet_wrap(~year)
```

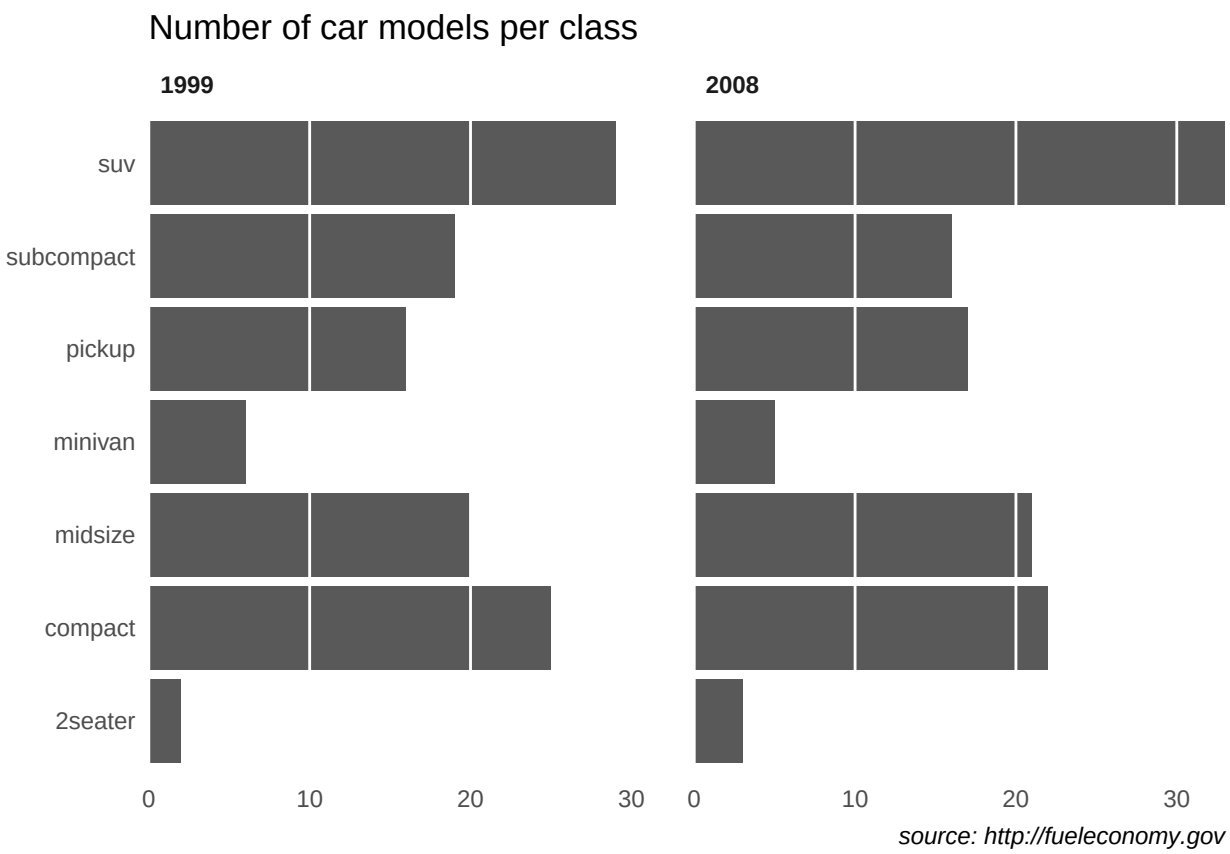



```
ggplot(mpg) +
  geom_bar(aes(y = class)) +
  facet_wrap(~year) +
  theme_minimal()
```



Further adjustments can be done in the end to get exactly the look you want

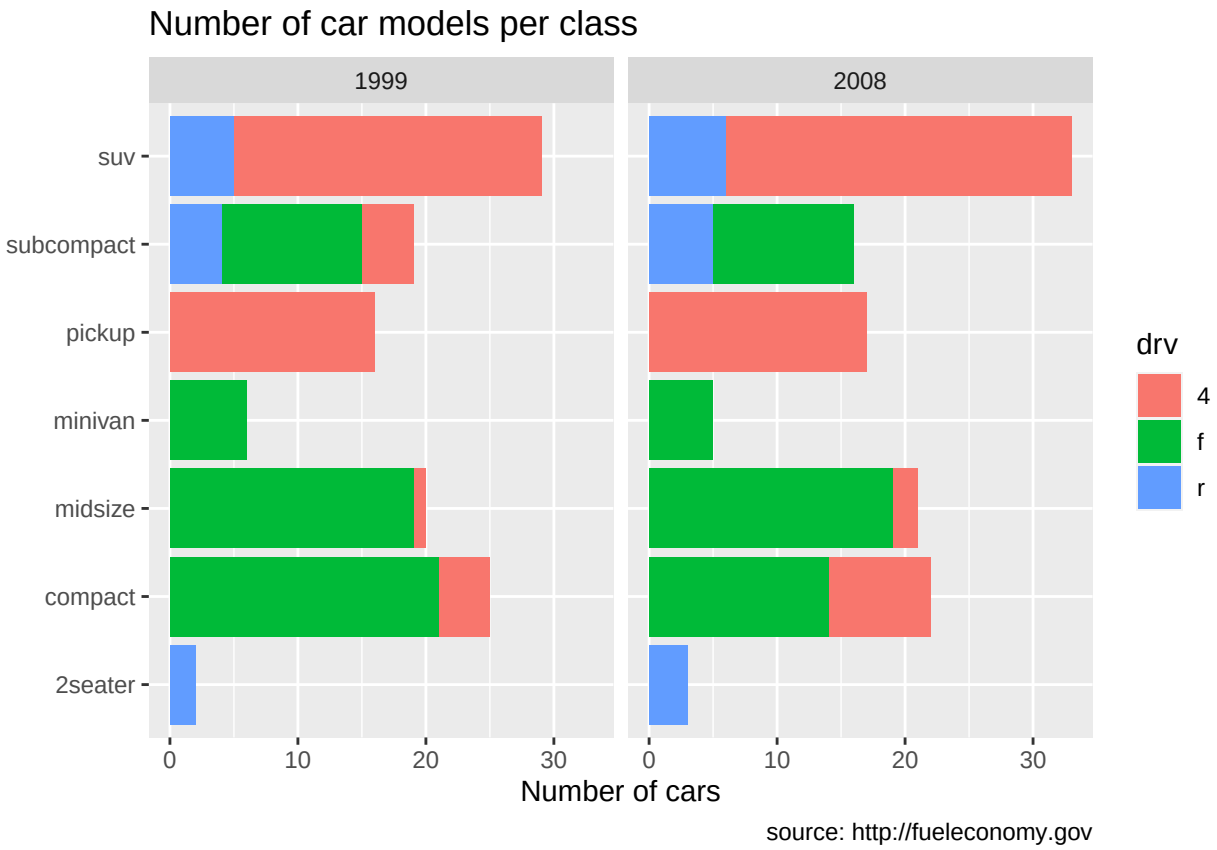
```
ggplot(mpg) +
  geom_bar(aes(y = class)) +
  facet_wrap(~year) +
  labs(title = "Number of car models per class",
       caption = "source: http://fueleconomy.gov",
       x = NULL,
       y = NULL) +
  scale_x_continuous(expand = c(0, NA)) +
  theme_minimal() +
  theme(
#    text = element_text('Avenir Next Condensed'),
    strip.text = element_text(face = 'bold', hjust = 0),
    plot.caption = element_text(face = 'italic'),
    panel.grid.major = element_line('white', size = 0.5),
    panel.grid.minor = element_blank(),
    panel.grid.major.y = element_blank(),
    panel.ontop = TRUE
  )
```



Exercises

Themes can be overwhelming, especially as you often try to optimise for beauty while you learn. To remove the last part of the equation, the exercise is to take the plot given below and make it as hideous as possible using the theme function. Go absolutely crazy, but take note of the effect as you change different settings.

```
ggplot(mpg) +
  geom_bar(aes(y = class, fill = drv)) +
  facet_wrap(~year) +
  labs(title = "Number of car models per class",
       caption = "source: http://fueleconomy.gov",
       x = 'Number of cars',
       y = NULL)
```


Number of car models per class
1999
2008
suv
subcompact
pickup
minivan
midsize
compact
2seater
0
10
20
30
Number of cars
drv
4
f
r
source: http://fueleconomy.gov