Notes on Ch 9: Layers

N. Lim

2025-07-02

Prerequisites

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

- According to John Tukey, the greatest value of a picture is when it forces us to notice what we never expected to see.

Inspecting `mpg`:

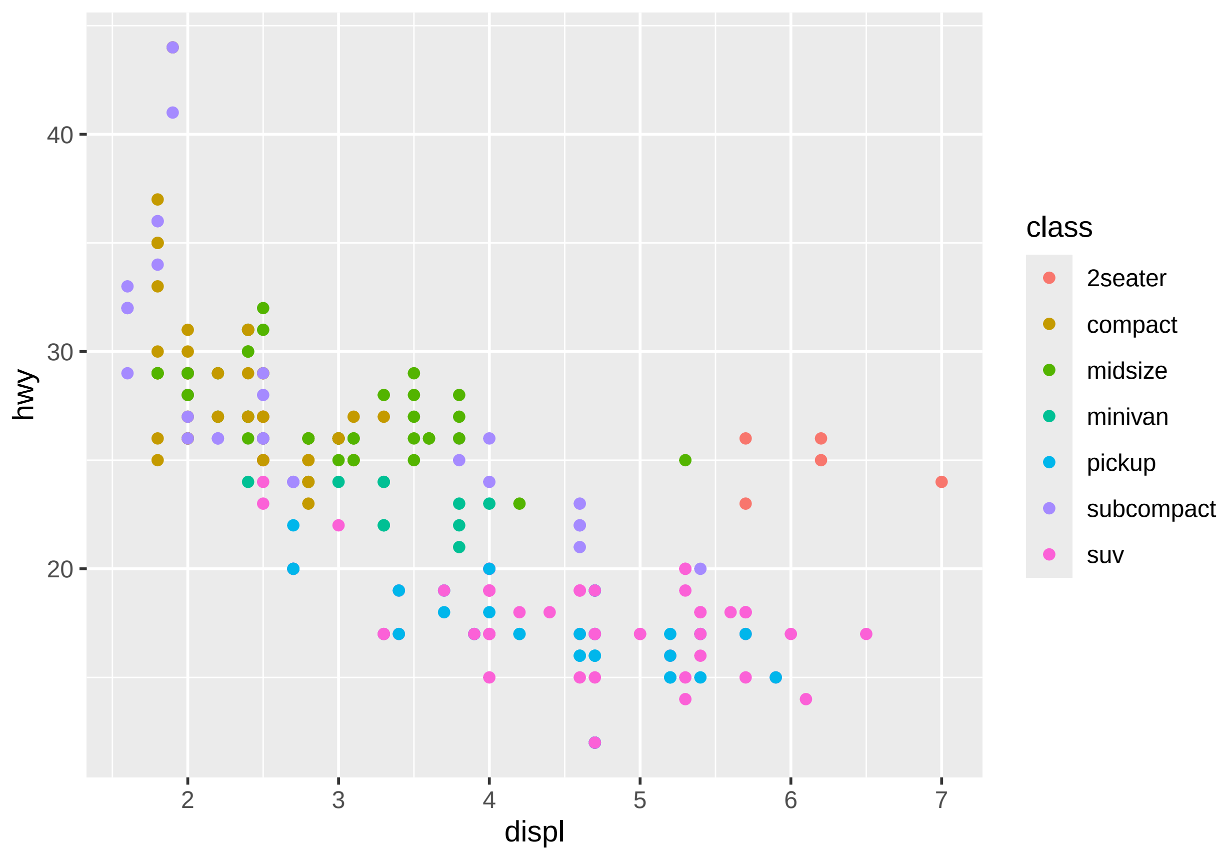
```r
mpg
```

```
## # 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~
## # i 224 more rows
```

where: 1. `displ` is the car's engine size (engine displacement) in liters. 2. `hwy` is the car's fuel efficiency on the highway, in mpg. 3. `class` is the type of car – a categorical value.

Visualizing the relationship between `displ` and `hwy` for various classes of cars:

```r
mpg |>
  ggplot(aes(x = displ, y = hwy, color = class)) +
  geom_point()
```

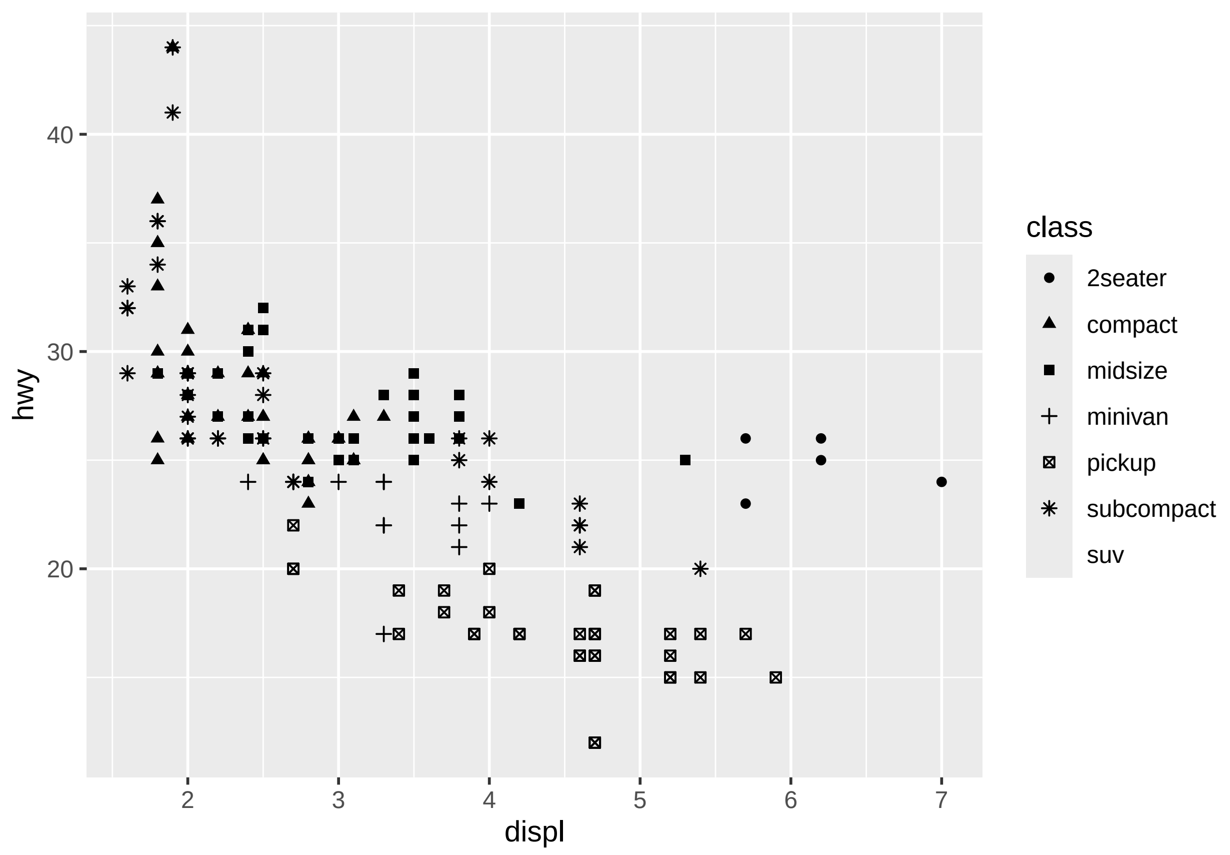


Another way to visualize the relationship between `displ` and `hwy` by assigning different shapes to each class:

```
mpg |>
  ggplot(aes(x = displ, y = hwy, shape = class)) +
  geom_point()
```

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

```
## Warning: Removed 62 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

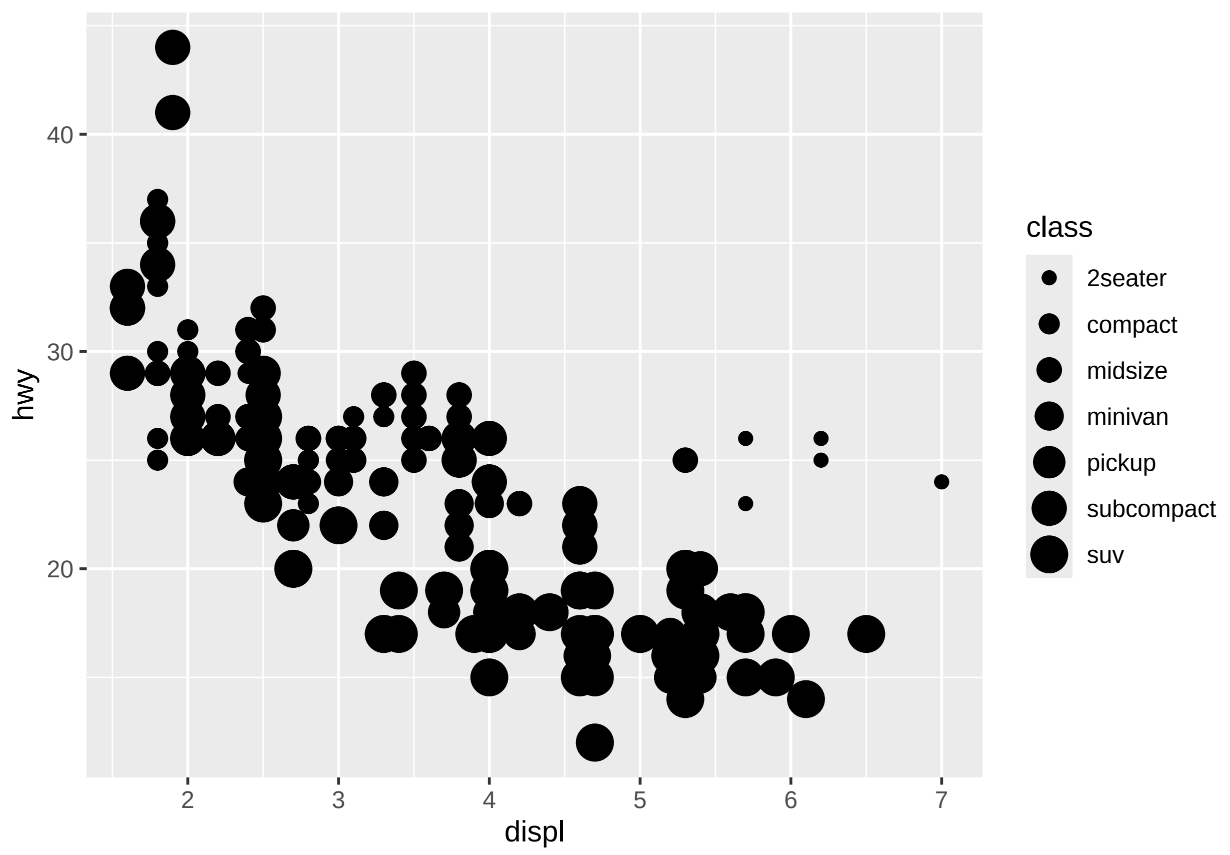


Note: when the class is mapped to shape we get a waringing because the shape palette can only deal with 6 discrete values.

Mapping `class` to size:

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

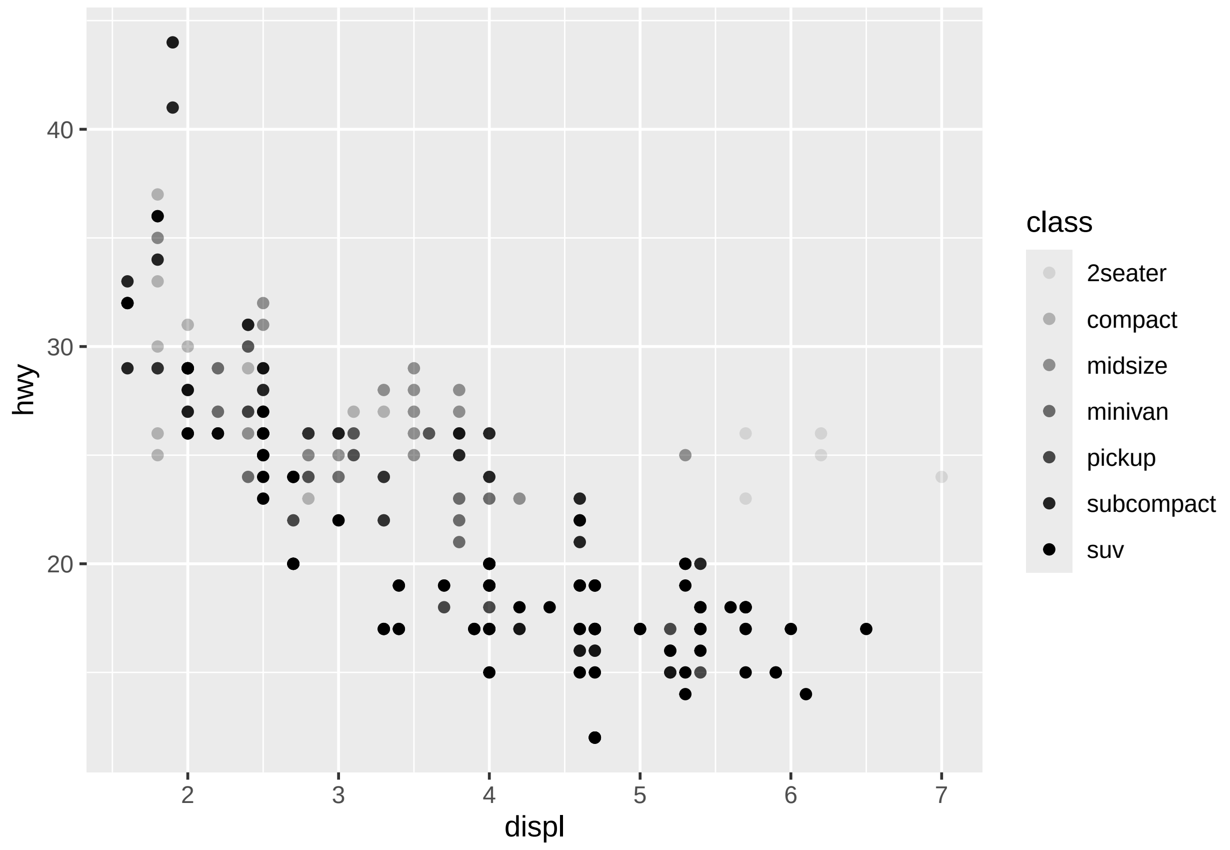
```
## Warning: Using size for a discrete variable is not advised.
```



Mapping `class` to alpha (transparency):

```
mpg |>
  ggplot(aes(x = displ, y = hwy, alpha = class)) +
  geom_point()
```

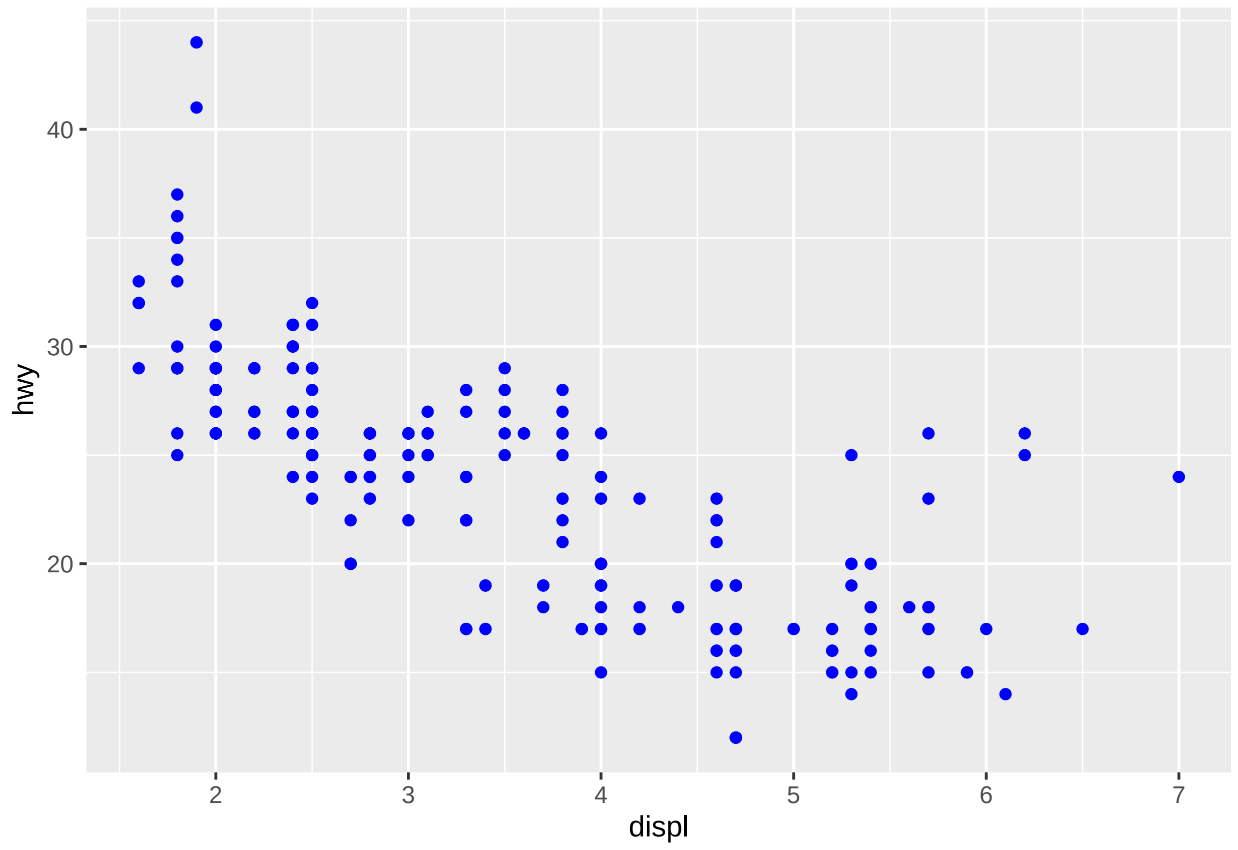
```
## Warning: Using alpha for a discrete variable is not advised.
```



Note: Both of these produced warnings stating that mapping a discrete variable to alpha or size is not advised. Mapping an unordered discrete or categorical variable to an ordered aesthetic like `size` or `alpha` is generally not a good idea because it implies a ranking that does not exist.

Example: Setting the visual properties of `geom` function *outside* of `aes()`:

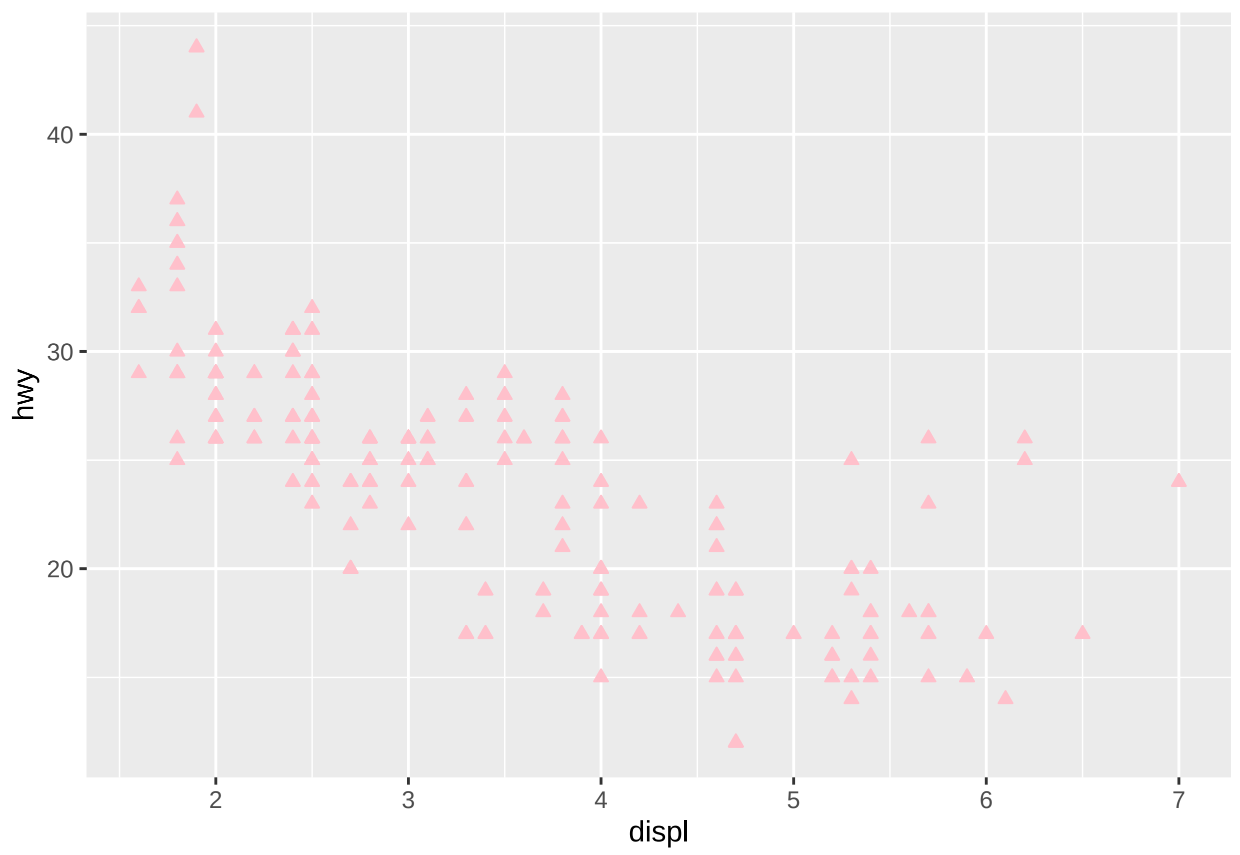
```
mpg |>
  ggplot(aes(x = displ, y = hwy)) +
  geom_point(color = "blue")
```



Exercises

E1. Create a scatterplot of `hwy` vs. `displ` where the points are pink filled in triangles.

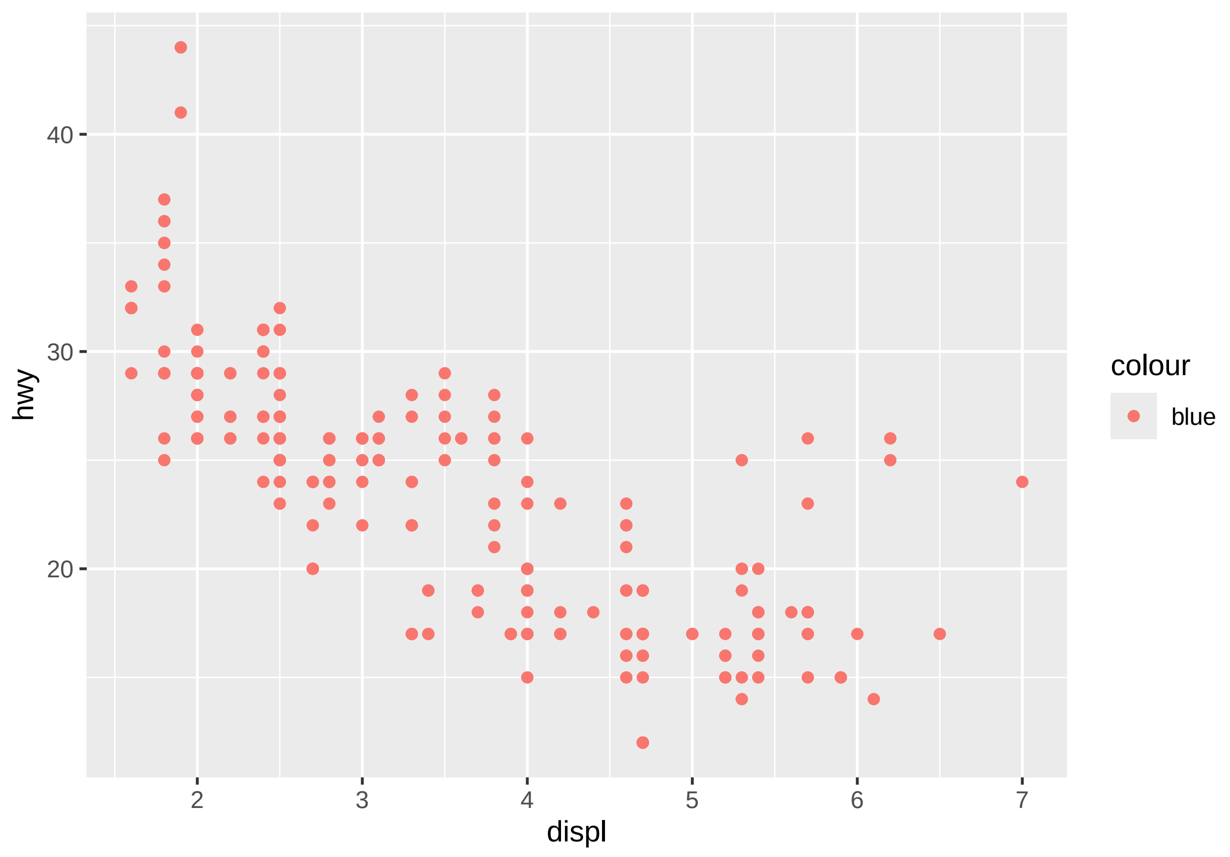
```
mpg |>
  ggplot(aes(x = displ, y = hwy)) +
  geom_point(shape = c(24),
             fill = "pink",
             color = "pink"
  )
```



E2. Why did the following code not result in a plot with blue points?

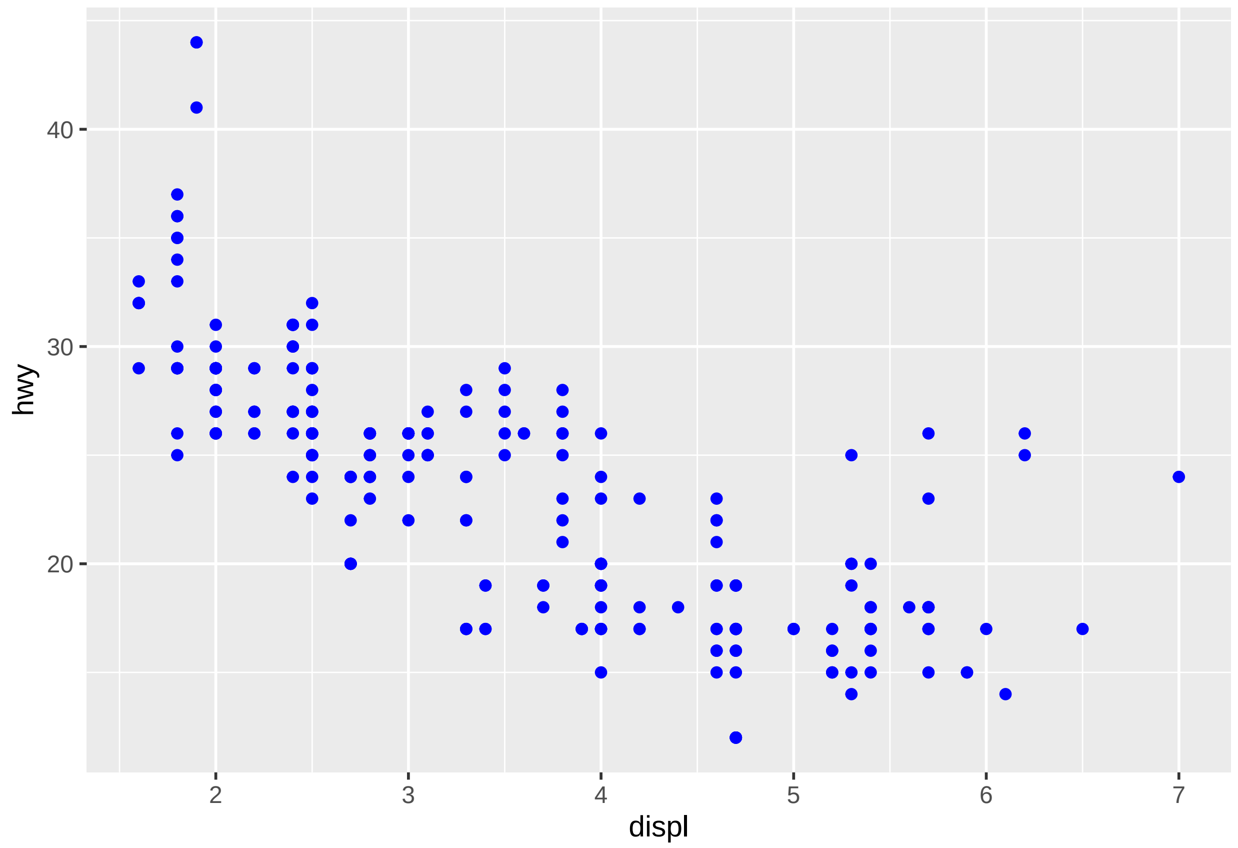
```r
ggplot(mpg) +
geom_point(aes(x = displ, y = hwy, color = "blue"))
```

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



Because in the case above, we are mapping "blue" to x and y and not to the points. To remedy this, we can move the `color` argument outside of `aes()`:

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



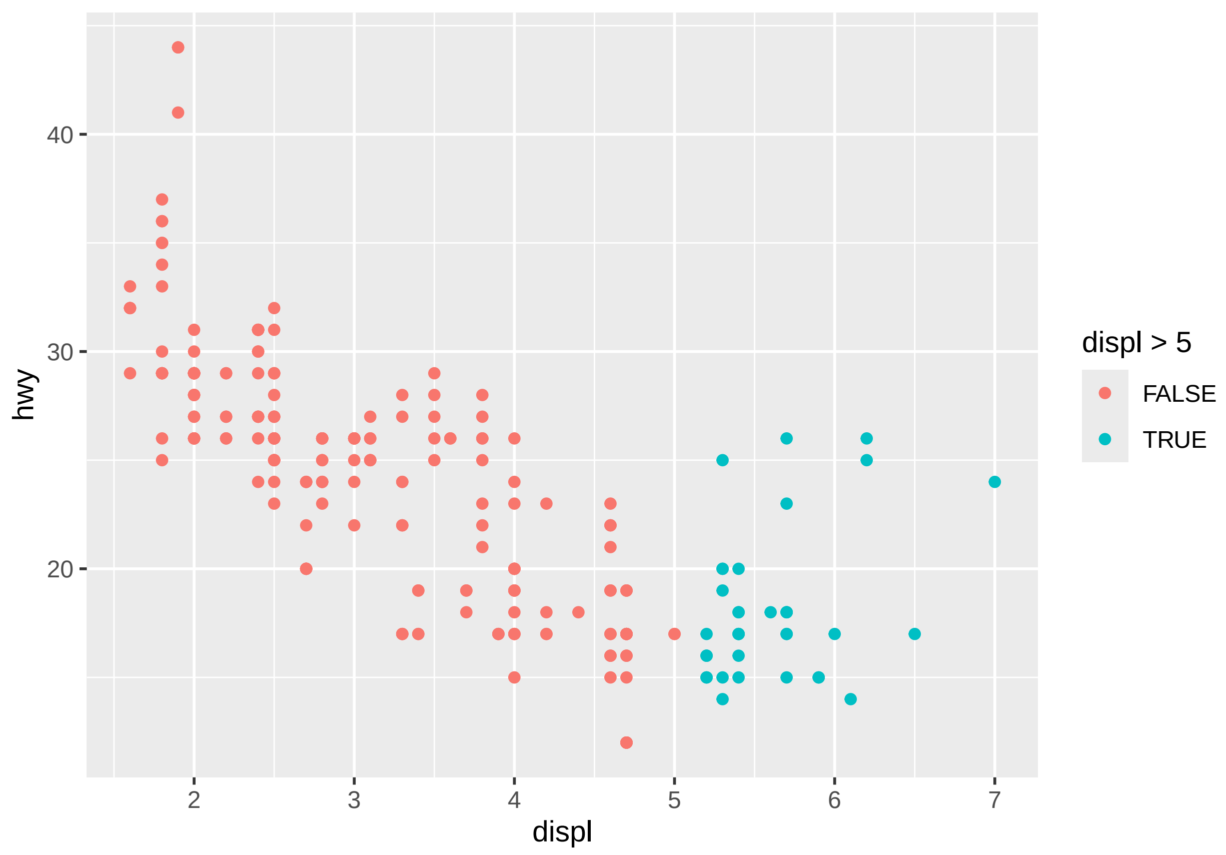
E3. What does the `stroke` aesthetic do? What shapes does it work with?

Ans. Since the shape aesthetic is not a "font", we uset the `stroke` parameter to control the size of the shape, whether it is a square, a cross, or anything in between.

E4. What happens if you map an aesthetic to something other than a variable name, like `aes(color = displ < 5)`? Note, you'll also need to specify `x` and `y`.

Ans:

```
mpg |>
  ggplot(aes(x = displ, y = hwy, color = displ > 5)) +
  geom_point()
```

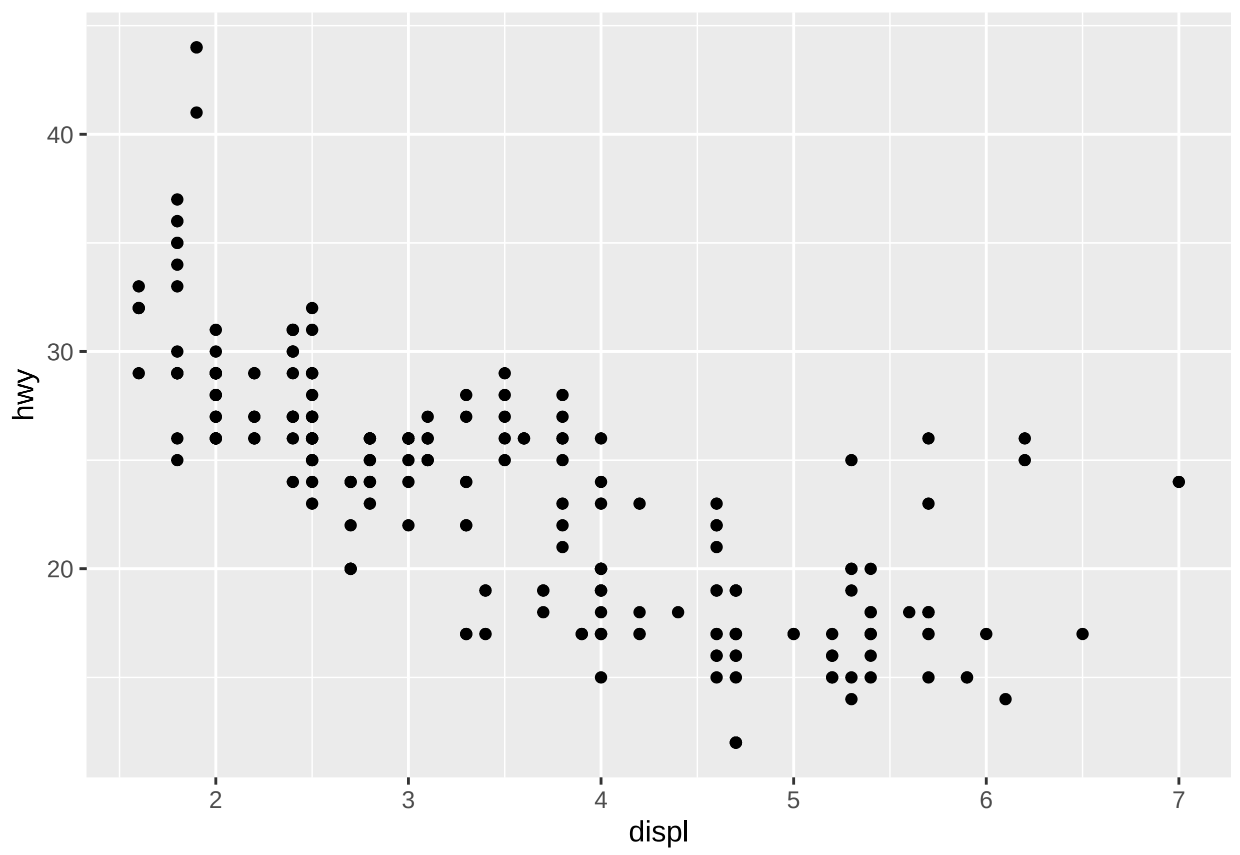


It works, but since we mapped color to a true-false variable, we get two colors in return: one color for when the condition is true, another color for when it is false.

Geometric objects

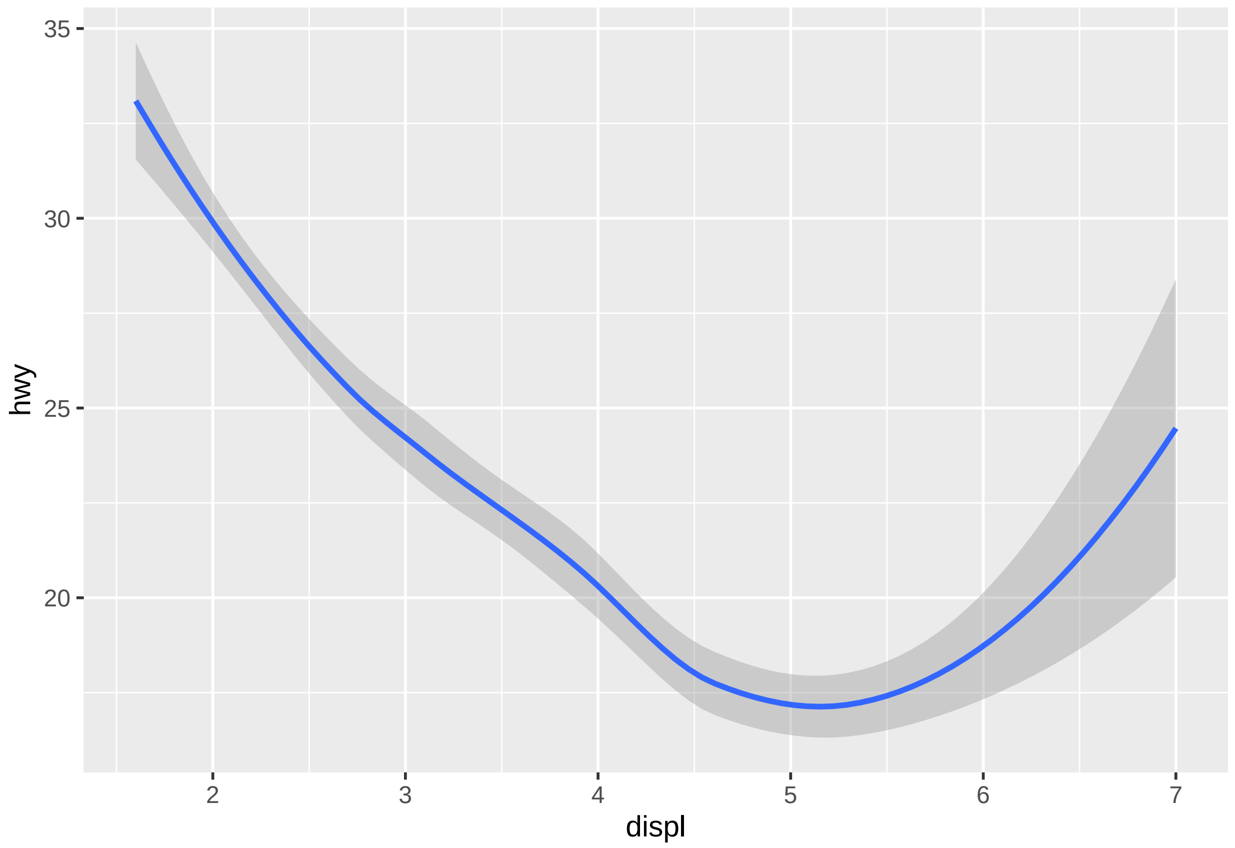
Consider these plots:

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



```r
# Second
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth()
```

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

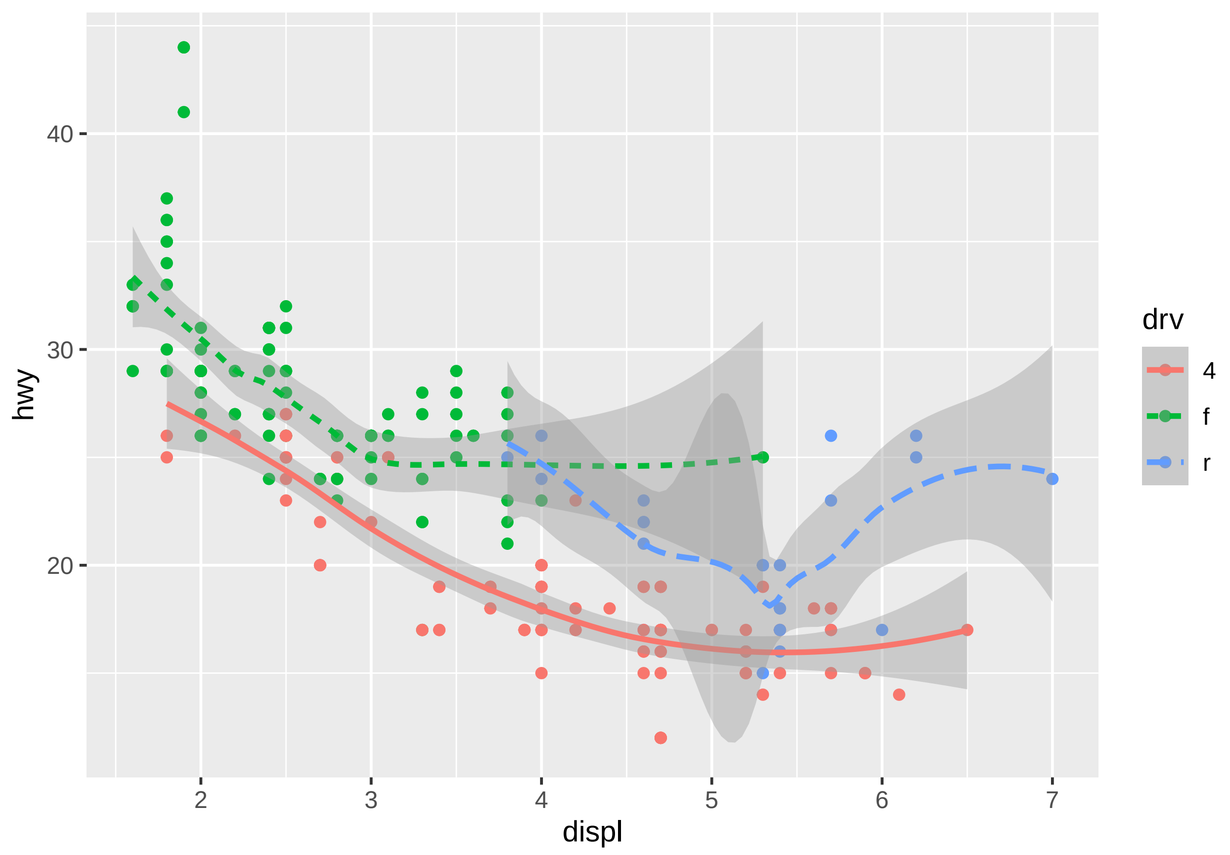


- The only change made was the geometric mapping. The `geom` layer takes a mapping argument, either from the `ggplot()` layer (global definition), or from the `aes()` call of the `geom` (local definition).
- If an aspect of the global mapping is not applicable, the `geom` layer will ignore it.

Example: Plot with 3 layers:

```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(aes(linetype = drv))
```

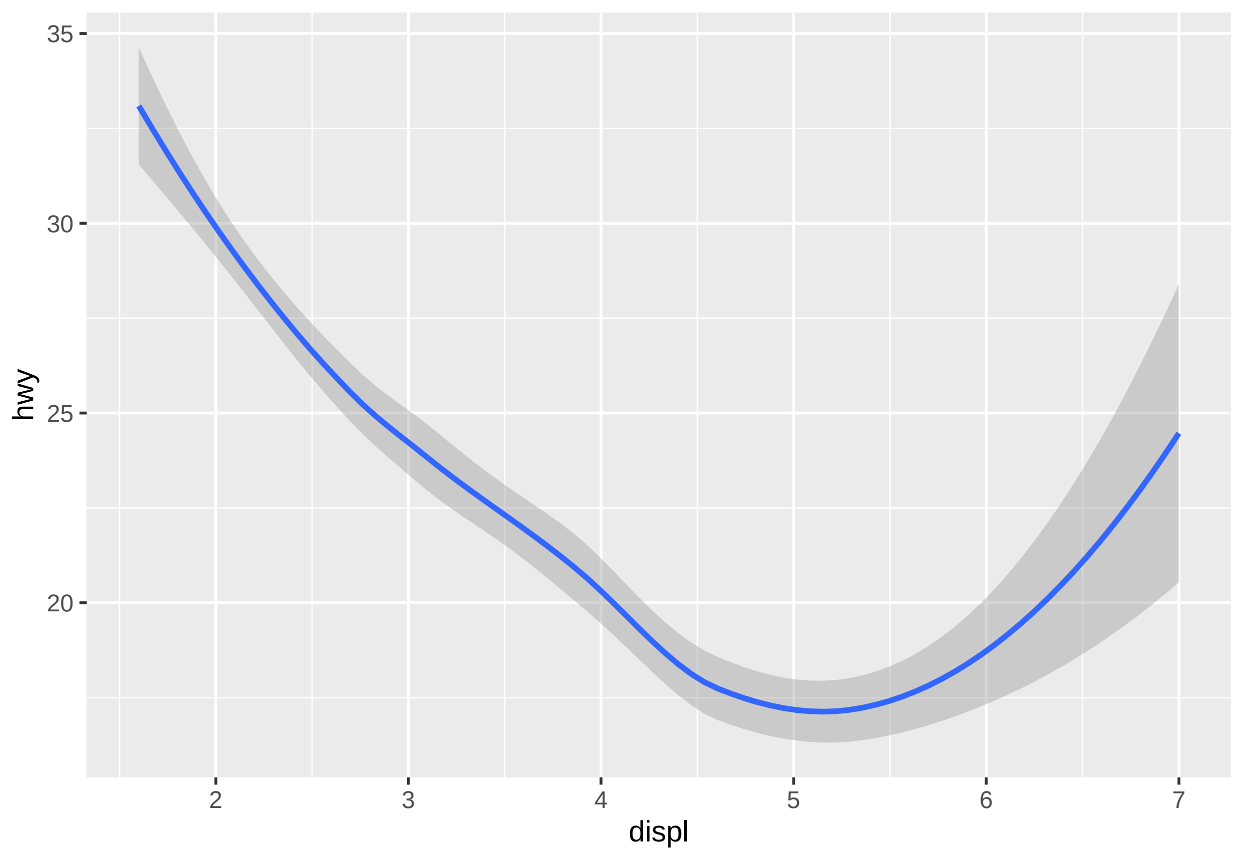
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



More examples: Example1:

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

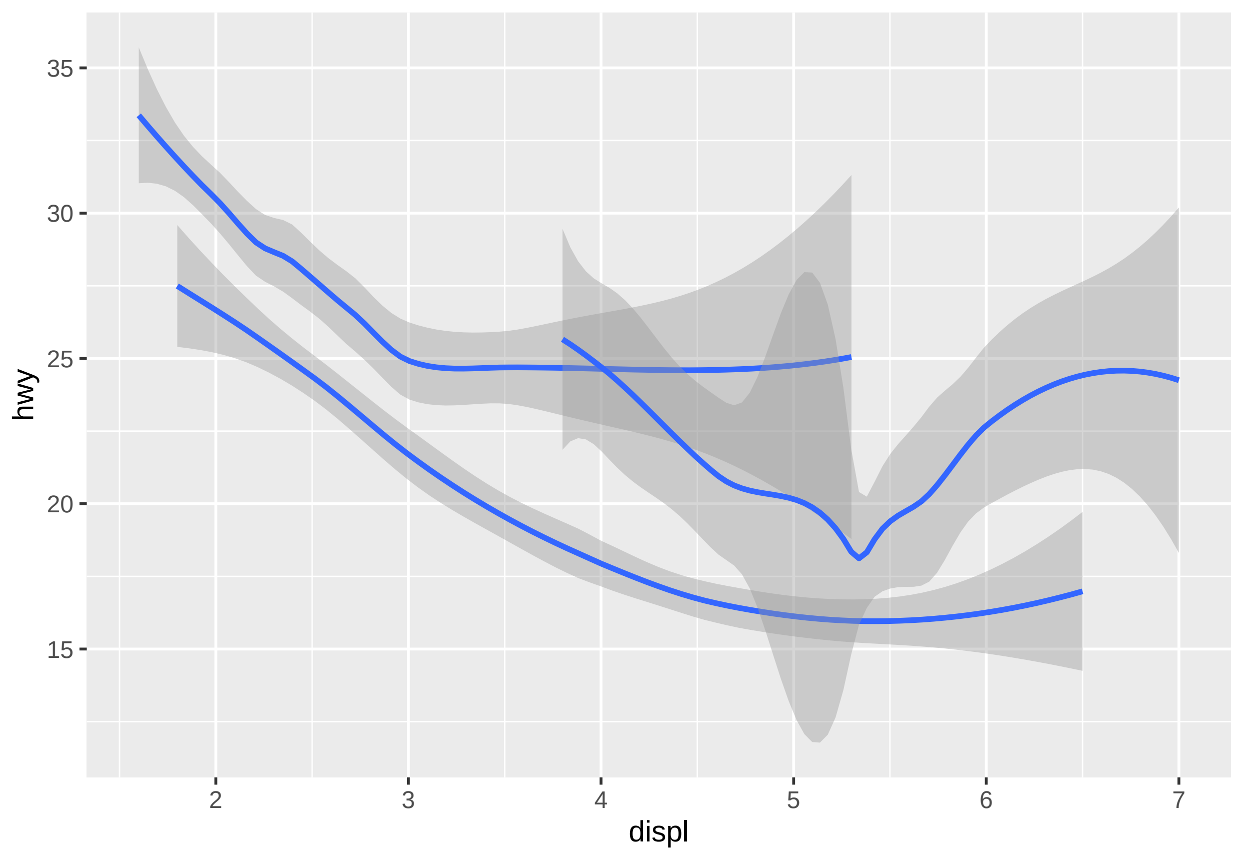
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Example2:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth(aes(group = drv))
```

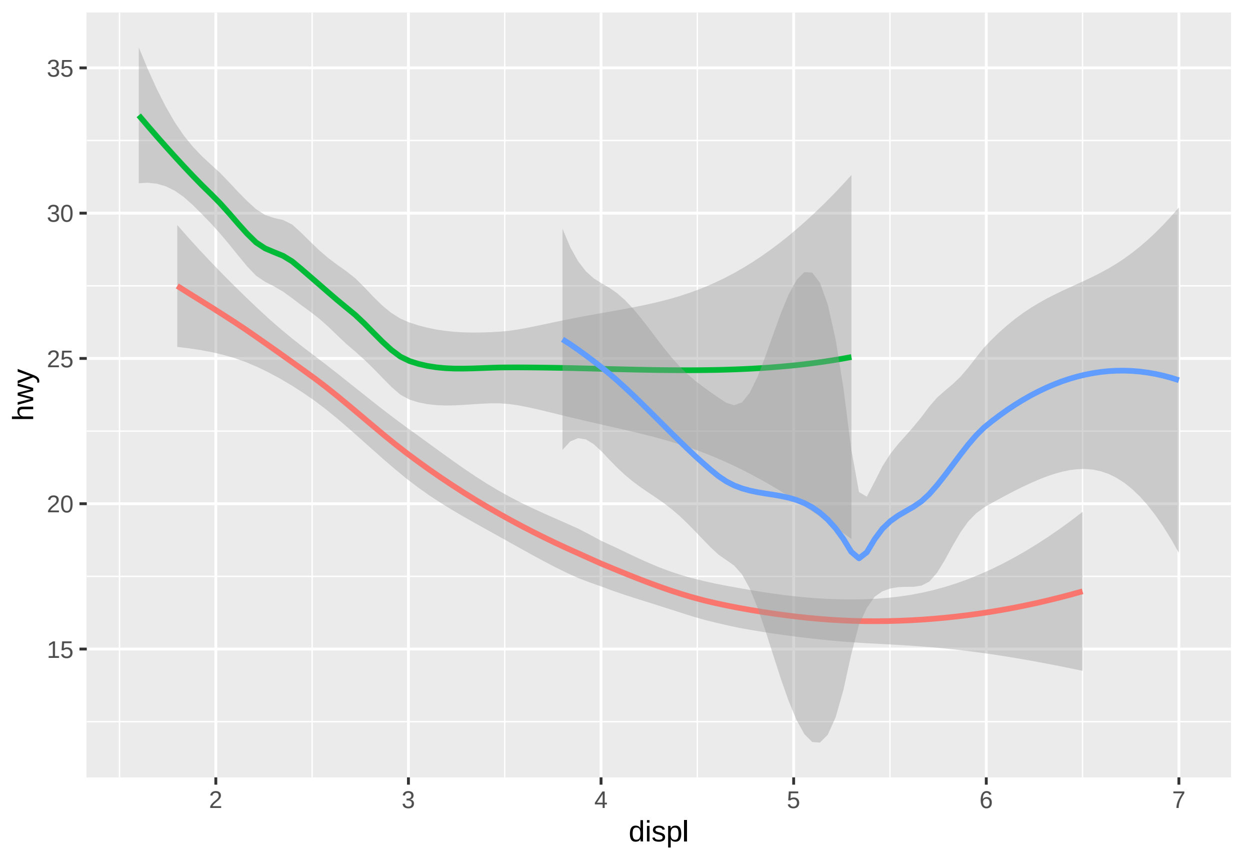
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Example 3:

```r
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth(aes(color = drv), show.legend = FALSE)
```

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

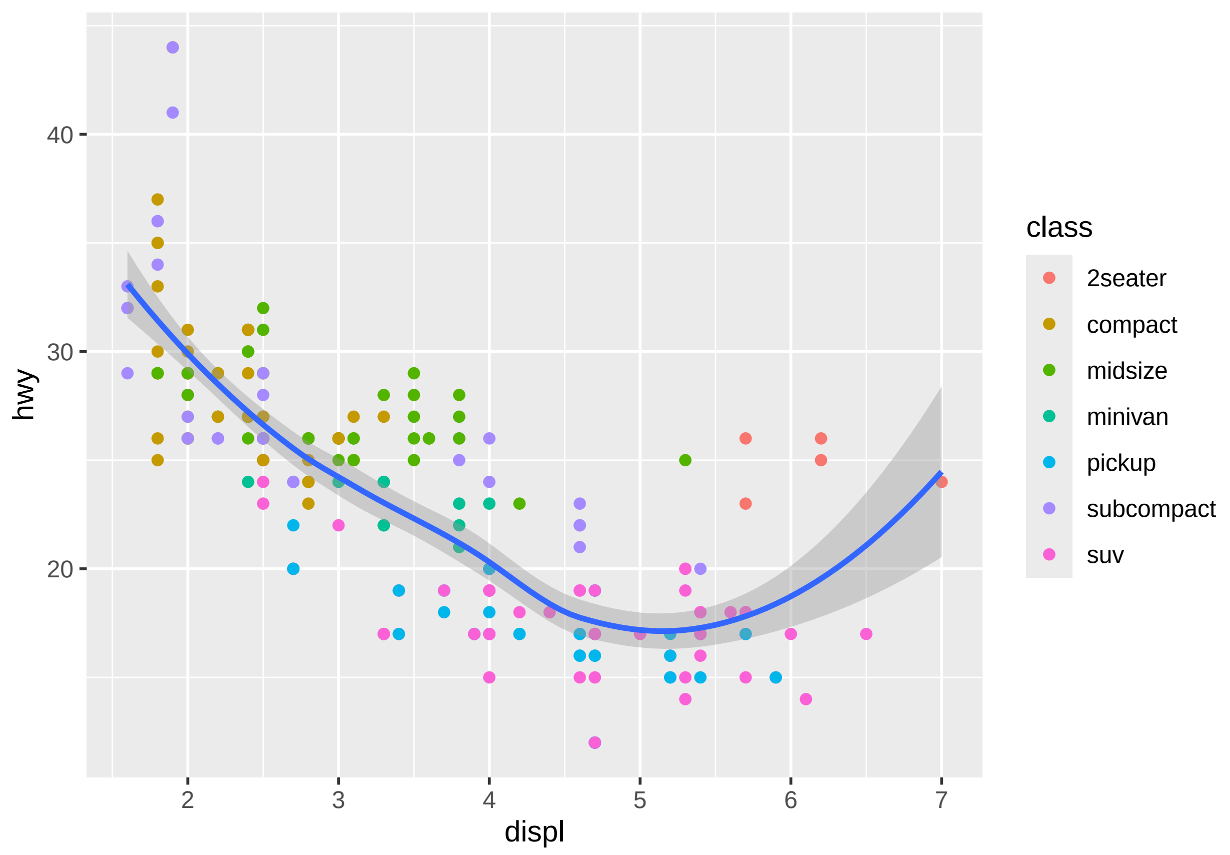


- Local mappings will override global mapping for that particular layer only. (This is similar to how a local styling will override the style defined in the .css in a modern html page.)

Example:

```r
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth()
```

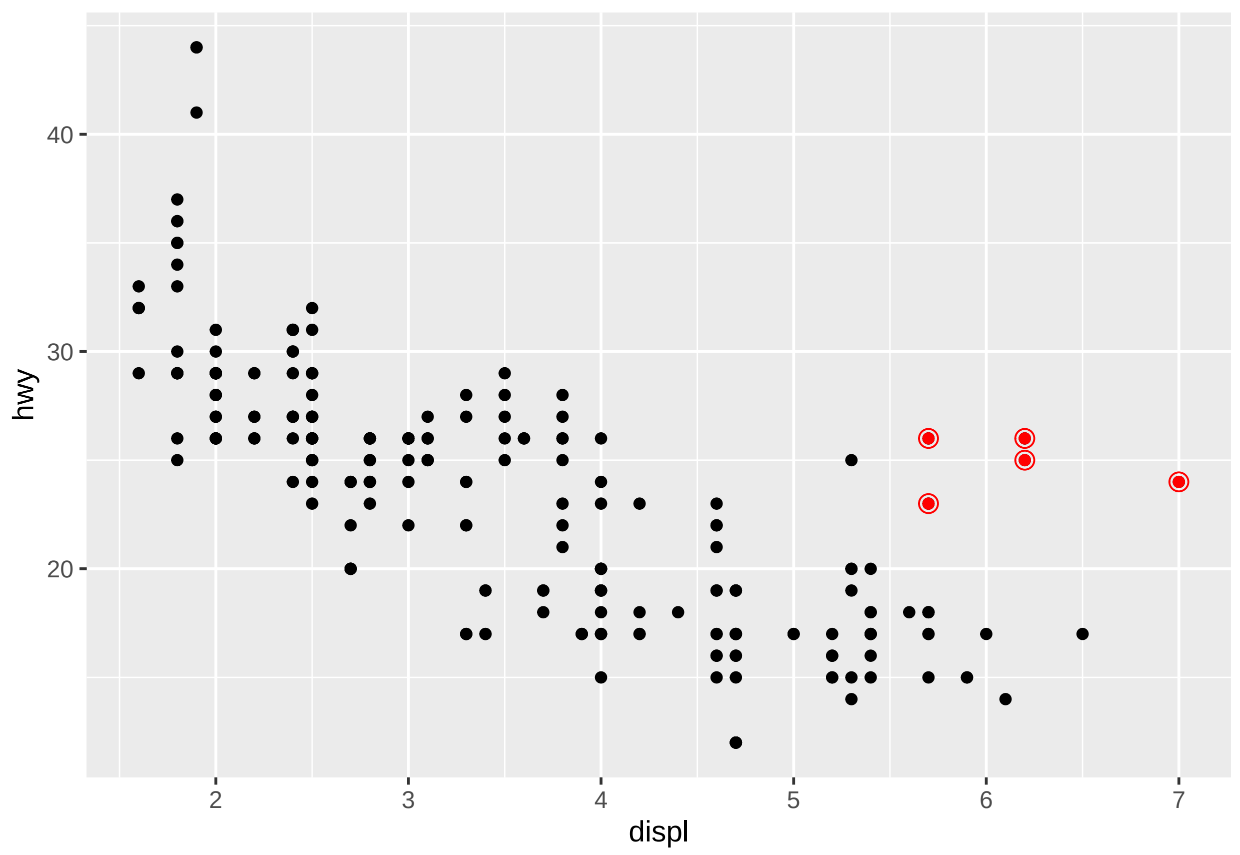
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



- The same idea can be used to specify different data for each layer. The local data argument will override the global data argument in `ggplot()`.

Example:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  geom_point(
    data = mpg |> filter(class == "2seater"),
    color = "red"
  ) +
  geom_point(
    data = mpg |> filter(class == "2seater"),
    shape = "circle open", size = 3, color = "red"
  )
```



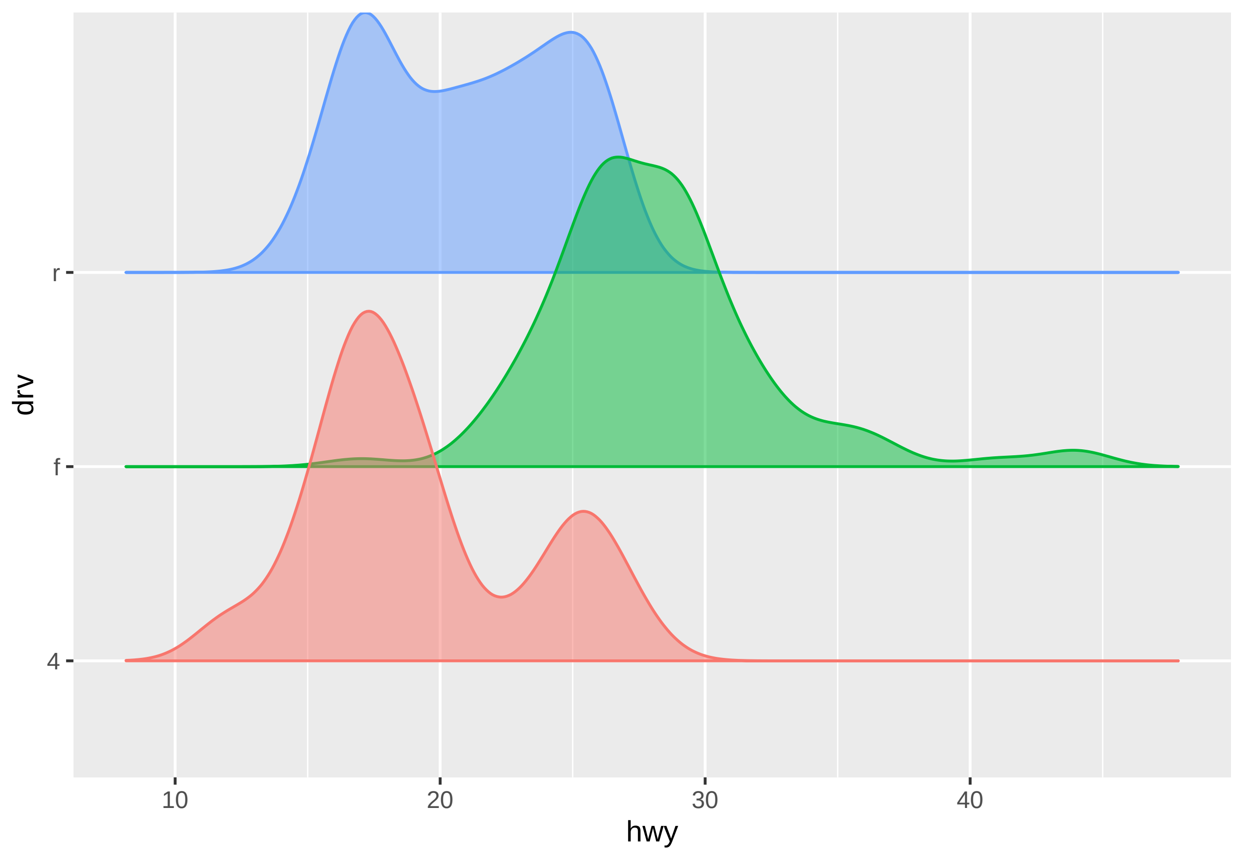
- `ggplot()` covers more than 40 geoms. See the help pages or the cheatsheets.

ExampleL Ridge plot

```
library(ggridges)

ggplot(mpg, aes(x = hwy, y = drv, fill = drv, color = drv)) +
  geom_density_ridges2(alpha = 0.5, show.legend = FALSE)
```

```
## Picking joint bandwidth of 1.28
```



Exercises

E1. What geom would you use to draw a line chart? A boxplot? A histogram? An area chart?

Ans. `geom_line()`, `geom_boxplot()`, `geom_histogram()`, `geom_area()`

E2. What does `show.legend = FALSE` do here?

```
ggplot(mpg, aes(x = displ, y = hwy)) +
geom_smooth(aes(color = drv), show.legend = FALSE)
```

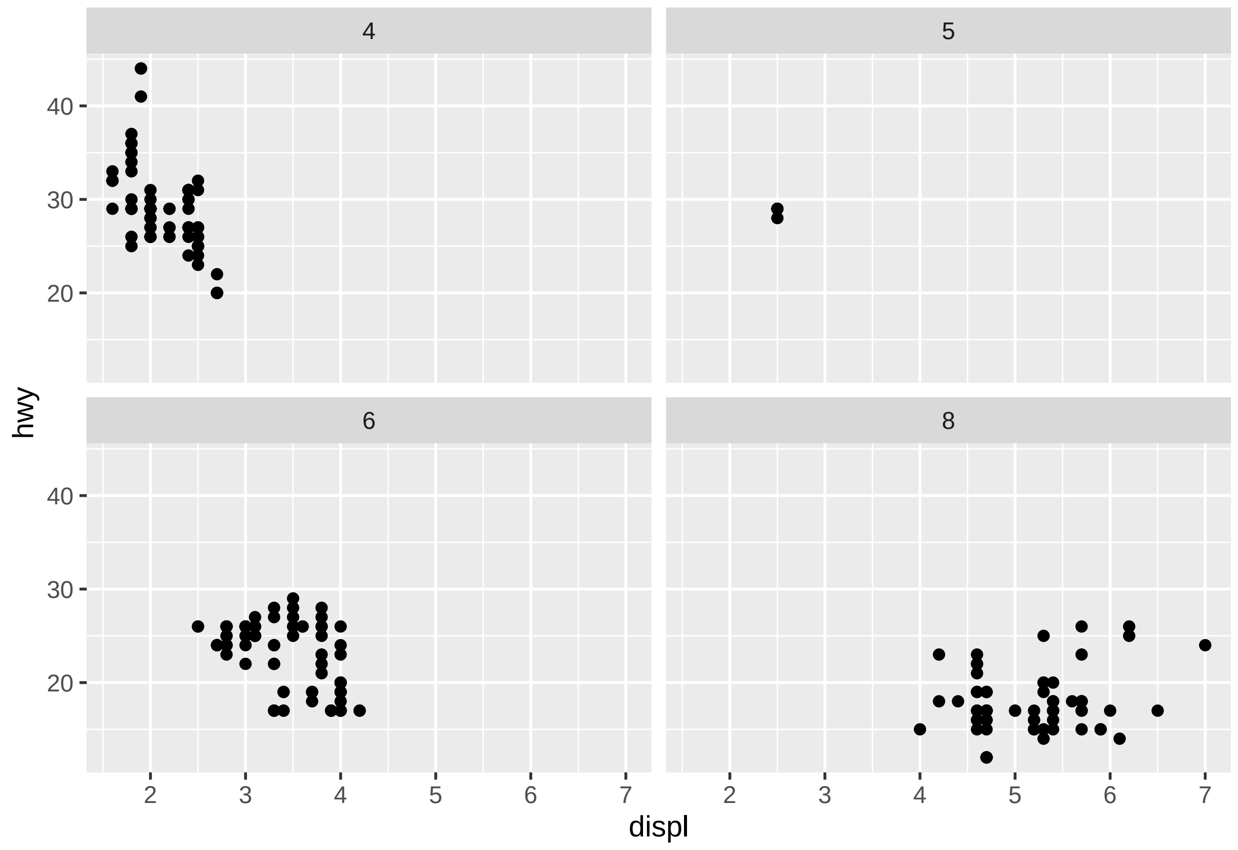
What happens if you remove it? Why do you think we used it earlier?

It prevents ggplot from showing the legend. If you remove this line, then the legend will be displayed. We suppressed the legend because we are making side-by-side comparisons with other plots that do not have legends. This way, the width of all the plots remain consistent.

Facets

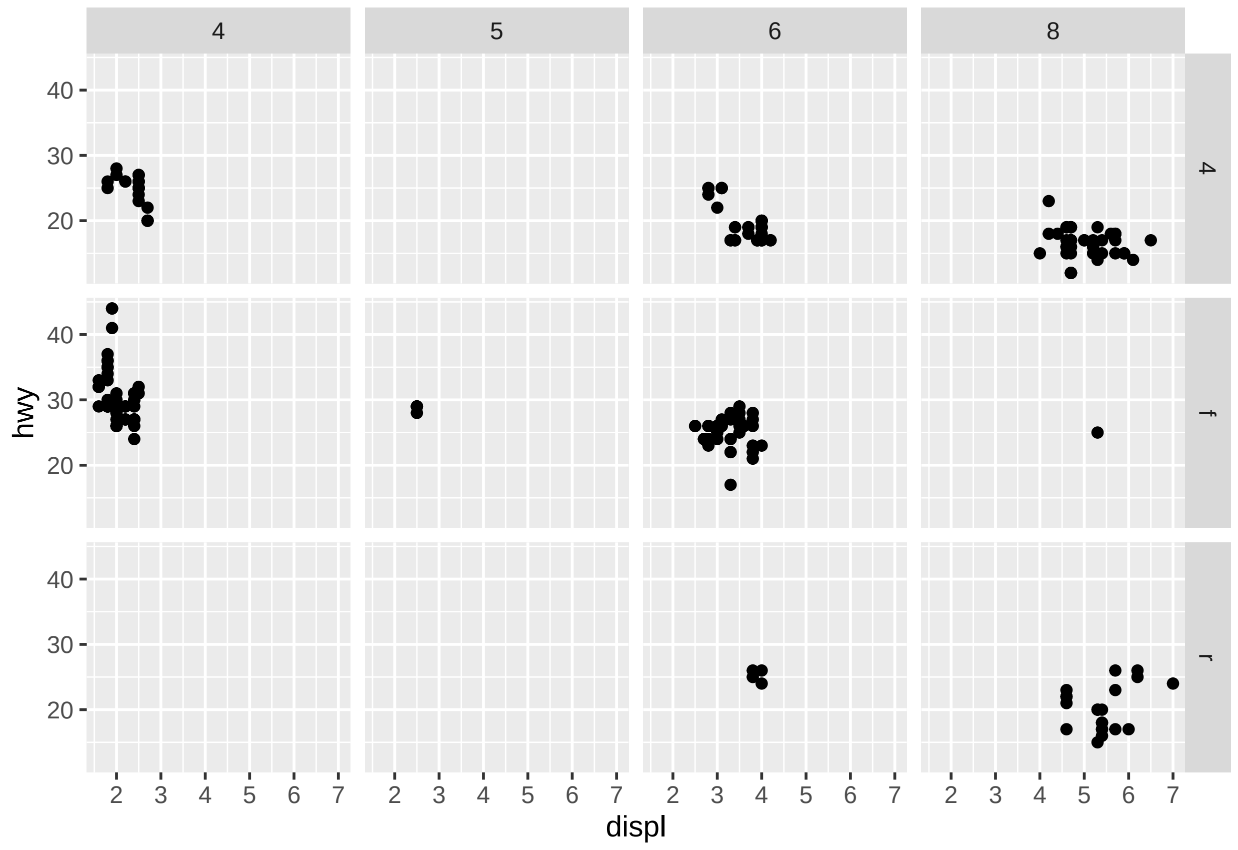
Splitting the plot into subplots that each display one subset of data based on categorical variable:

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



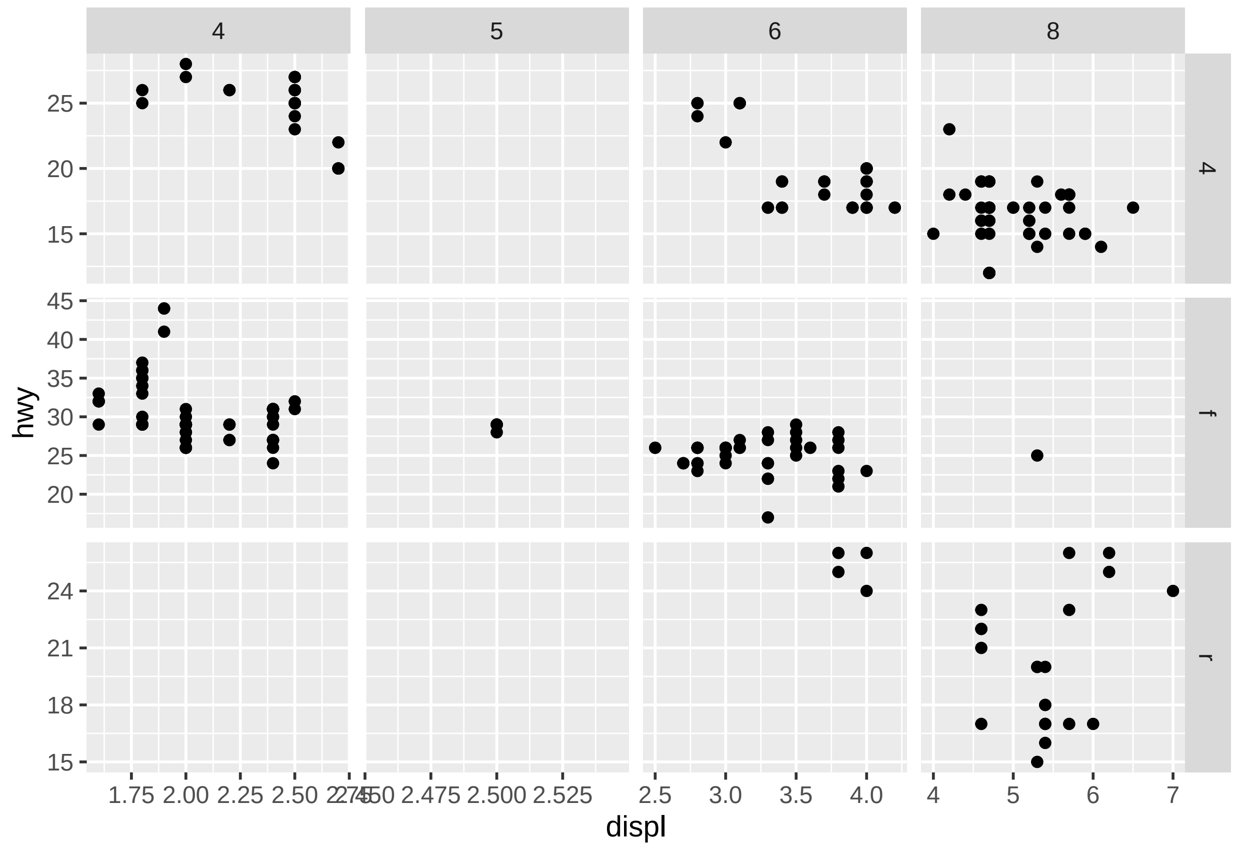
Faceting the plot with a combination of two variables using `facet_grid()`:

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



Faceting with the scales "unlinked":

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(drv ~ cyl, scales = "free")
```

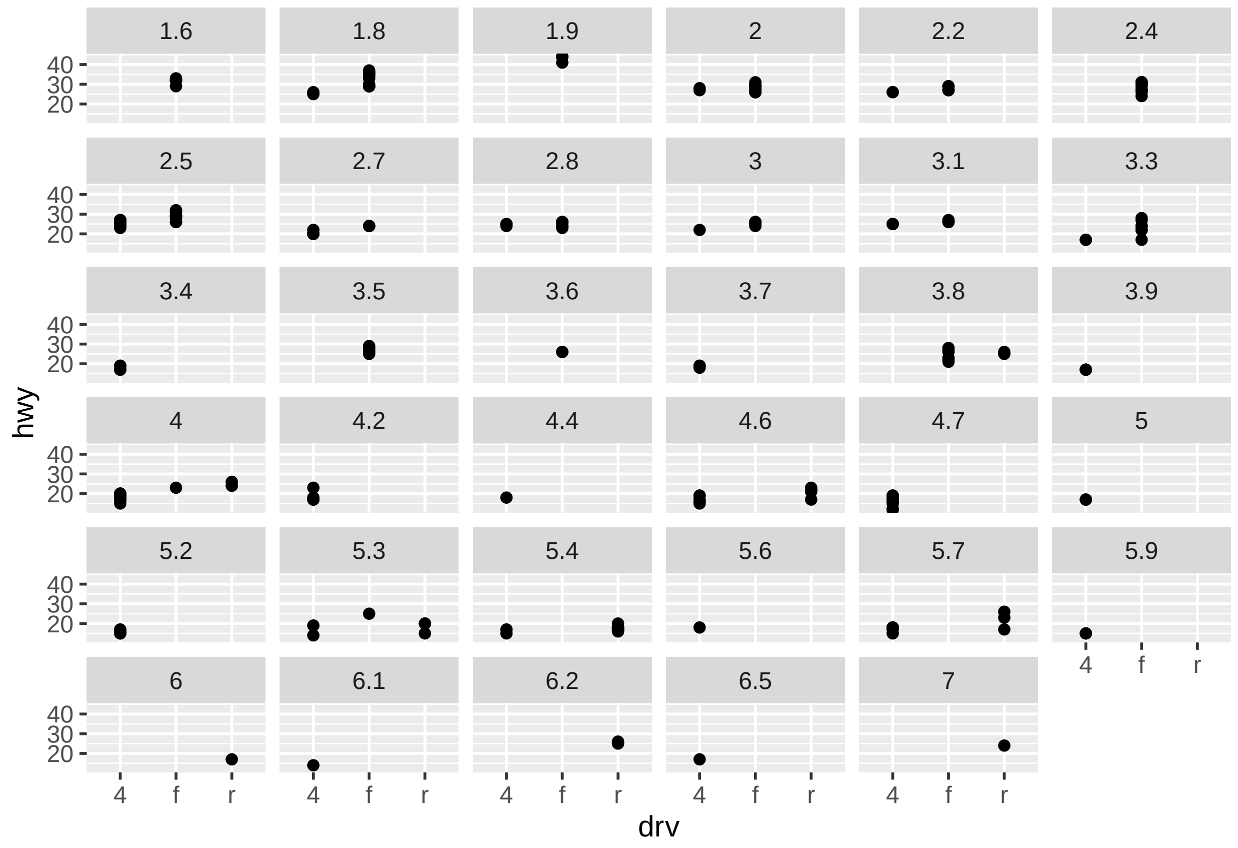


Exercises

E1. What happens if you facet on a continuous variable?

Ans.

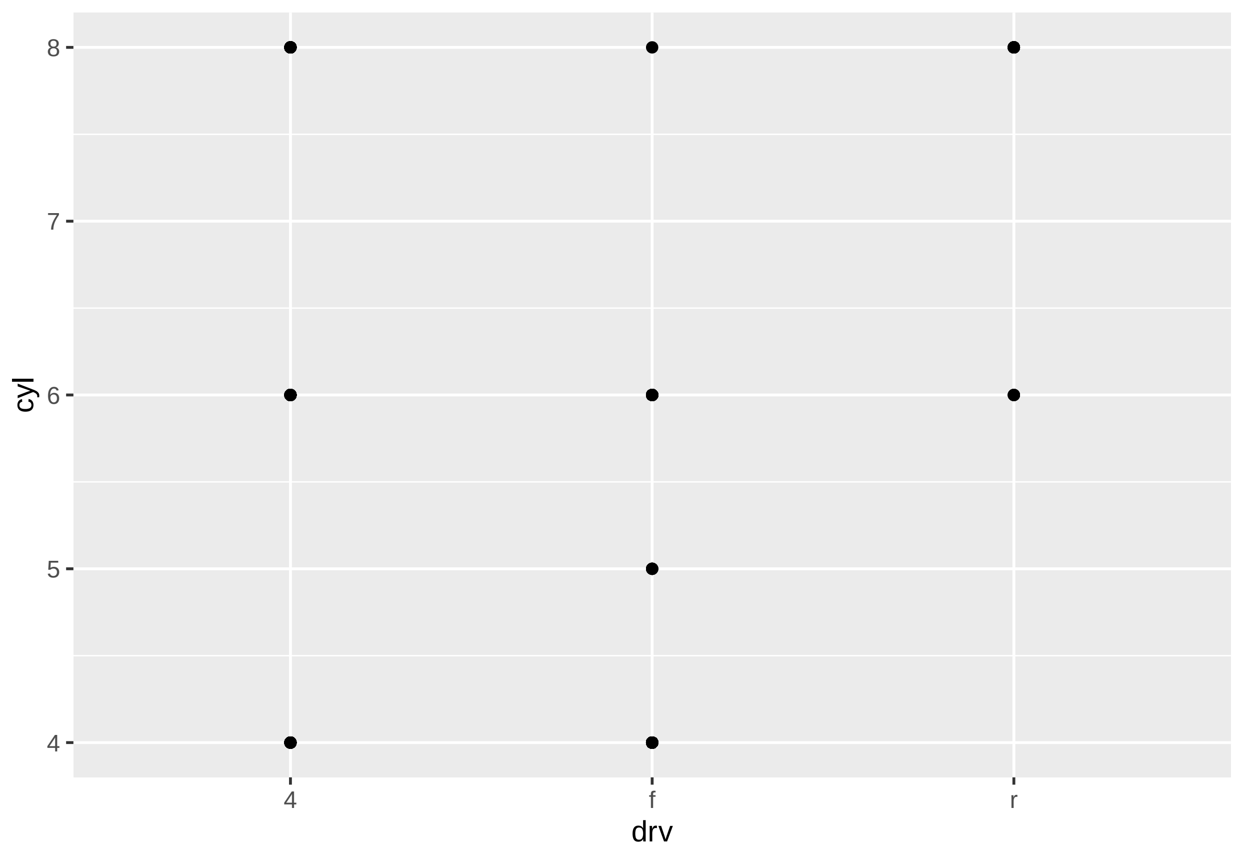
```r
ggplot(mpg, aes(x = drv, y = hwy)) +
  geom_point() +
  facet_wrap(~ displ)
```



Ans. Each value is treated as a category.

E2. What do the empty cells in the plot above with 'facet_grid(drv ~ cyl) mean? Run the following code. How do they relate to the resulting plot?

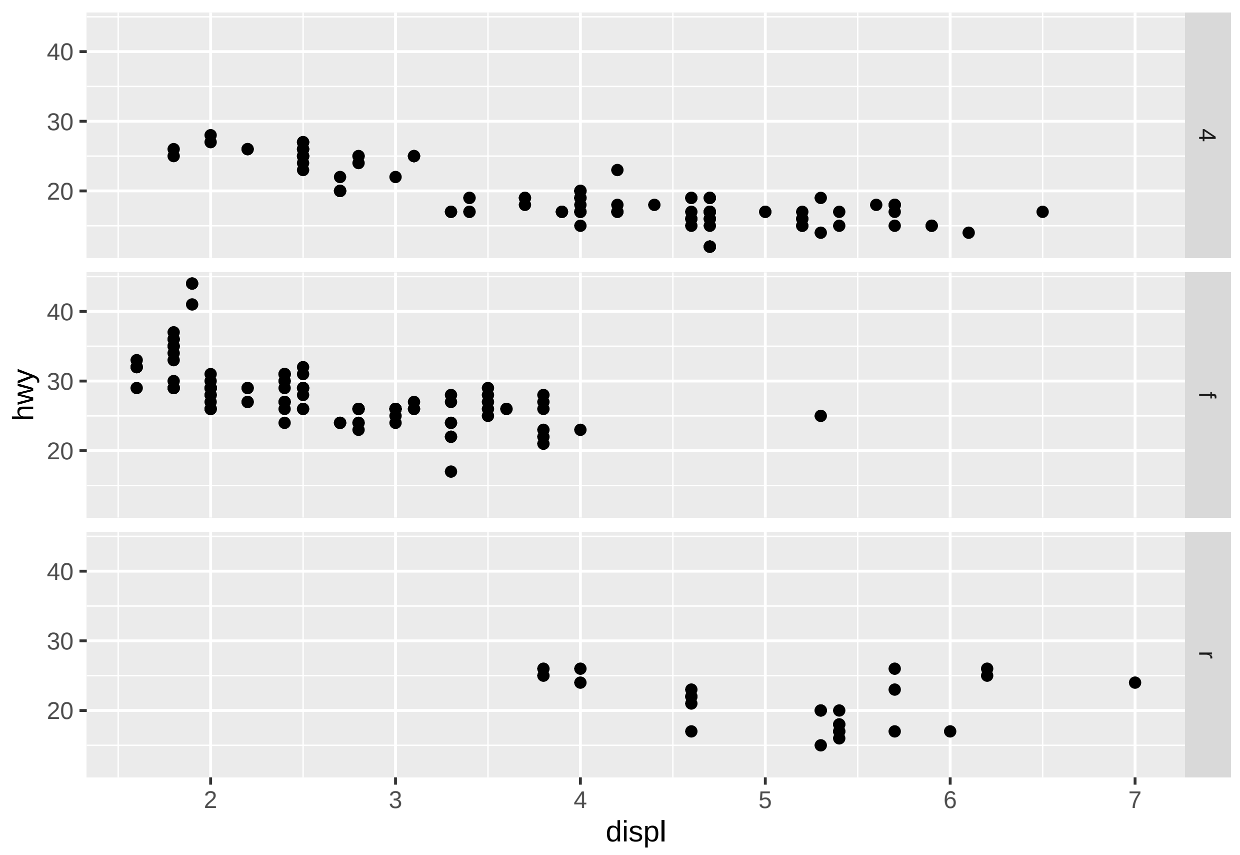
```r
ggplot(mpg) +
  geom_point(aes(x = drv, y = cyl))
```



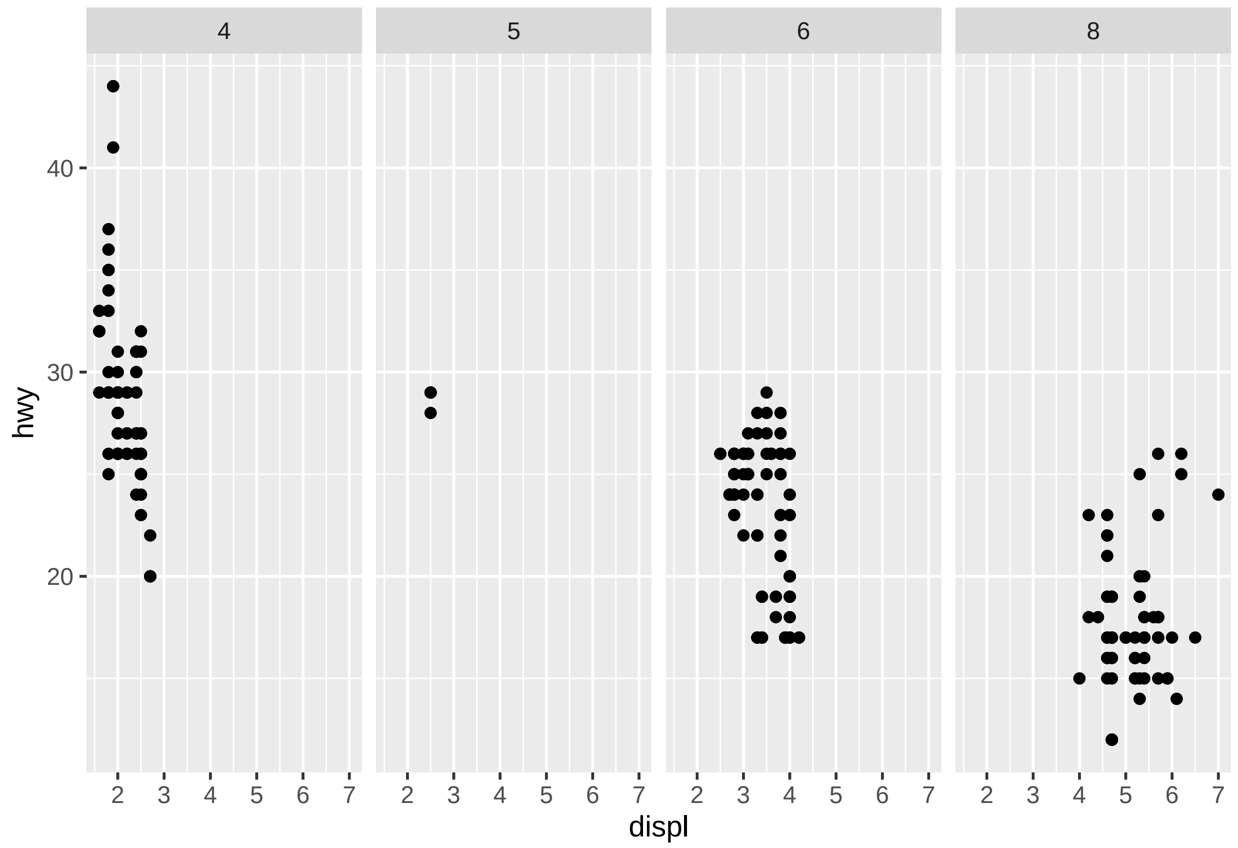
Ans. The empty cells mean that there are no data that support that combination of number of cylinders and the drive type of the car.

E3. What plots does the following code make? What does `.` do?

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



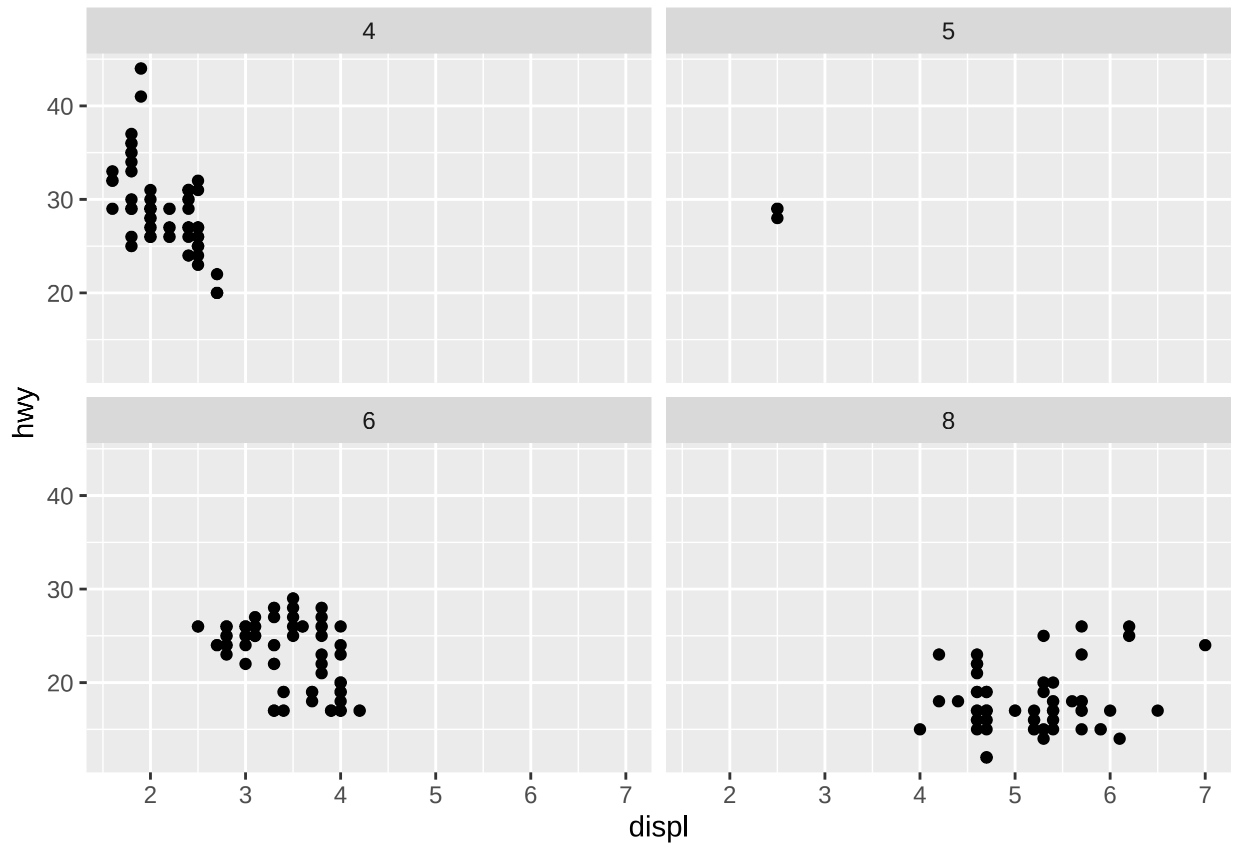
```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_grid(. ~ cyl)
```



Ans. The `.` acts as a placeholder when you don't want to specify what to facet on "x" or "y" axis.

E4. Take the first faceted plot in this section:

```r
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  facet_wrap(~ cyl, nrow = 2)
```



What are the advantages to using faceting instead of the color aesthetic? What are the disadvantages? How might the balance change if you had a larger dataset?

Ans. It is generally easier on the eyes to look at a faceted plot than to look at the unfaceted version, but with the legend showing. Our eyes do not have to move back and forth between the legend and what is plotted. But when you have too many columns or when you have too many categories, each facet might end up looking like tiny strips of chart. In this case, instead of faceting the plot, showing the legent might be better.

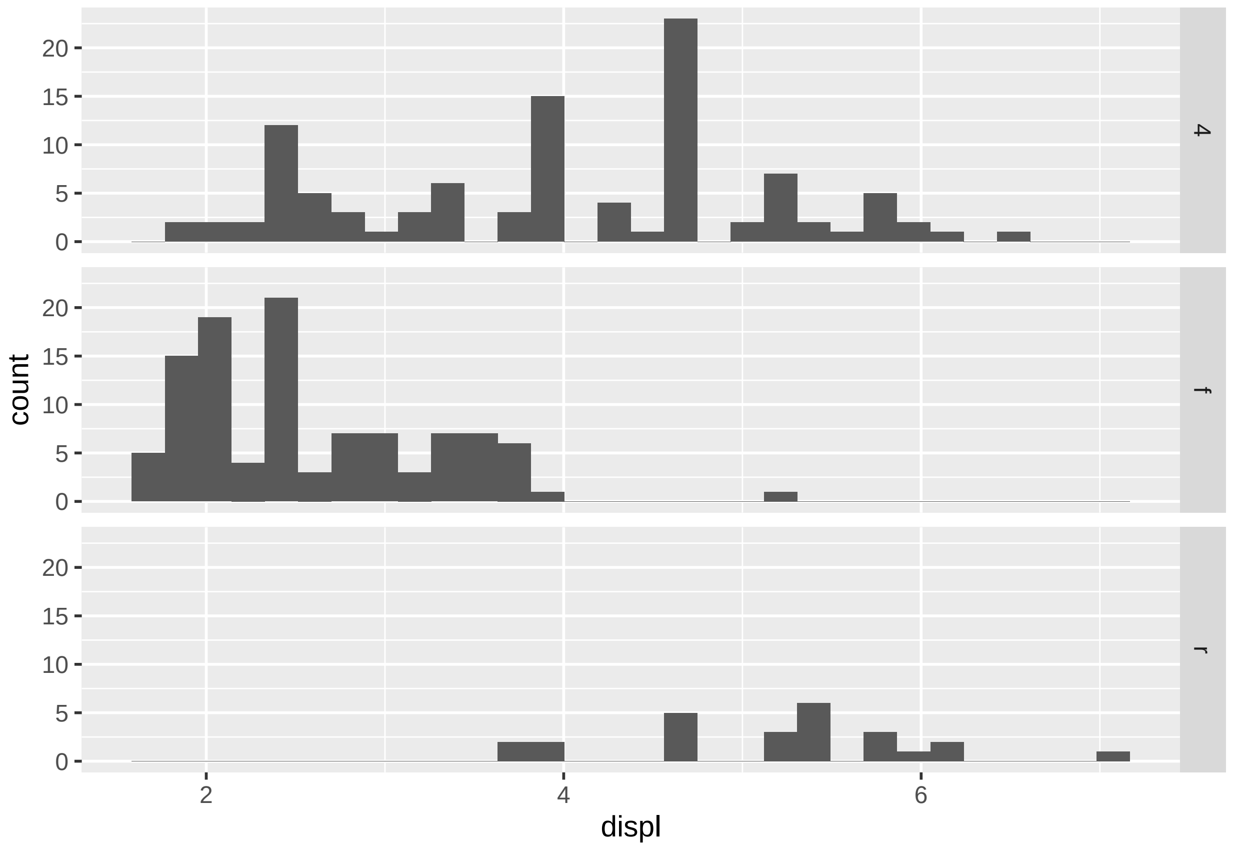
E5. Read `?facet_wrap`. What does `nrow` do? What does `ncol` do? What other options control the layout of the individual panels? Why doesn't `facet_grid()` have `nrow` and `ncol` arguments?

Ans. The `ncol` and `nrow` arguments specify the number of rows and number of columns to display our small multiples. ~facet_grid()‘ doesn't have those arguments since the dimensions of the grid will be dictated by how many categories there are in our chosen x-category and y-category.

E6. Which of the following plots makes it easier to compare engine size (`displ`) across cars with different drive trains? What does this say about when to place a faceting variable across rows and columns?

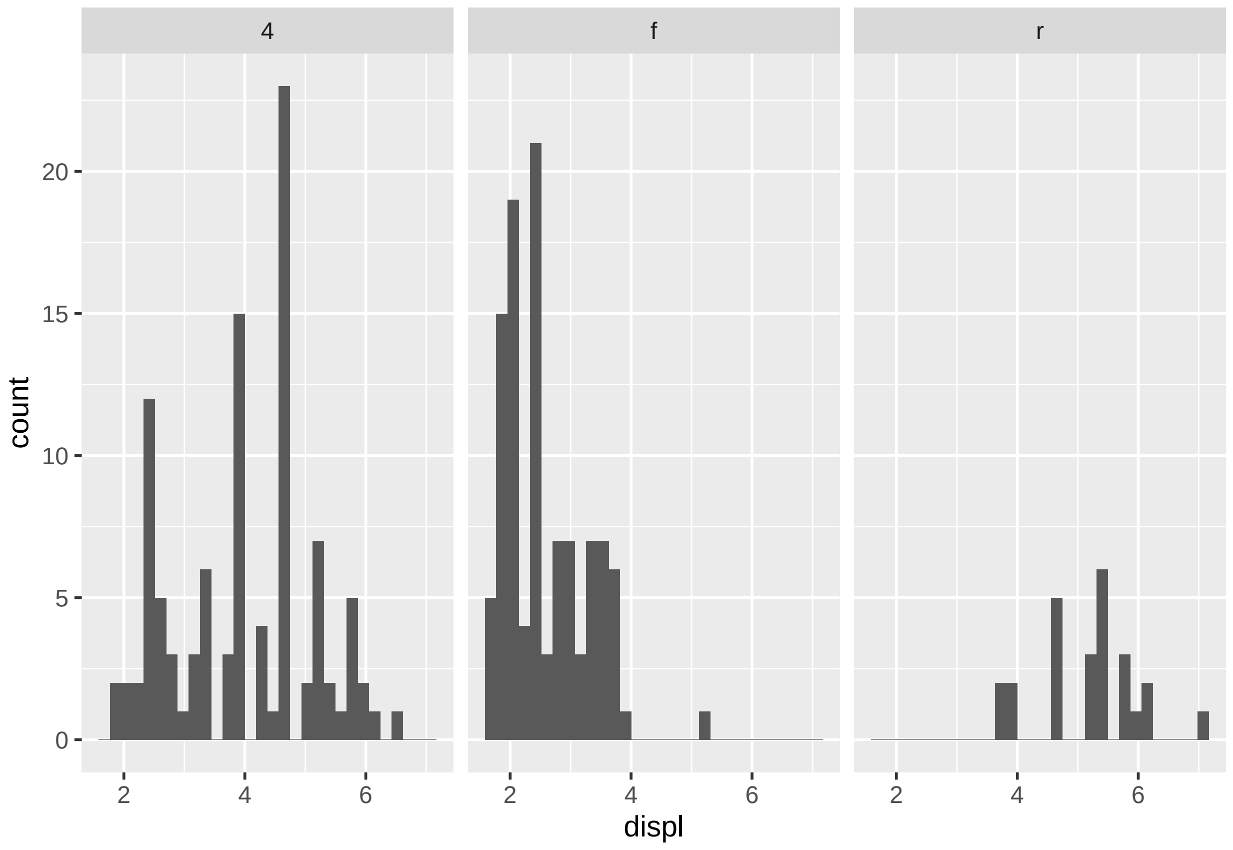
```r
ggplot(mpg, aes(x = displ)) +
  geom_histogram() +
  facet_grid(drv ~ .)
```

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



```r
ggplot(mpg, aes(x = displ)) +
  geom_histogram() +
  facet_grid(. ~ drv)
```

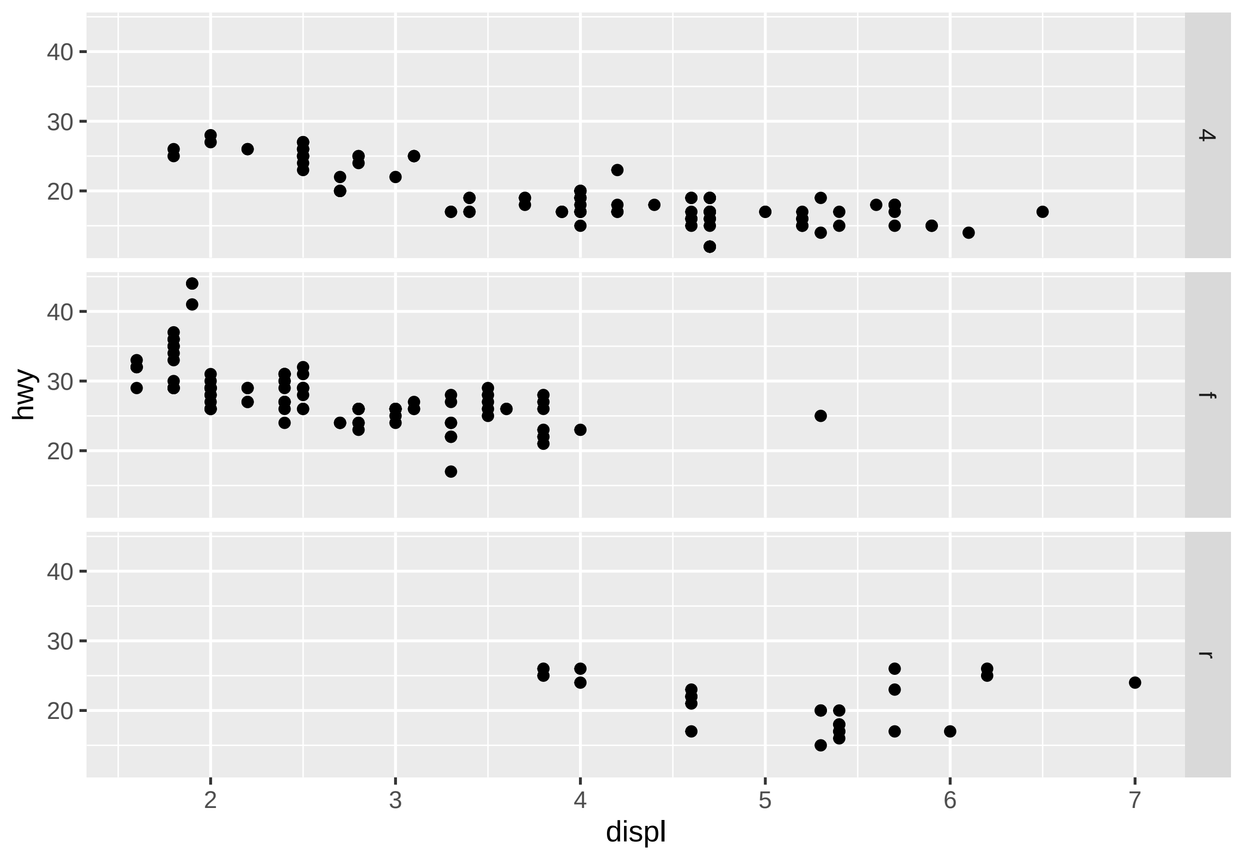
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Ans. It is easier to make the comparison looking at the first plot. I don't know how to put it in words, but it seems that a "wide" layout is easier to look at than a "tall" layout. There is a reason why our laptops and computer monitors are laid out in a landscape orientation by default.

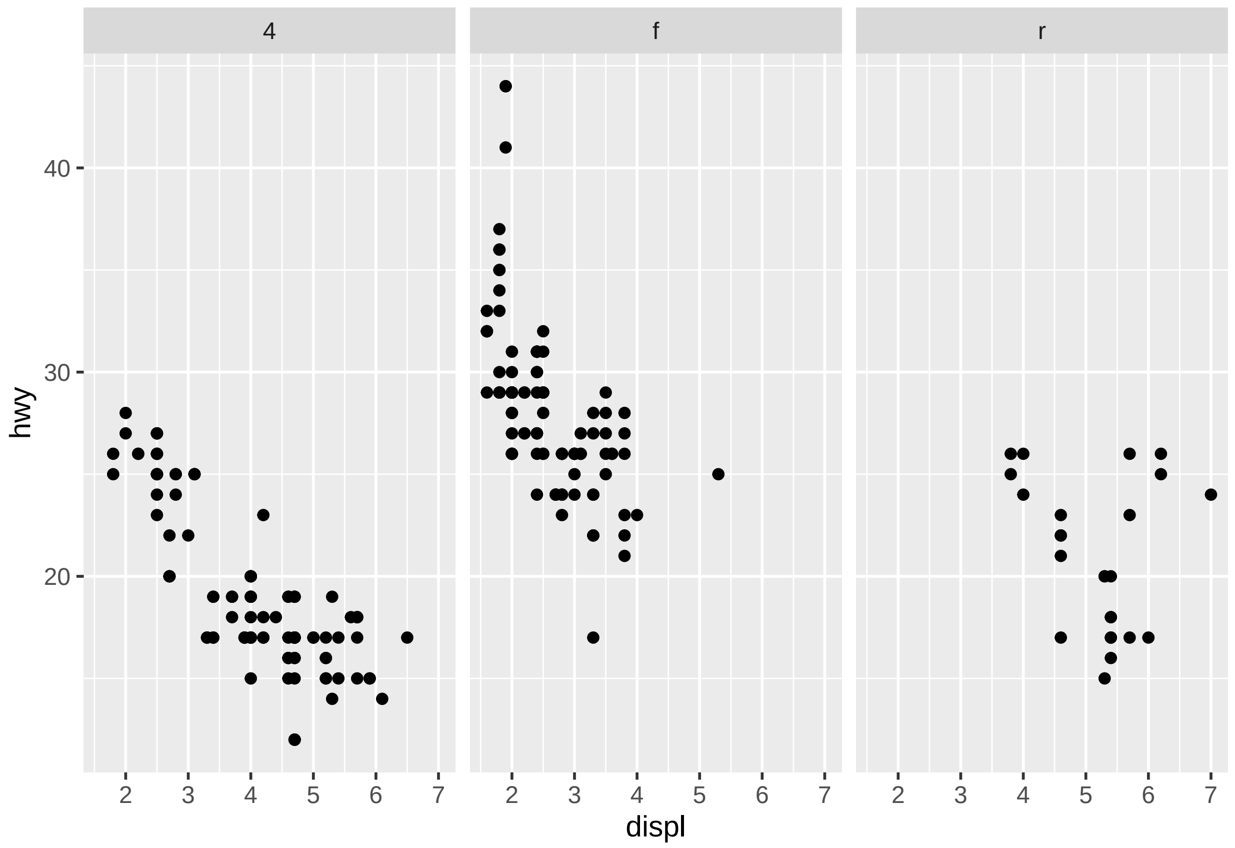
E7. Recreate the following plot using `facet_wrap() ins tead of`facet_grid()‘. How do the positions of the facet labels change?

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



Ans.

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

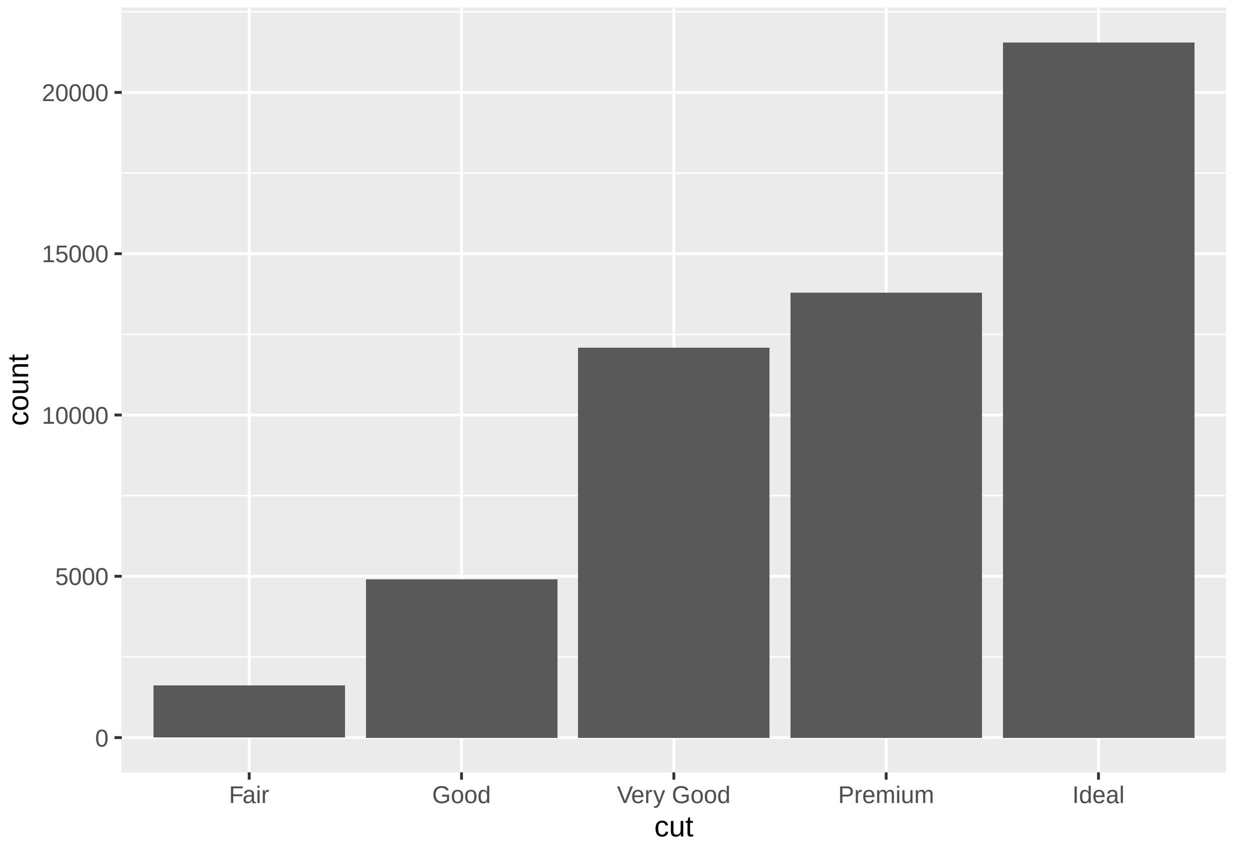


With `facet_wrap()`, the labels are rendered on top.

Statistical transformations

Basic bar plot:

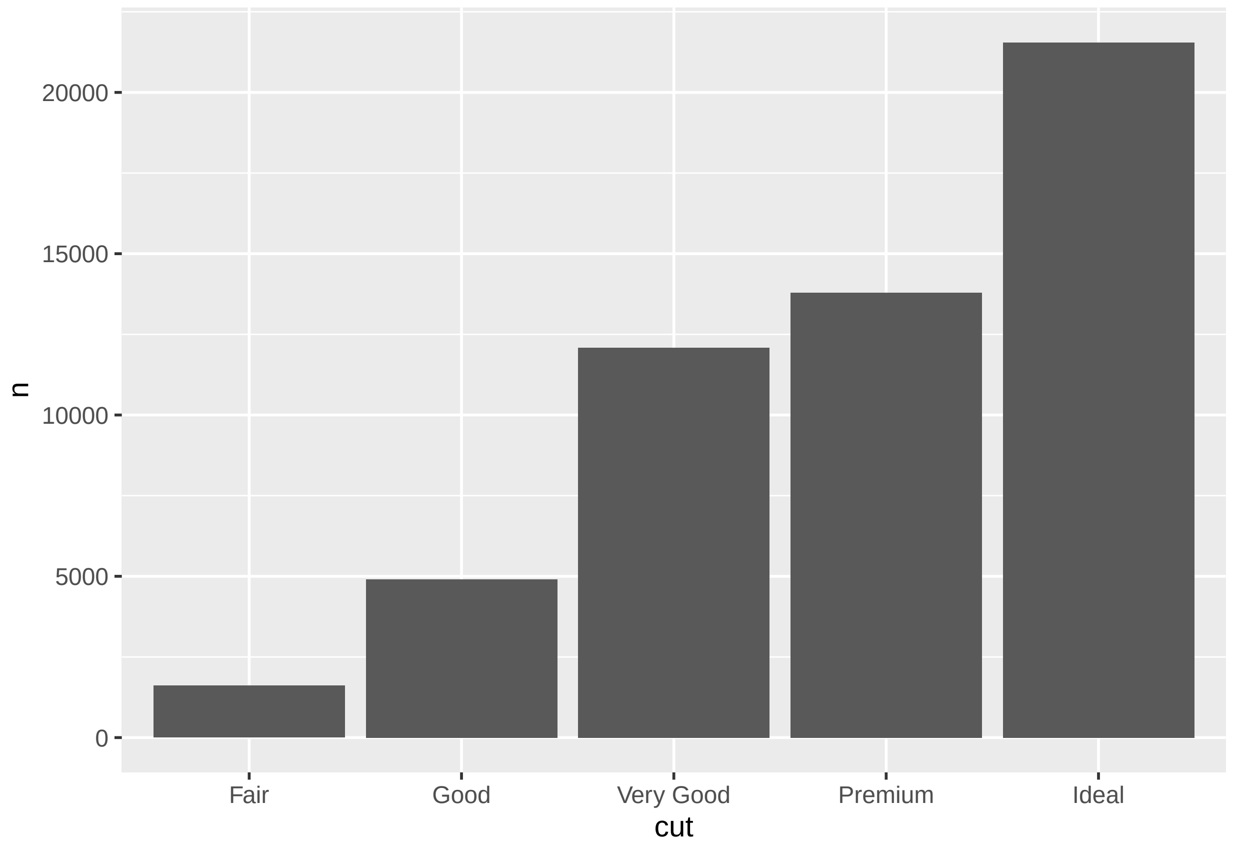
```
ggplot(diamonds, aes(x = cut)) +
  geom_bar()
```



- The default value of the `stat` argument is "count".

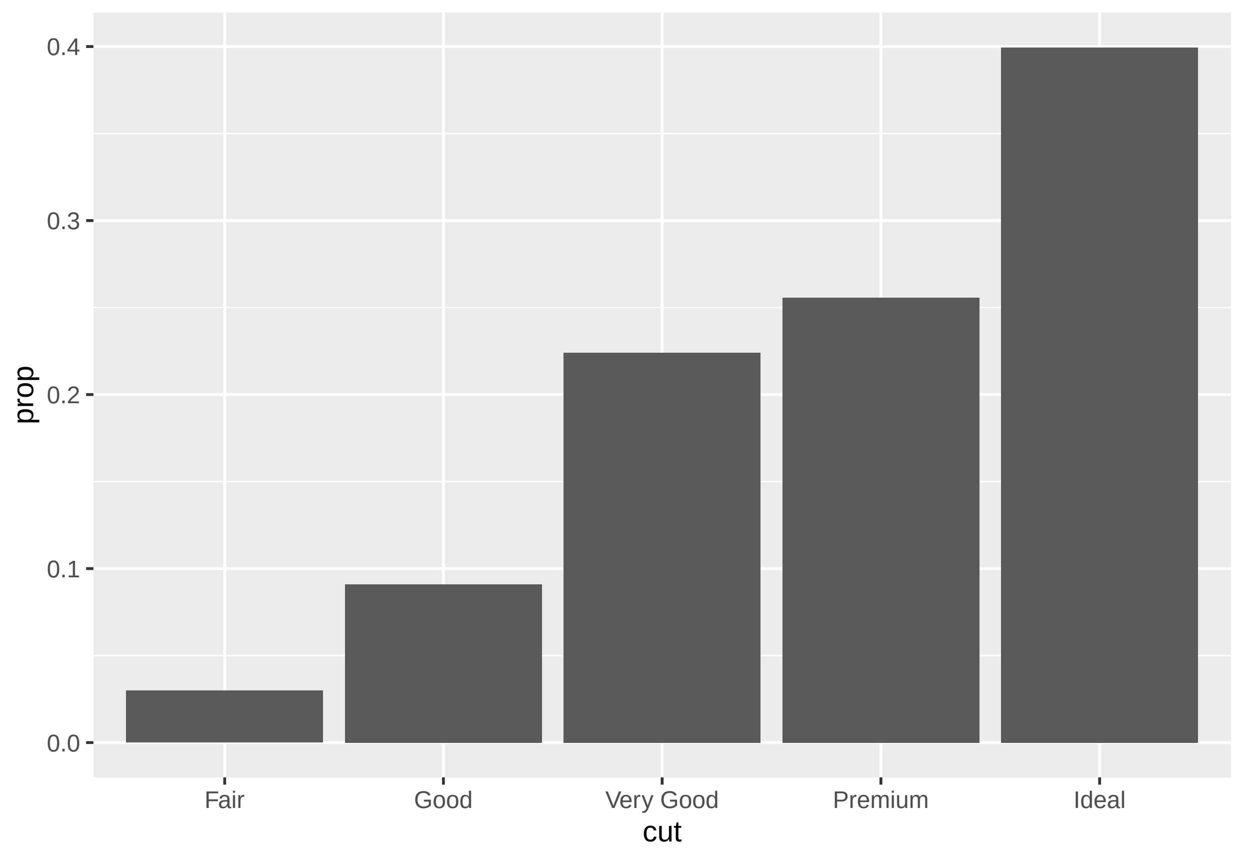
To plot the raw values of a y variable in a barplot:

```r
diamonds |>
  count(cut) |>
  ggplot(aes(x = cut, y = n)) +
  geom_bar(stat = "identity")
```



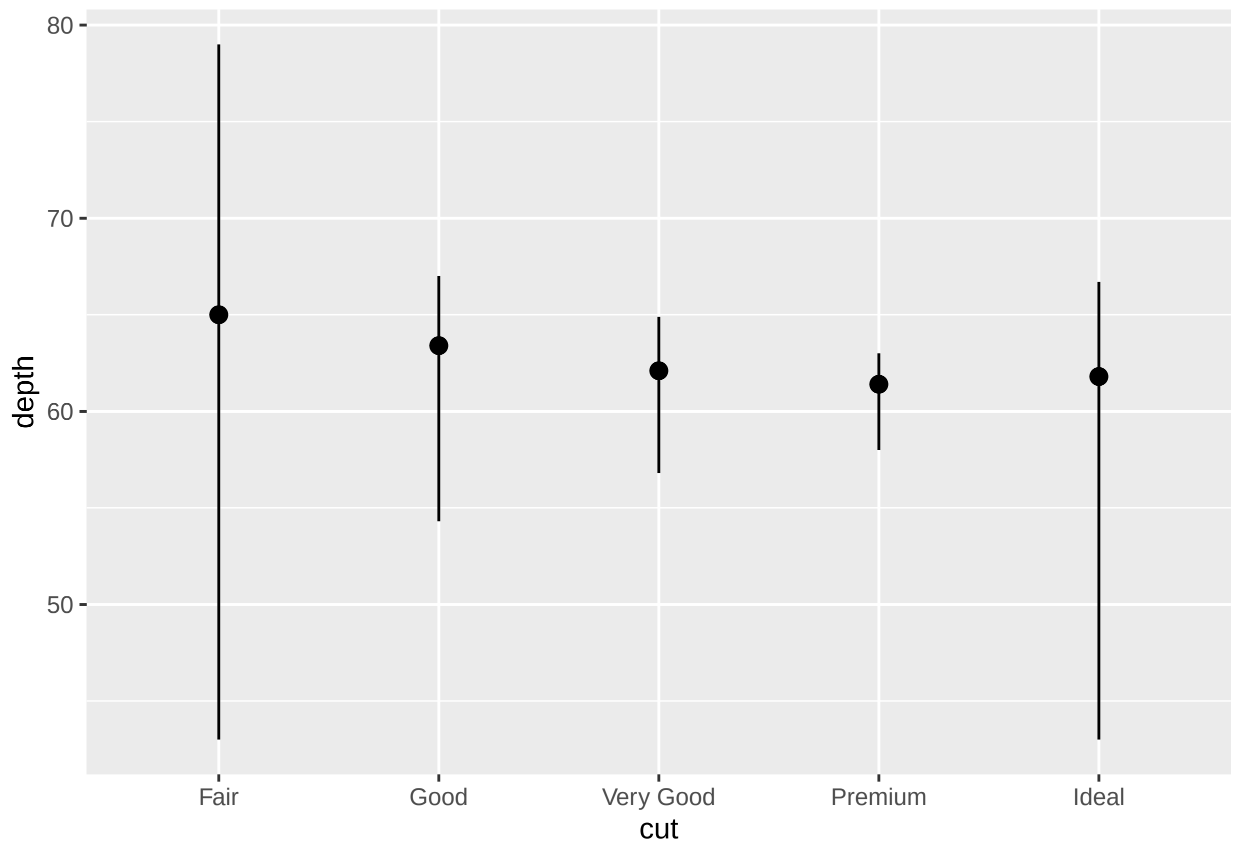
To plot the proportion of x values instead of count in a bar plot:

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



To plot a “stat summary”:

```
ggplot(diamonds) +
  stat_summary(
    aes(x = cut, y = depth),
    fun.min = min,
    fun.max = max,
    fun = median
  )
```

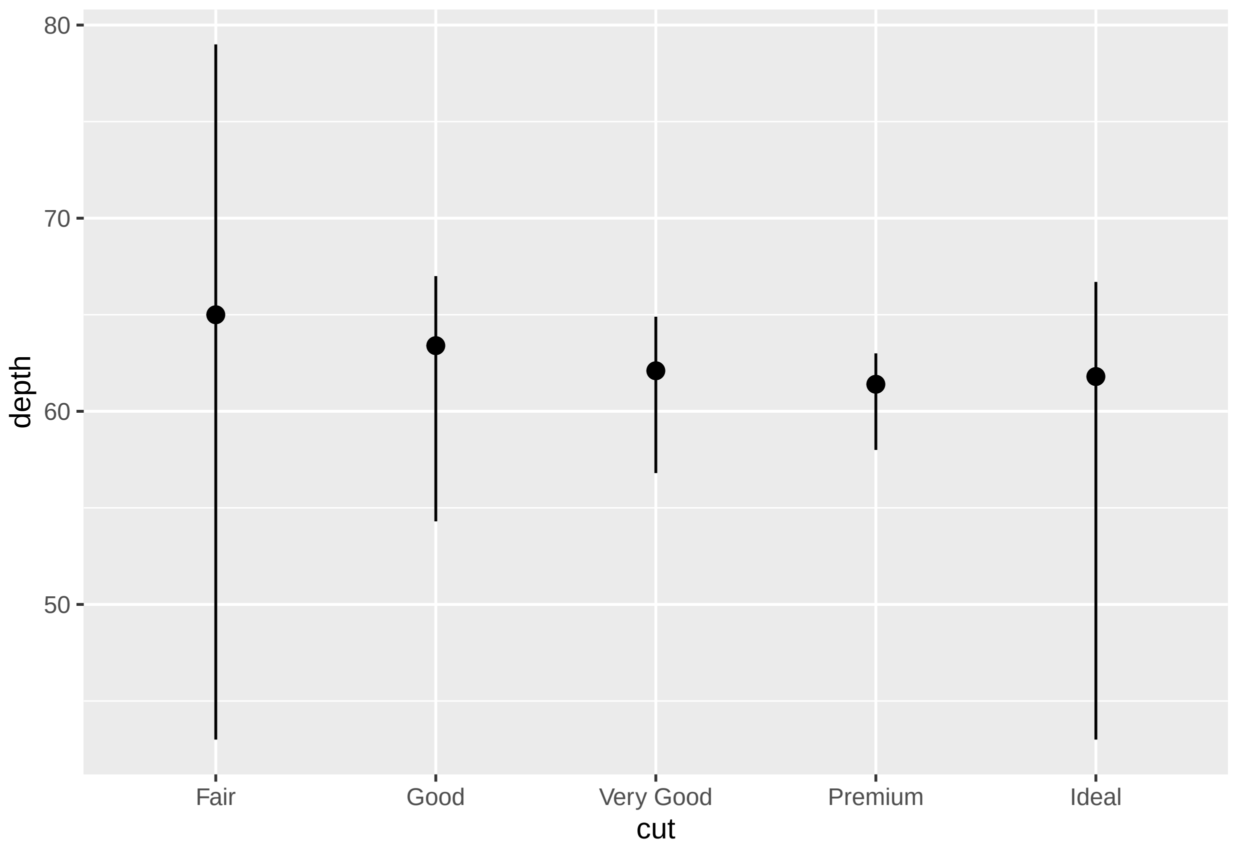


Exercises

E1. What is the default geom associated with `stat_summary()`? How could you rewrite the previous plot to use that geom function instead of the stat function?

Ans. pointrange.

```
ggplot(diamonds, aes(x = cut, y = depth)) +
  geom_pointrange(
    stat = "summary",
    fun.min = min,
    fun.max = max,
    fun = median
  )
```



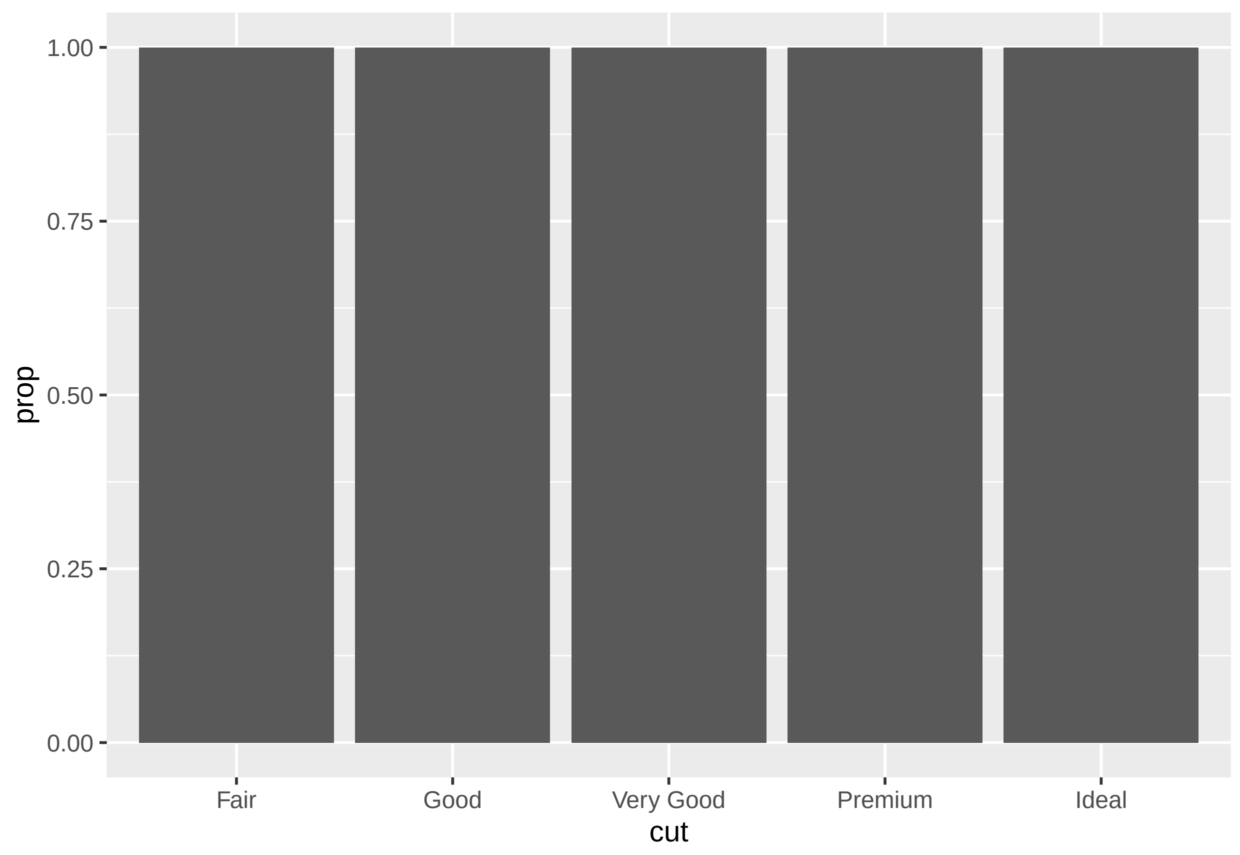
E2. What does `geom_col()` mean? How is it different from `geom_bar()`?

Ans. `geom_col()` has a default "stat" of "identity" which applies the raw data to create the plot. It also expects 2 values: the category on x-axis, and the values that represent the height of the bars.

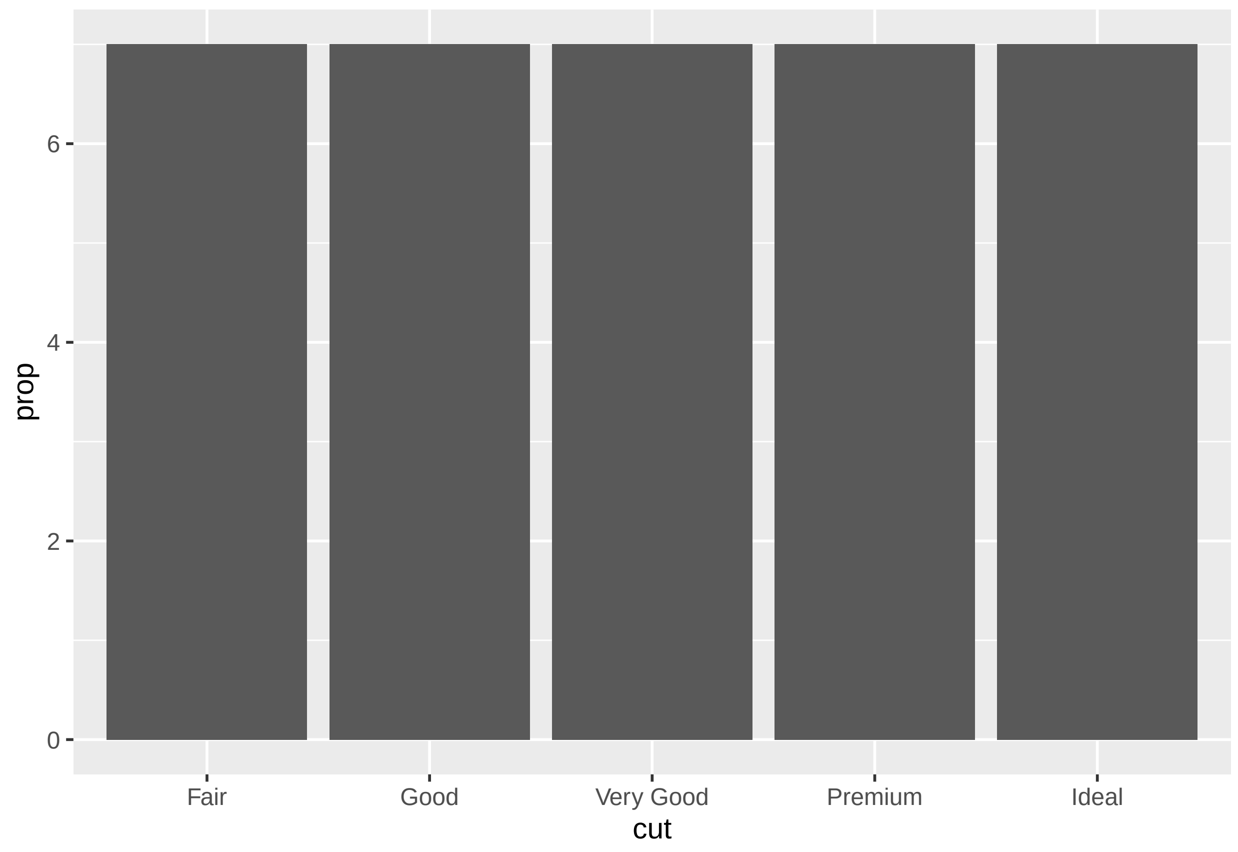
`geom_bar()` has a default "stat" of "count". It doesn't need a second argument to determine the height of the bars.

E5. In our proportion bar chart, we needed the set `group = 1`. Why? In other words, what is the problem with these two graphs?

```r
ggplot(diamonds, aes(x = cut, y = after_stat(prop))) +
  geom_bar()
```



```r
ggplot(diamonds, aes(x = cut, full = color, y = after_stat(prop))) +
  geom_bar()
```

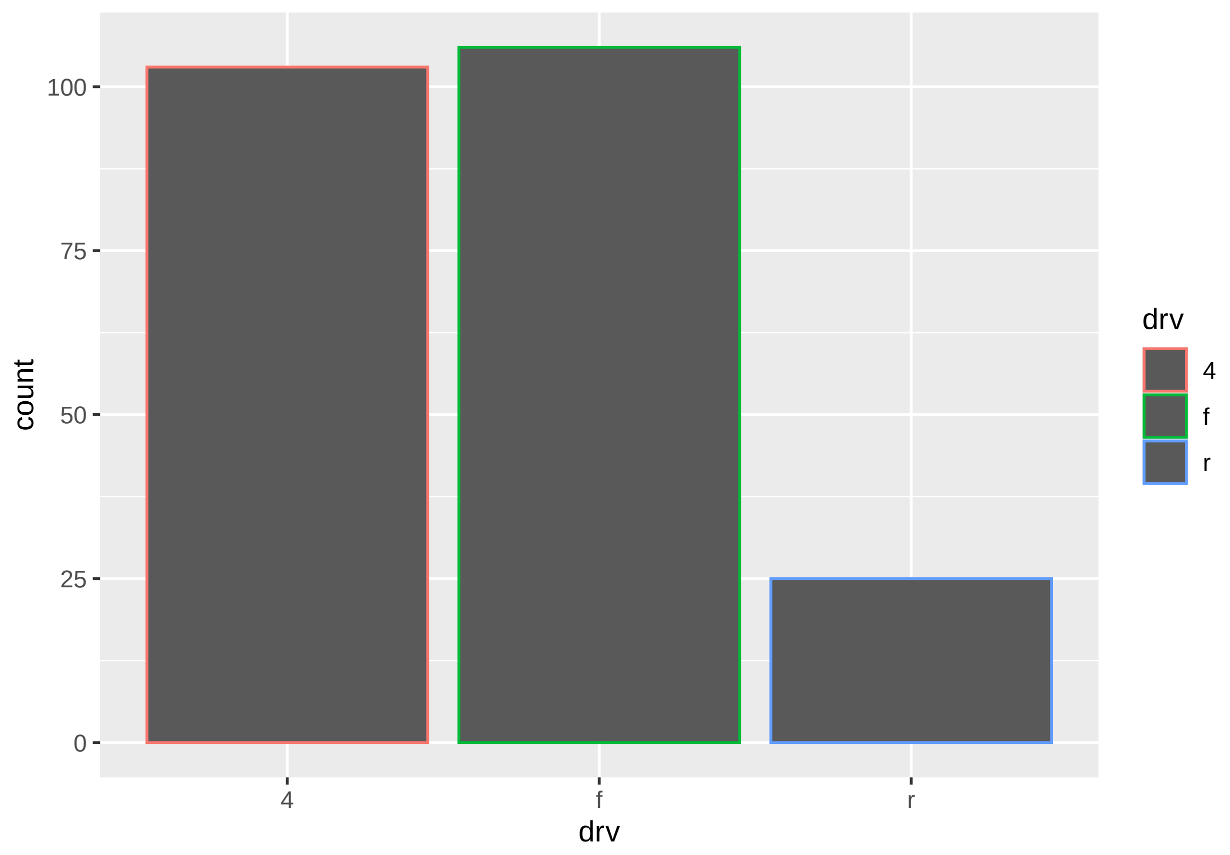


Ans. If there is no group supplied, each "prop" will be calculated in proportional to each x. Adding `group = 1` tells `geom_bar()` to do calculate the proportion to the entire group (and not only in proportion to itself).

Position adjustments

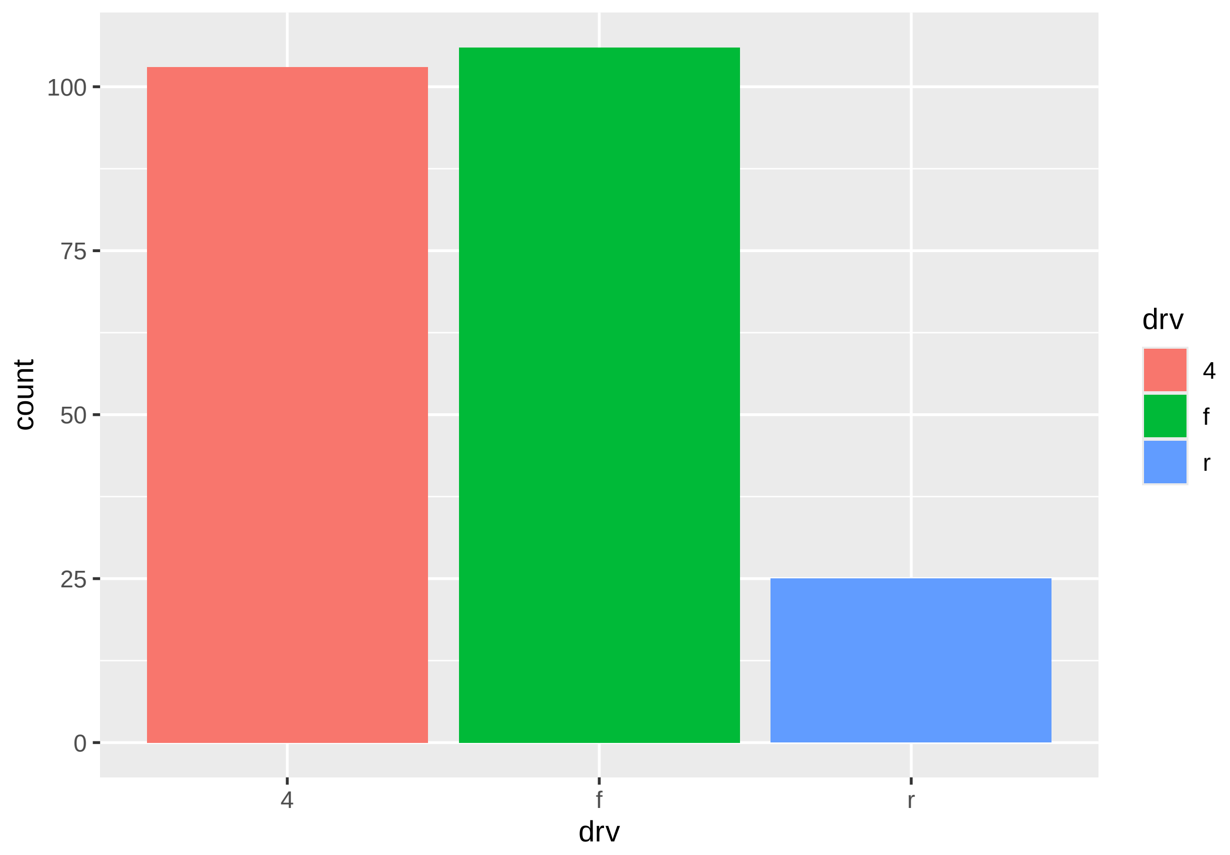
Coloring a bar chart using the `color` aesthetic:

```r
ggplot(mpg, aes(x = drv, color = drv)) +
  geom_bar()
```



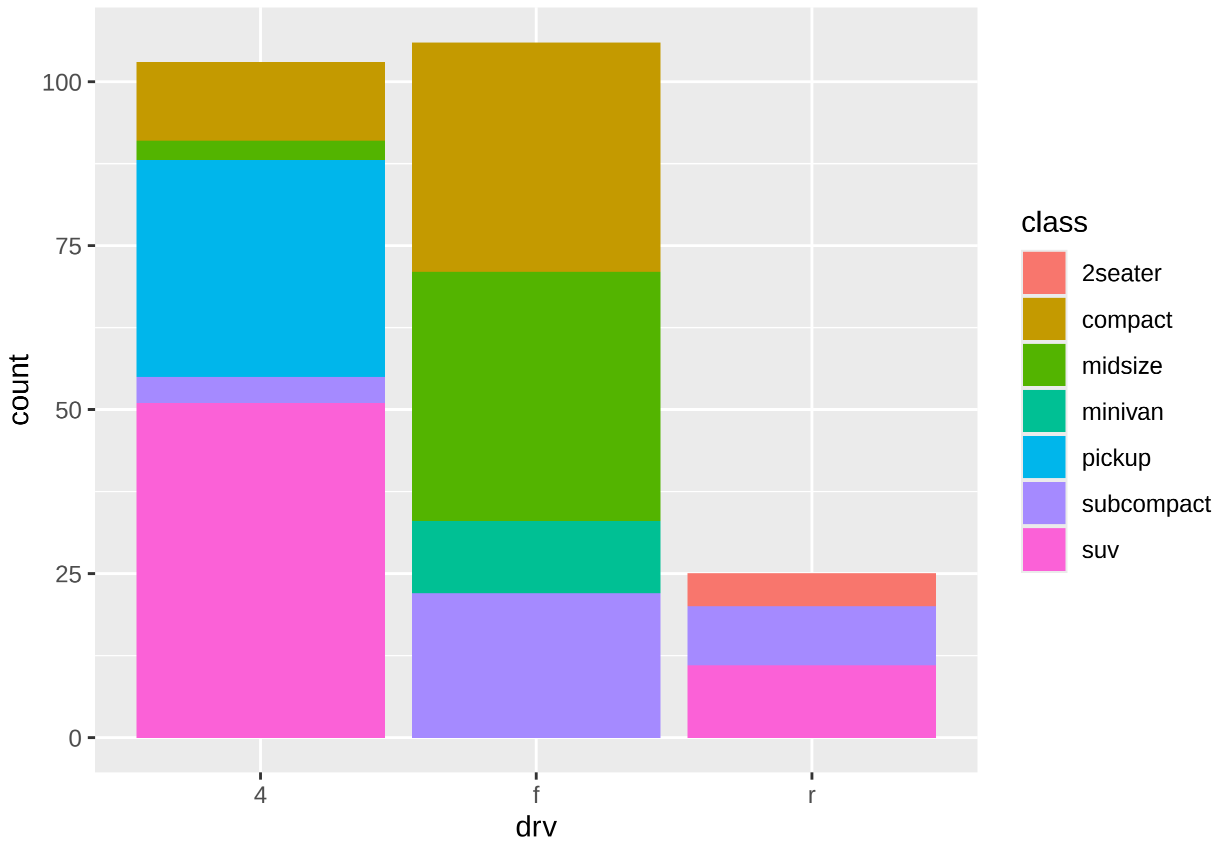
Coloring a bar chart using the `fill` aesthetic:

```
ggplot(mpg, aes(x = drv, fill = drv)) +
  geom_bar()
```



If we map the `fill` aesthetic to another variable, we will get a stacked bar plot:

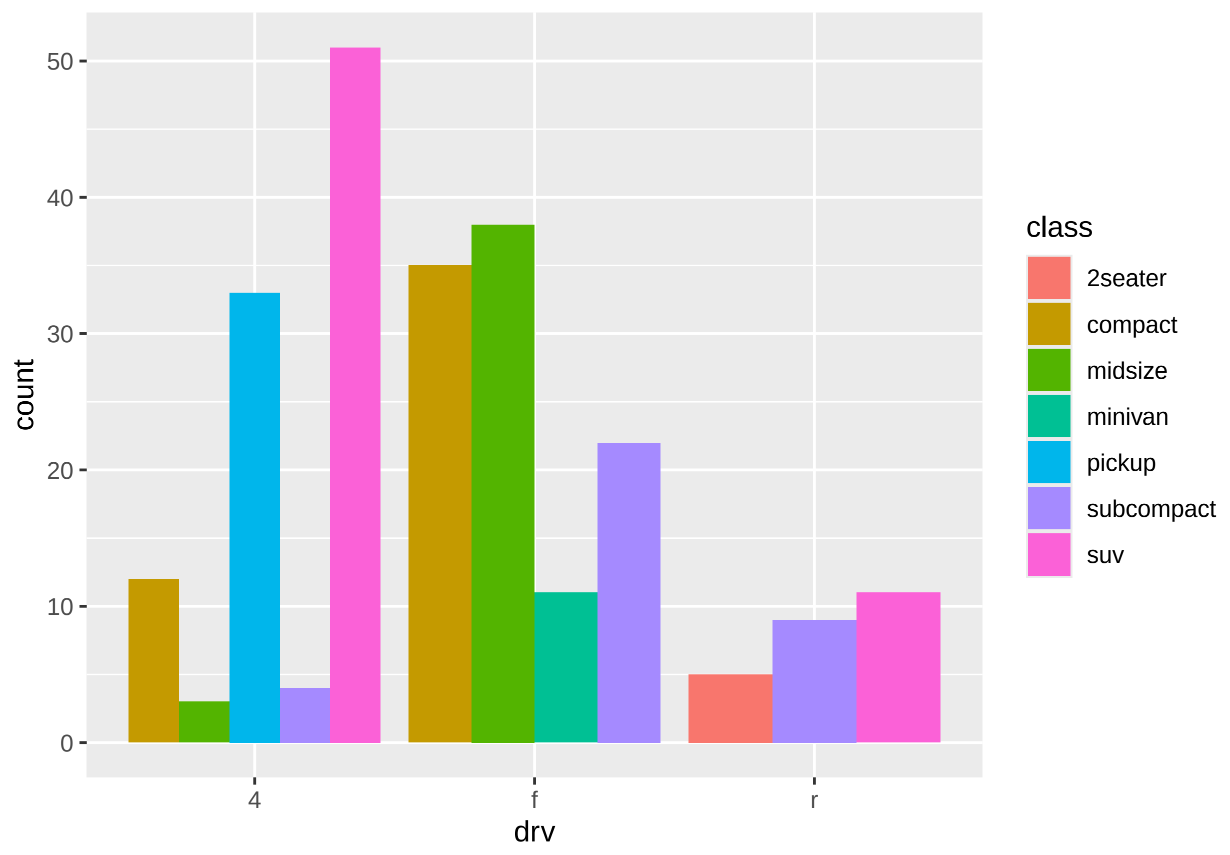
```r
ggplot(mpg, aes(x = drv, fill = class)) +
  geom_bar()
```



- this stacking is determined by the default value of the`position` argument ("stack"). Other options are:
- `"identity"`: each object will be placed exactly where it falls in the context of the graph.
- `"dodge"`: places overlapping objects directly beside one another.
- `"fill"`: similar to "stack" but each set of stacked bars will be of the same height.

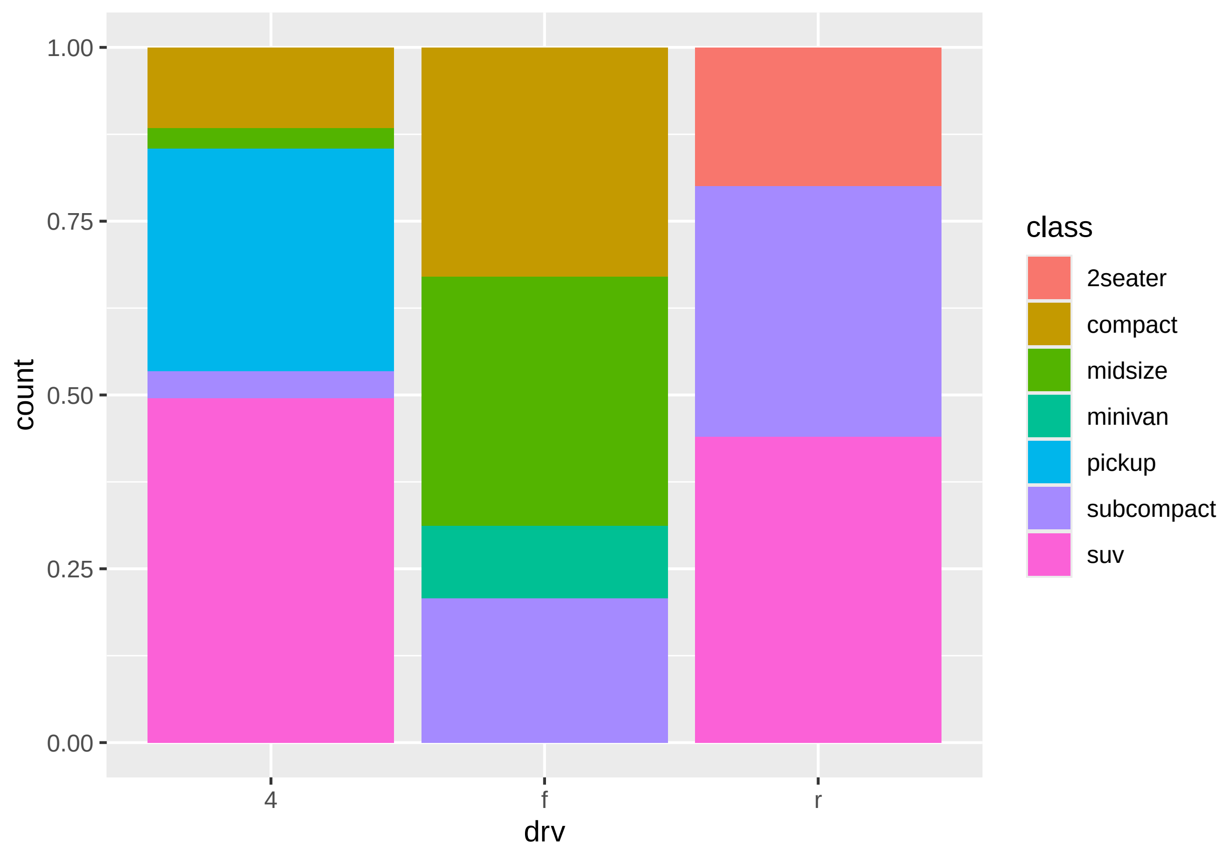
Bar plot with `position = "dodge"`:

```r
ggplot(mpg, aes(x = drv, fill = class)) +
  geom_bar(position = "dodge")
```



Bar plot with 'position = "fill"':

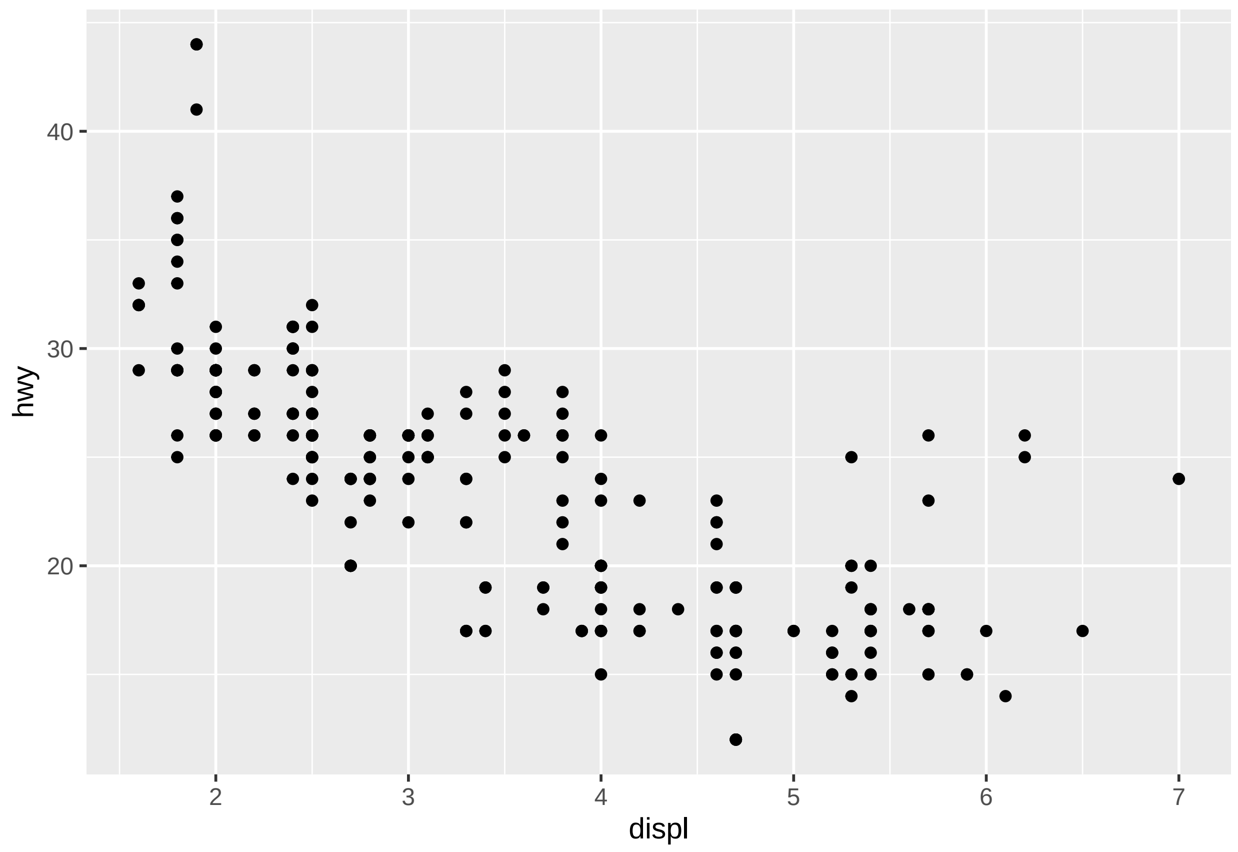
```r
ggplot(mpg, aes(x = drv, fill = class)) +
  geom_bar(position = "fill")
```



Jittering to prevent "overplotting"

Scatterplot without jitter:

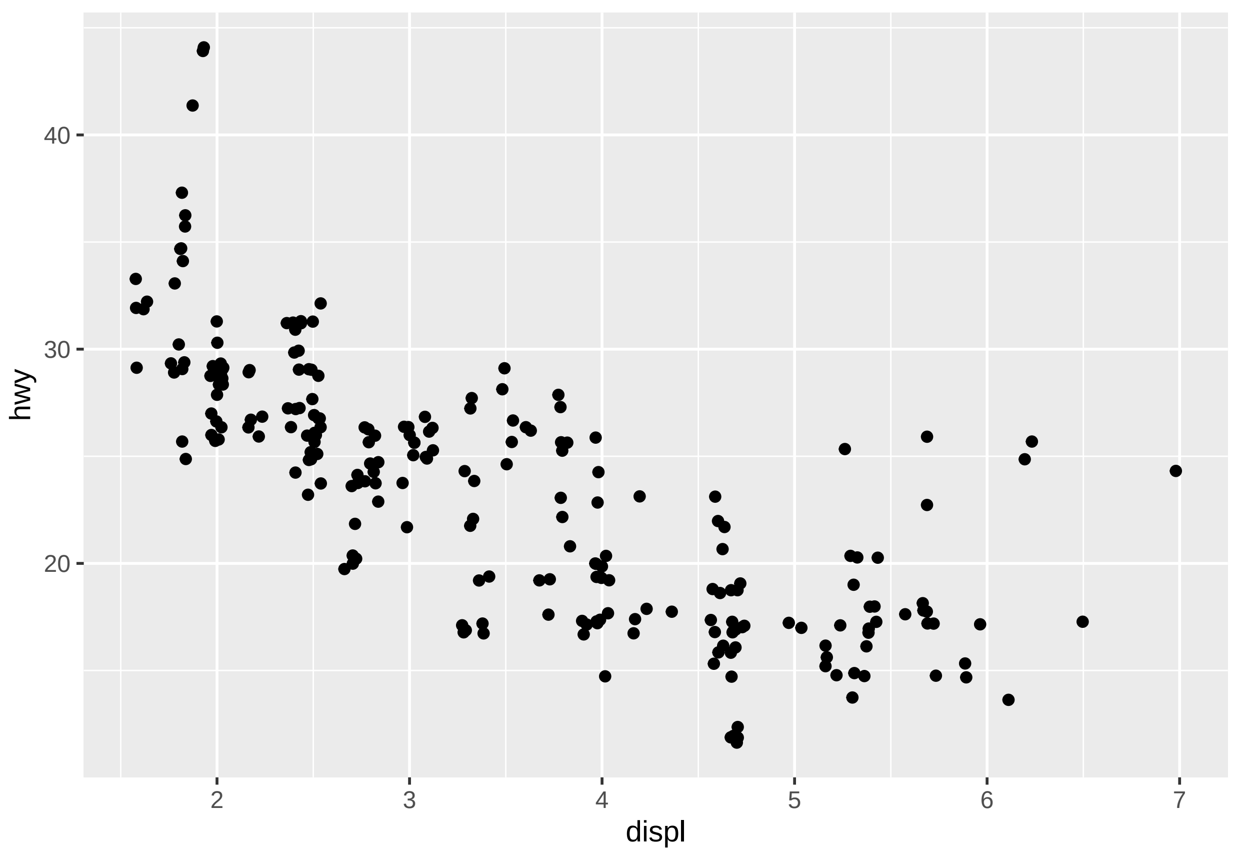
```r
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point()
```



Here, some dots are plotted on top of each other. We can only see 126 points even though there are 234 observations in the dataset.

Scatterplot with jitter:

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

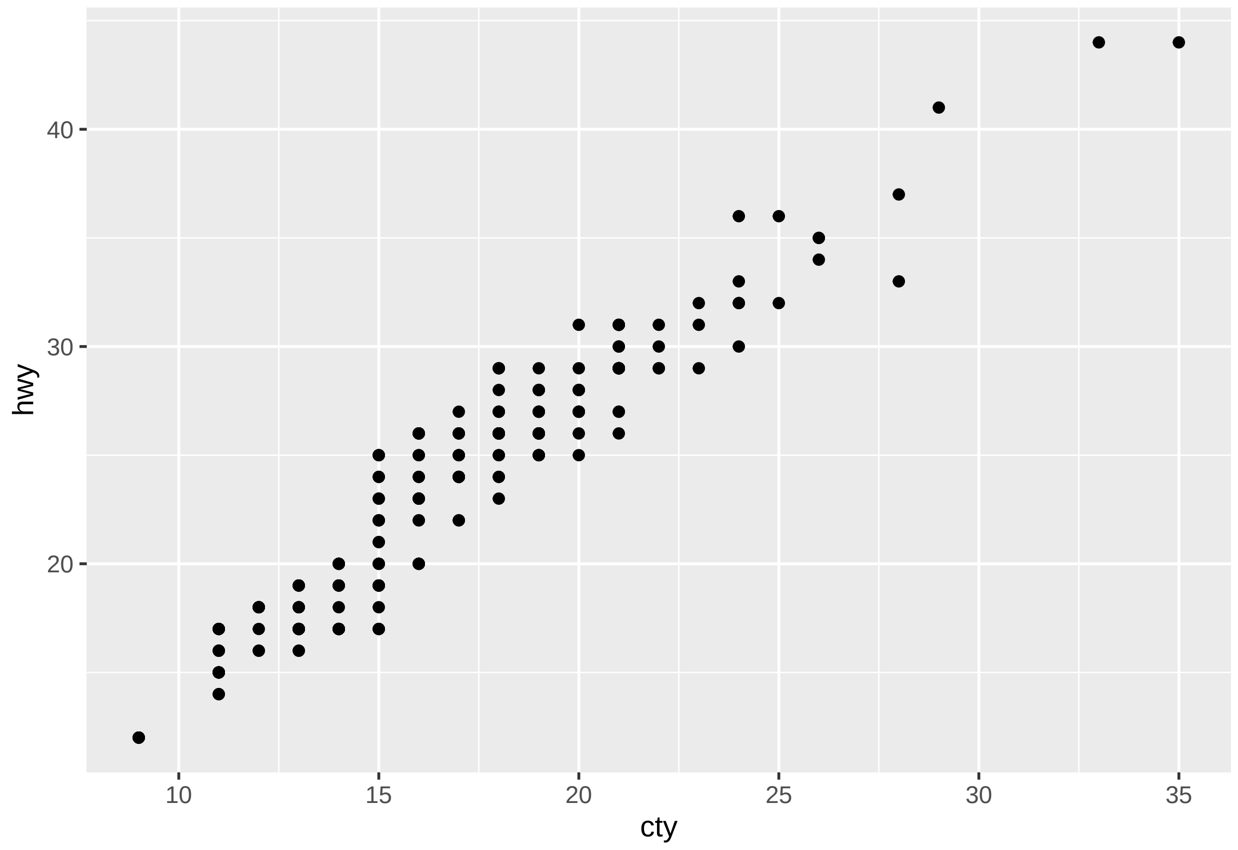


- The "jitter" argument adds a small amount of random noise to each point, preventing overplotting, since no two points are likely to get the same amount of random noise.
- instead of `geom_point(position = "jitter")`, we can also use the shorthand `geom_jitter()`

Exercises

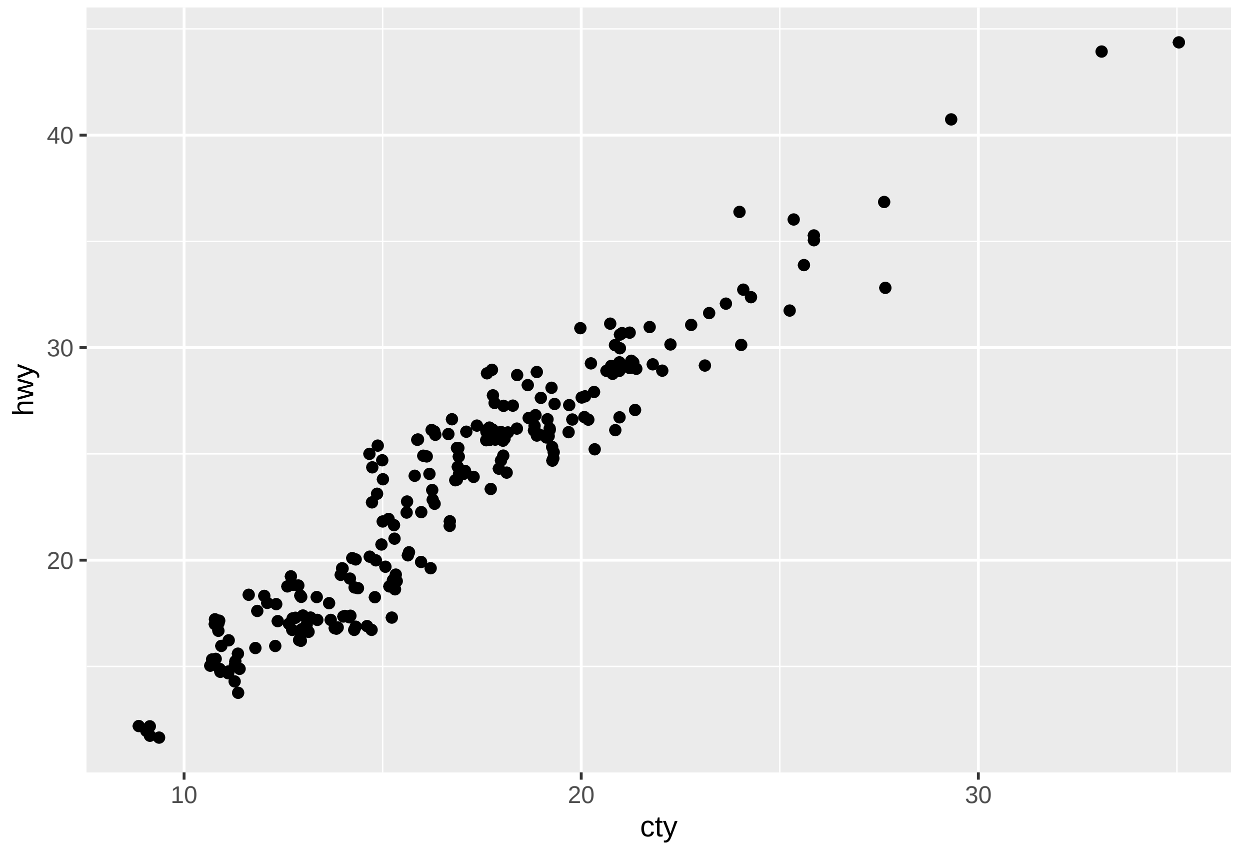
E1. What is the problem with the following plot? How could you improve it?

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



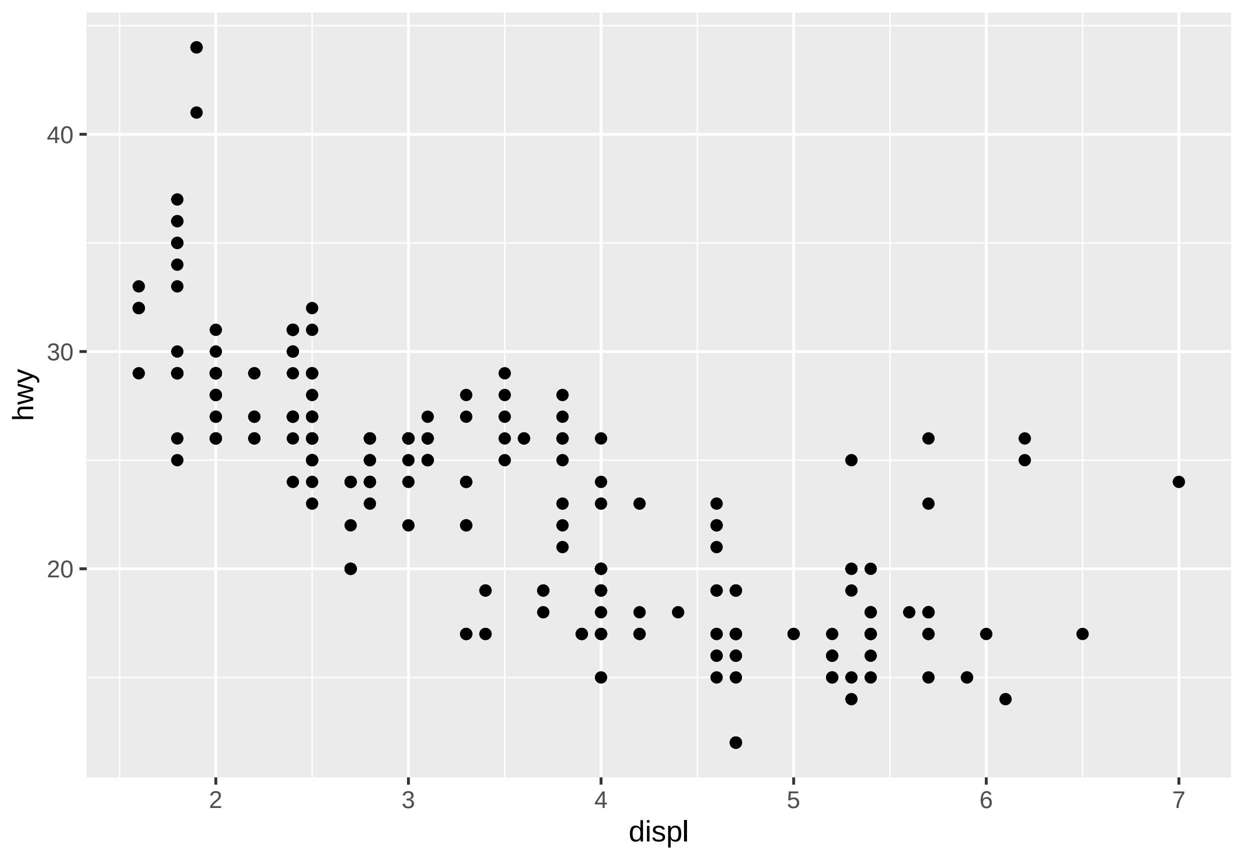
Ans. By applying a jiter to prevent overplotting.

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

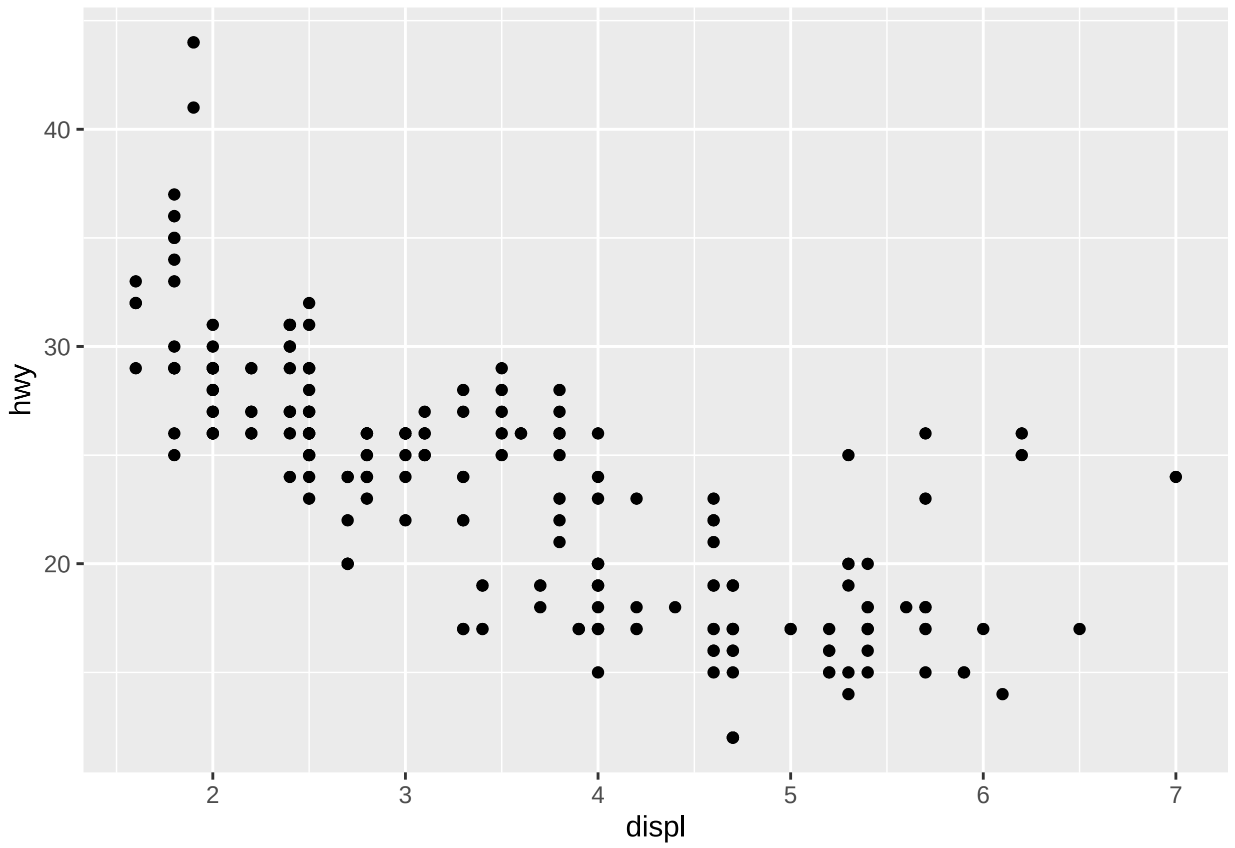


E2. What, if anything, is the difference between the two plots? Why?

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



```r
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(position = "identity")
```



Ans. They practically the same, since 'position = "identity" is the default setting.

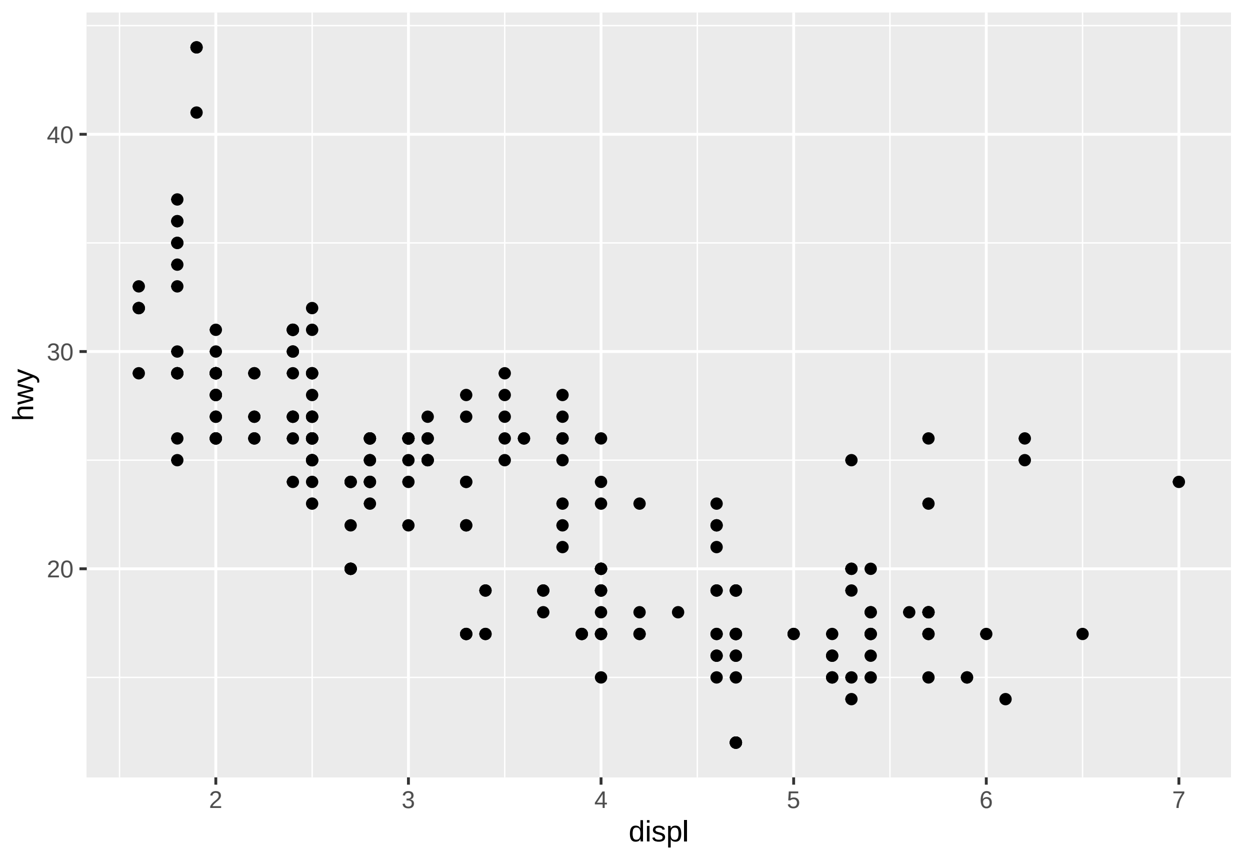
E3. What parameters to `geom_jitter()` control the amount of jittering?

Ans. `width` and `height`. These two controls the amount of vertical and horizontal jitter.

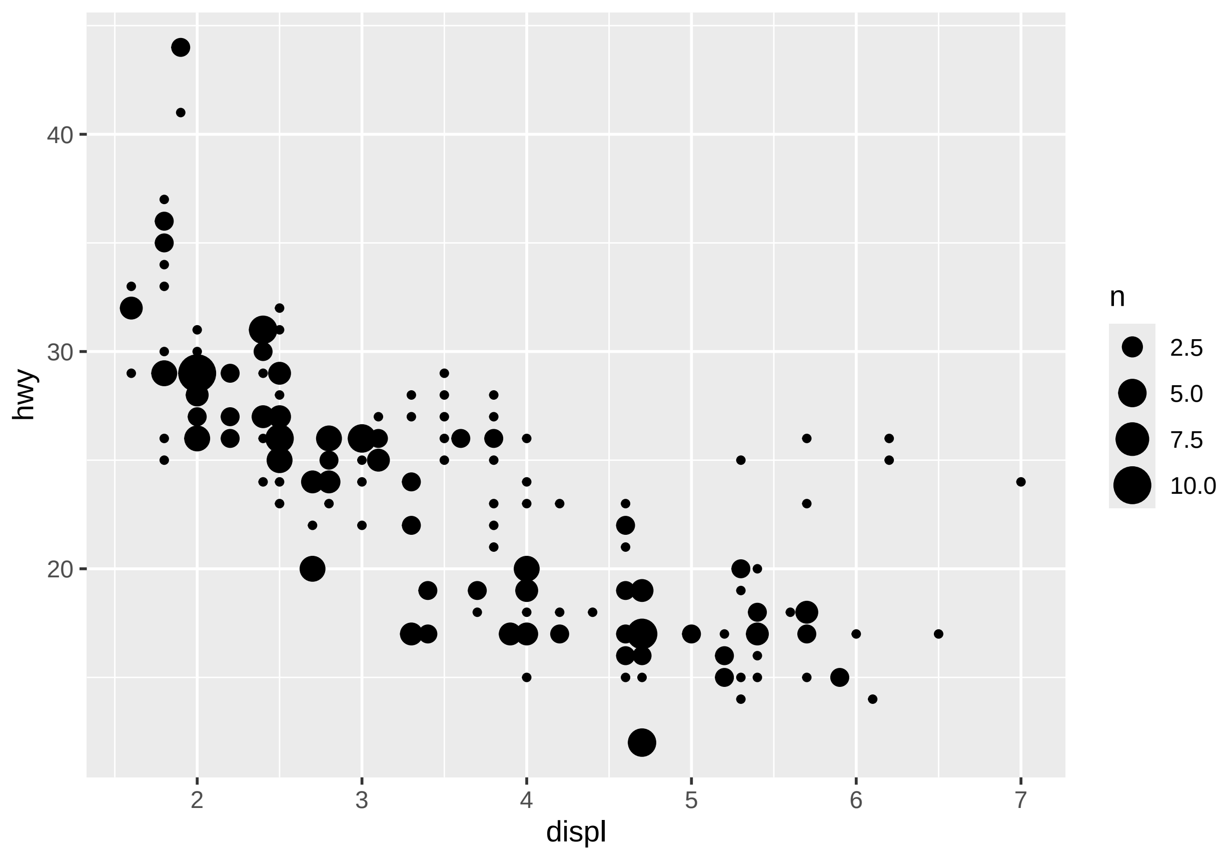
$4. Compare and contrast `geom_jitter()` with `geom_count()`.

Ans.

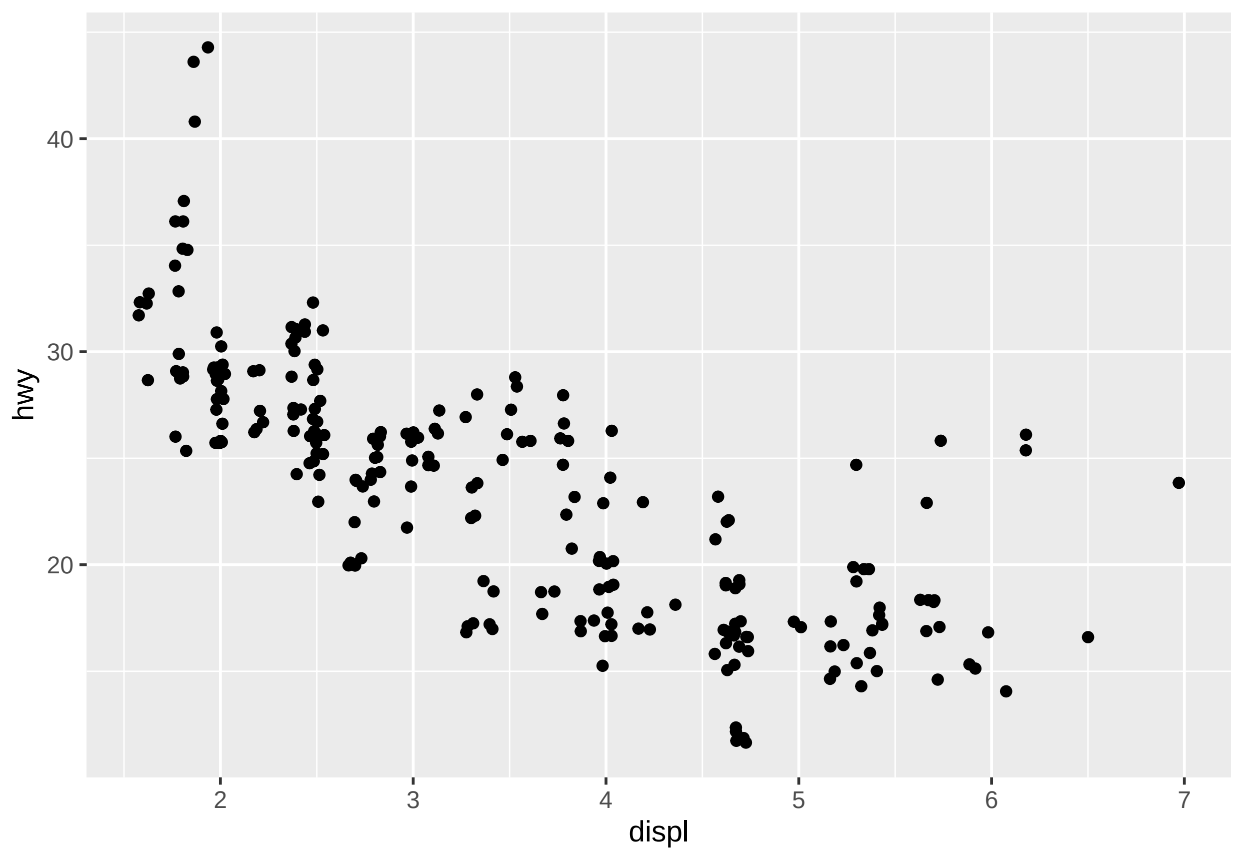
```
# normal
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point()
```



```
# with geom_count()
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_count()
```



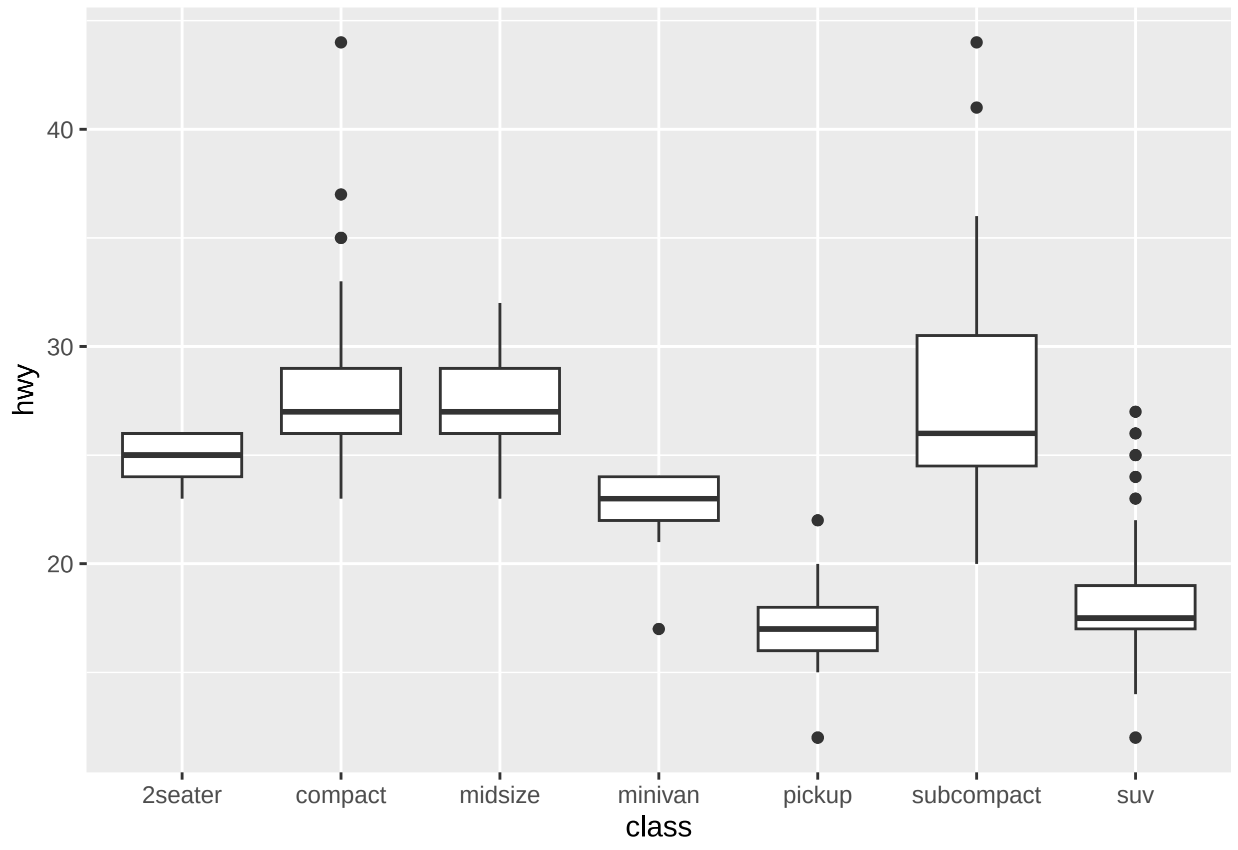
```r
# with geom_jitter()
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_jitter()
```



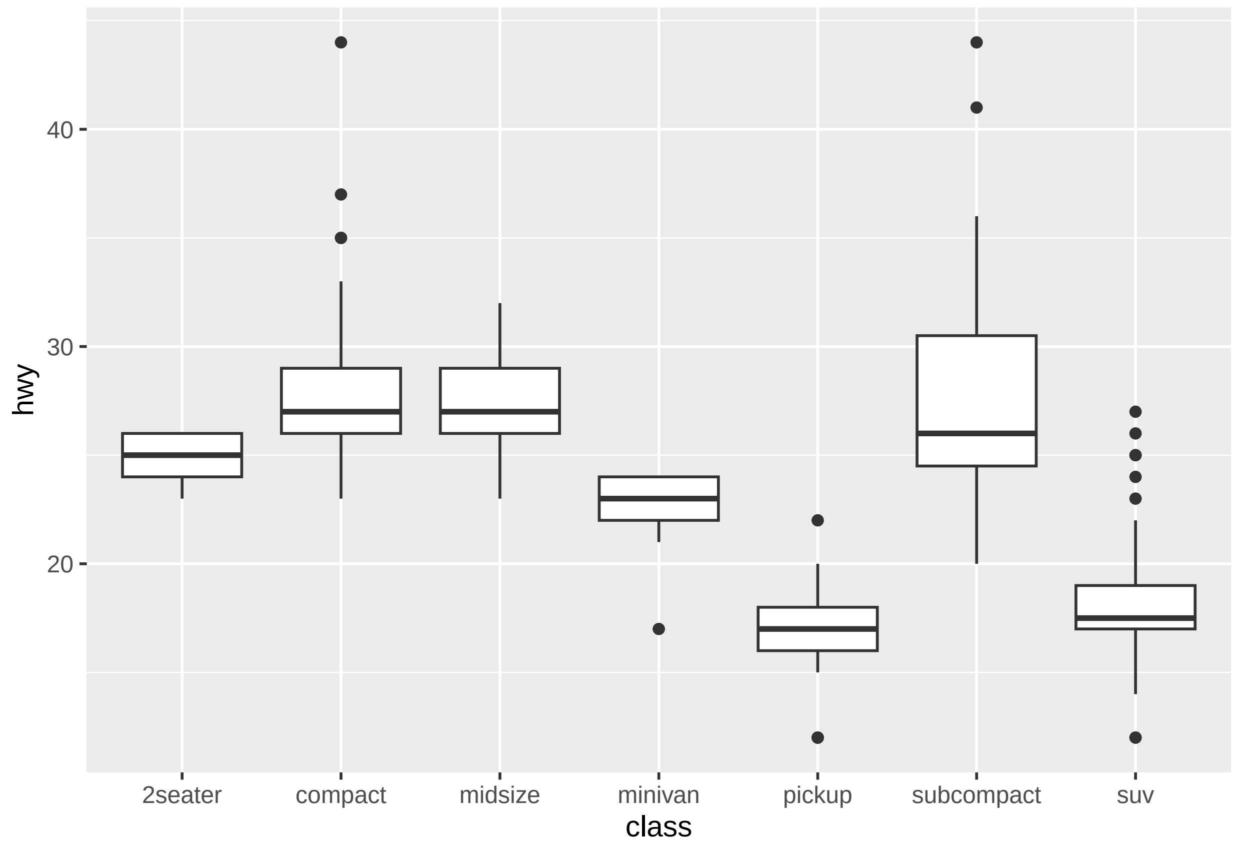
Both are used to counter overplotting. `geom_jitter()` does that by adding a small amount of random noise to slightly alter the position of each point on the plot. `geom_count()` maps the size of the dots to the number of observations at each location.

E5. What is the default position adjustment for `geom_boxplot()`? Create a visualization of the `mpg` dataset that demonstrates it.

```
# using default position argument
ggplot(mpg, aes(x = class, y = hwy)) +
  geom_boxplot()
```



```r
# using position = "dodge2"
ggplot(mpg, aes(x = class, y = hwy)) +
  geom_boxplot(position = "dodge2")
```



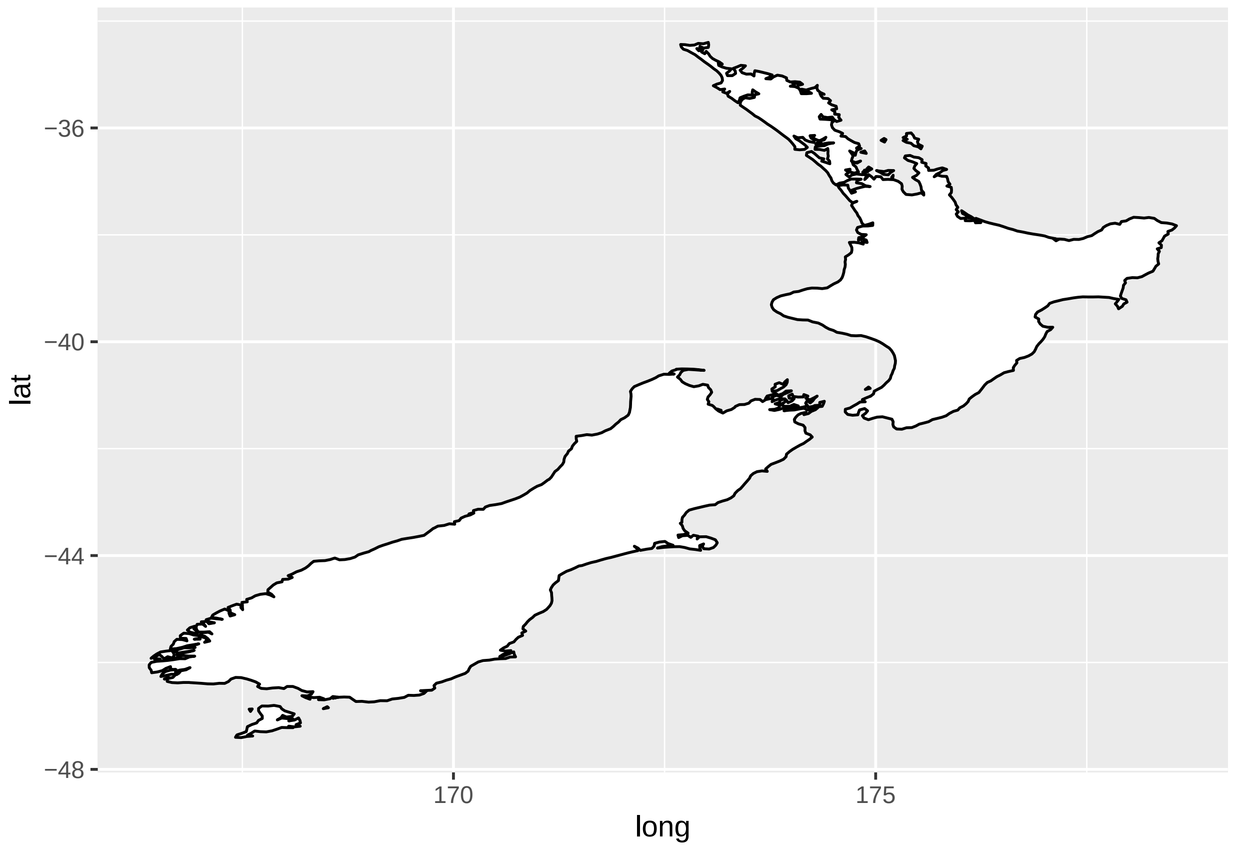
Coordinate systems

- Default coordinate system is the Cartesian coordinate system.

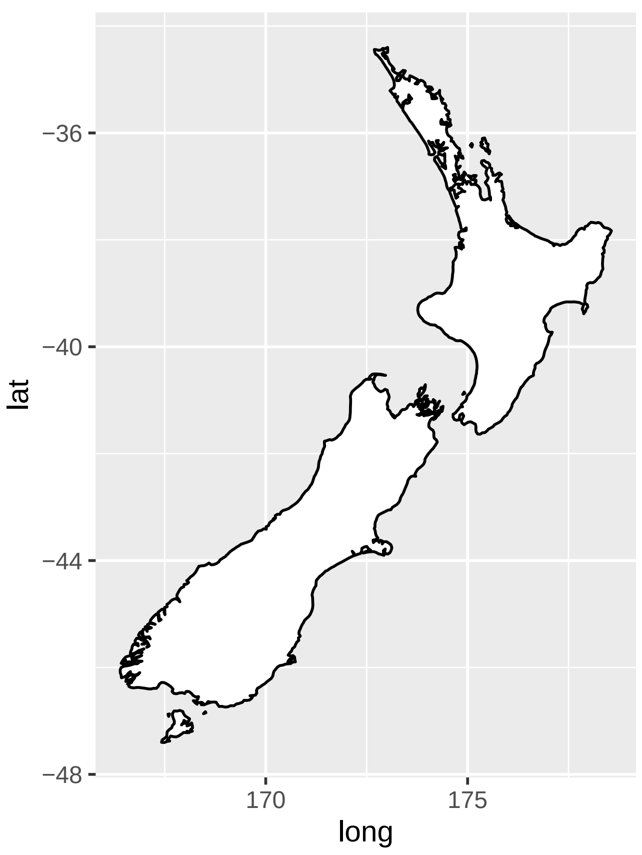
Mapping example using `coord_quickmap()`

```
nz <- map_data("nz")

ggplot(nz, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill = "white", color = "black")
```



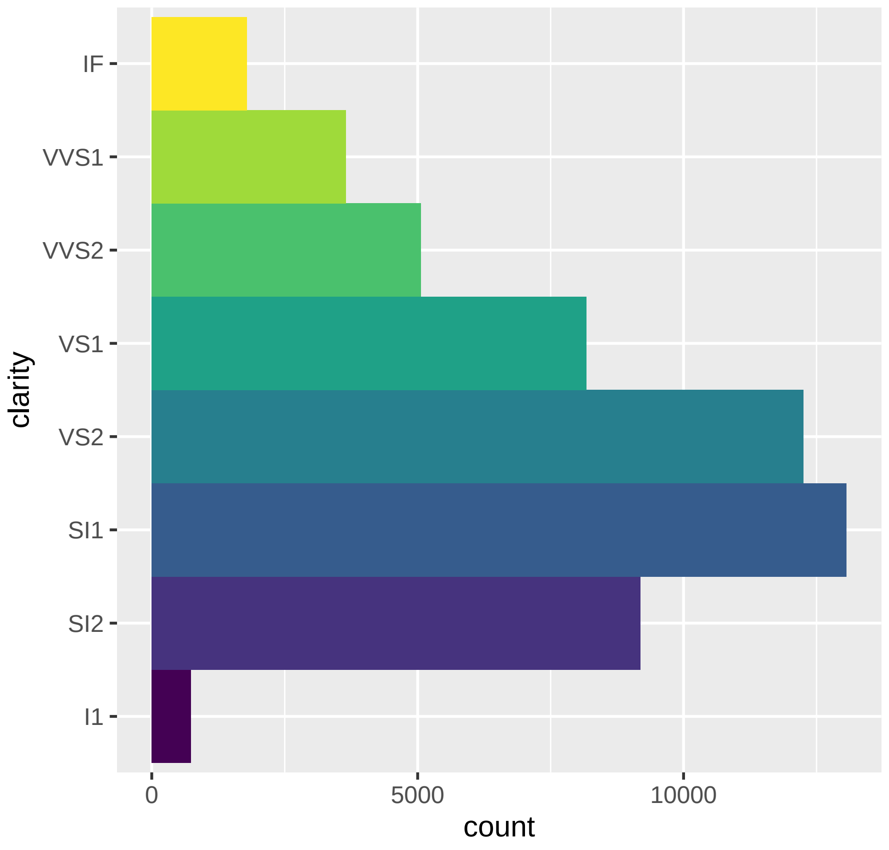
```
ggplot(nz, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill = "white", color = "black") +
  coord_quickmap()
```



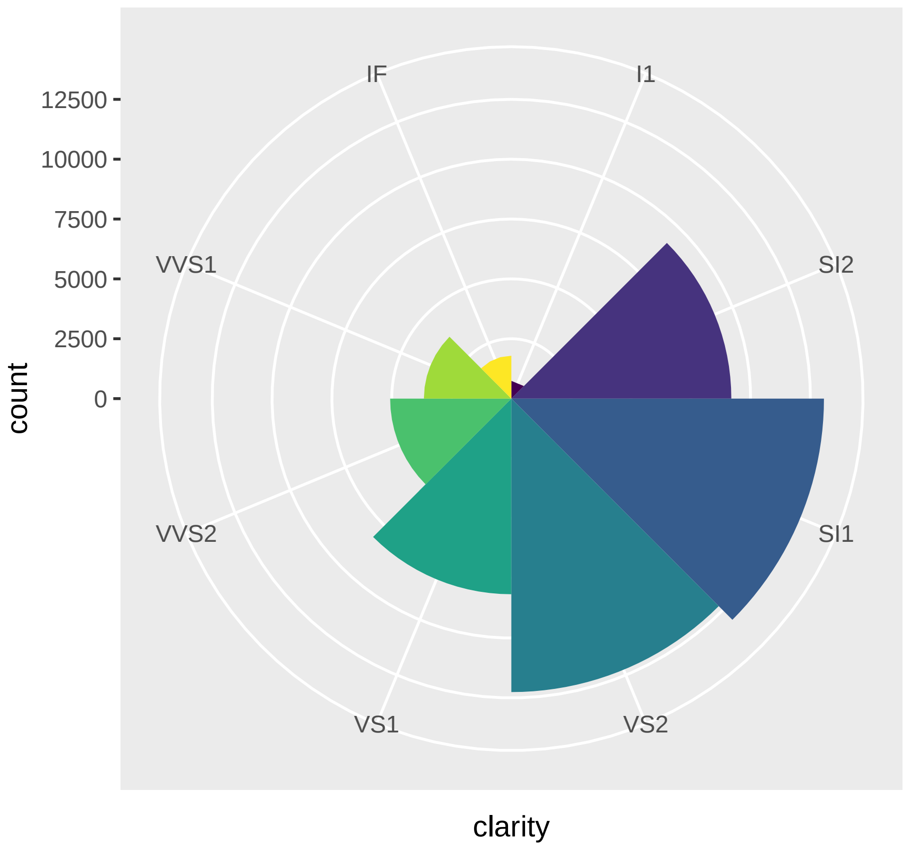
Using `coord_polar()` to make a Coxcomb chart:

```
bar <- ggplot(data = diamonds) +
  geom_bar(
    mapping = aes(x = clarity, fill = clarity),
    show.legend = FALSE,
    width = 1
  ) +
  theme(aspect.ratio = 1)

bar + coord_flip()
```



```
bar + coord_polar()
```



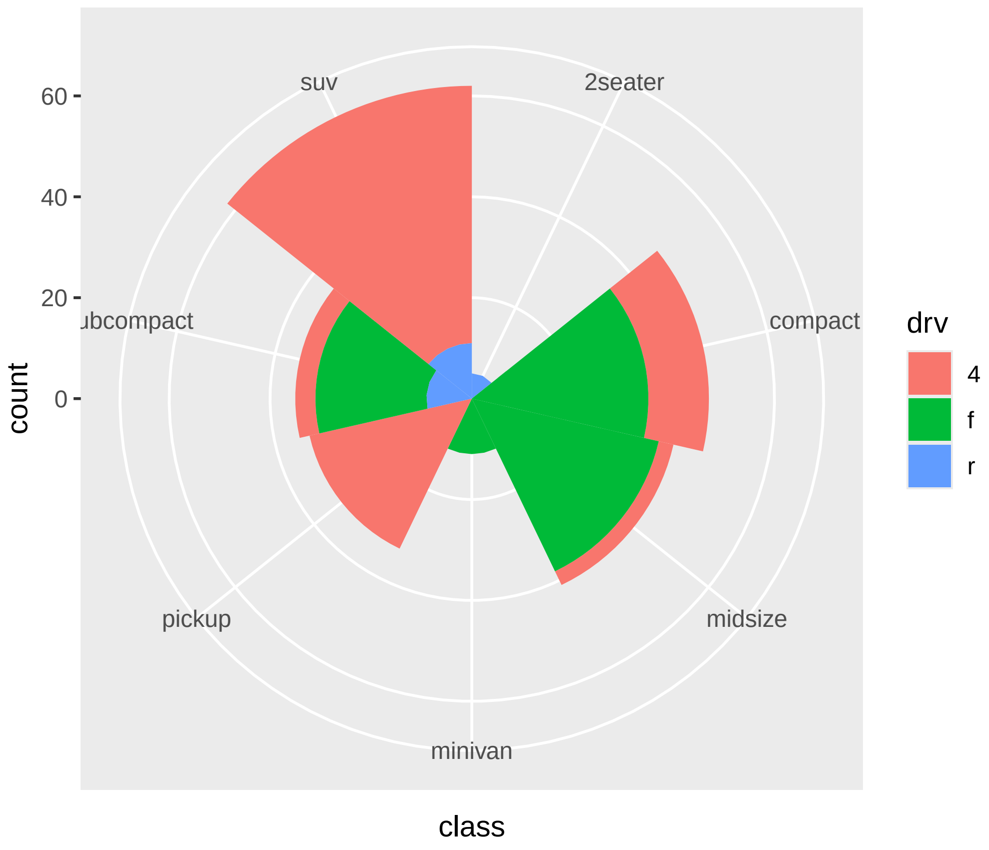
Exercises

E1. Turn a stacked bar chart into a pie chart using `coord_polar()`.

Ans:

```r
ggplot(mpg) +
  geom_bar(
    aes(x = class, fill = drv),
    width = 1
  ) + coord_flip() +
  coord_polar()
```

```
## Coordinate system already present. Adding new coordinate system, which will
## replace the existing one.
```

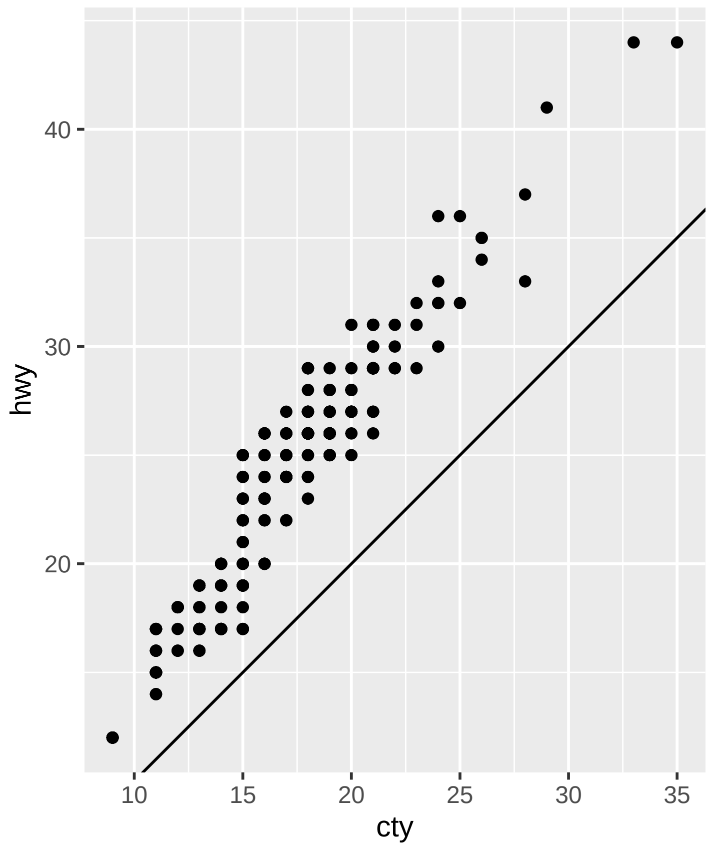


E2. What's the difference between `coord_quickmap()` and `coord_map()`?

Ans. `coord_map()` uses a mapping projection to convert a 3d surface of the earth into a 2d map. `coord_quickmap()` is a quick approximation of `coord_map()`. It works well for areas closer to the equator.

E3. What does the following plot tell you about the relationship between city and highway mpg? Why is `coord_fixed()` important? What does `geom_abline()` do?

```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_point() +
  geom_abline() +
  coord_fixed()
```



Ans. Cars are a bit more efficient on highways than on city streets. `geom_abline()` is used to draw straight lines on the plot. `coord_fixed()` is used to preserve the aspect ratio of the plot (in this case 1:1)

Layered grammar of graphics

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
    mapping = aes(<MAPPINGS>),
    stat = <STAT>,
    position = <POSITION>
) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION>
```