03-Data-Visualisation

Contents

3.1 Prerequisites

```
> ## load all tidyverse packages
> library(tidyverse)
> ## this bring the first time in my installation the following message:
> #> Loading tidyverse: ggplot2
> #> Loading tidyverse: tibble
> #> Loading tidyverse: tidyr
> #> Loading tidyverse: readr
> #> Loading tidyverse: purrr
> #> Loading tidyverse: dplyr
> #> Conflicts with tidy packages -----------------------------------------------------------------
> #> filter(): dplyr, stats
> #> lag():    dplyr, stats
> #
> ## but generally it loads the following packages:
> tidyverse_packages(include_self = TRUE)
```

```
 [1] "broom"     "dplyr"     "feather"   "forcats"   "ggplot2"
 [6] "haven"     "httr"      "hms"       "jsonlite"  "lubridate"
[11] "magrittr"  "modelr"    "purrr"     "readr"     "readxl"
[16] "stringr"   "tibble"    "rvest"     "tidyr"     "xml2"
[21] "tidyverse"
```

```
> ## for more information about the rationale behind `tidyverse` read:
> ## https://cran.r-project.org/web/packages/tidyverse/vignettes/manifesto.html
> ## if you need updates of all these packages, run:
> # tidyverse_update()
```

3.2 First steps

3.2.1 The mpg data frame

```
> # inspect the `mpg`data frame
> mpg # show the complete data frame
```

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

```
> my.mpg <- mpg # copy to a local variable (to inspect in RStudio)
> class(mpg)
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

```
> head(mpg)
```

```
# A tibble: 6 × 11
  manufacturer model displ  year   cyl      trans   drv   cty   hwy    fl
         <chr> <chr> <dbl> <int> <int>      <chr> <chr> <int> <int> <chr>
```

```
1         audi    a4   1.8  1999     4   auto(l5)     f    18    29     p
2         audi    a4   1.8  1999     4 manual(m5)     f    21    29     p
3         audi    a4   2.0  2008     4 manual(m6)     f    20    31     p
4         audi    a4   2.0  2008     4   auto(av)     f    21    30     p
5         audi    a4   2.8  1999     6   auto(l5)     f    16    26     p
6         audi    a4   2.8  1999     6 manual(m5)     f    18    26     p
# ... with 1 more variables: class <chr>
```

```
> tail(mpg)
```

```
# A tibble: 6 × 11
  manufacturer  model displ  year   cyl      trans   drv   cty   hwy    fl
         <chr>  <chr> <dbl> <int> <int>      <chr> <chr> <int> <int> <chr>
1   volkswagen passat   1.8  1999     4   auto(l5)     f    18    29     p
2   volkswagen passat   2.0  2008     4   auto(s6)     f    19    28     p
3   volkswagen passat   2.0  2008     4 manual(m6)     f    21    29     p
4   volkswagen passat   2.8  1999     6   auto(l5)     f    16    26     p
5   volkswagen passat   2.8  1999     6 manual(m5)     f    18    26     p
6   volkswagen passat   3.6  2008     6   auto(s6)     f    17    26     p
# ... with 1 more variables: class <chr>
```

```
> str(mpg)
```

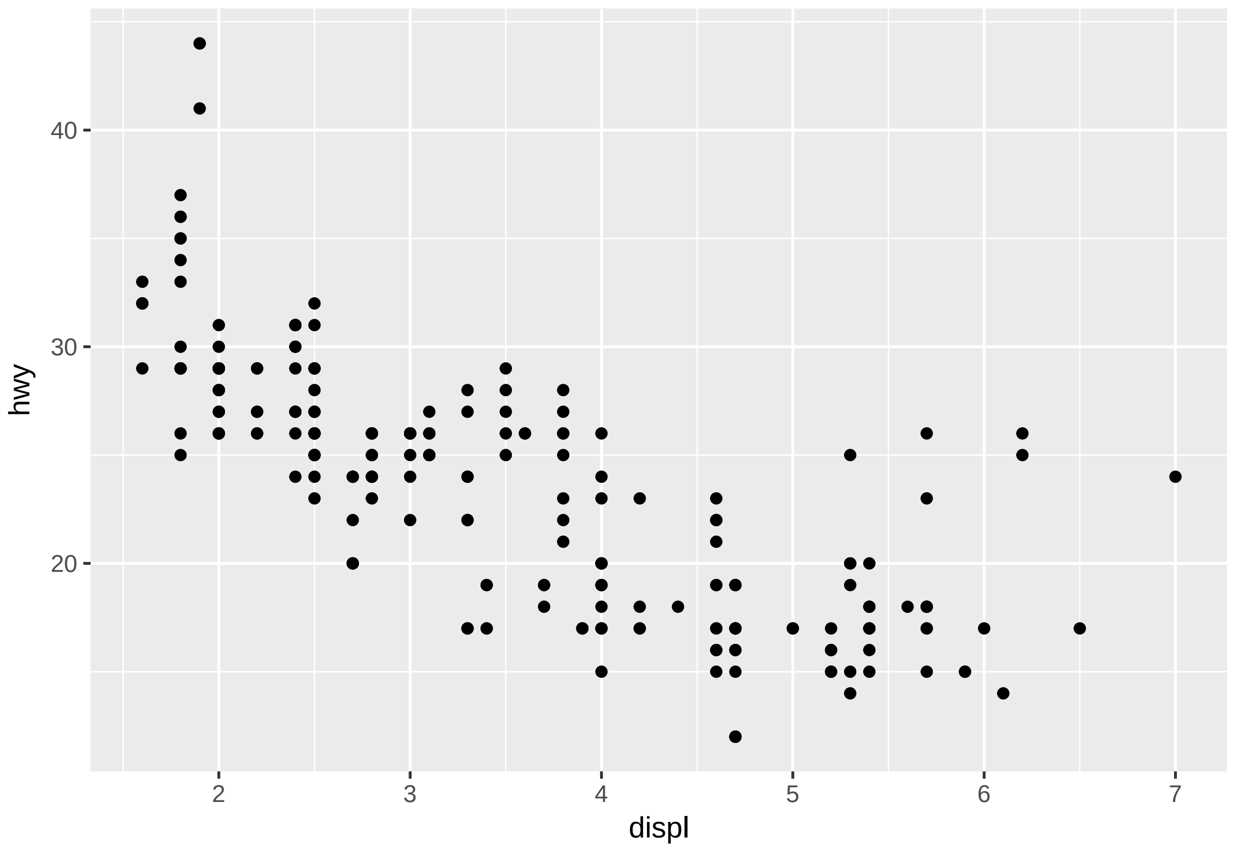
```
Classes 'tbl_df', 'tbl' and 'data.frame':   234 obs. of  11 variables:
 $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
 $ model       : chr  "a4" "a4" "a4" "a4" ...
 $ displ       : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
 $ year        : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
 $ cyl         : int  4 4 4 4 6 6 6 4 4 4 ...
 $ trans       : chr  "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
 $ drv         : chr  "f" "f" "f" "f" ...
 $ cty         : int  18 21 20 21 16 18 18 18 16 20 ...
 $ hwy         : int  29 29 31 30 26 26 27 26 25 28 ...
 $ fl          : chr  "p" "p" "p" "p" ...
 $ class       : chr  "compact" "compact" "compact" "compact" ...
```

```
> head(as.factor(my.mpg$class))
```

```
[1] compact compact compact compact compact compact
Levels: 2seater compact midsize minivan pickup subcompact suv
```

3.2.2 Creating a ggplot

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



You can make a template, where you just have to substitute the expressions with brackets with your data.

3.2.3 A graphic template

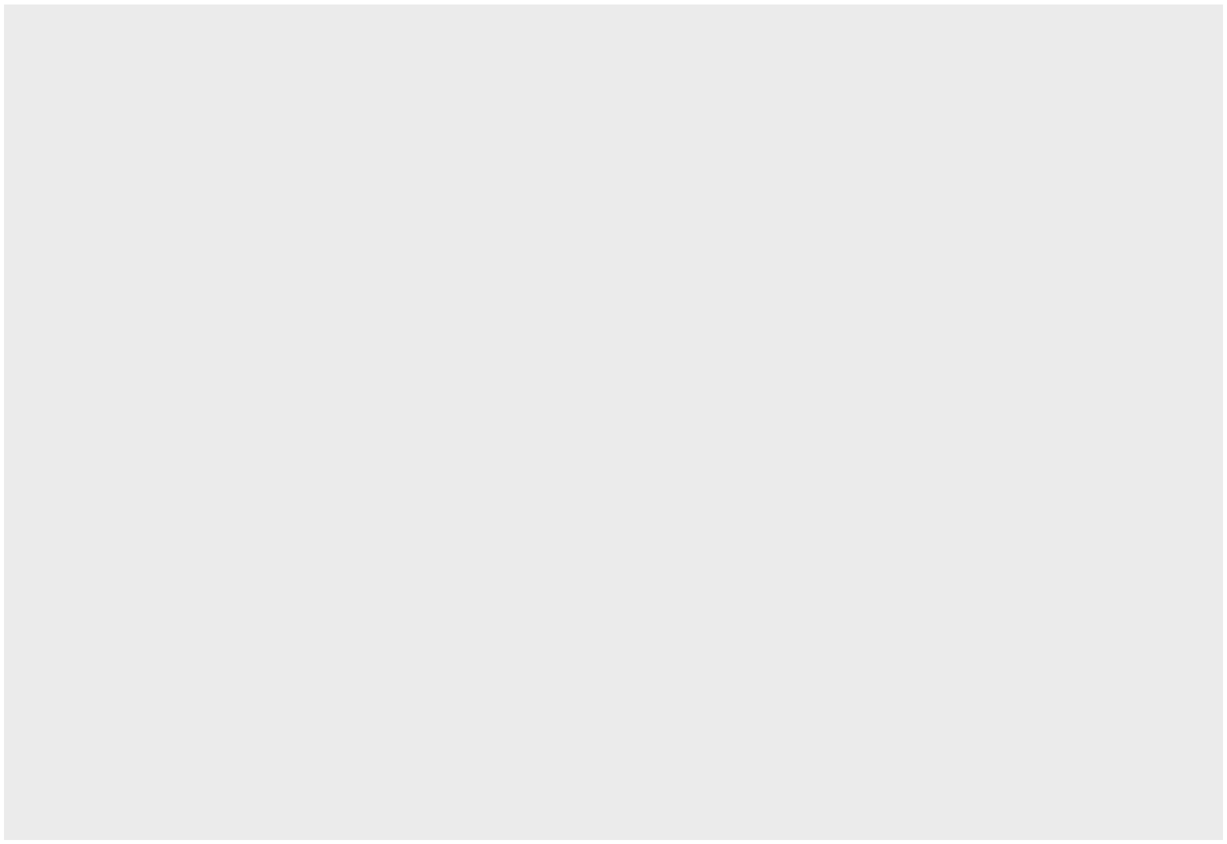
Summarizing the essential parameters using ggplot we get the following template:

```
ggplot(data = <DATA>) +
        <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

3.2.4 Exercises

(1) Run `ggplot(data = mpg)` – what do you see?

```
> ggplot(data = mpg) # an empty graphic frame
```

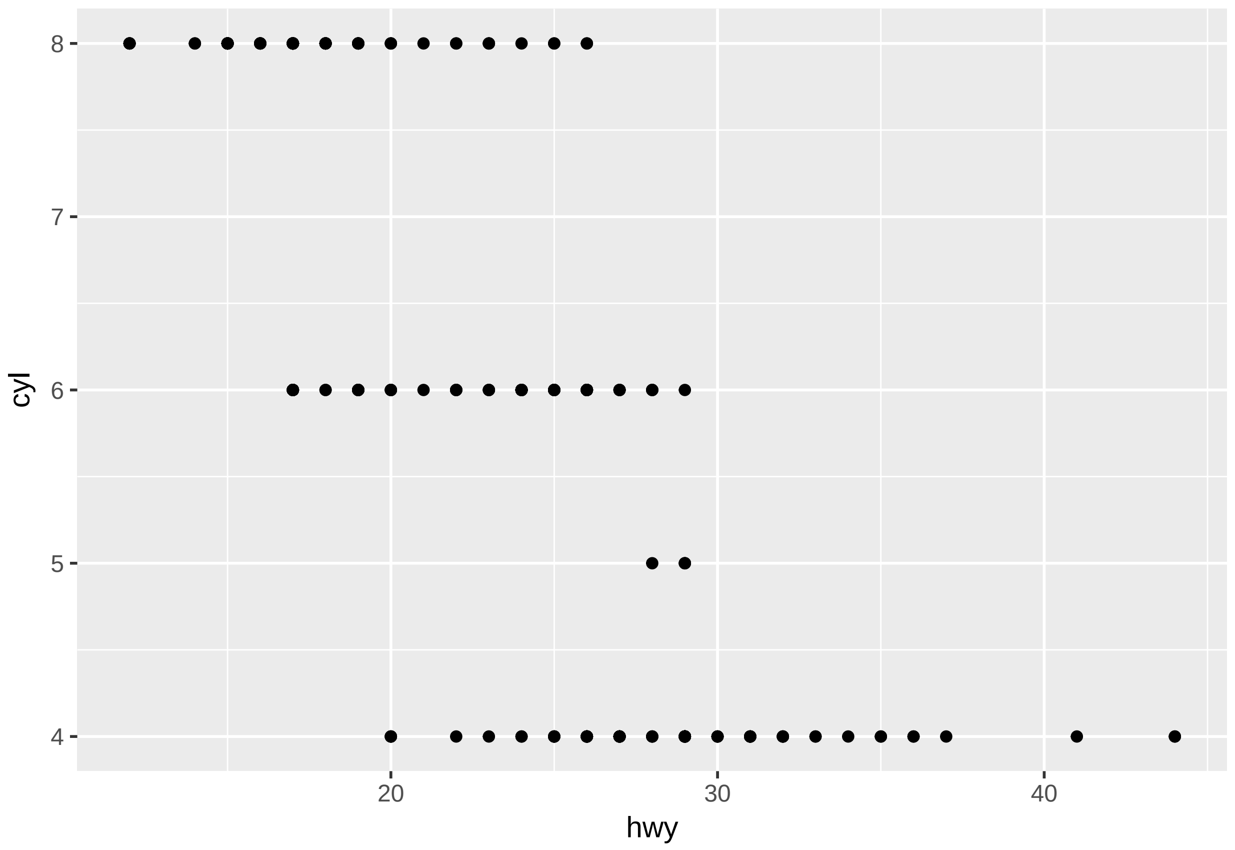


(2) How many rows are in mtcars? How many columns? 234 rows and 11 columns.

(3) What does the drv variable describe? Read the help for ?mpg to find out. f = front-wheel drive, r = rear wheel drive, 4 = 4wd

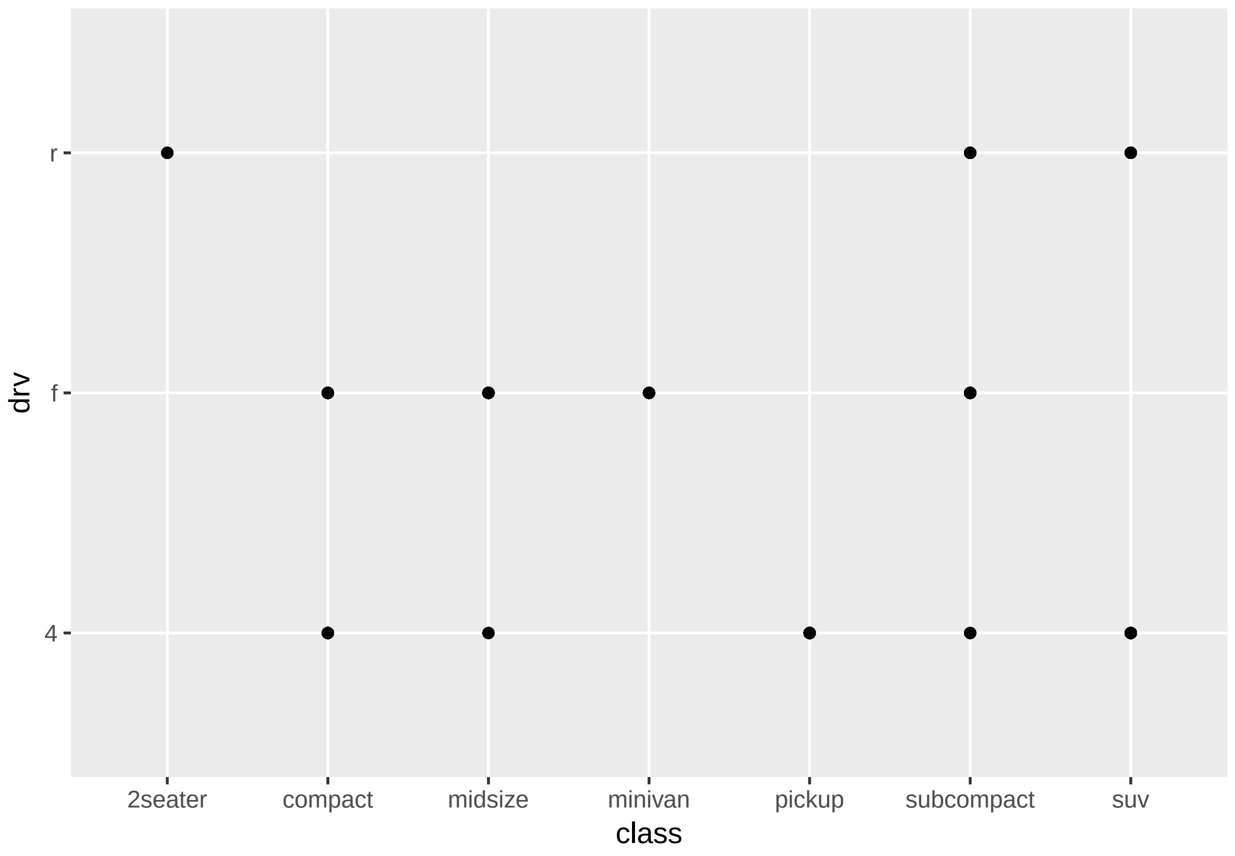
(4) Make a scatterplot of hwy vs cyl.

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



(5) What happens if you make a scatterplot of class vs drv. Why is the plot not useful?

```
> ggplot(data = mpg) +
+           geom_point(mapping = aes(x = class, y = drv))
```



This plot is not very useful, because both axis are discrete variable. One cannot see how many occurences one data point represent, e.g. for instance there are 35 compact cars with f (front wheel drive)

```
> nrow(filter(mpg, class == "compact", drv == "f"))
```

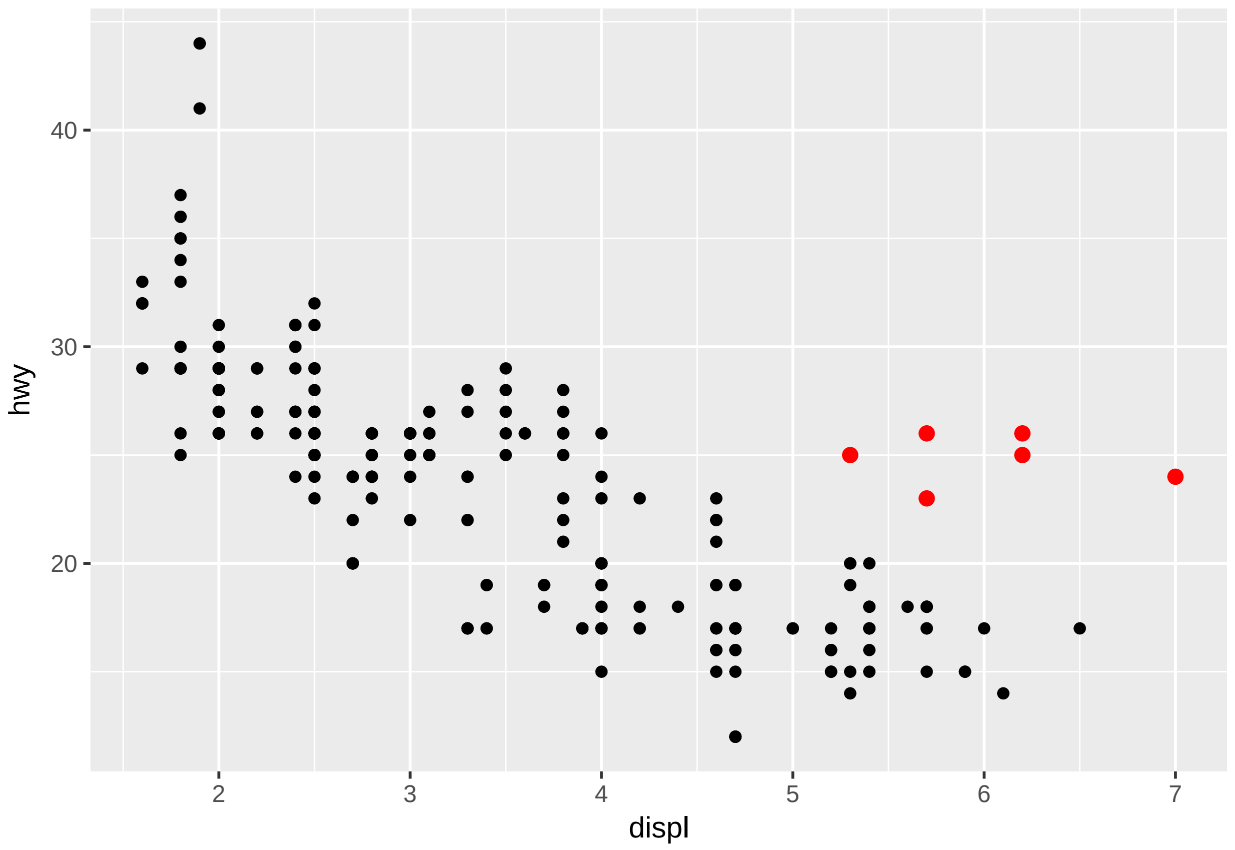
```
[1] 35
```

3.3 Aesthetic mappings

3.3.1 Aeshetic overlapping

The graph in section 3.2.2 Creating a ggplot display some points which do not confirm the general (negative) linear trend. I am refering to the red points in the following graph:

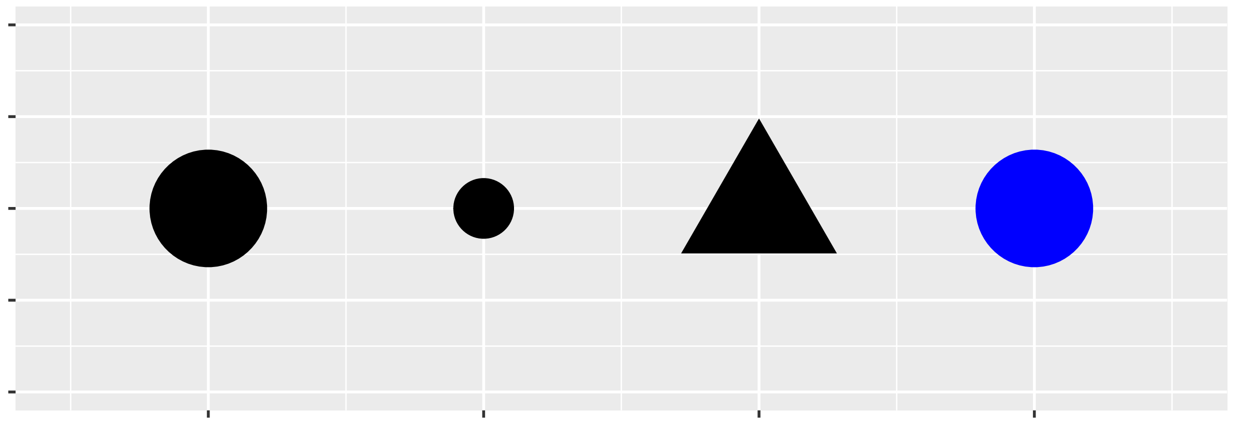
```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
+         geom_point() +
+         geom_point(data = dplyr::filter(mpg, displ > 5, hwy > 20),
+                    colour = "red", size = 2.2)
```



3.3.2 Changing aesthetic properties

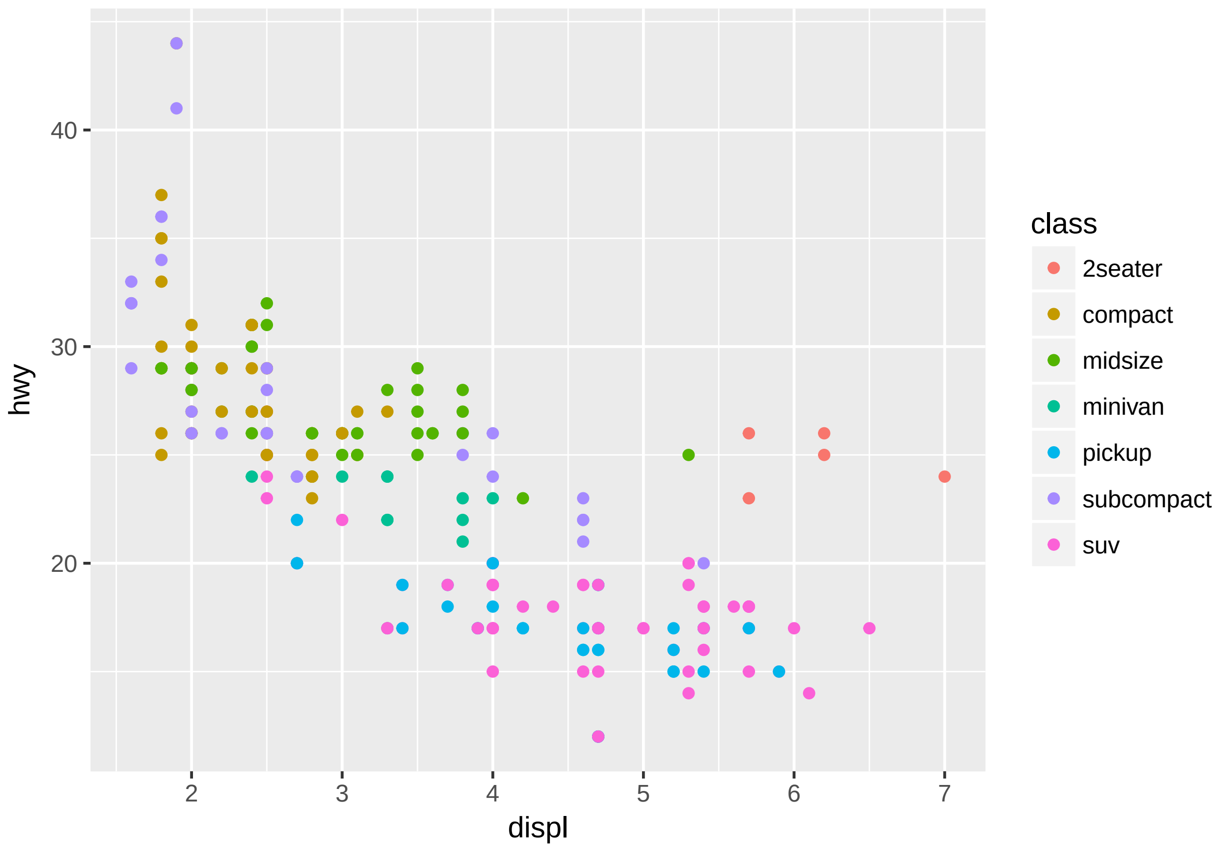
You can display a point (like the one below) in different ways by changing the values of its aesthetic properties.

```
> ggplot() +
+   geom_point(aes(1, 1), size = 20) +
+   geom_point(aes(2, 1), size = 10) +
+   geom_point(aes(3, 1), size = 20, shape = 17) +
+   geom_point(aes(4, 1), size = 20, colour = "blue") +
+   scale_x_continuous(NULL, limits = c(0.5, 4.5), labels = NULL) +
+   scale_y_continuous(NULL, limits = c(0.9, 1.1), labels = NULL) +
+   theme(aspect.ratio = 1/3)
```



3.3.3 Mapping aesthetics as (additional, e.g.third) variable You can convey information about your data by mapping the aesthetics in your plot to the variables in your dataset. For example, you can map the colors of your points to the class variable to reveal the class of each car.

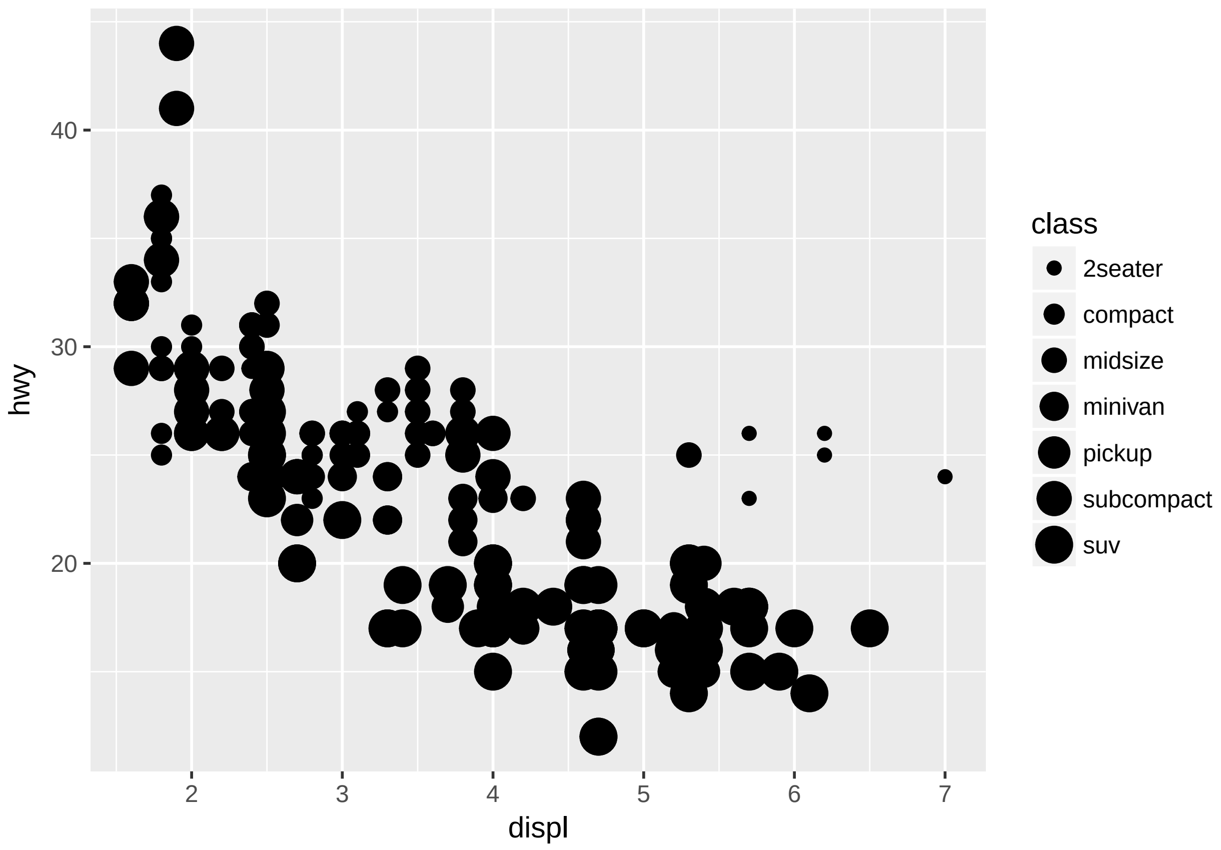
```
> ggplot(data = mpg) +
+          geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



In the above example, we mapped class to the color aesthetic, but we could have mapped class to the size aesthetic in the same way. In this case, the exact size of each point would reveal its class affiliation. We get a warning here, because mapping an unordered variable (class) to an ordered aesthetic (size) is not a good idea.

3.3.4 Mapping unordered to ordered variable = warning

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

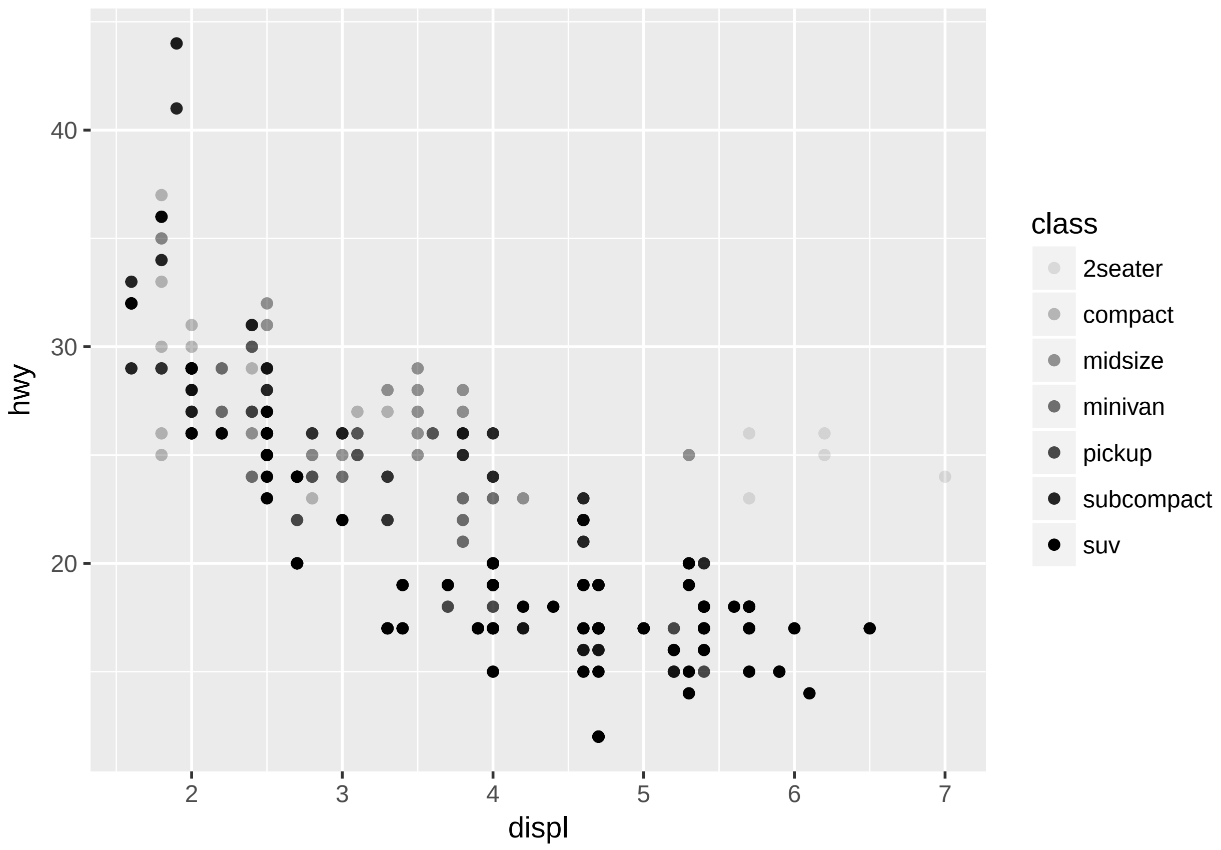


```
> #> Warning: Using size for a discrete variable is not advised.
```

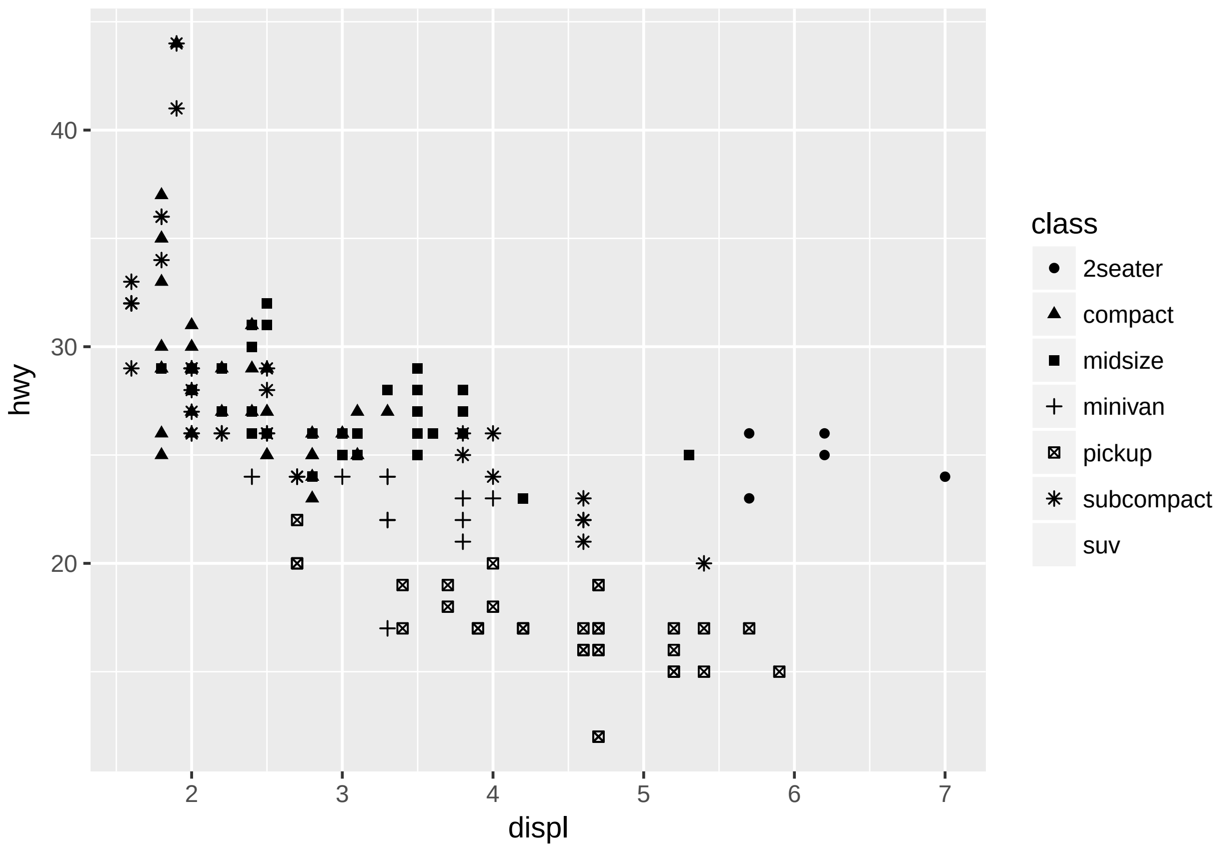
3.3.5 Other examples of mapping a third variable

Or we could have mapped class to the alpha aesthetic, which controls the transparency of the points, or the shape of the points.

```
> # Left
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy, alpha = class))
```



```
> # Right
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy, shape = class))
```

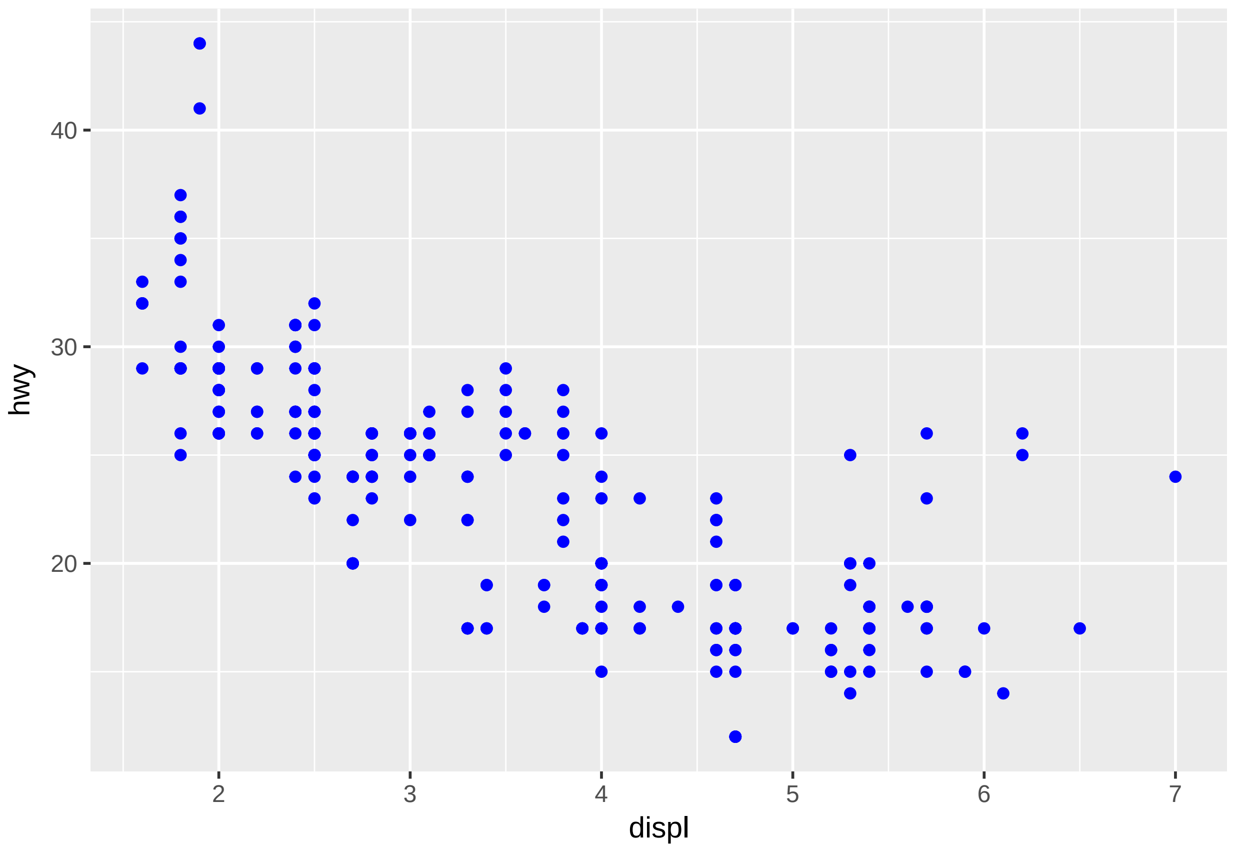


```
> # Warning:
> # The shape palette can deal with a maximum of 6 discrete values
> # because more than 6 becomes difficult to discriminate; you have 7.
> # Consider specifying shapes manually if you must have them.
> # Removed 62 rows containing missing values (geom_point)
```

3.3.6 Seting aesthetic properties manually

Youn can also set the aesthetic properties of your geom manually, e.g. plotting the graph not with black (= default) but with blue points. Here, the color doesn't convey information about a variable, but only changes the appearance of the plot.

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



To set an aesthetic manually, set the aesthetic by name as an argument of your geom function; i.e. it goes outside of `aes()`.

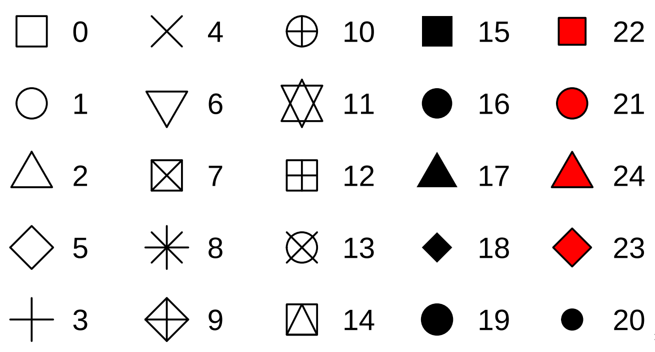
You'll need to pick a value that makes sense for that aesthetic:

- The name of a color as a character string.
- The size of a point in mm.
- The shape of a point as a number.

3.3.7 Different forms to plot the aesthetic

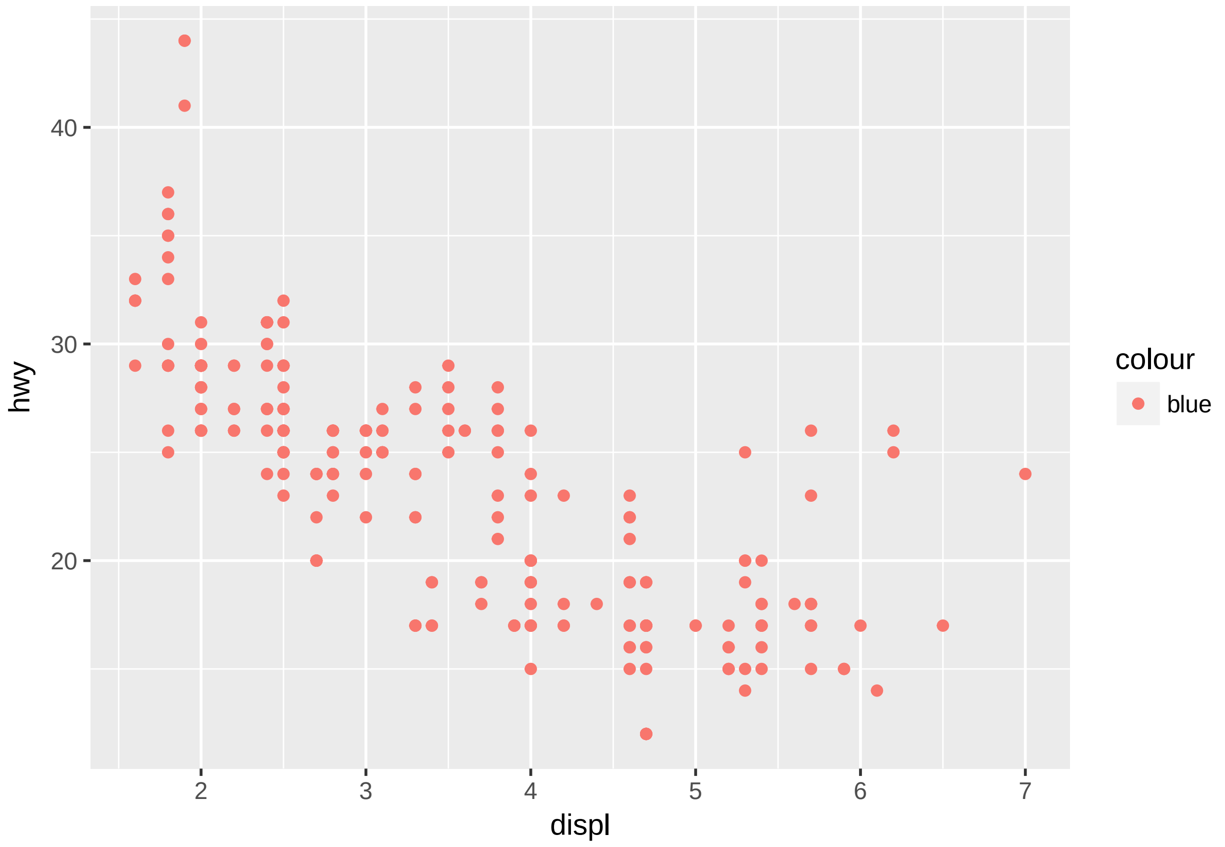
You can choose of many different forms to plot the aesthetic:

```
> shapes <- tibble(
+   shape = c(0, 1, 2, 5, 3, 4, 6:19, 22, 21, 24, 23, 20),
+   x = (0:24 %/% 5) / 2,
+   y = (-(0:24 %% 5)) / 4
+ )
> ggplot(shapes, aes(x, y)) +
+   geom_point(aes(shape = shape), size = 5, fill = "red") +
+   geom_text(aes(label = shape), hjust = 0, nudge_x = 0.15) +
+   scale_shape_identity() +
+   expand_limits(x = 4.1) +
+   scale_x_continuous(NULL, breaks = NULL) +
+   scale_y_continuous(NULL, breaks = NULL, limits = c(-1.2, 0.2)) +
+   theme_minimal() +
+   theme(aspect.ratio = 1/2.75)
```



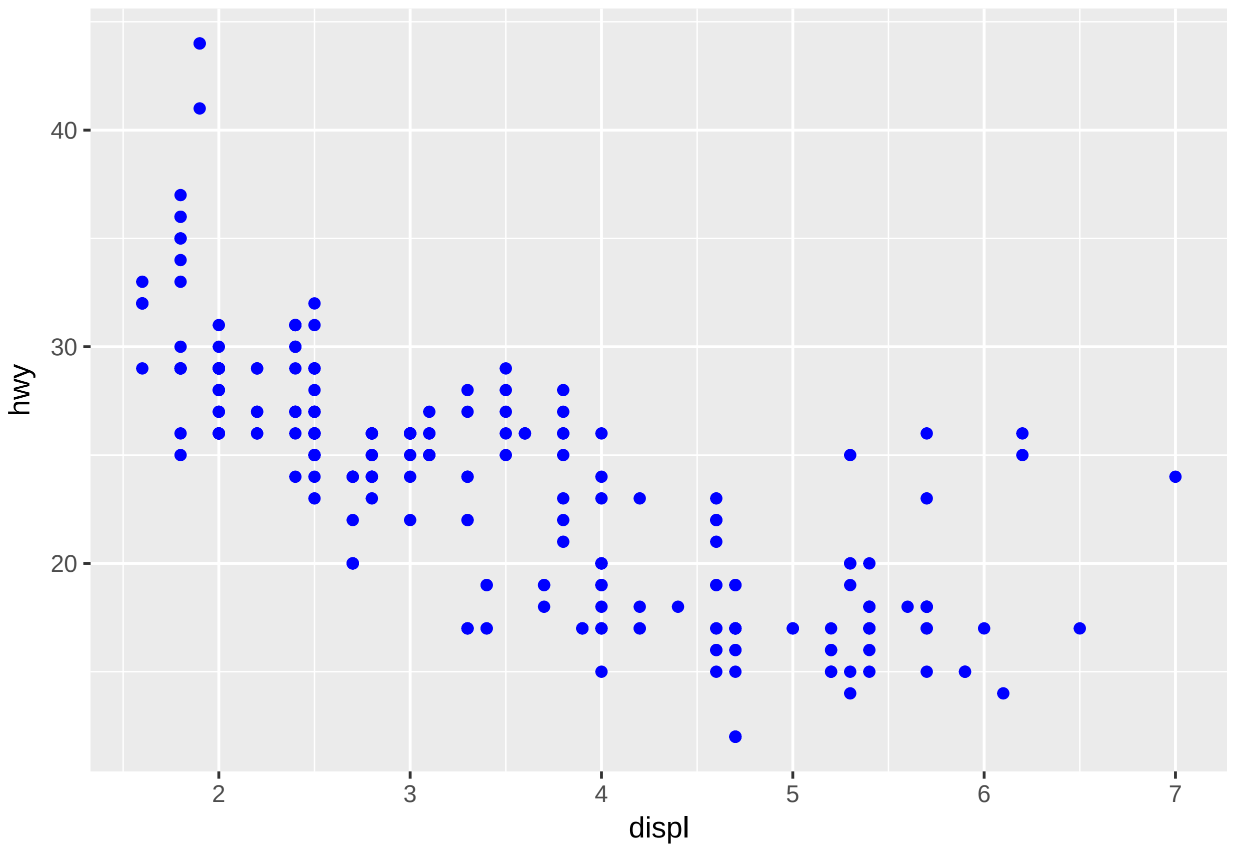
3.3.8 Exercises (1) What's gone wrong with this code? Why are the points not blue?

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



My Solution: The points aren't blue because the command to set aestethics manually has to be outside of the `aes()` function

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



(2) Which variables in `mpg` are categorical? Which variables are continuous? (Hint: type `?mpg` to read the documentation for the dataset). How can you see this information when you run `mpg`?

My Solution: All character variables are categorial. Integer and Numeric are continuous. You can see this information by calling the str() function: `str(mpg)`

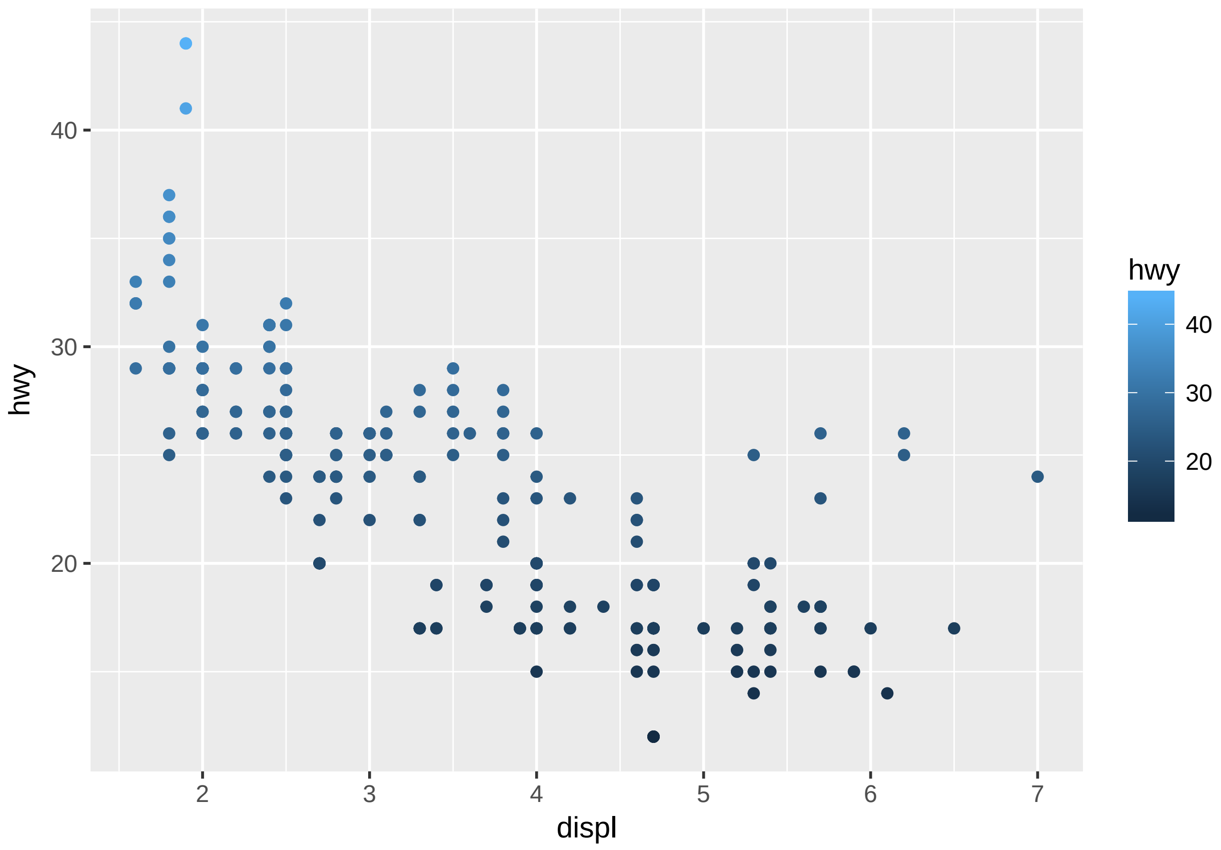
```
> str(mpg)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   234 obs. of  11 variables:
 $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
 $ model       : chr  "a4" "a4" "a4" "a4" ...
 $ displ       : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
 $ year        : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
 $ cyl         : int  4 4 4 4 6 6 6 4 4 4 ...
 $ trans       : chr  "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
 $ drv         : chr  "f" "f" "f" "f" ...
 $ cty         : int  18 21 20 21 16 18 18 18 16 20 ...
 $ hwy         : int  29 29 31 30 26 26 27 26 25 28 ...
 $ fl          : chr  "p" "p" "p" "p" ...
 $ class       : chr  "compact" "compact" "compact" "compact" ...
```

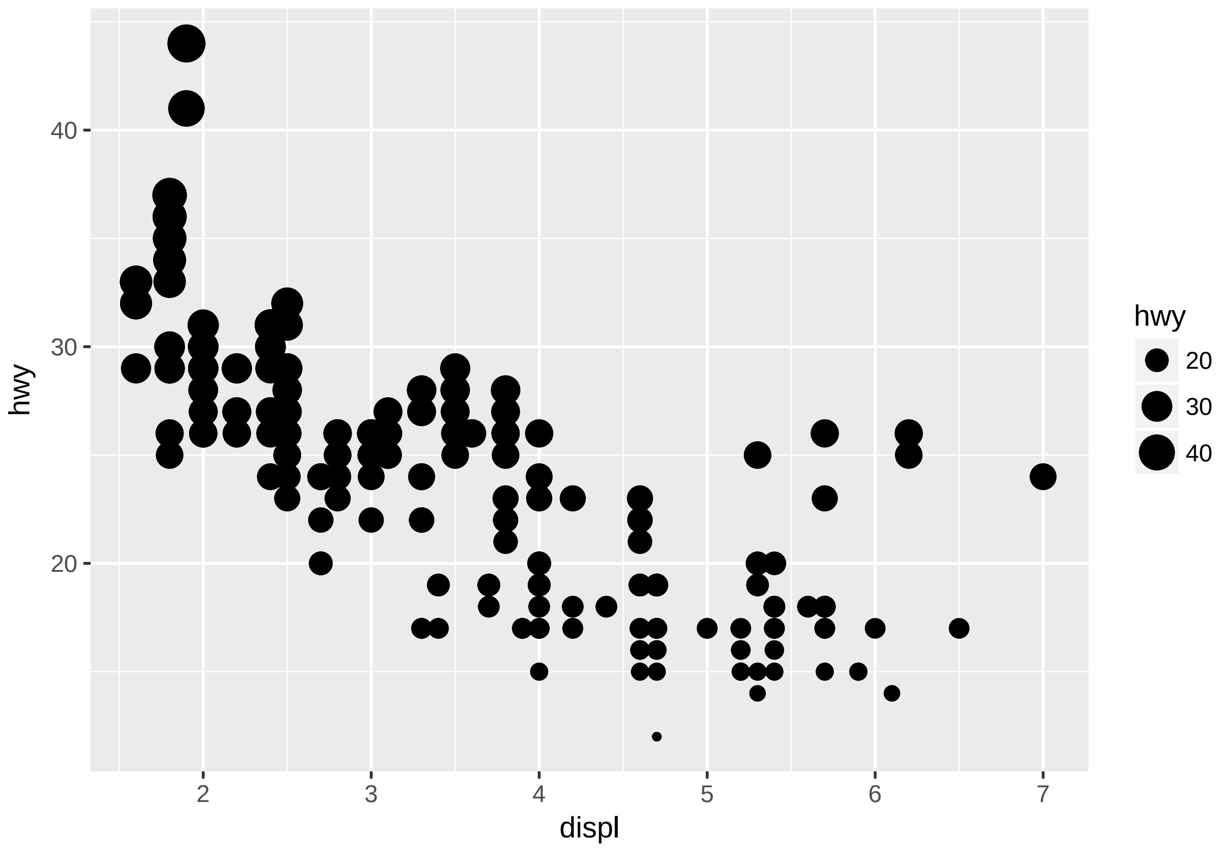
(3) Map a continuous variable to color, size, and shape. How do these aesthetics behave differently for categorical vs. continuous variables?

My Solution: For discrete variables aesthetics do not make much sense. Color behaves as a continiuos aesthetic, as different shades of one color (= blue) are taken. Detto size. But shape generates an error as it behaves as a discrete aesthetic. Summary: Use color or size for continuous and shape for discrete variables.

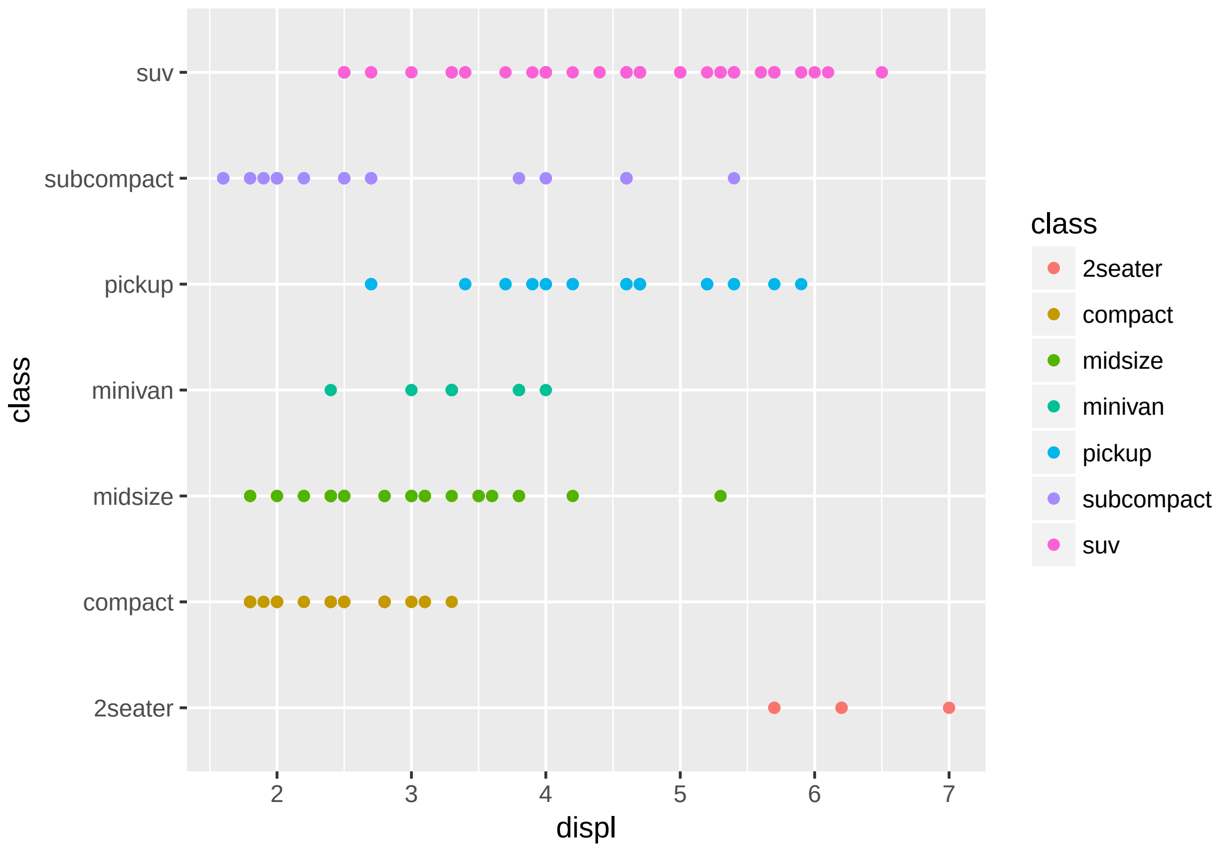
```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy, color = hwy))
```



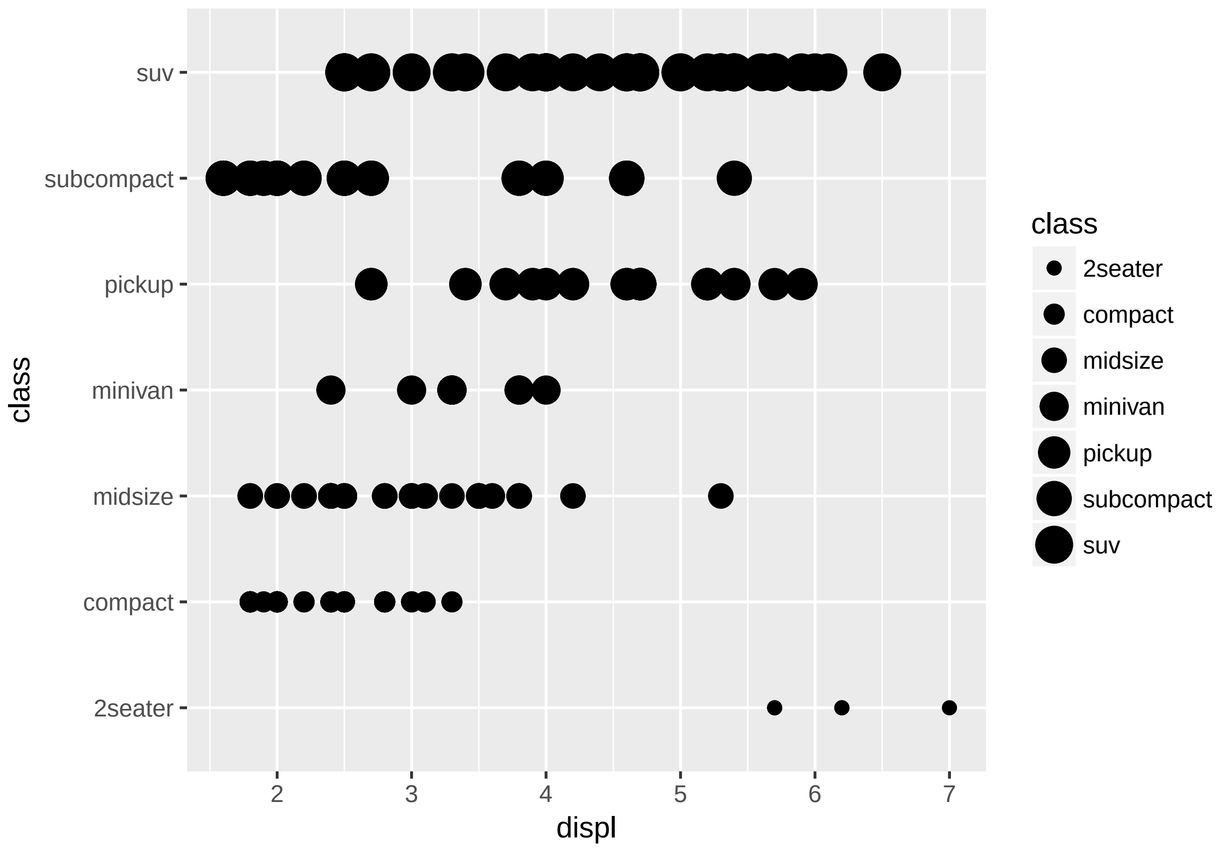
```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy, size = hwy))
```



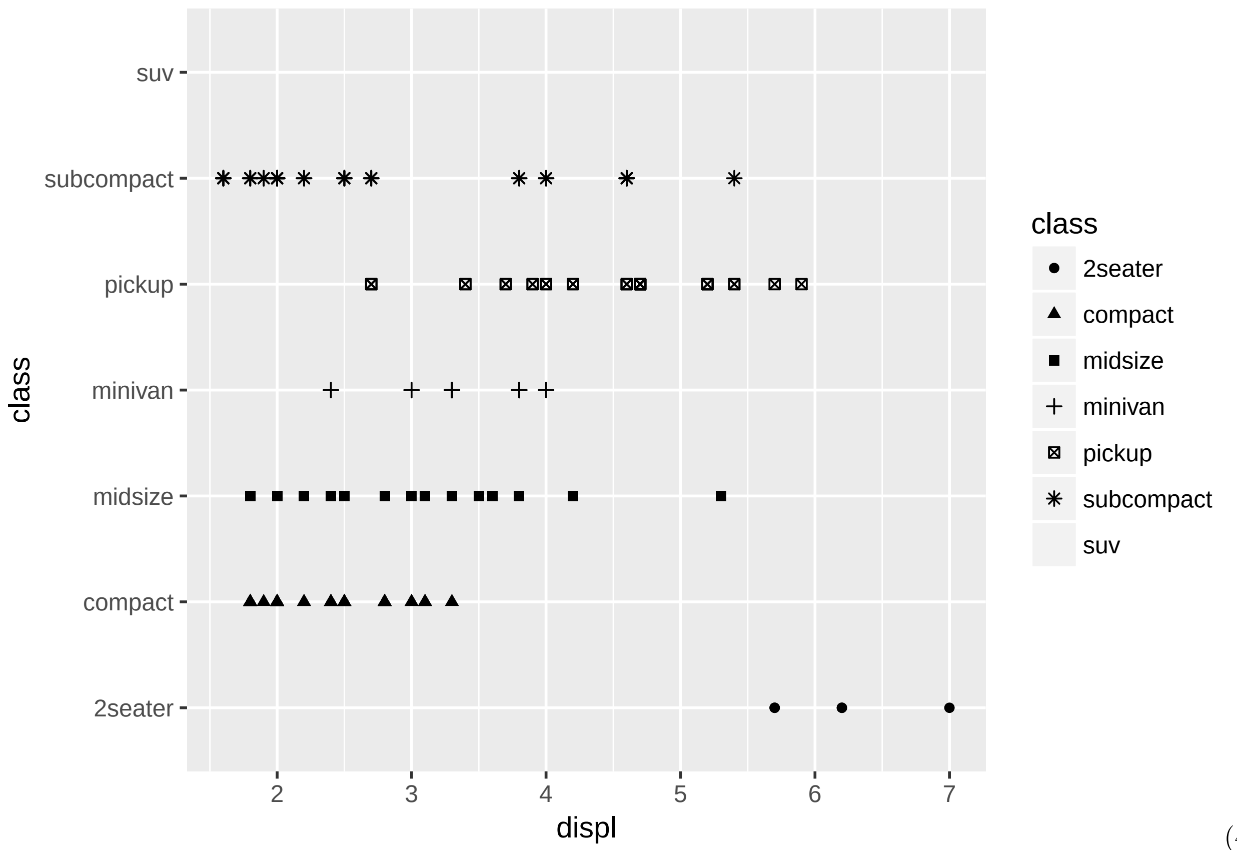
```
> ggplot(data = mpg) +
+ # ggplot(data = mpg) +
+ #         geom_point(mapping = aes(x = displ, y = hwy, shape = hwy))
+ # Error: A continuous variable can not be mapped to shape
+   geom_point(mapping = aes(x = displ, y = class, color = class))
```



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



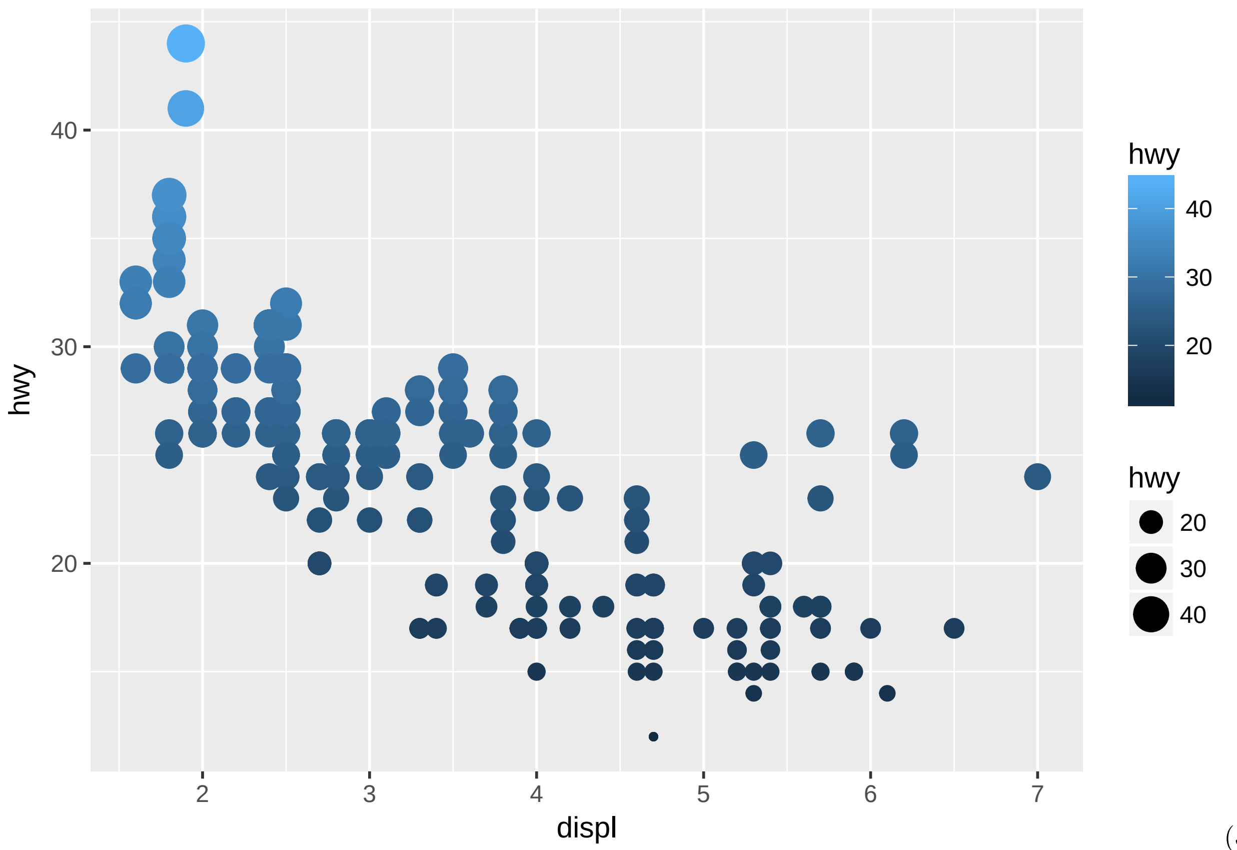
```
> # Warning:
> # Using size for a discrete variable is not advised.
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = class, shape = class))
```



(4)

What happens if you map the same variable to multiple aesthetics? **My Solution:** ggplot displays both aesthetics in the same graph.

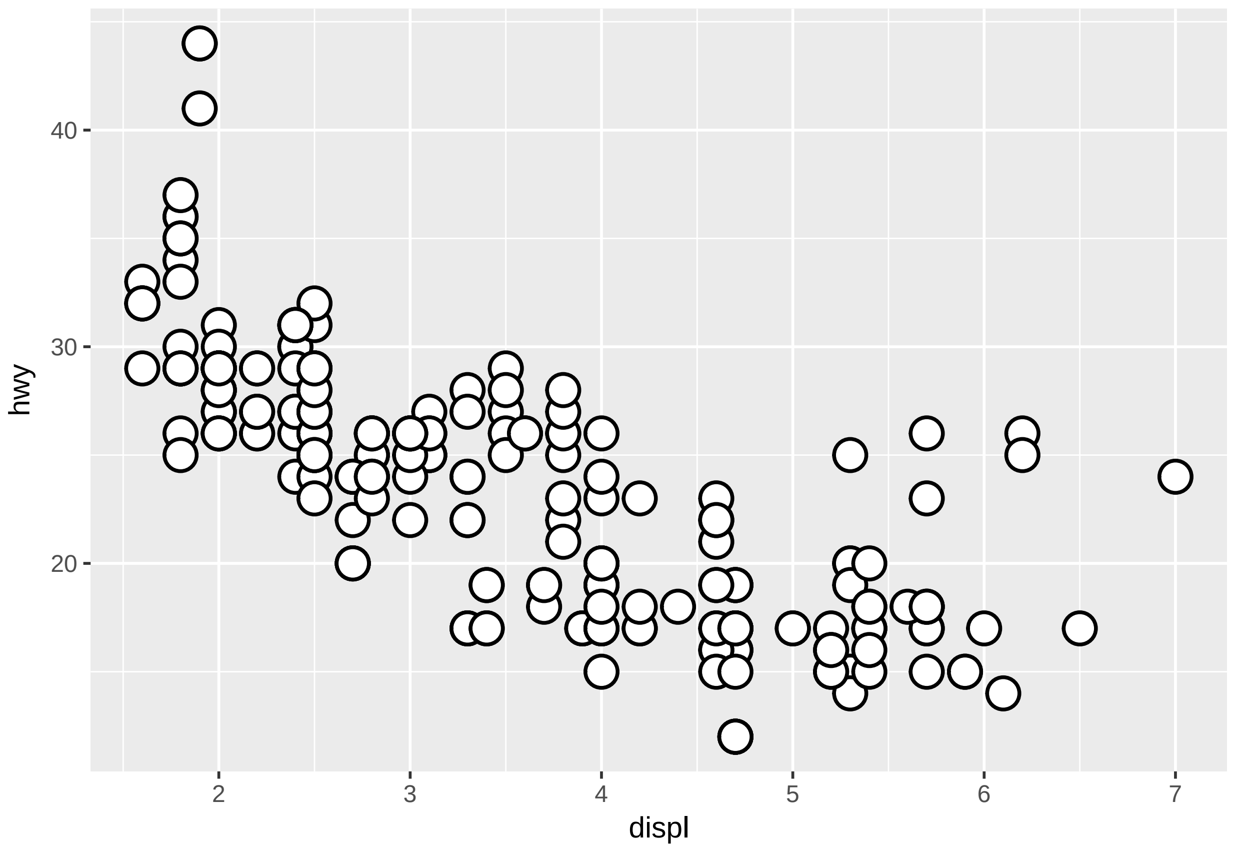
```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy, color = hwy, size = hwy))
```



(5)

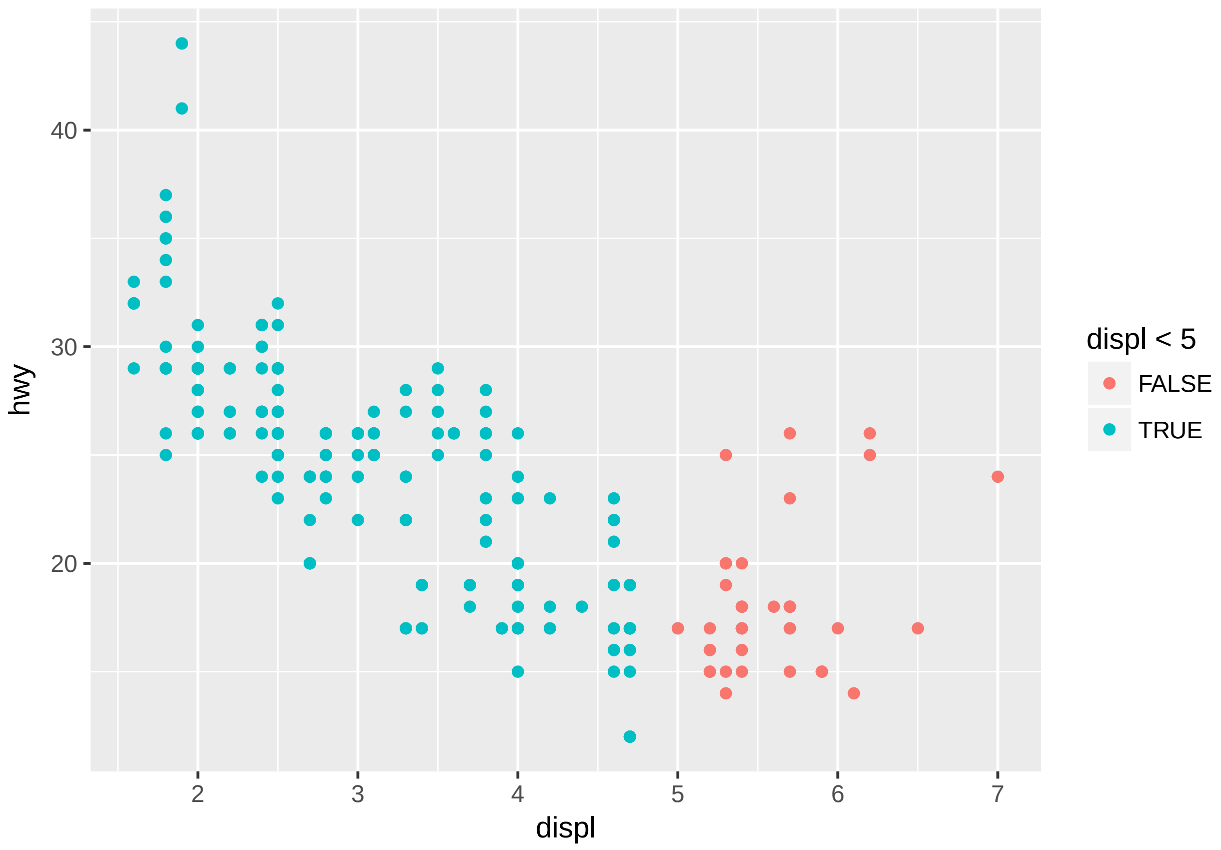
What does the stroke aesthetic do? What shapes does it work with? (Hint: use ?geom_point) **My Solution:** Stroke aesthetic is useful with shapes number 21,22,23,24,25 aesthetics which have a stroke and fill color. The size of the filled part is controlled by size, the size of the stroke is controlled by stroke. Each is measured in mm, and the total size of the point is the sum of the two.

```
> ggplot(mpg, aes(displ, hwy)) +
+   geom_point(shape = 21, color = "black", fill = "white", size = 5, stroke = 1)
```



(6) What happens if you map an aesthetic to something other than a variable name, like aes(color = displ < 5)? **My Solution:** ggplot evalaute the condition and uses the aesthetic to present the outcome.

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



#

3.4 Common problems This section discusses trouble-shooting and where to get help.

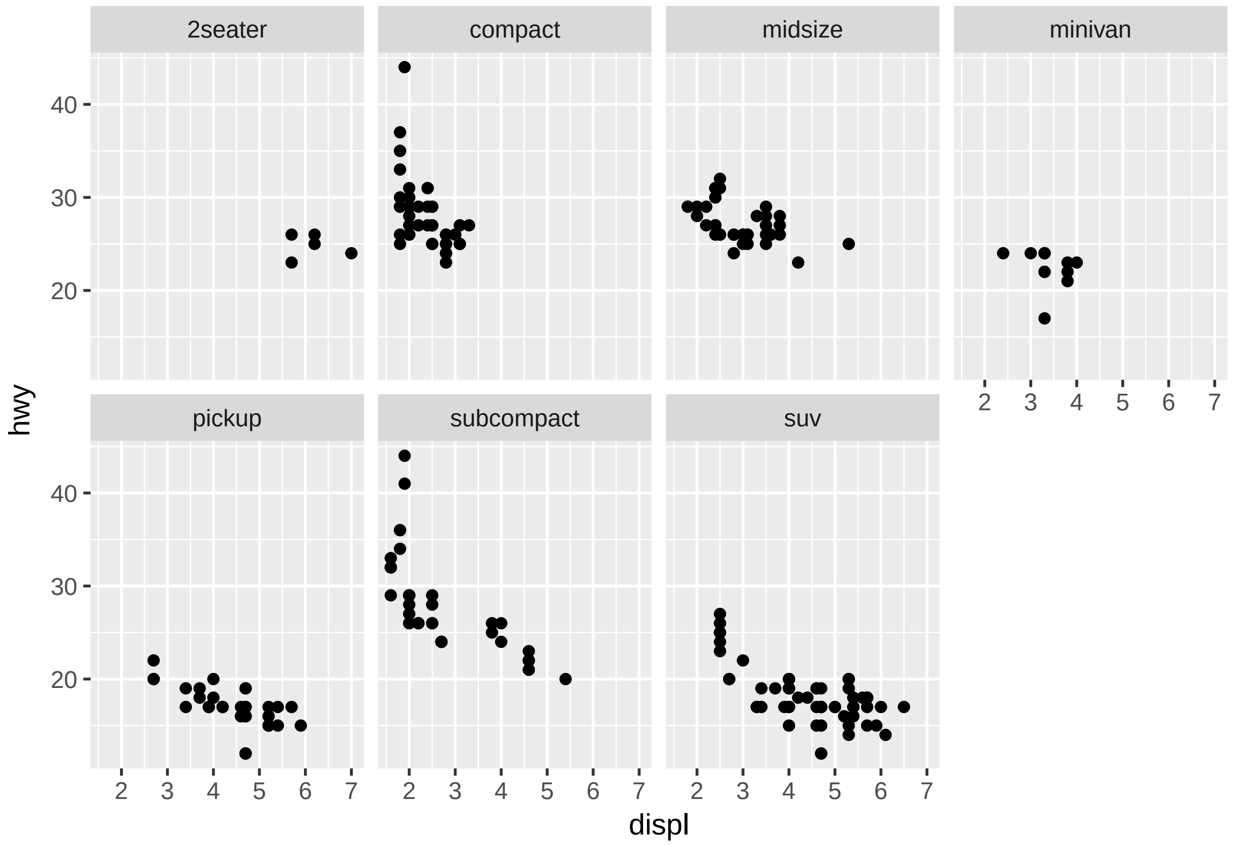
3.5 Facets

One way to add additional variables is with aesthetics. Another way, particularly useful for categorical variables, is to split your plot into facets, subplots that each display one subset of the data.

3.5.1 Facet by a single variable

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.

```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_wrap(~ class, nrow = 2)
```



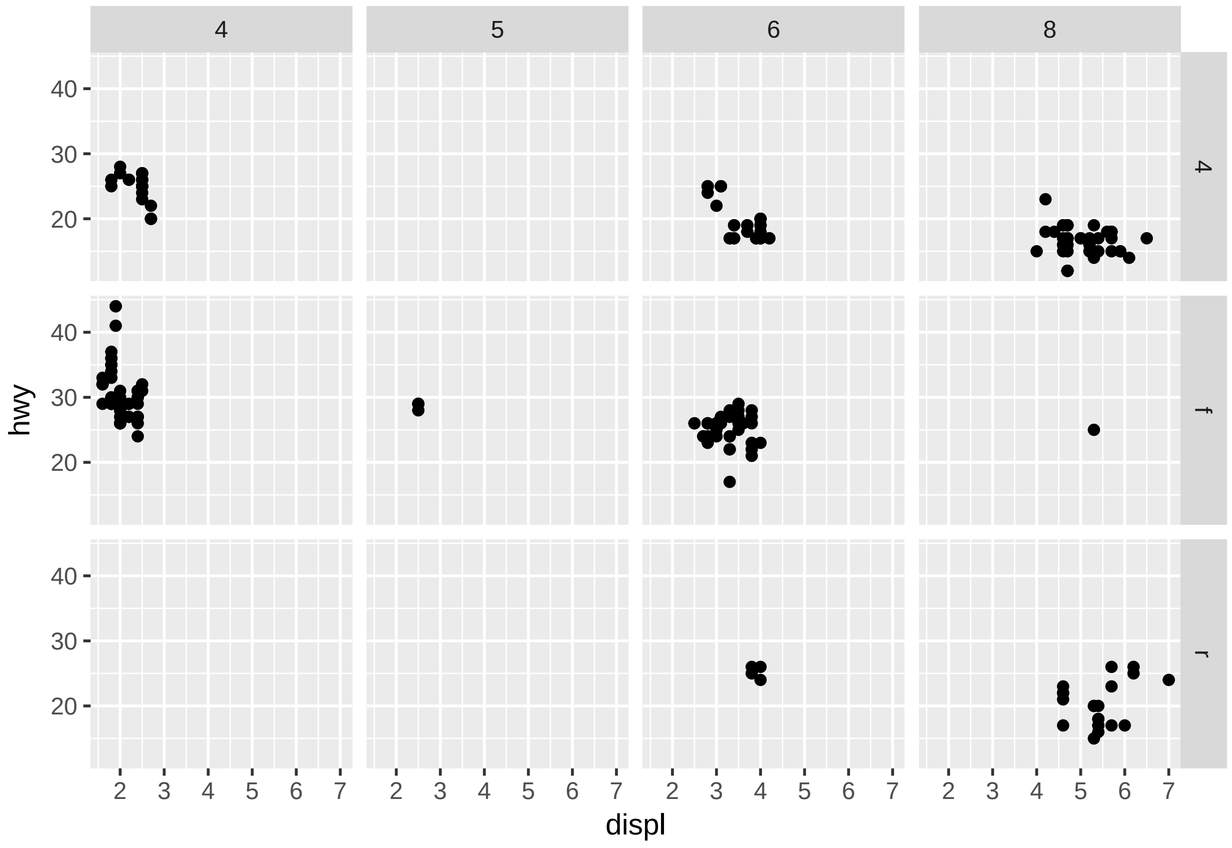
From the documentation:

> `facet_wrap` wraps a 1d sequence of panels into 2d. This is generally a better use of screen space than facet_grid because most displays are roughly rectangular.

3.5.2 Facet on the combination of two variables

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 `~`.

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



I would like to spread out the y-dimension, because there are many points overlapping each other. In `facet_wrap` I have some control of the display with nrow and ncol, but these parameters do not work in `facet_grid`. But I do not know how. `space = "free_y"` does not work. There is one questions, which seems similar to mine in SO, but the (aeustion) and also the answer is still to complex for me to understand. (see: http://stackoverflow.com/questions/28111413/how-to-set-different-y-axis-scale-in-a-facet-grid-with-ggplot).

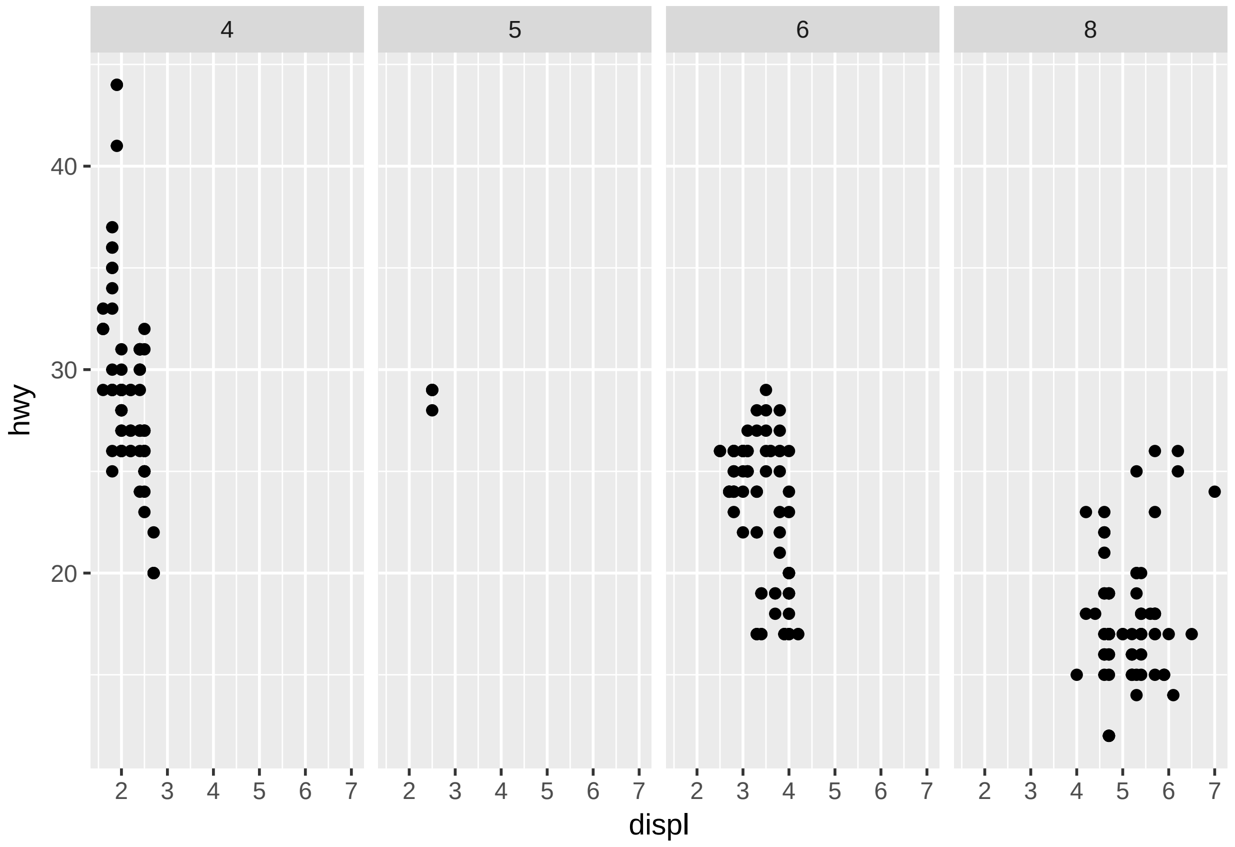
From the documentation:

> `facet_grid` forms a matrix of panels defined by row and column facetting variables. It is most useful when you have two discrete variables, and all combinations of the variables exist in the data.

3.5.3 Facet without row or column dimension

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

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

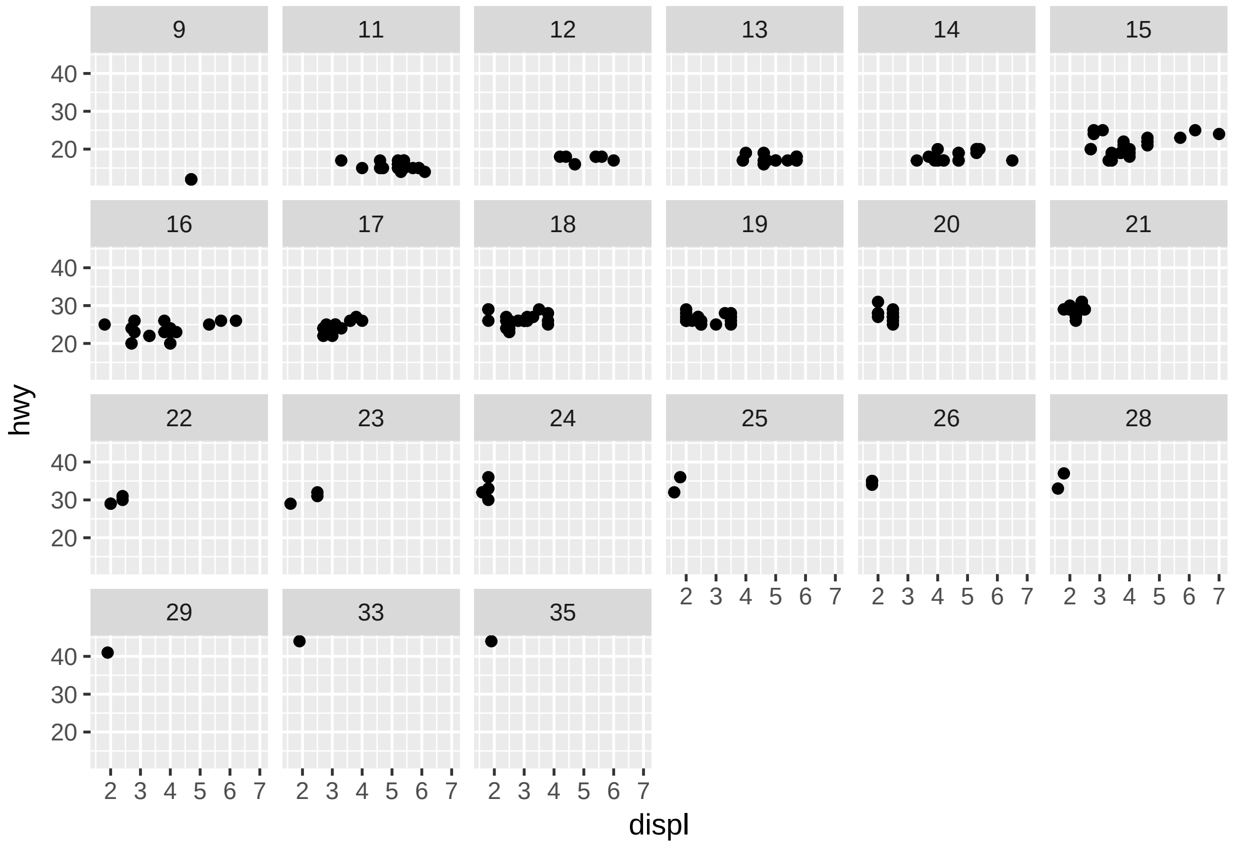


3.5.4 Exercises

(1) What happens if you facet on a continuous variable?

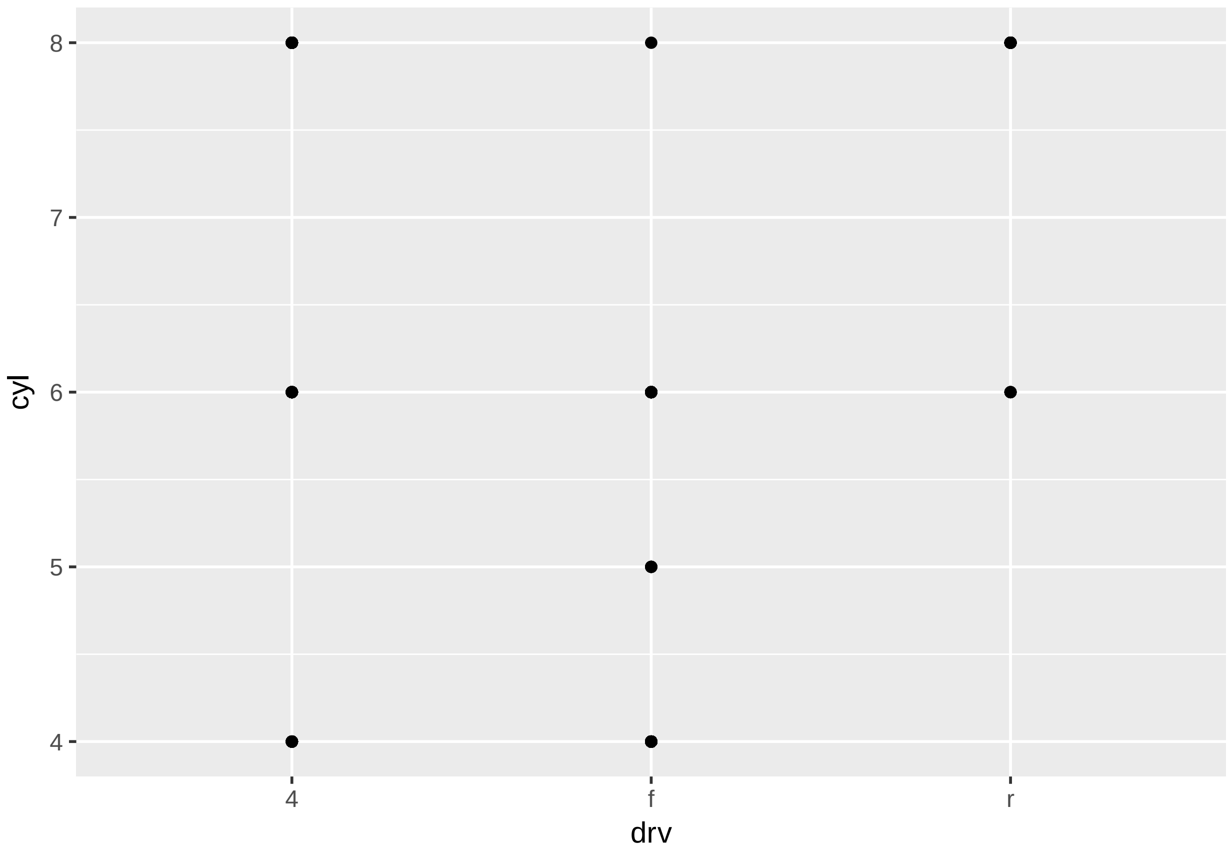
My Solution: A plot will be drawn for *every* value in the data. This could result to a very large number of plots with little information.

```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_wrap(~ cty, nrow = 4)
```



(2) What do the empty cells in plot with `facet_grid(drv ~ cyl)` mean? How do they relate to this plot?

```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = drv, y = cyl))
```

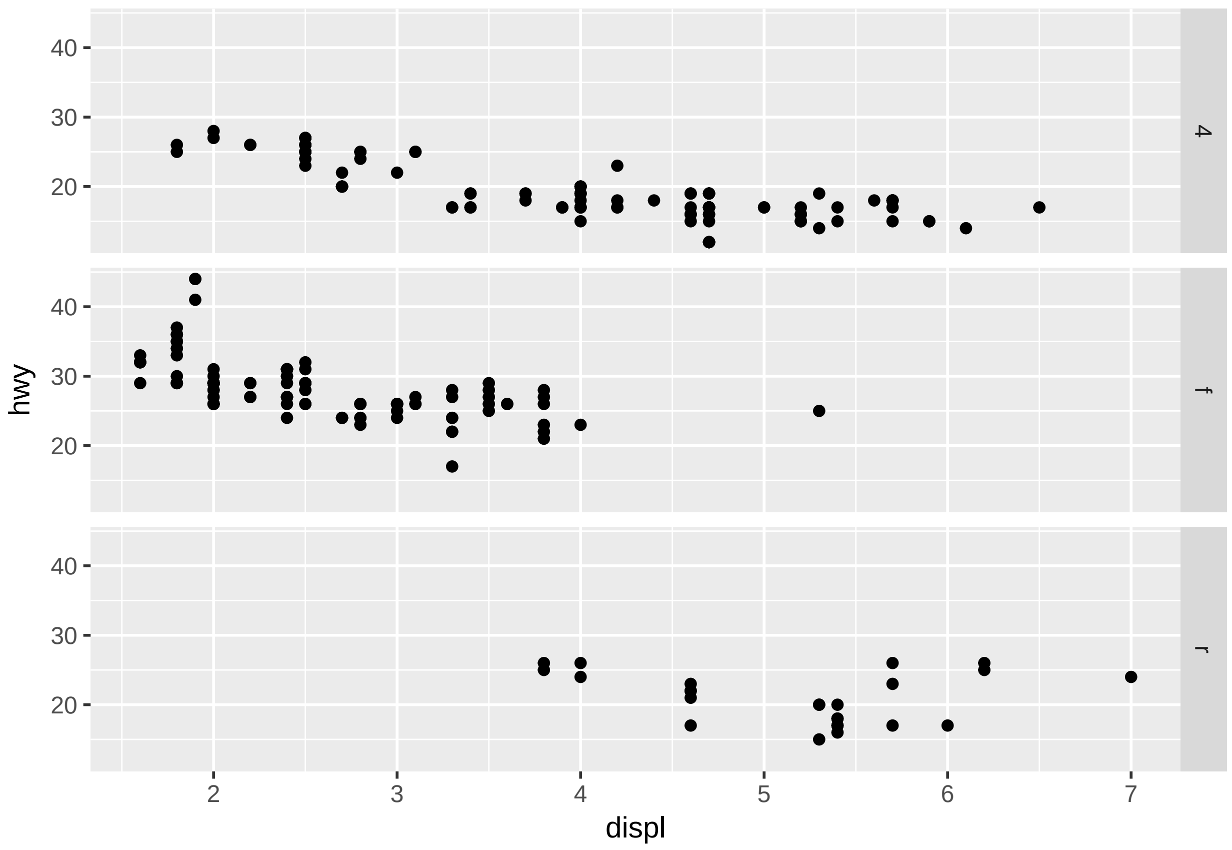


My Solution: An empty cell means that there are no data for the specified combination, e.g. there are no cars with a four-wheel-drive and 5 cylinder. The plot above shows what combinations are missing, e.g. there are no cars with 7 cylinder.

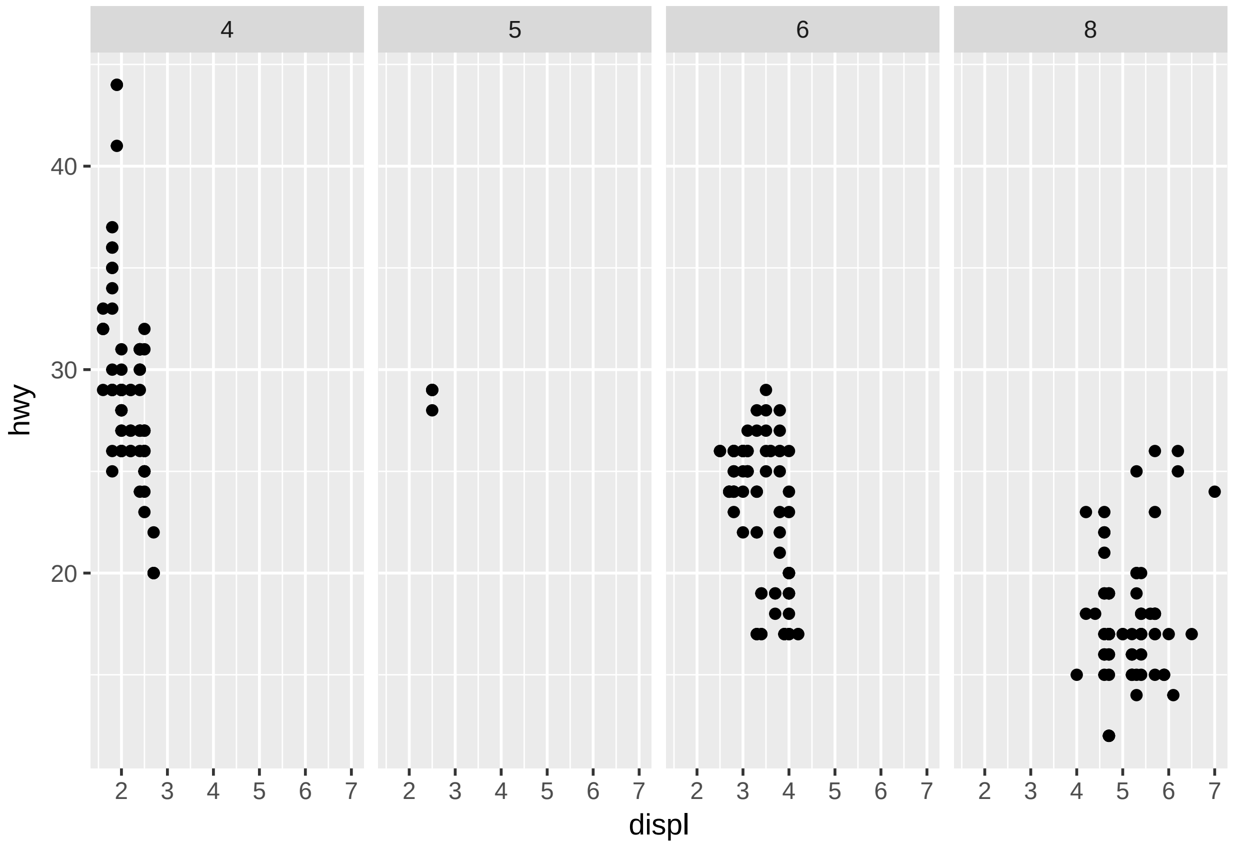
(3) What plots does the following code make? What does '. do?

My Solution: The . suppresses the facet in the row or column dimension so that the plot in the row resp. in the column has no subsection, e.g. separate grids.

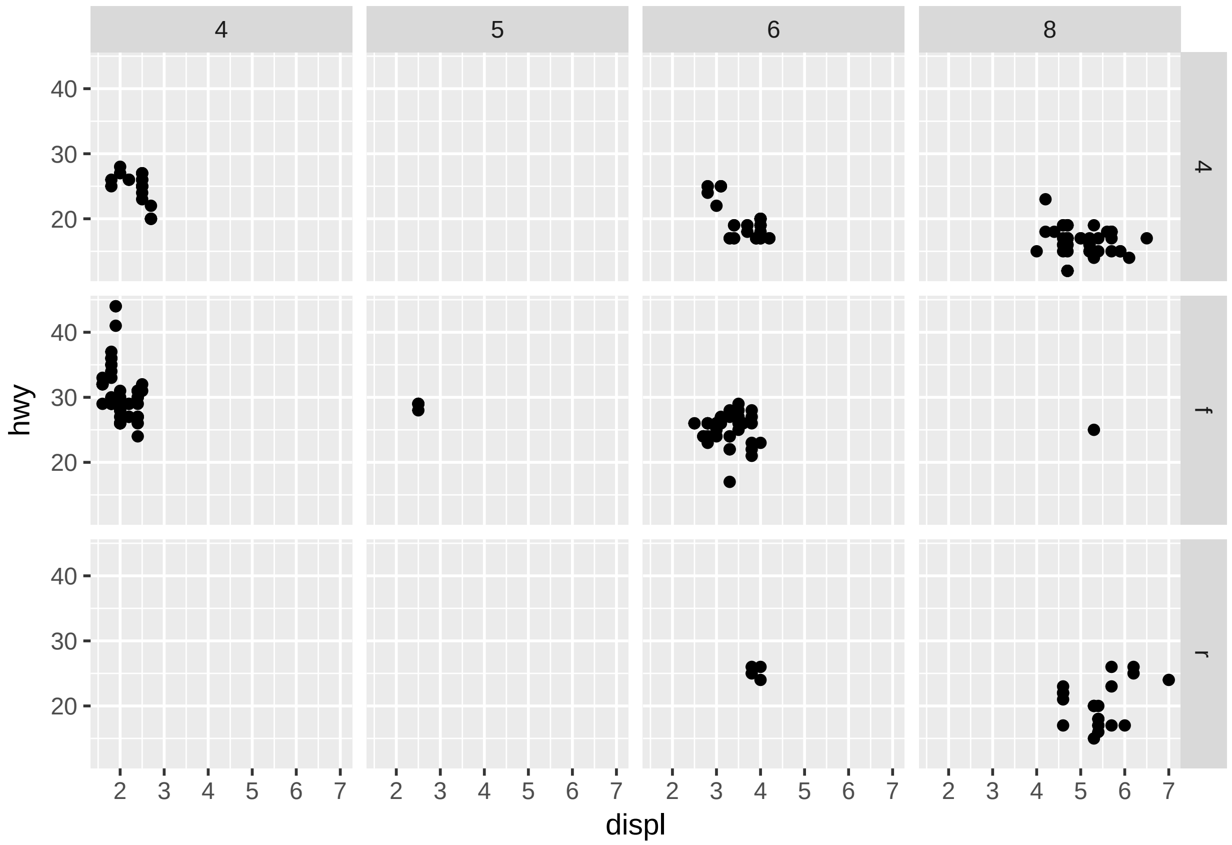
```
> # no column facet, `displ` will not be broken up in subparts
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_grid(drv ~ .)
```



```
> # no row facet, `drv` will not be broken up in subparts
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_grid(. ~ cyl)
```

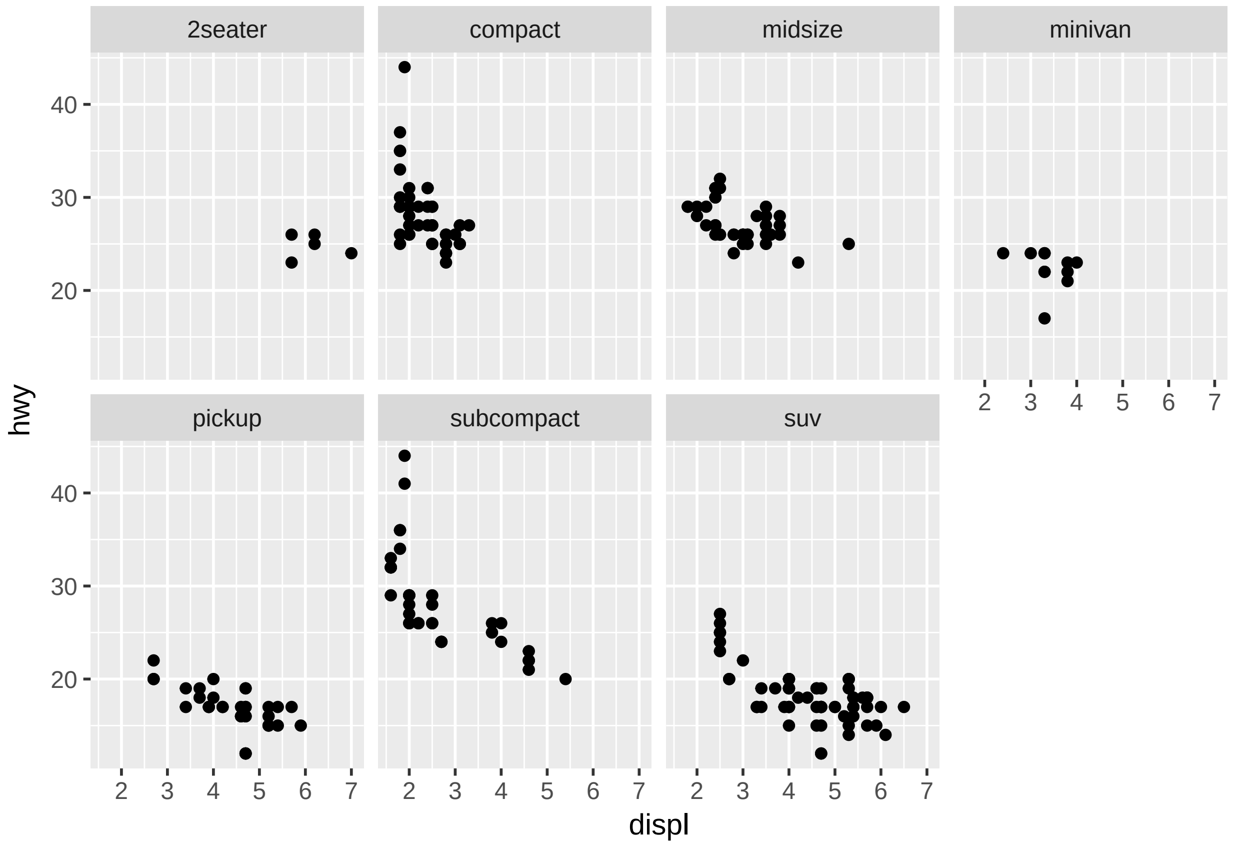


```
> # row and column facet
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_grid(drv ~ cyl)
```



(4) Take the first faceted plot in this section:

```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_wrap(~ class, nrow = 2)
```

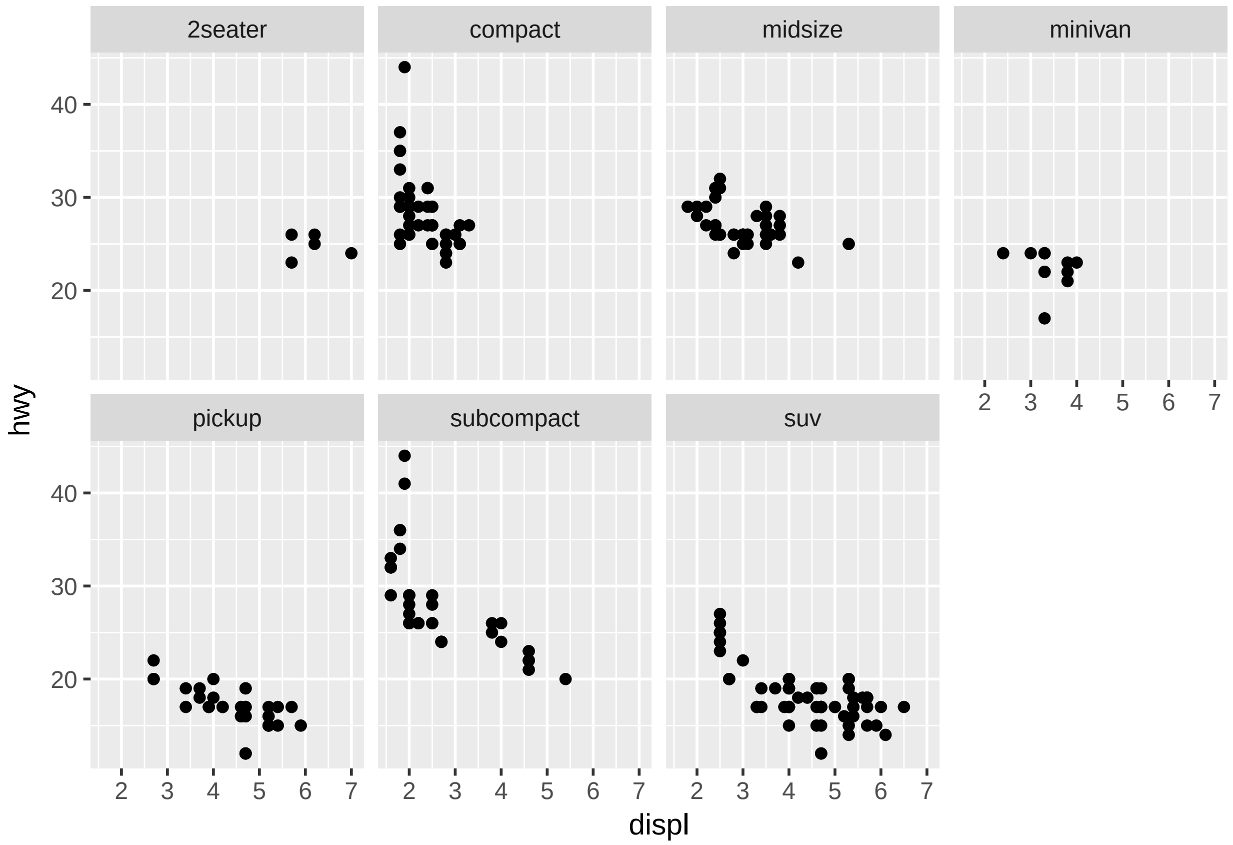


What are the advantages to using faceting instead of the color aesthetic? What are the disadvantages? How might the balance change if you had a larger dataset?

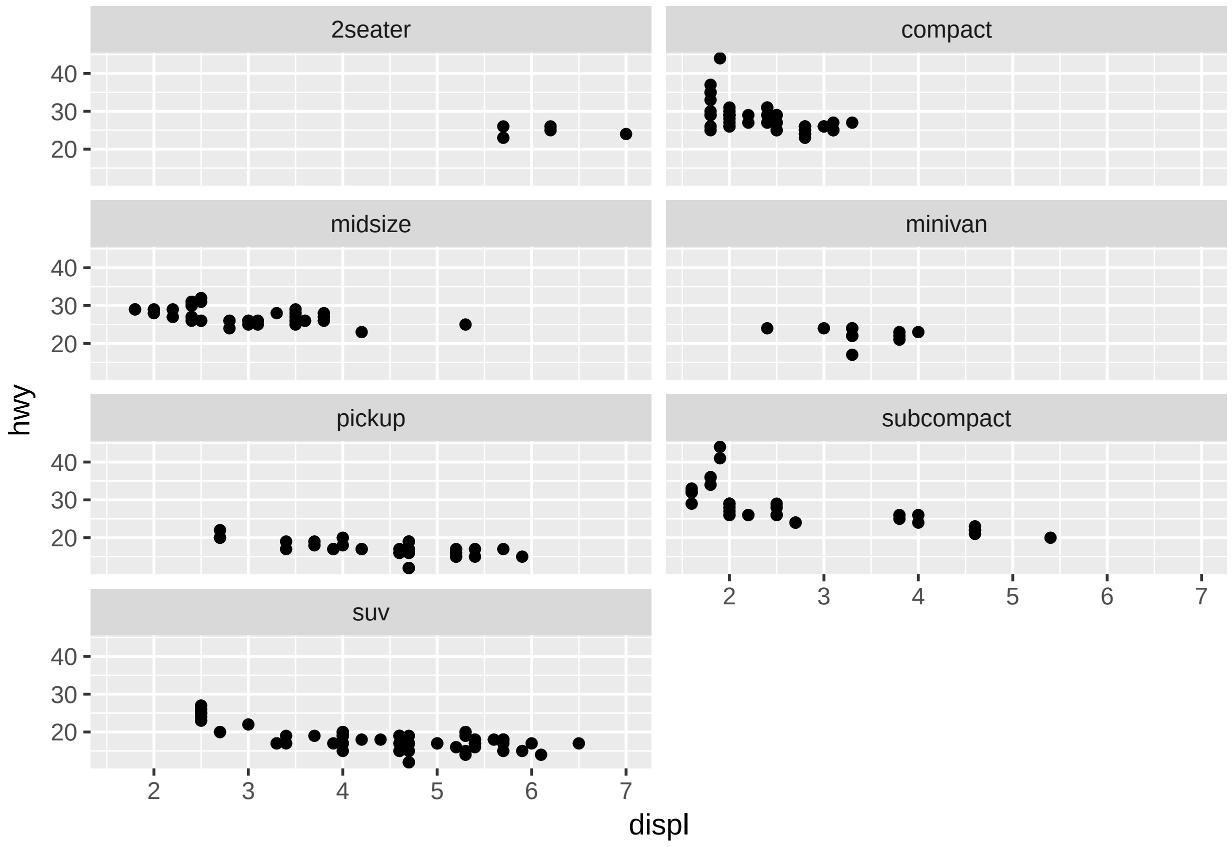
My Solution: Because of all the separated grids with the facet version you see better the relationship of the x and y variable under the conditions of the third variable. There is the danger that with the color aesthetic — as it is plotted just in one graph — are many points overlapping each other. For instance you can see in the suv grid about 5 vehicles with 2,5 litre displacement, whereas in the plot with the color aesthetic you see more but only two suv cases. The disadvantage of the facet version is that it is difficult to get an overview of the relationship of all data combined. For data sets with many different values of the third variable the facet versions becomes unhandy. But in that case maybe the color aesthetics is not better as many points overlap each other. – In my opinion one needs a plot where you can see how many points are overlapping each other.

(5) 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 variables?

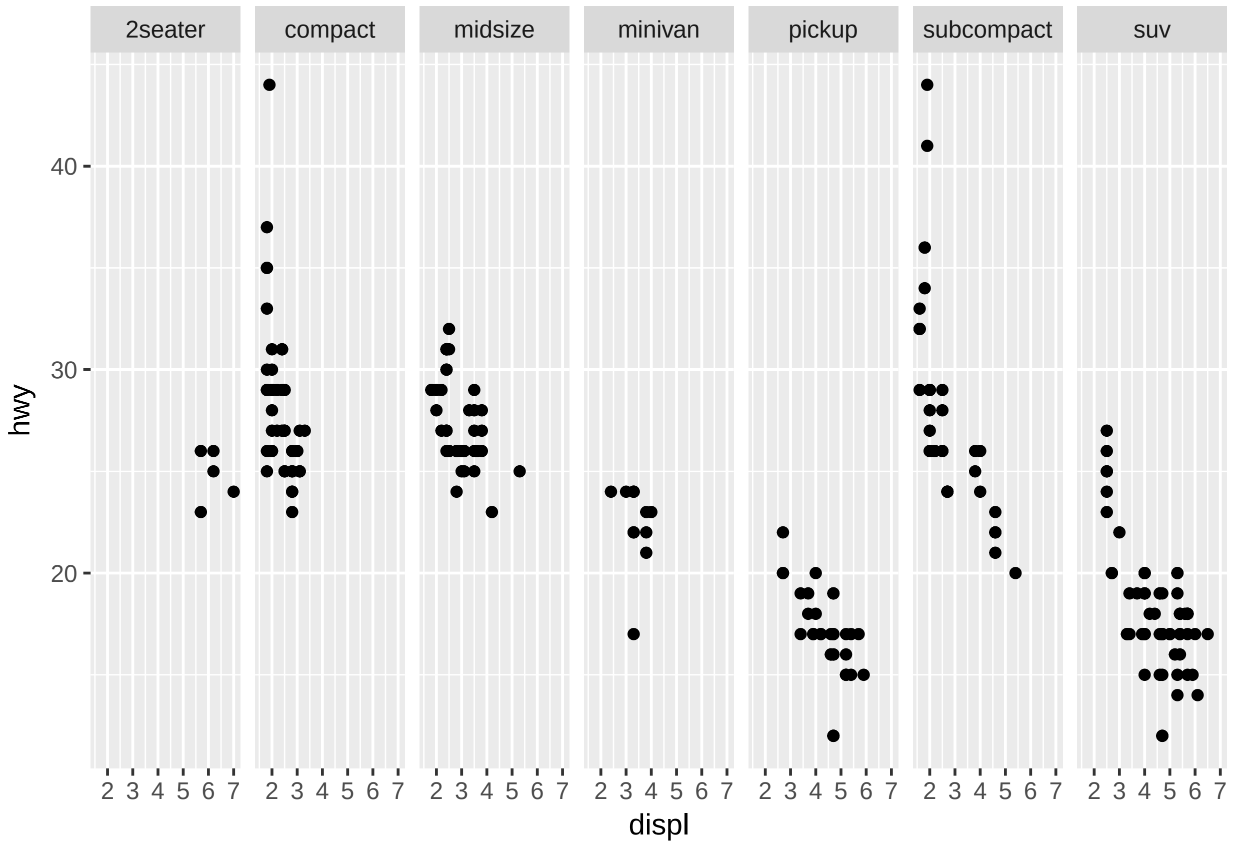
```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_wrap(~ class, nrow = 2)
```



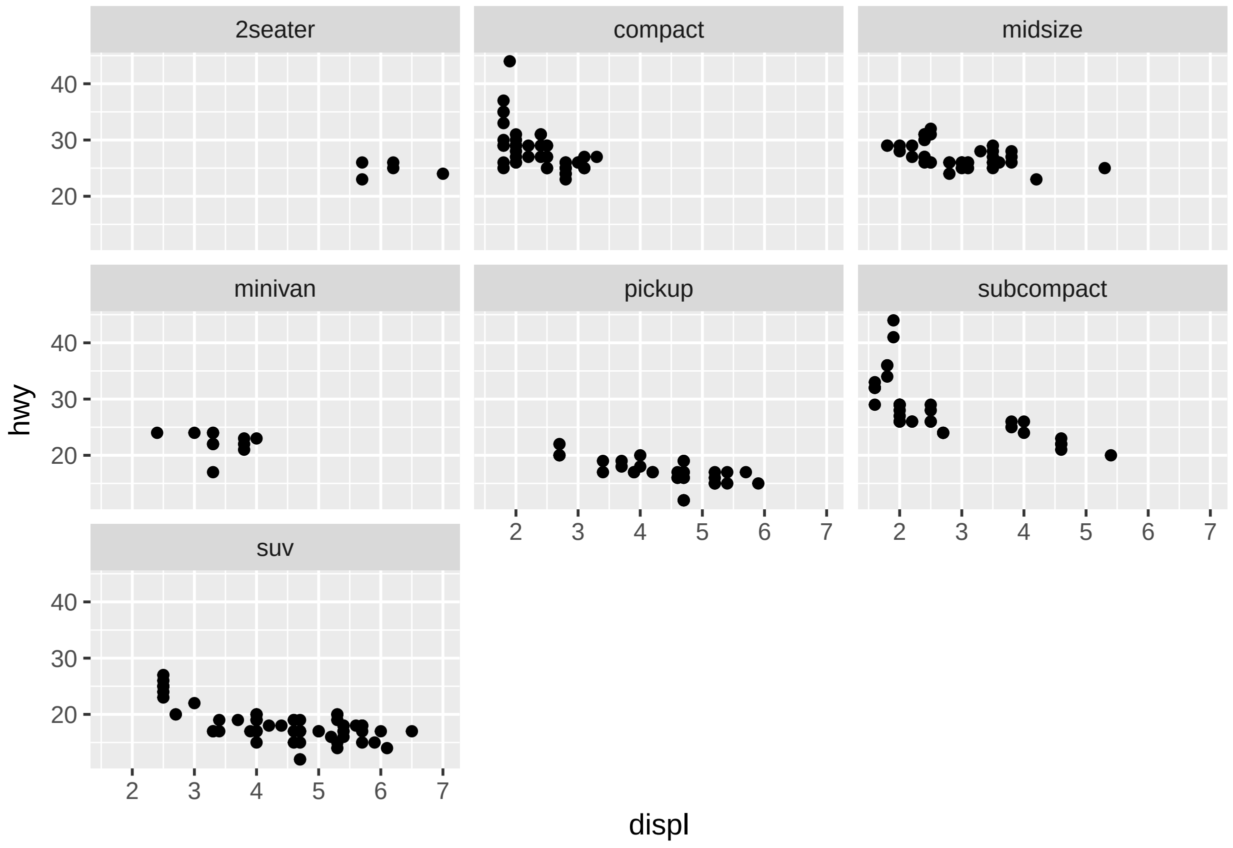
```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_wrap(~ class, nrow = 4) # more rows results in grids with a greater width
```



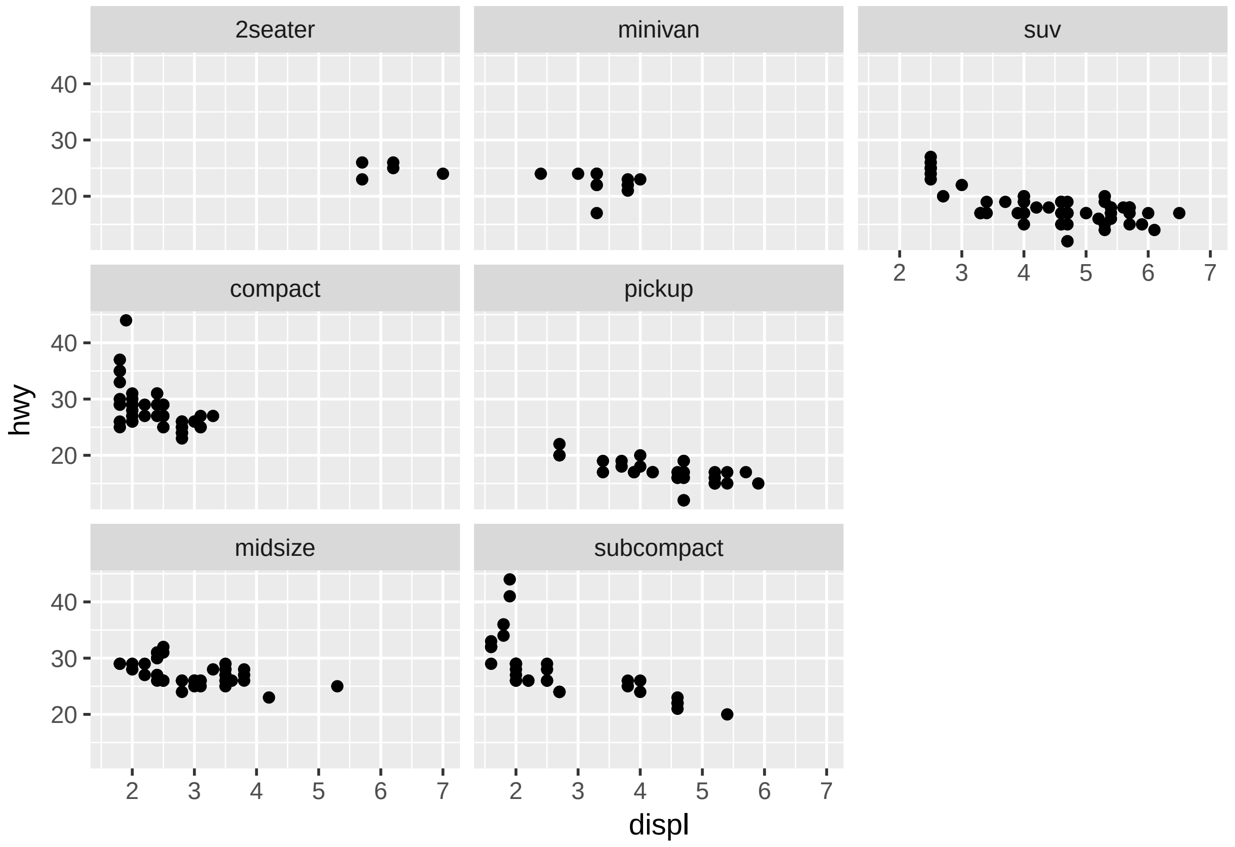
```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_wrap(~ class, nrow = 1) # less rows results in thinner but longer grids
```



```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_wrap(~ class, ncol = 3) # display 3 grids in one row with order from left to right
```



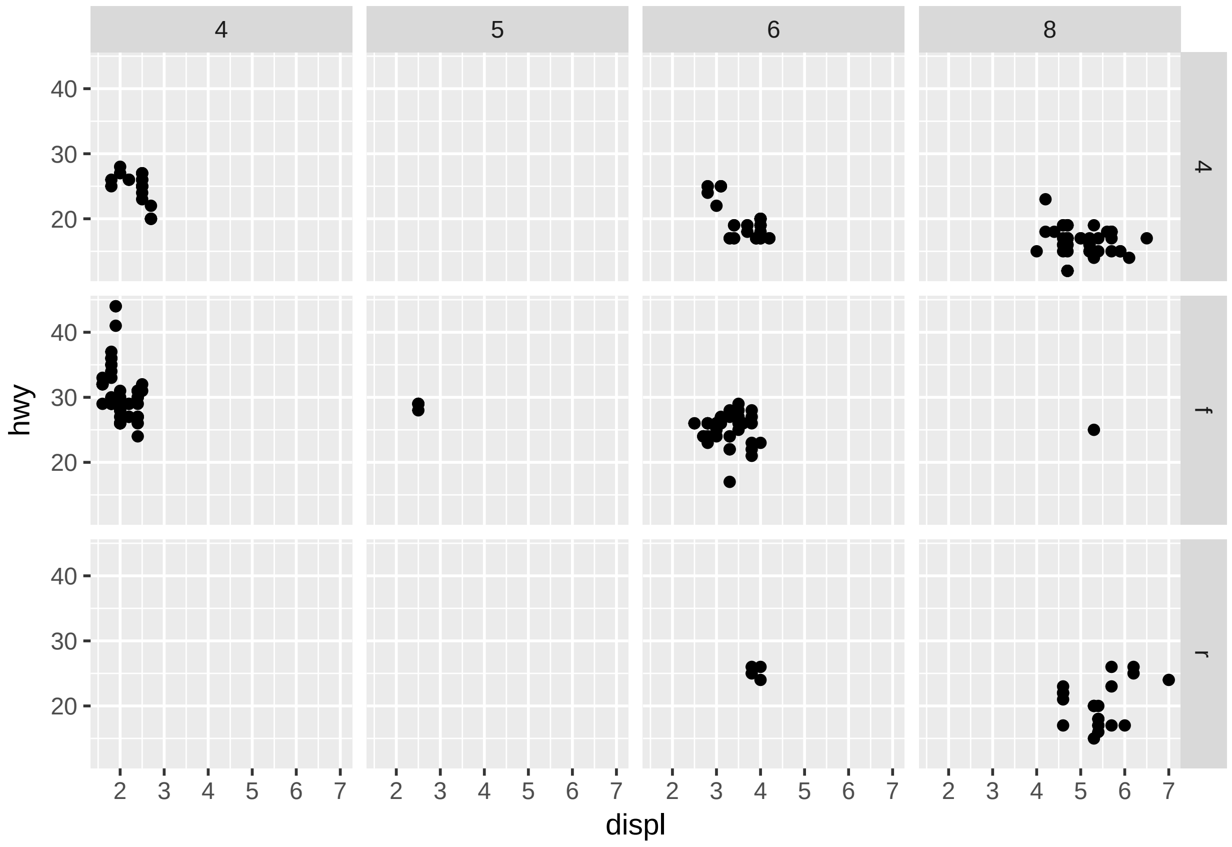
```
> ggplot(data = mpg) +
+   geom_point(mapping = aes(x = displ, y = hwy)) +
+   facet_wrap(~ class, ncol = 3, dir = "v") # display 3 grids in one row, from top to down
```



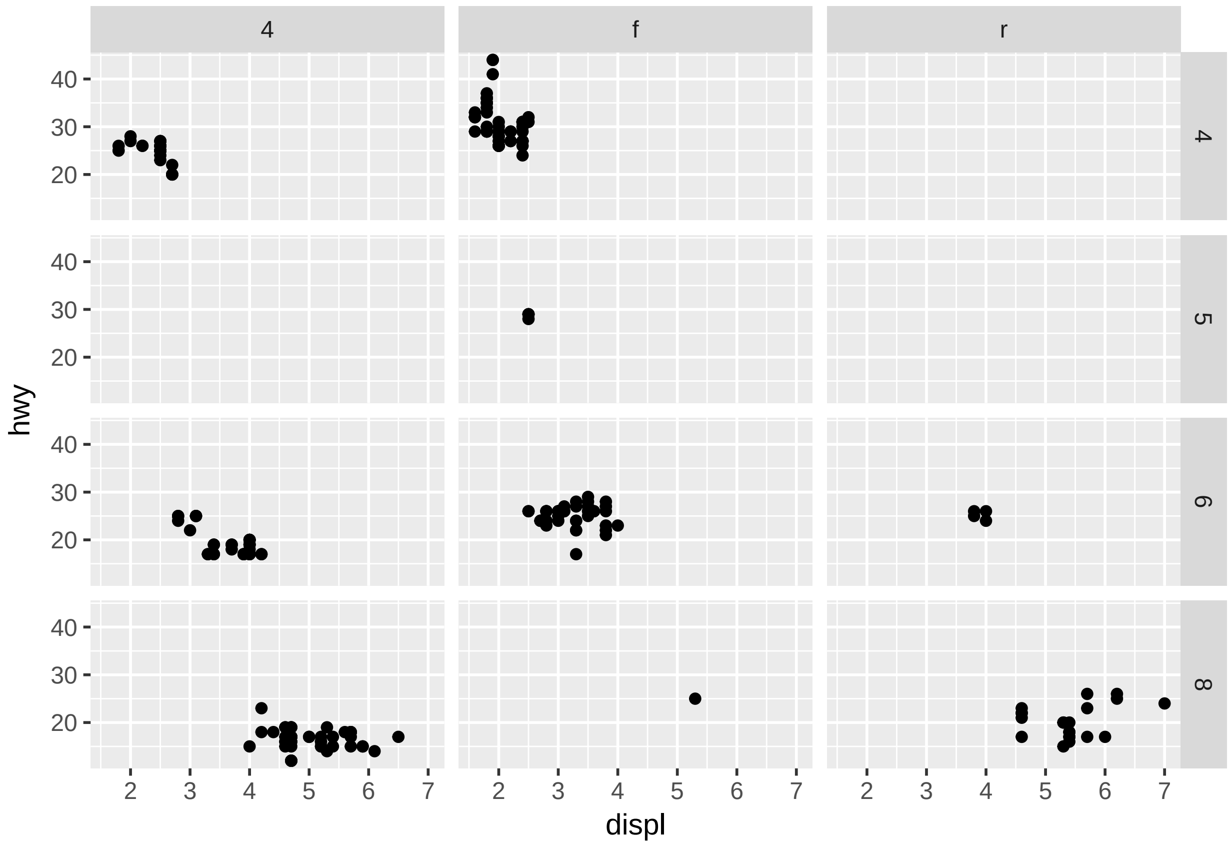
My Solution: With nrow and ncol you can control the distribution or layout of the grids panels and therefore width and length of the panels. `dir` controls the layout as well by ordering from left to right (`h`= horizontal layout) or from top to down (`v`= vertical layout). `facet_grid()` does not have sewttings for `nrow` and `ncol` as they are already integrated in the function.

(6) When using `facet_grid()` you should usually put the variable with more unique levels in the columns. Why?

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



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



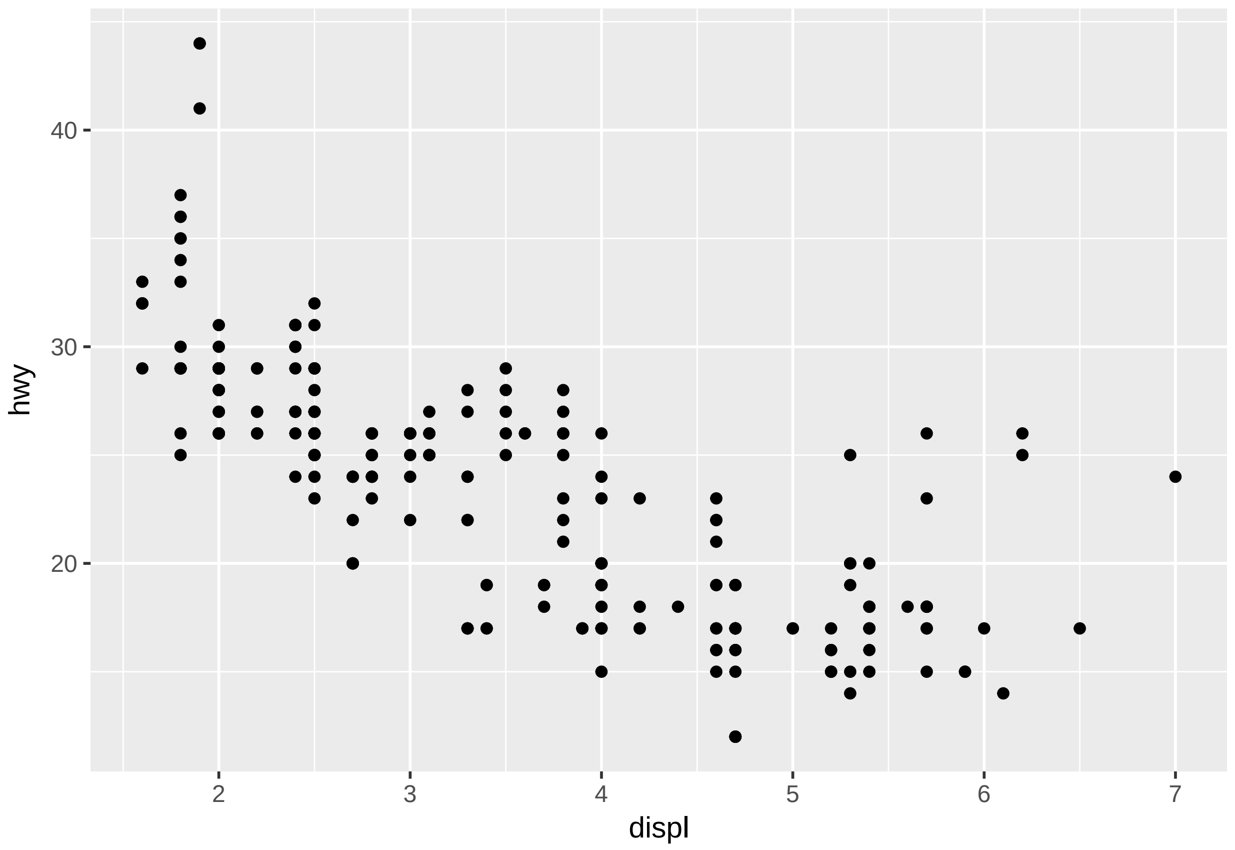
My Solution: To save space and to procude an compact display you should usually put the variable with more unique levels in the columns. If it doesn't fit the line, ggplots starts another line of panels. Otherwise each value of the variable would generate a new line.

3.6 Geometric objects

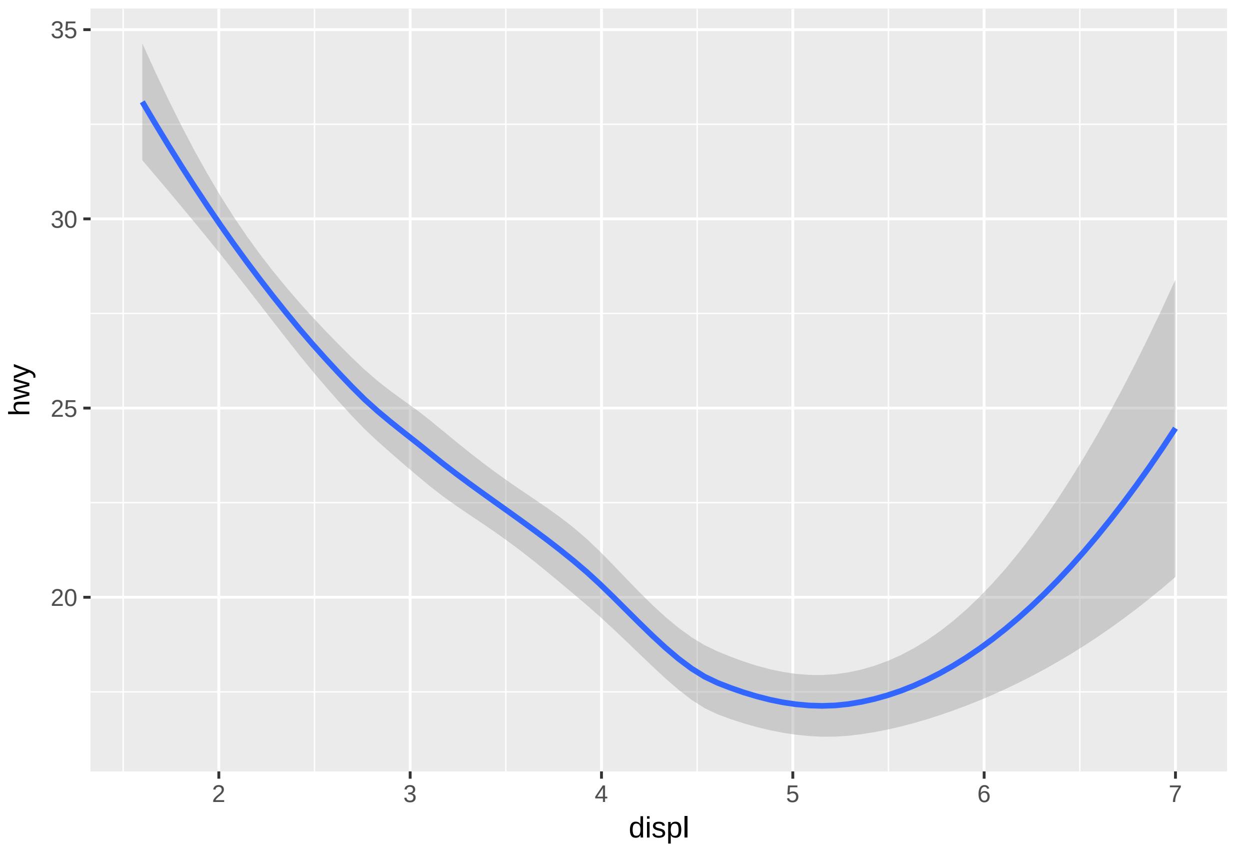
3.6.1 Same data but different geoms

How are these two plots similar?

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



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



>

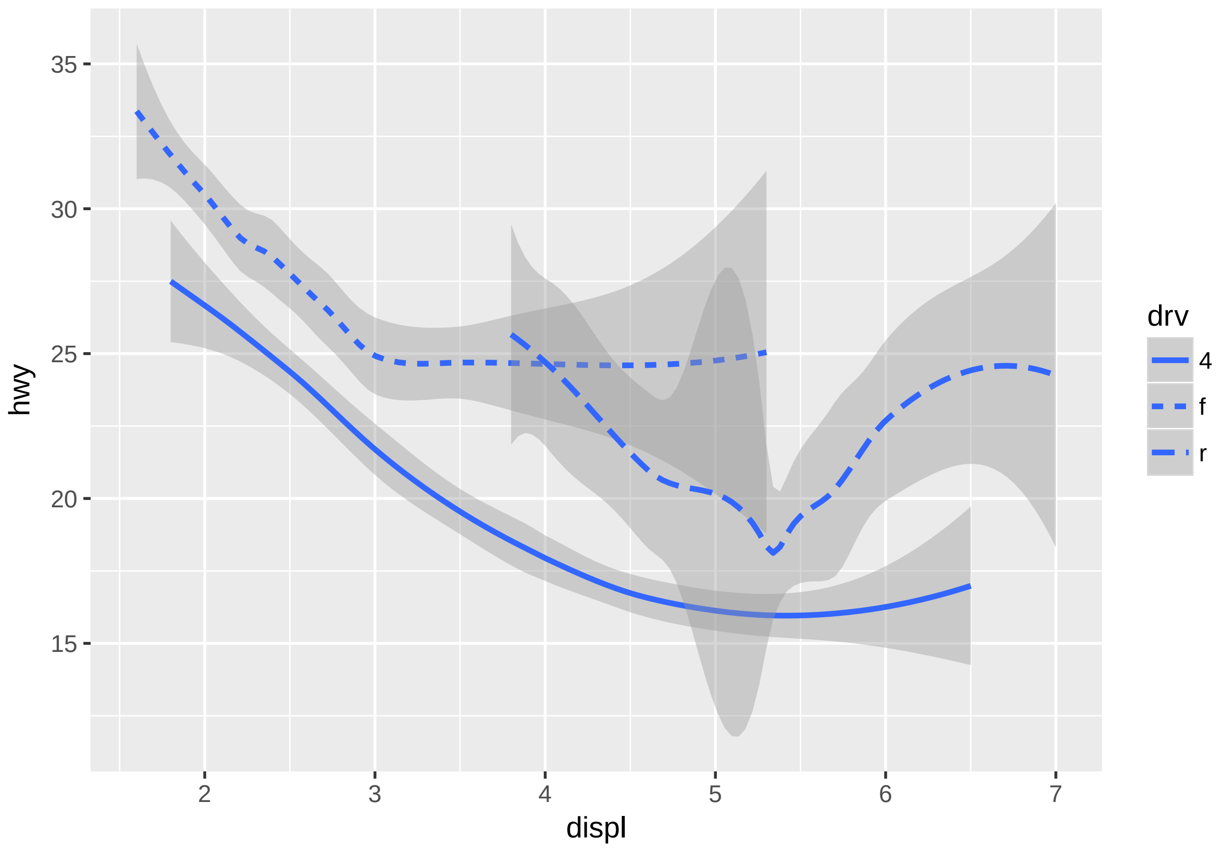
A geom is the geometrical object that a plot uses to represent data. … For example, bar charts use bar geoms, line charts use line geoms, boxplots use boxplot geoms, and so on. Scatterplots break the trend; they use the point geom.

3.6.2 Not every aesthetic works with every geom

Every geom function in ggplot2 takes a mapping argument. However, not every aesthetic works with every geom. You could set the shape of a point, but you couldn't set the "shape" of a line. On the other hand, you could set the linetype of a line. geom_smooth() will draw a different line, with a different linetype, for each unique value of the variable that you map to linetype.

On the other hand: For each geom exist different kinds of appropriate aesthetics to choose from.

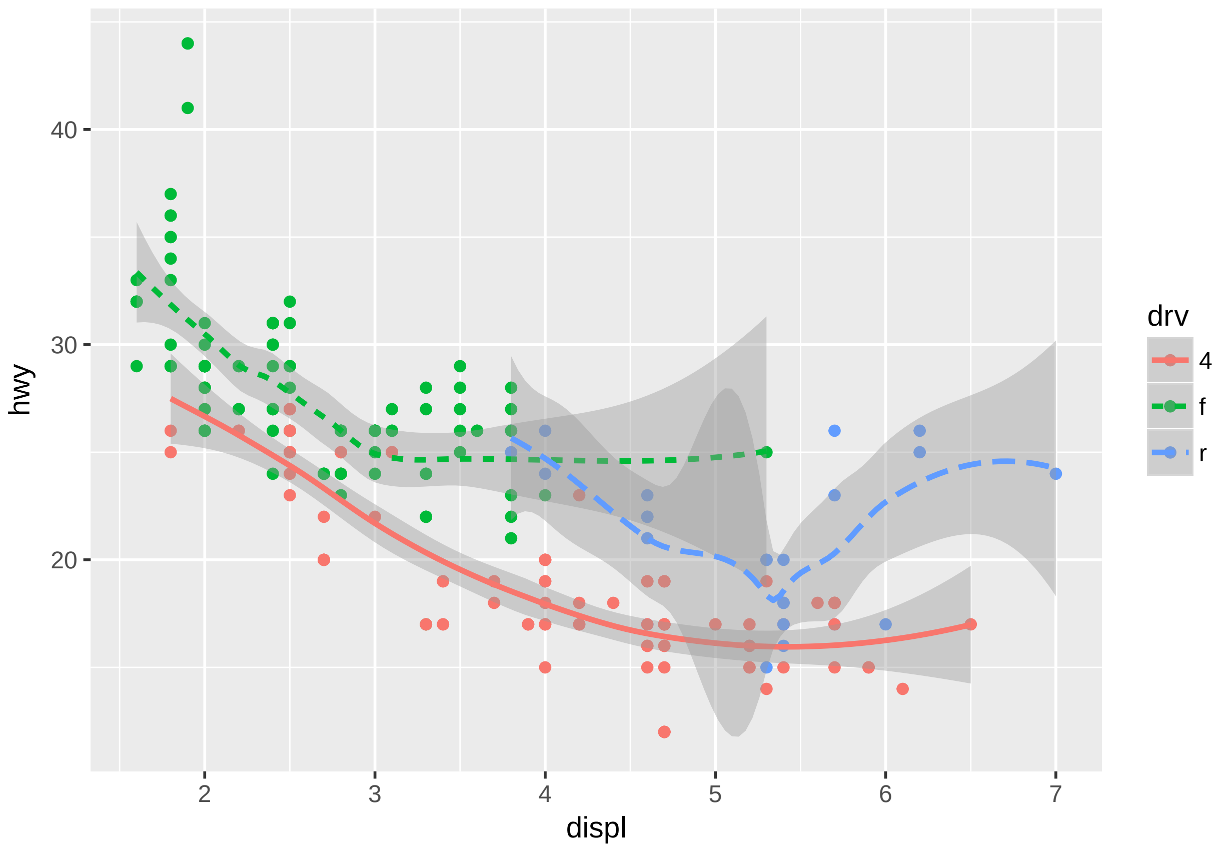
```
> ggplot(data = mpg) +
+   geom_smooth(mapping = aes(x = displ, y = hwy, linetype = drv))
```



3.6.3 Two geoms in one graph

You can also overlay the lines on top of the raw data and then color everything according to the values of `drv`:

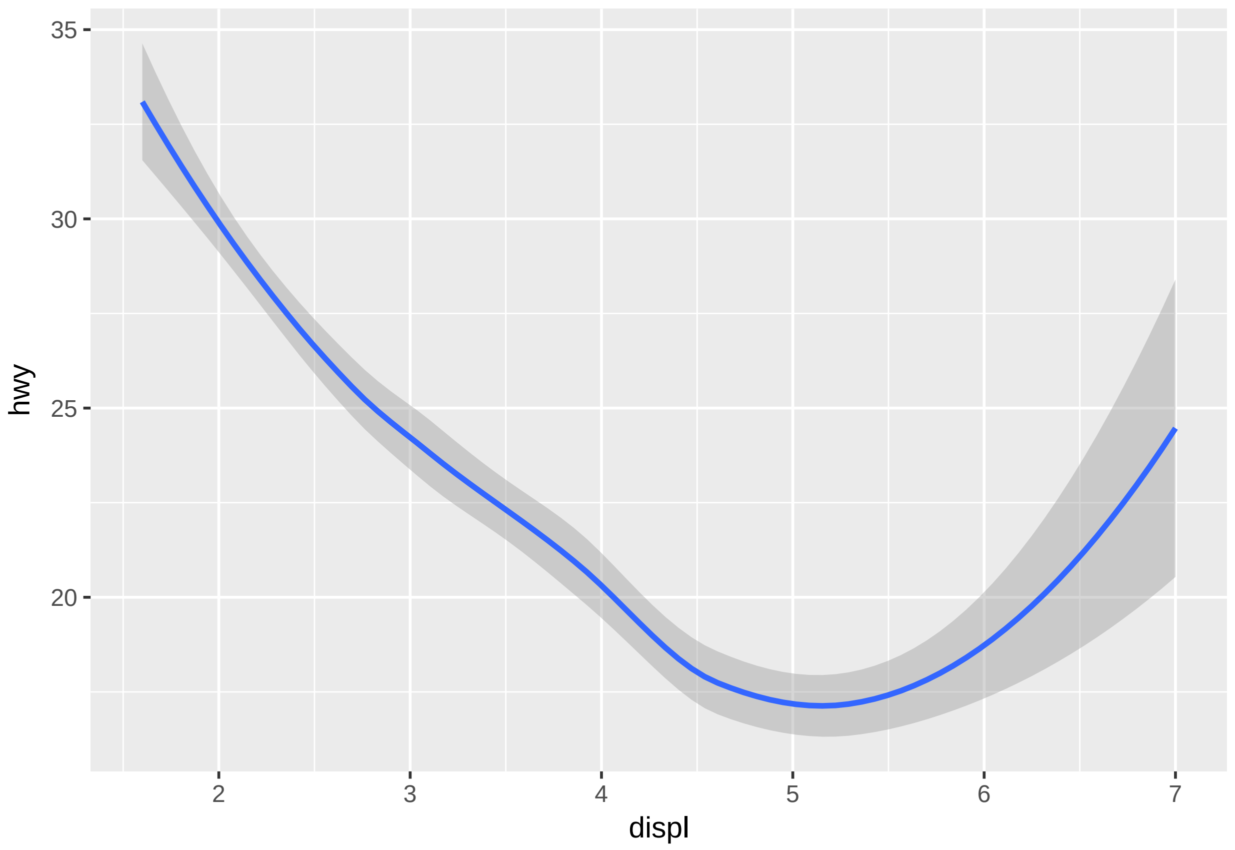
```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +
+         geom_point() +
+         geom_smooth(mapping = aes(x = displ, y = hwy, linetype = drv))
```



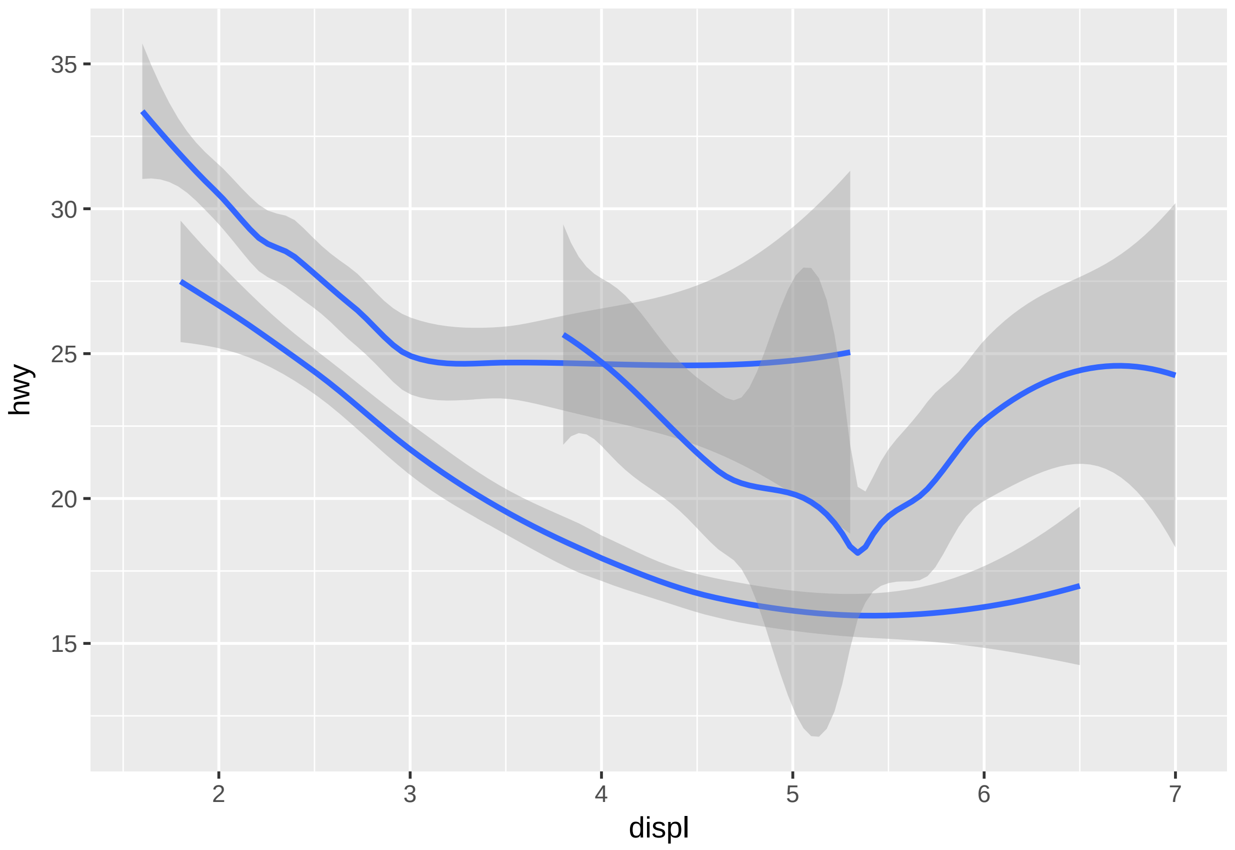
3.6.4 Multiple rows of data and grouping with discrete/categorial variable

Many geoms, like `geom_smooth()`, use a single geometric object to display multiple rows of data. For these geoms, you can set the `group` aesthetic to a categorical variable to draw multiple objects. `ggplot2` will draw a separate object for each unique value of the grouping variable. In practice, `ggplot2` will automatically group the data for these geoms whenever you map an aesthetic to a discrete variable (as in the `linetype` example). It is convenient to rely on this feature because the group aesthetic by itself does not add a legend or distinguishing features to the geoms.

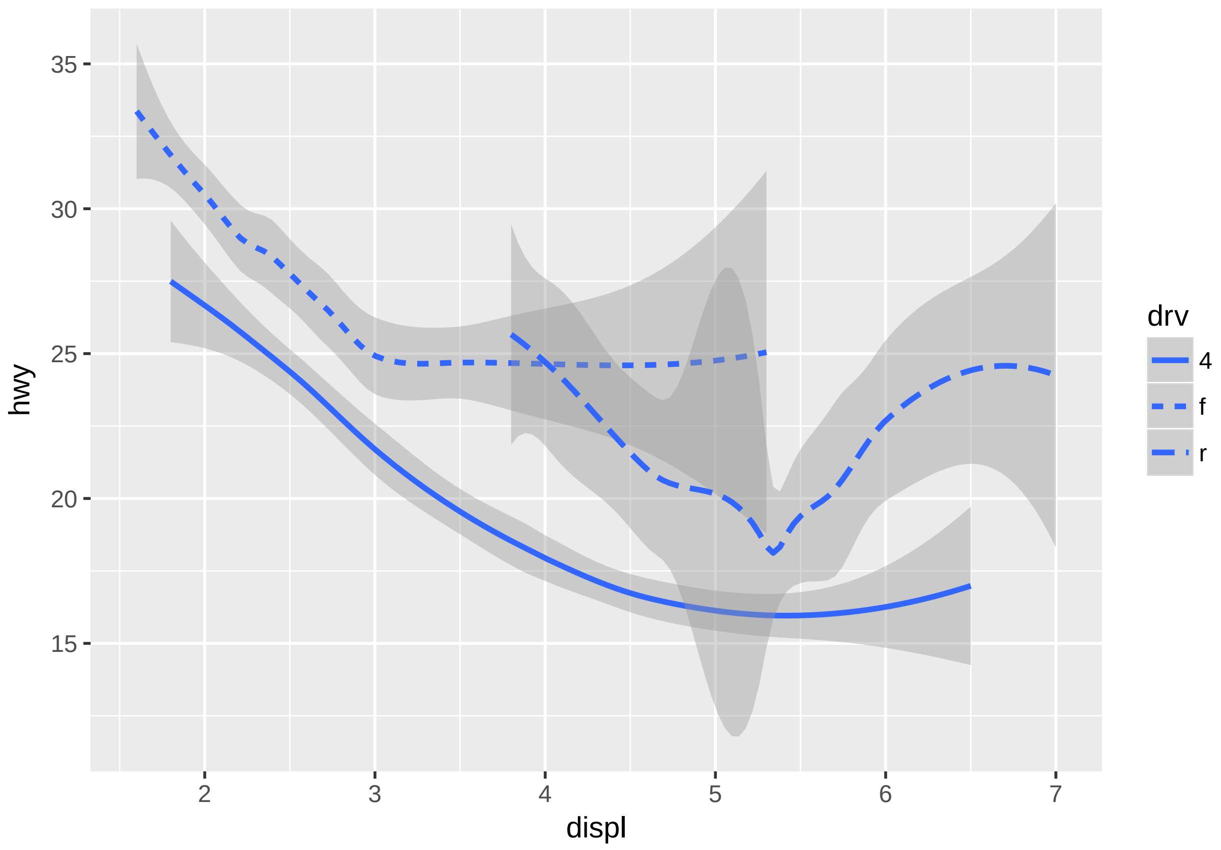
```
> # just
> ggplot(data = mpg) +
+   geom_smooth(mapping = aes(x = displ, y = hwy))
```



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

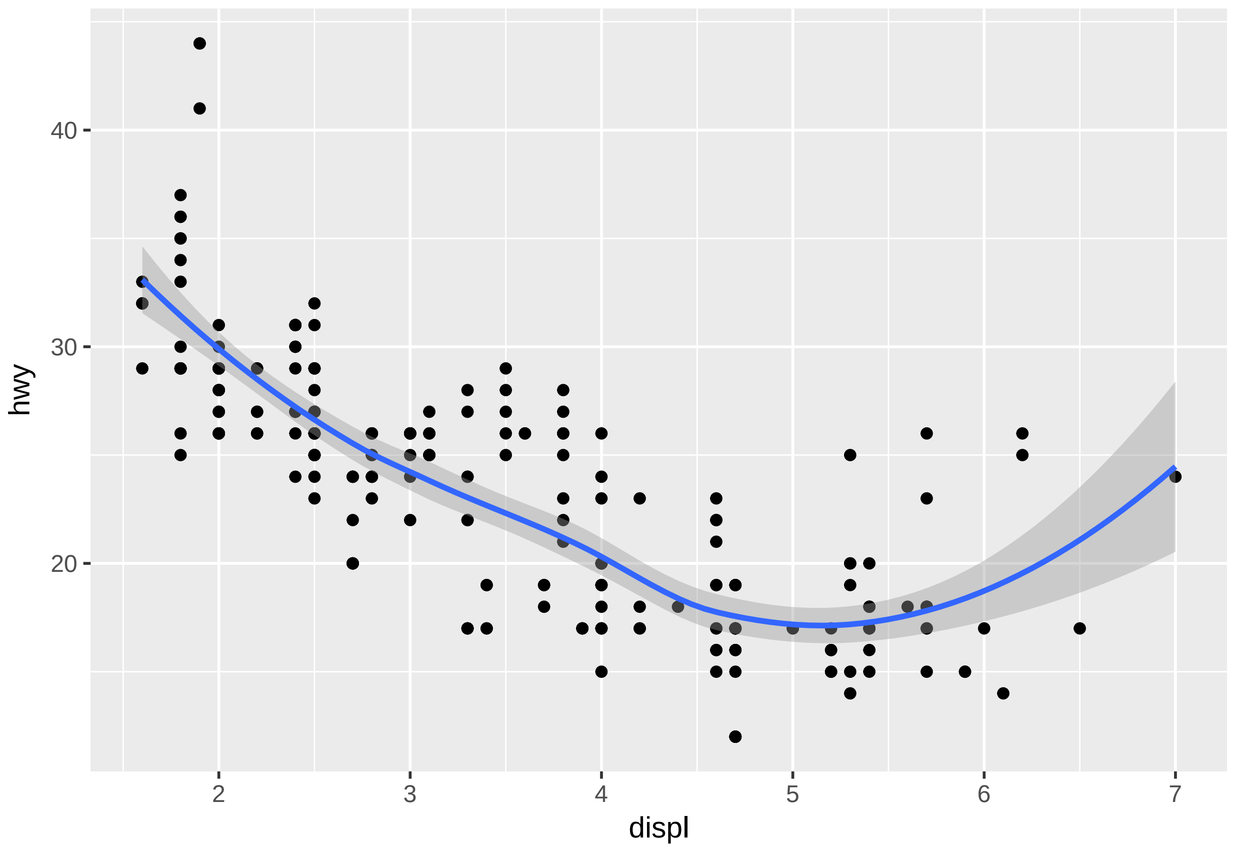


```
> # the last one is different than in the book
> # I think this is an error, as 2 + 3 are in the book identical
> # Here I used `linetype` to demonstrate the legend feature
> ggplot(data = mpg) +
+   geom_smooth(
+     mapping = aes(x = displ, y = hwy, linetype = drv)
+   )
```



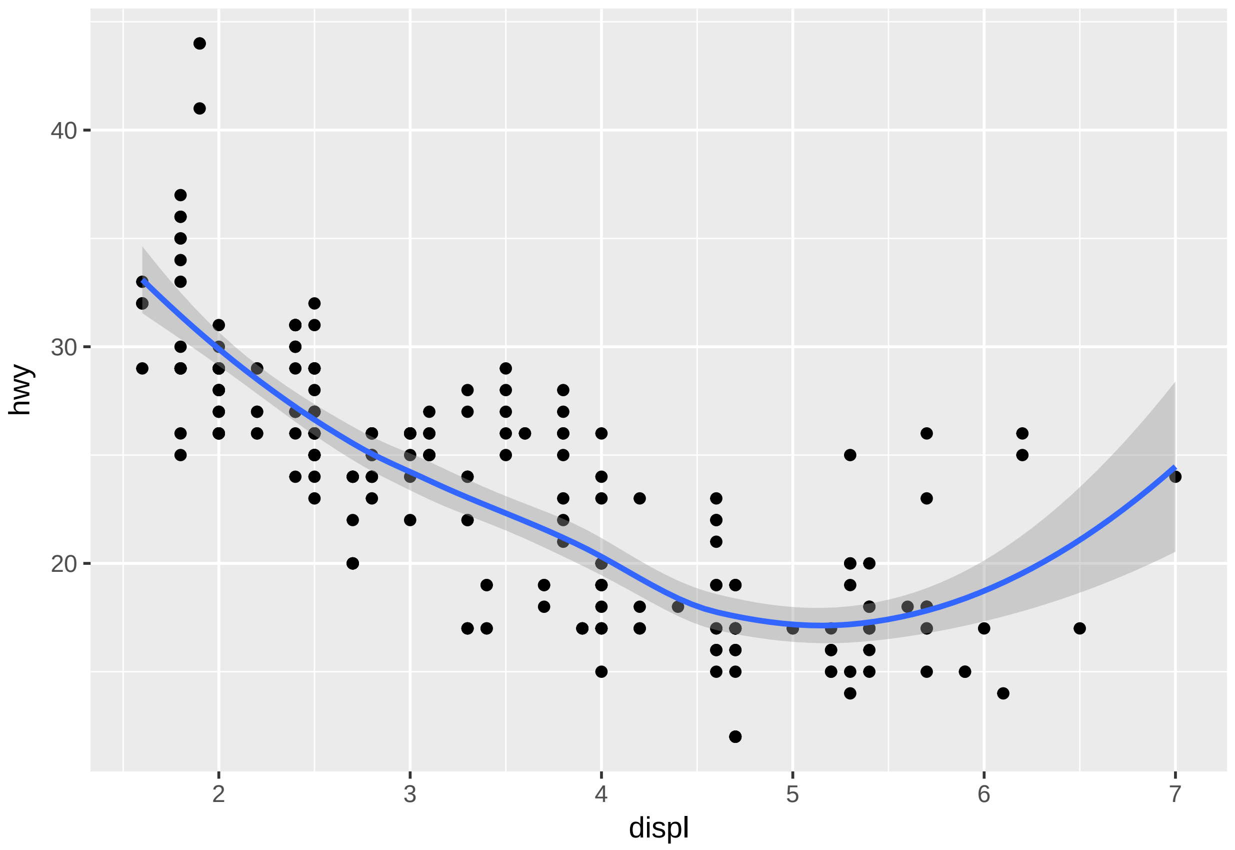
3.6.5 Multiple geoms in the same plot To display multiple geoms in the same plot, add multiple geom functions to ggplot():

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



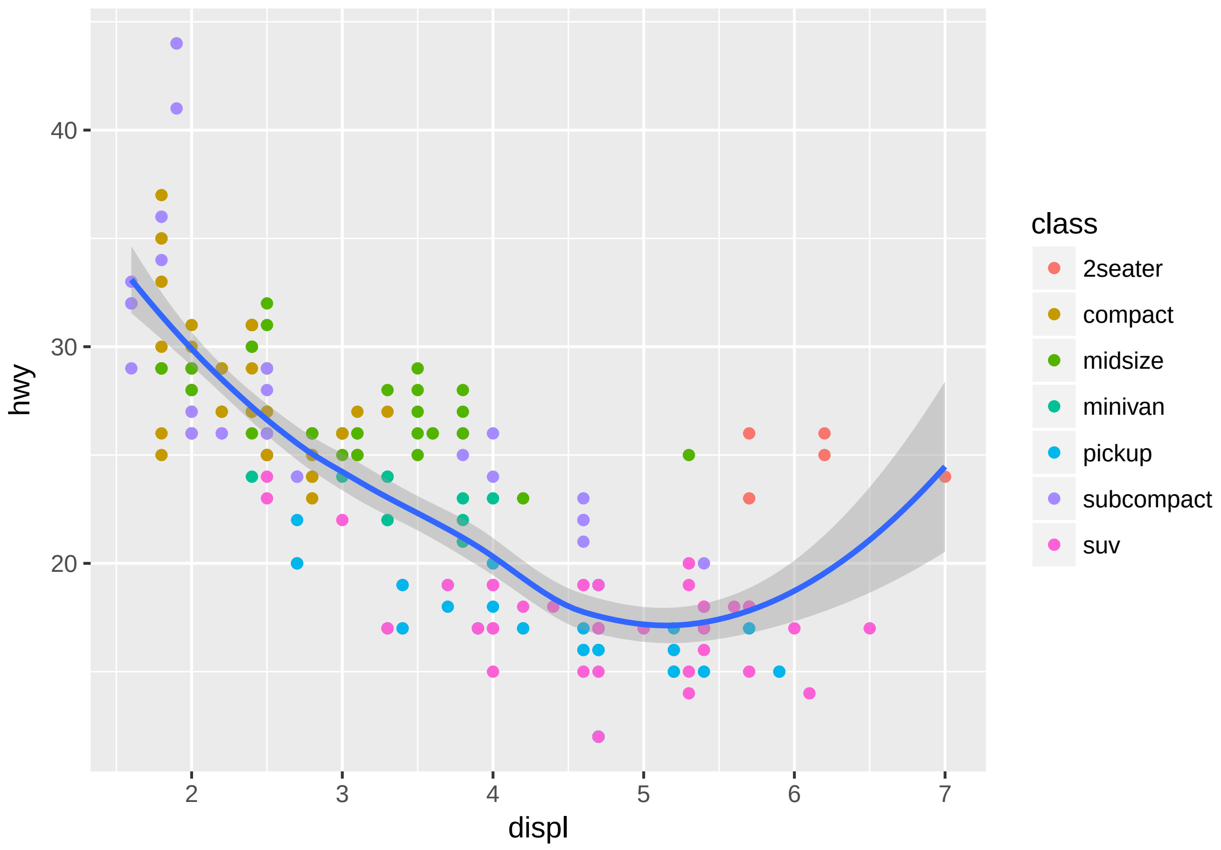
This, however, introduces some duplication in our code. One can avoid this type of repetition by passing a set of mappings to ggplot(). ggplot2 will treat these mappings as global mappings that apply to each geom in the graph.

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



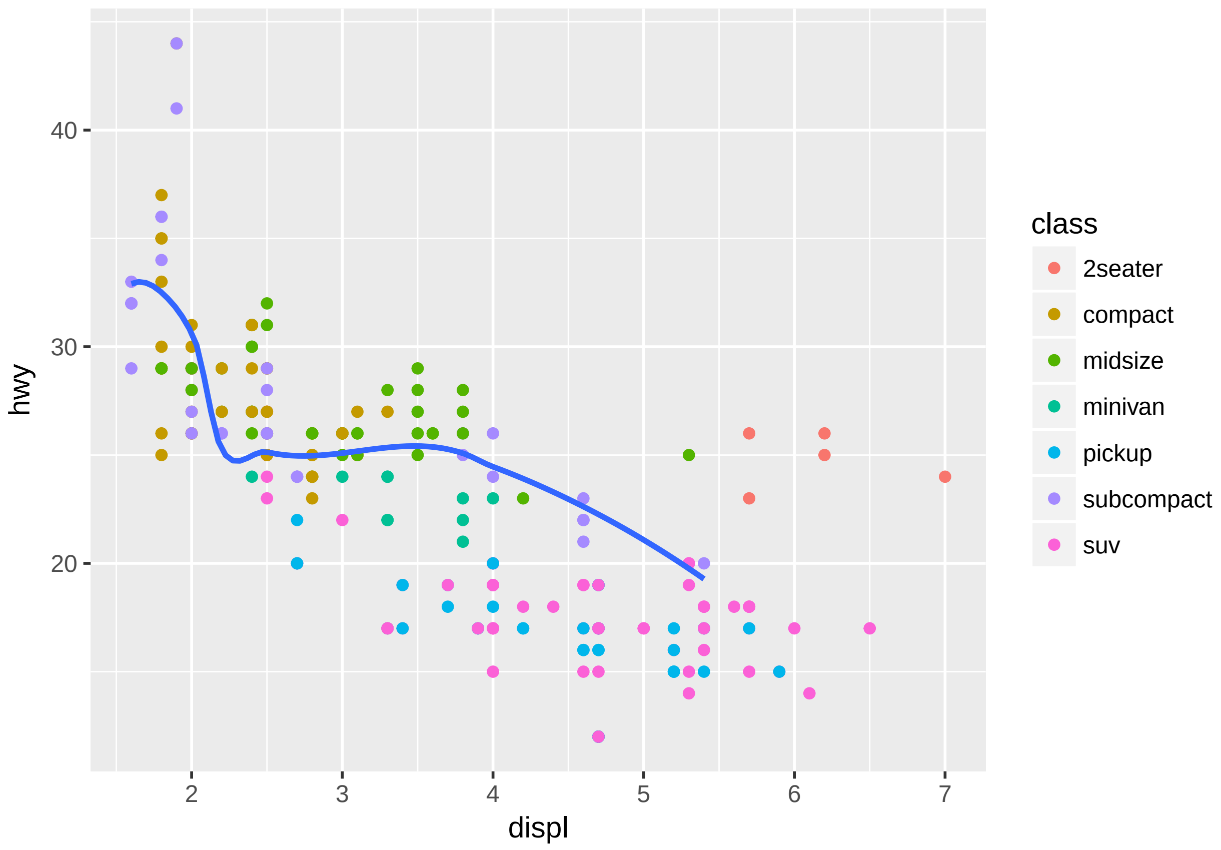
3.6.6 Global and local mappings If you place mappings in a geom function, ggplot2 will treat them as local mappings for the layer. It will use these mappings to extend or overwrite the global mappings for that layer only. This makes it possible to display different aesthetics in different layers.

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

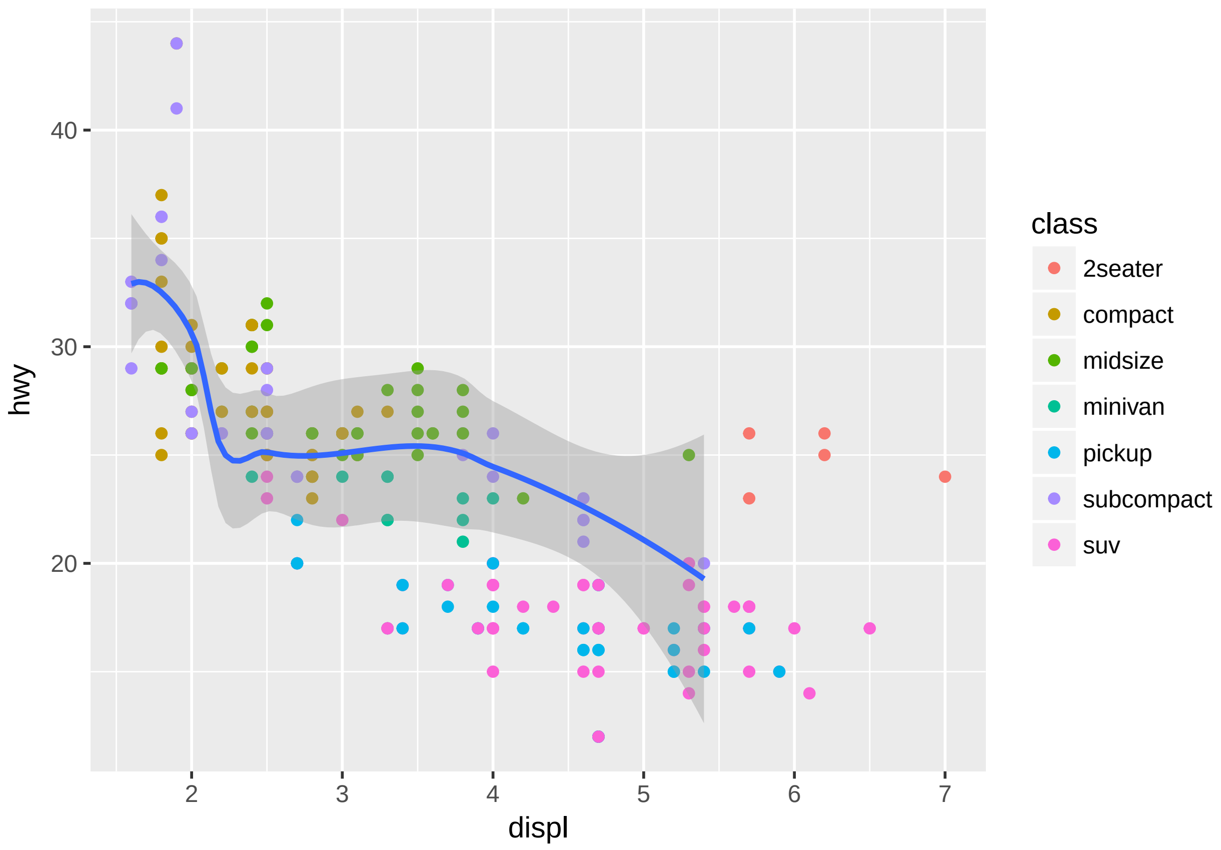


You can use the same idea to specify different data for each layer. Here, our smooth line displays just a subset of the mpg dataset, the subcompact cars. The local data argument in `geom_smooth()` overrides the global data argument in `ggplot()` for that layer only.

```
> # without confidence intervall
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
+   geom_point(mapping = aes(color = class)) +
+   geom_smooth(data = filter(mpg, class == "subcompact"), se = FALSE)
```



```
> # with confidence intervall
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
+   geom_point(mapping = aes(color = class)) +
+   geom_smooth(data = filter(mpg, class == "subcompact"), se = TRUE)
```



3.6.7 Exercises and ToDos

(0) ggplot2 provides over 30 geoms, and extension packages provide even more (see https://www.ggplot2-exts.org for a sampling). The best way to get a comprehensive overview is the ggplot2 cheatsheet, which you can find at https://www.rstudio.com/wp-content/uploads/2016/11/ggplot2-cheatsheet-2.1.pdf. To learn more about any single geom, use help: e.g. ?geom_smooth.

Personal Note: The cheatsheet is really informative. All the different geoms are displayed with their parameters and with an example code. Besides the `mpg` data set of this chapter, there is also another data set used to demonstrate how to apply the ggplot2 package. This `economics` data set is a data frame with 574 rows (not 478 as in the help file is stated) and 6 variables and consists of US economic time series from July 1967 until April 2015. These data are published regularly by the Federal Reserve Bank of St. Louis.(http://research.stlouisfed.org/fred2). The FRED site (Federal Reserve Ecoonomic Data) "offers a wealth of economic data and information to promote economic education and enhance economic research. The widely used database FRED is updated regularly and allows 24/7 access to regional and national financial and economic data." Information about the `economics` data set can be found at: http://svitsrv25.epfl.ch/R-doc/library/ggplot2/html/economics.html

Together with `mpg` I will use `economics` to show how to use. I will take the examples from the cheatsheet about data visualization.

```
> economics
```

```
# A tibble: 574 × 6
         date   pce    pop psavert uempmed unemploy
       <date> <dbl>  <int>   <dbl>   <dbl>    <int>
1  1967-07-01 507.4 198712    12.5     4.5     2944
2  1967-08-01 510.5 198911    12.5     4.7     2945
```

```
3  1967-09-01 516.3 199113    11.7     4.6     2958
4  1967-10-01 512.9 199311    12.5     4.9     3143
5  1967-11-01 518.1 199498    12.5     4.7     3066
6  1967-12-01 525.8 199657    12.1     4.8     3018
7  1968-01-01 531.5 199808    11.7     5.1     2878
8  1968-02-01 534.2 199920    12.2     4.5     3001
9  1968-03-01 544.9 200056    11.6     4.1     2877
10 1968-04-01 544.6 200208    12.2     4.6     2709
# ... with 564 more rows
```

```
> class(economics)
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

```
> head(economics)
```

```
# A tibble: 6 × 6
        date   pce    pop psavert uempmed unemploy
      <date> <dbl>  <int>   <dbl>   <dbl>    <int>
1 1967-07-01 507.4 198712    12.5     4.5     2944
2 1967-08-01 510.5 198911    12.5     4.7     2945
3 1967-09-01 516.3 199113    11.7     4.6     2958
4 1967-10-01 512.9 199311    12.5     4.9     3143
5 1967-11-01 518.1 199498    12.5     4.7     3066
6 1967-12-01 525.8 199657    12.1     4.8     3018
```

```
> tail(economics)
```

```
# A tibble: 6 × 6
        date     pce    pop psavert uempmed unemploy
      <date>   <dbl>  <int>   <dbl>   <dbl>    <int>
1 2014-11-01 12142.2 320013     4.5    12.8     9071
2 2014-12-01 12122.0 320201     5.0    12.6     8688
3 2015-01-01 12080.8 320367     5.5    13.4     8979
4 2015-02-01 12095.9 320534     5.7    13.1     8705
5 2015-03-01 12161.5 320707     5.2    12.2     8575
6 2015-04-01 12158.9 320887     5.6    11.7     8549
```

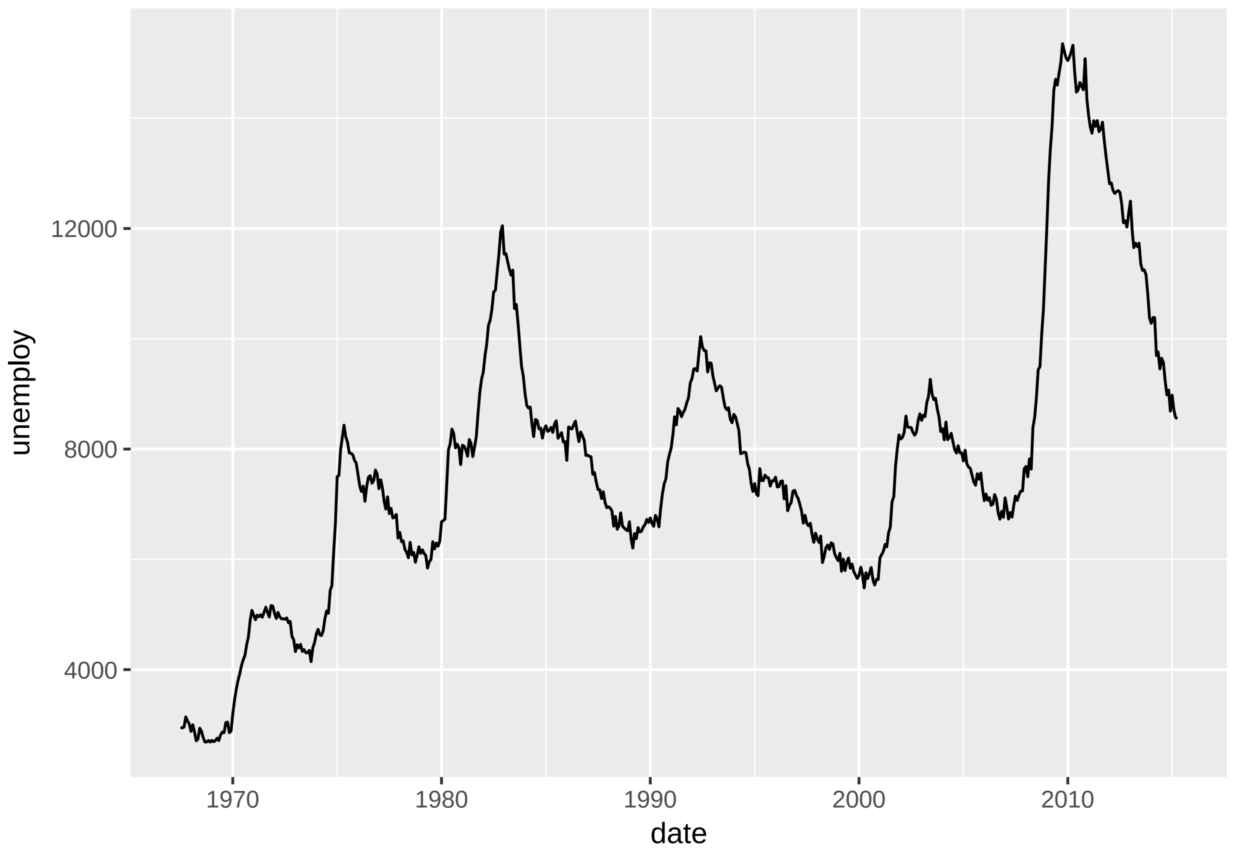
```
> str(economics)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   574 obs. of  6 variables:
 $ date    : Date, format: "1967-07-01" "1967-08-01" ...
 $ pce     : num  507 510 516 513 518 ...
 $ pop     : int  198712 198911 199113 199311 199498 199657 199808 199920 200056 200208 ...
 $ psavert : num  12.5 12.5 11.7 12.5 12.5 12.1 11.7 12.2 11.6 12.2 ...
 $ uempmed : num  4.5 4.7 4.6 4.9 4.7 4.8 5.1 4.5 4.1 4.6 ...
 $ unemploy: int  2944 2945 2958 3143 3066 3018 2878 3001 2877 2709 ...
```

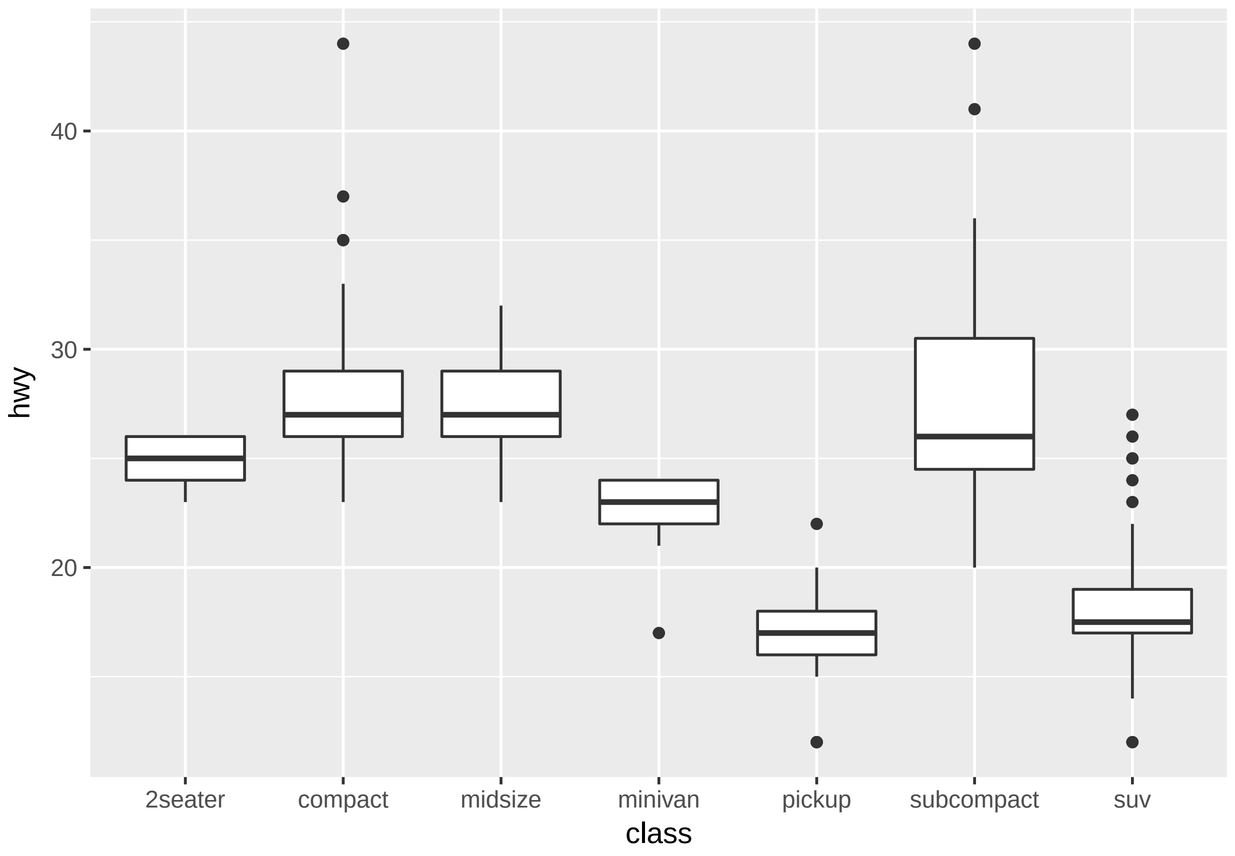
(1) What geom would you use to draw a line chart? A boxplot? A histogram? An area chart?

My Solution: geom_line(), geom_boxplot(), geom_histogram(), geom_area().

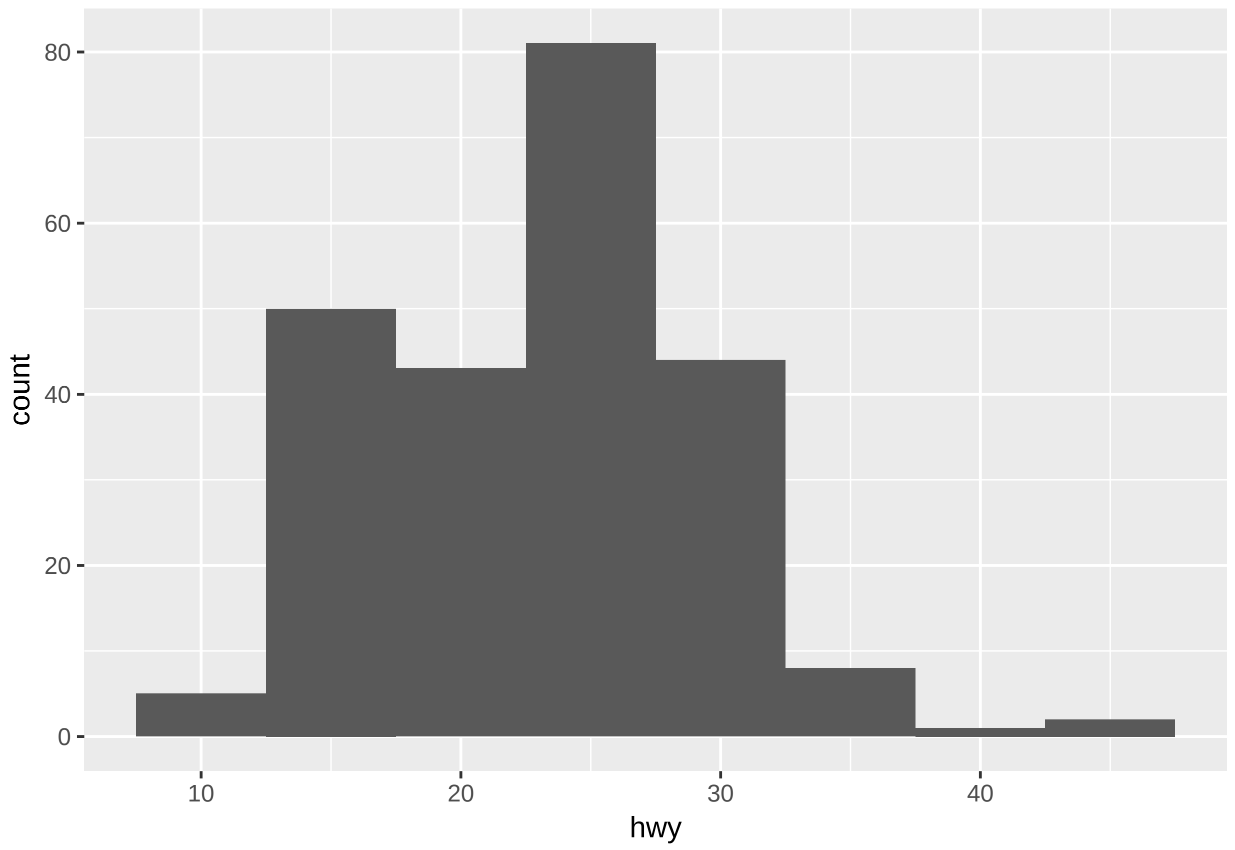
```
> ggplot(economics, aes(date, unemploy)) +
+         geom_line()
```



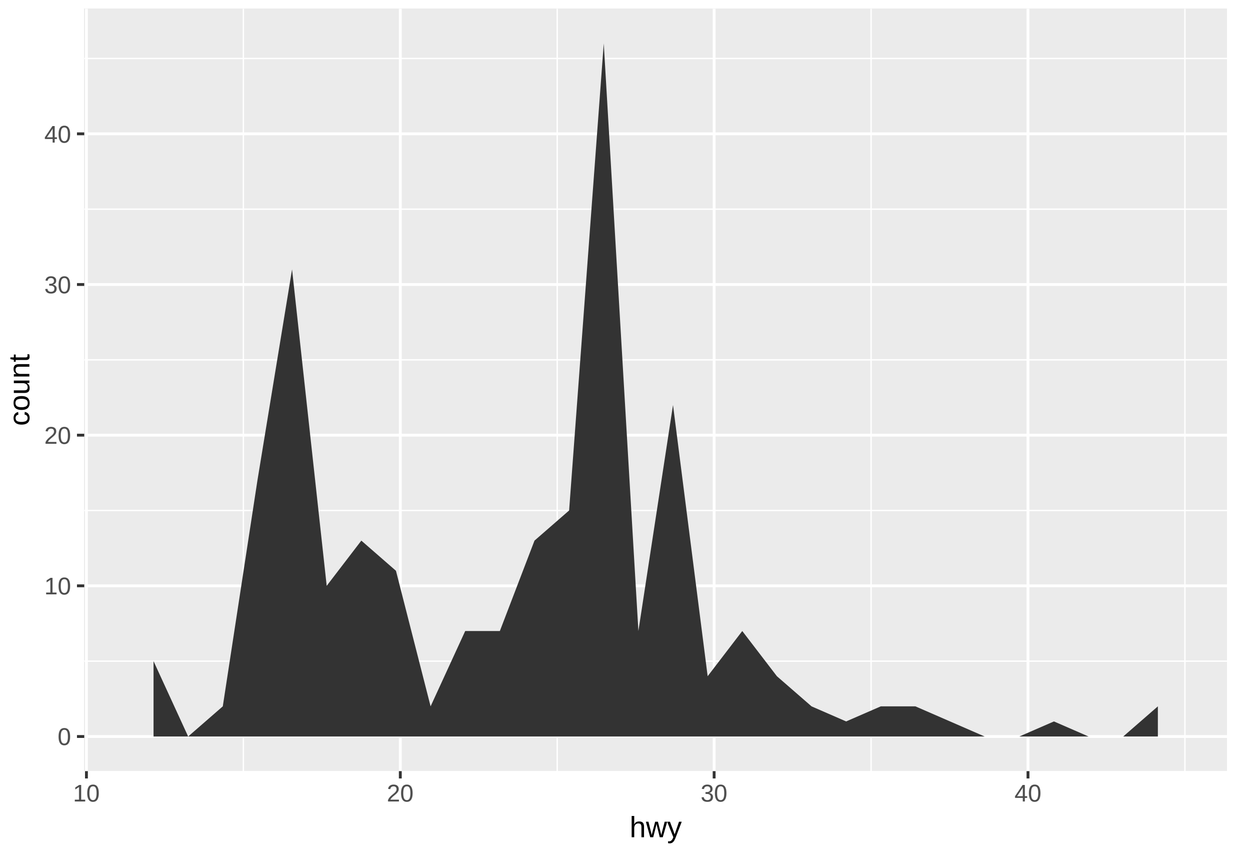
```
> ggplot(mpg, aes(class, hwy)) +
+         geom_boxplot()
```



```
> ggplot(mpg, aes(hwy)) +
+         geom_histogram(binwidth = 5)
```

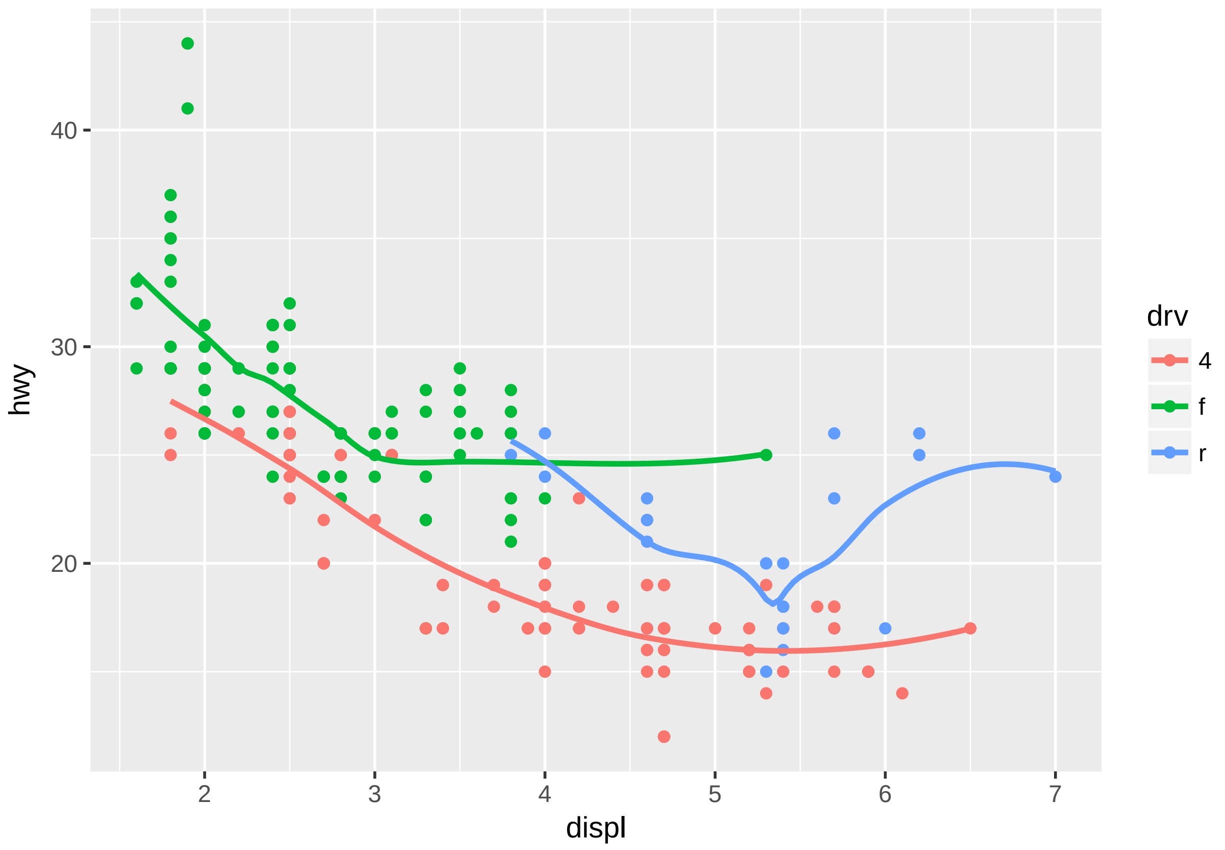


```
> ggplot(mpg, aes(hwy)) +
+         geom_area(stat="bin")
```



(2) Run this code in your head and predict what the output will look like. Then, run the code in R and check your predictions.

```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +
+   geom_point() +
+   geom_smooth(se = FALSE)
```



(3) What does show.legend = FALSE do? What happens if you remove it? Why do you think I used it earlier in the chapter?

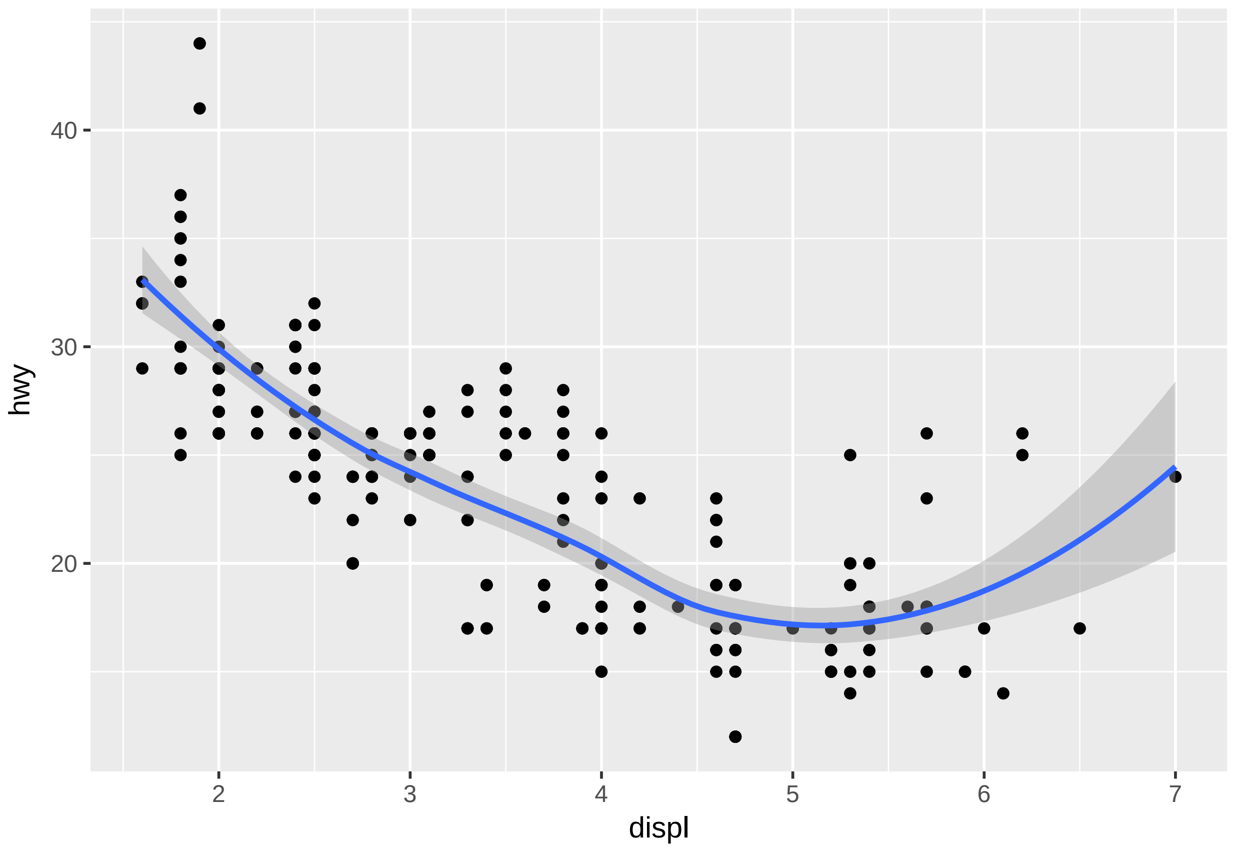
I cannot find any differences with T, F or NA??

(4) What does the se argument to geom_smooth() do?

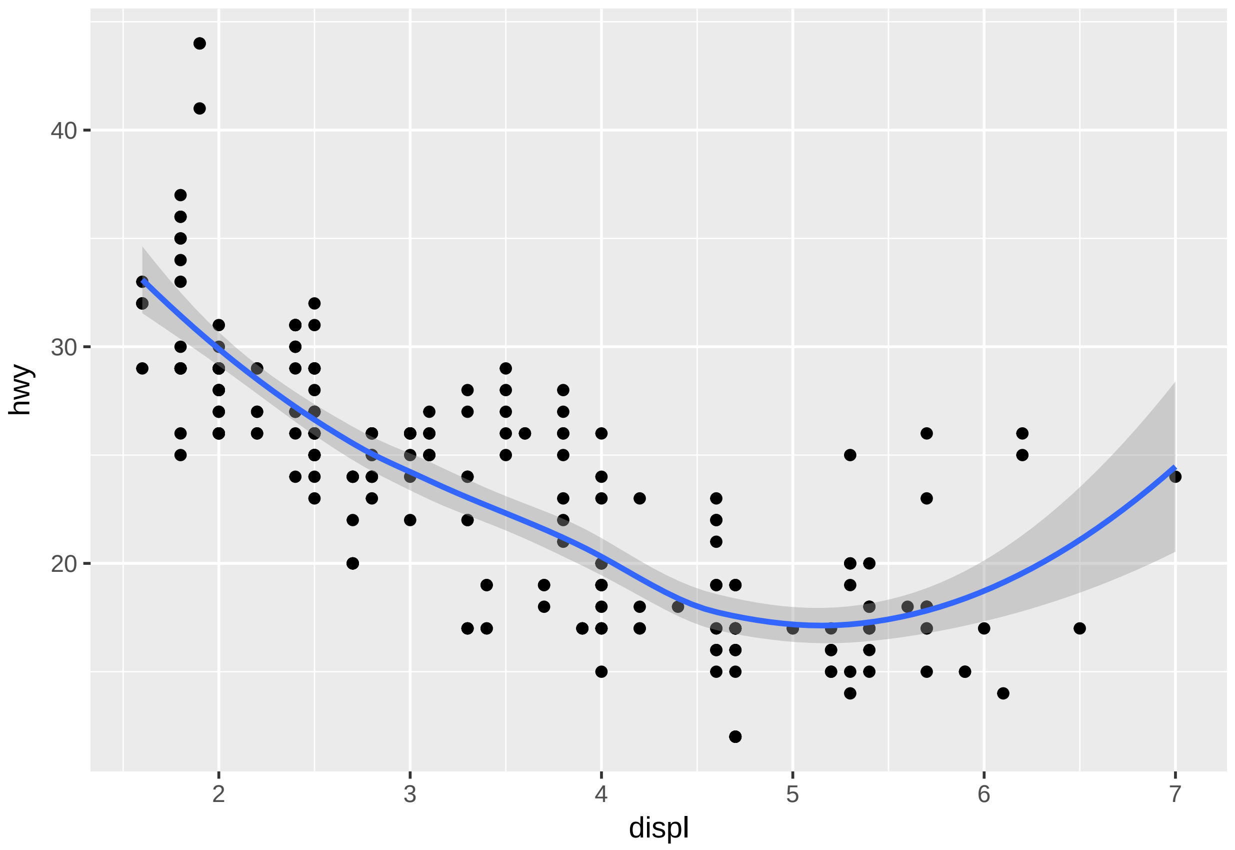
My Solution: se = TRUE shows the confidence intervall with geom_smoth().

(5) Will these two graphs look different? Why/why not?

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



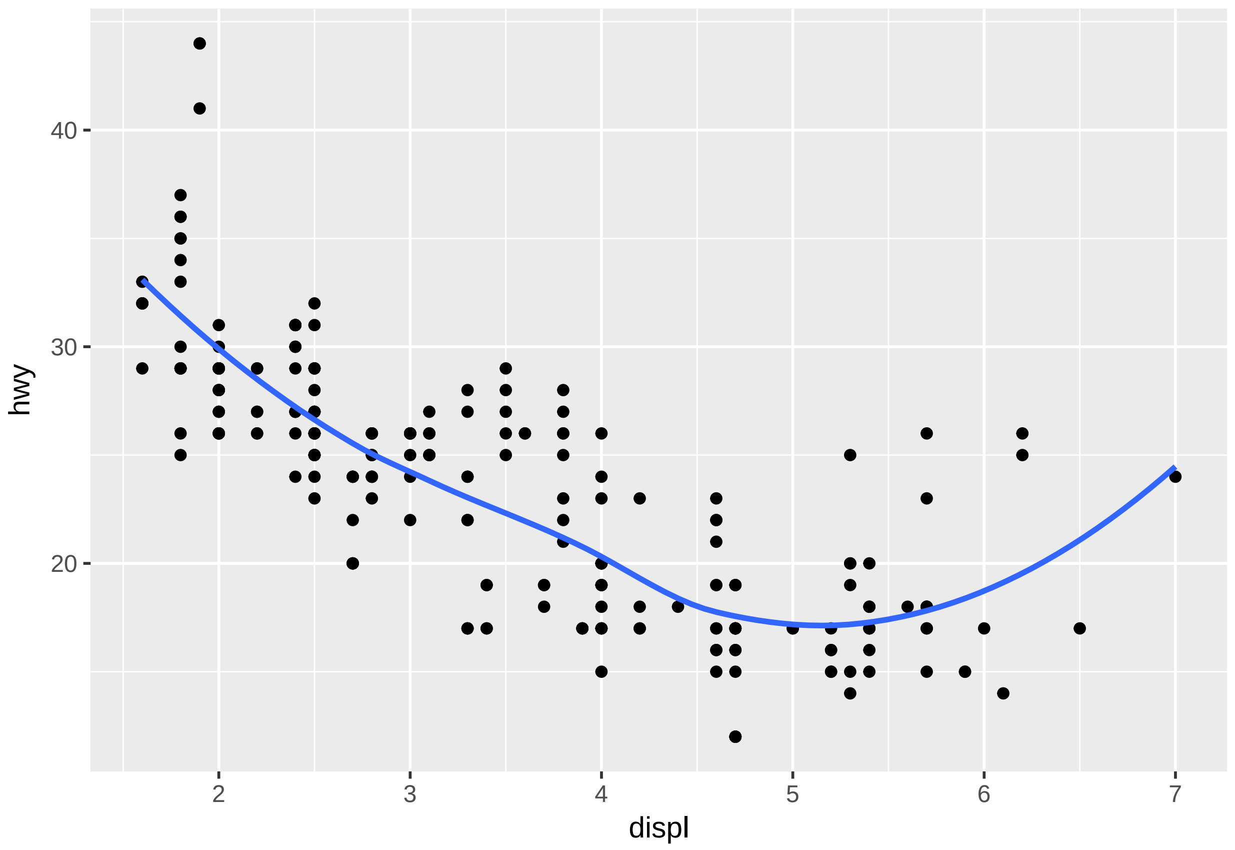
```
> ggplot() +
+   geom_point(data = mpg, mapping = aes(x = displ, y = hwy)) +
+   geom_smooth(data = mpg, mapping = aes(x = displ, y = hwy))
```



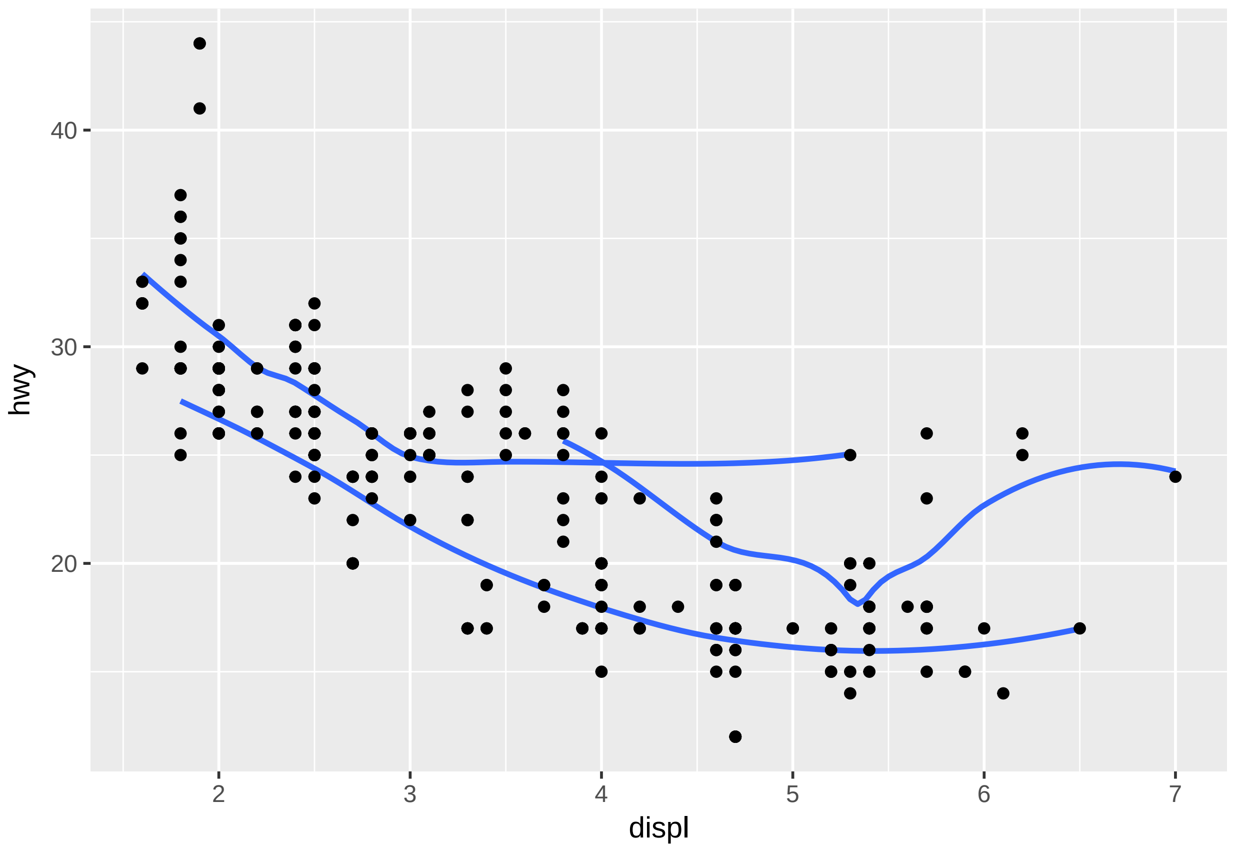
My Solution: Both graphs look alike: In the first case there is global mapping in the second case it is local mapping.

(6) Recreate the R code necessary to generate the graphs presented in the book:

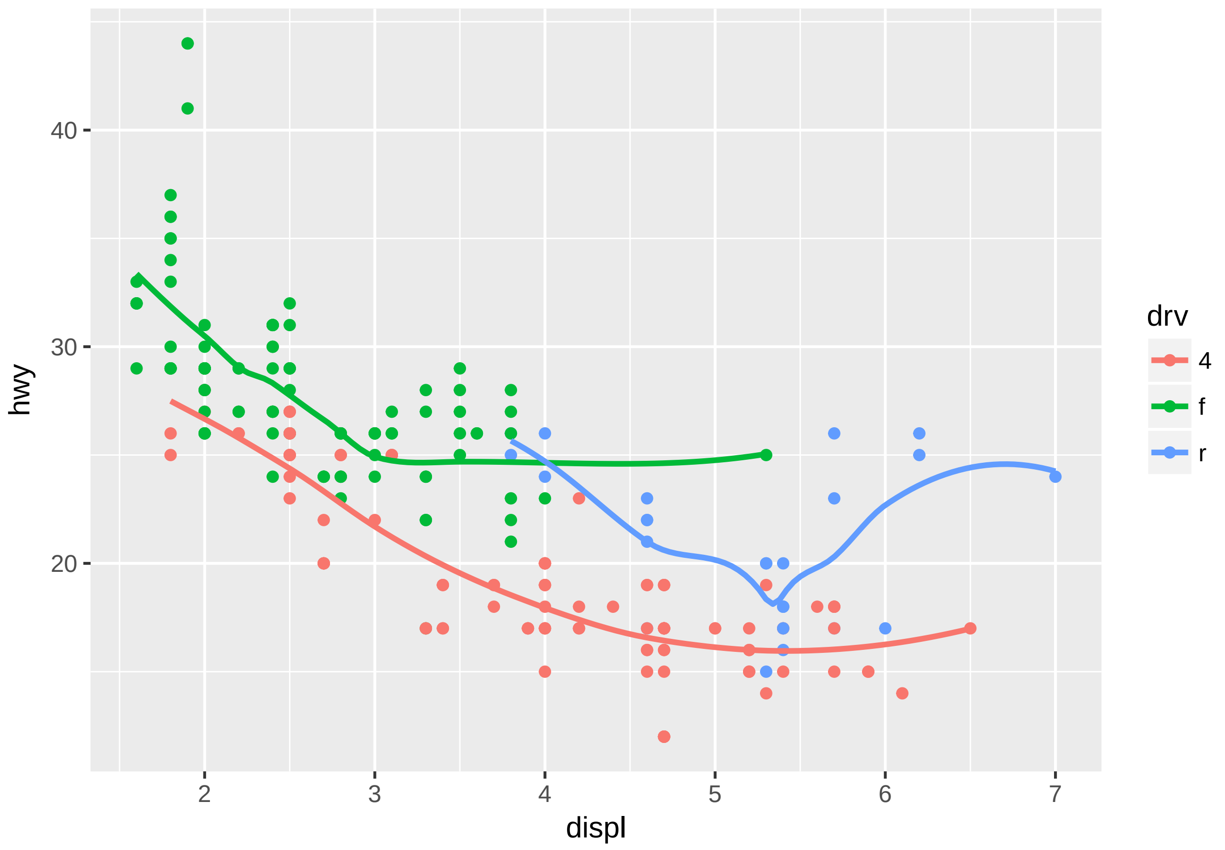
```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
+   geom_point() +
+   geom_smooth(se = FALSE)
```



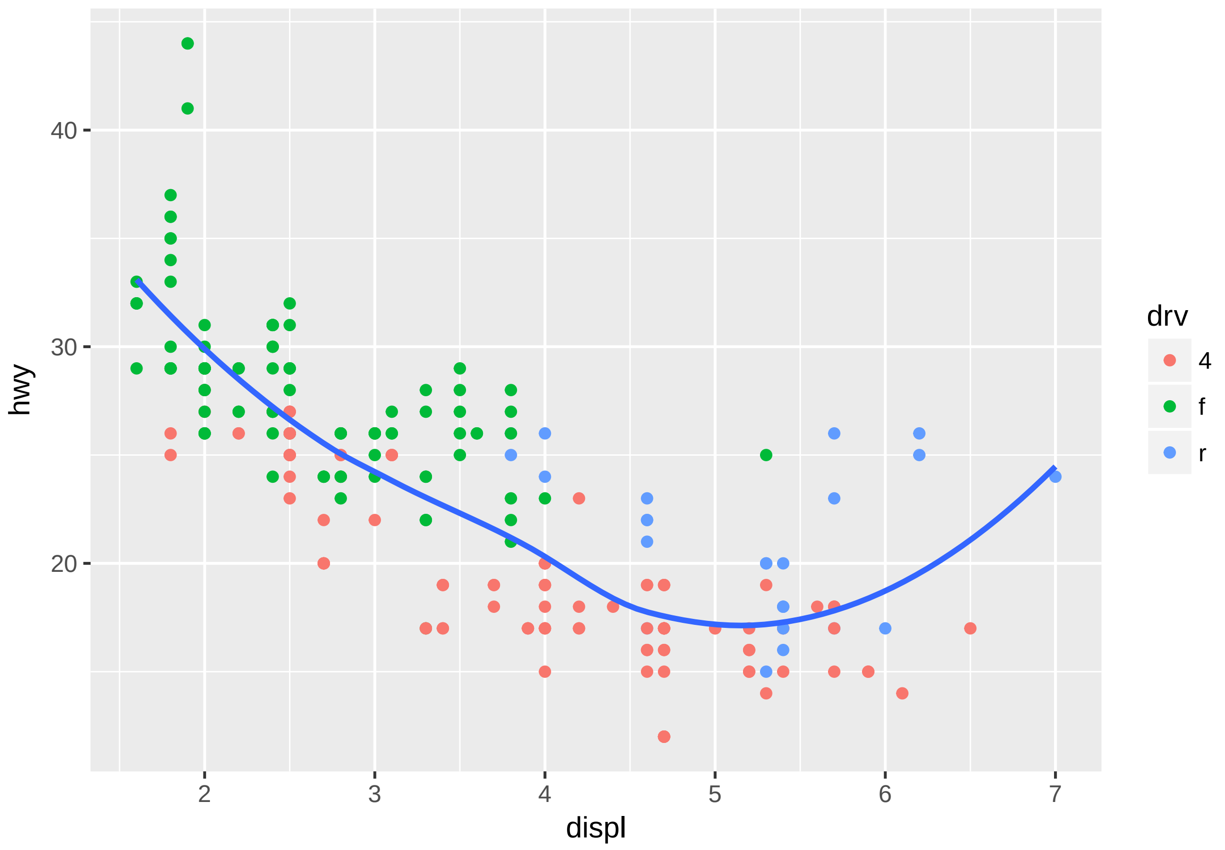
```
> # ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
> #   geom_point() +
> #   geom_smooth(mapping = aes(group = drv), se = FALSE)
> #   # mapping is not necessary, because it is already definined globally
>
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
+   geom_smooth(aes(group = drv), se = FALSE) +
+   geom_point()
```



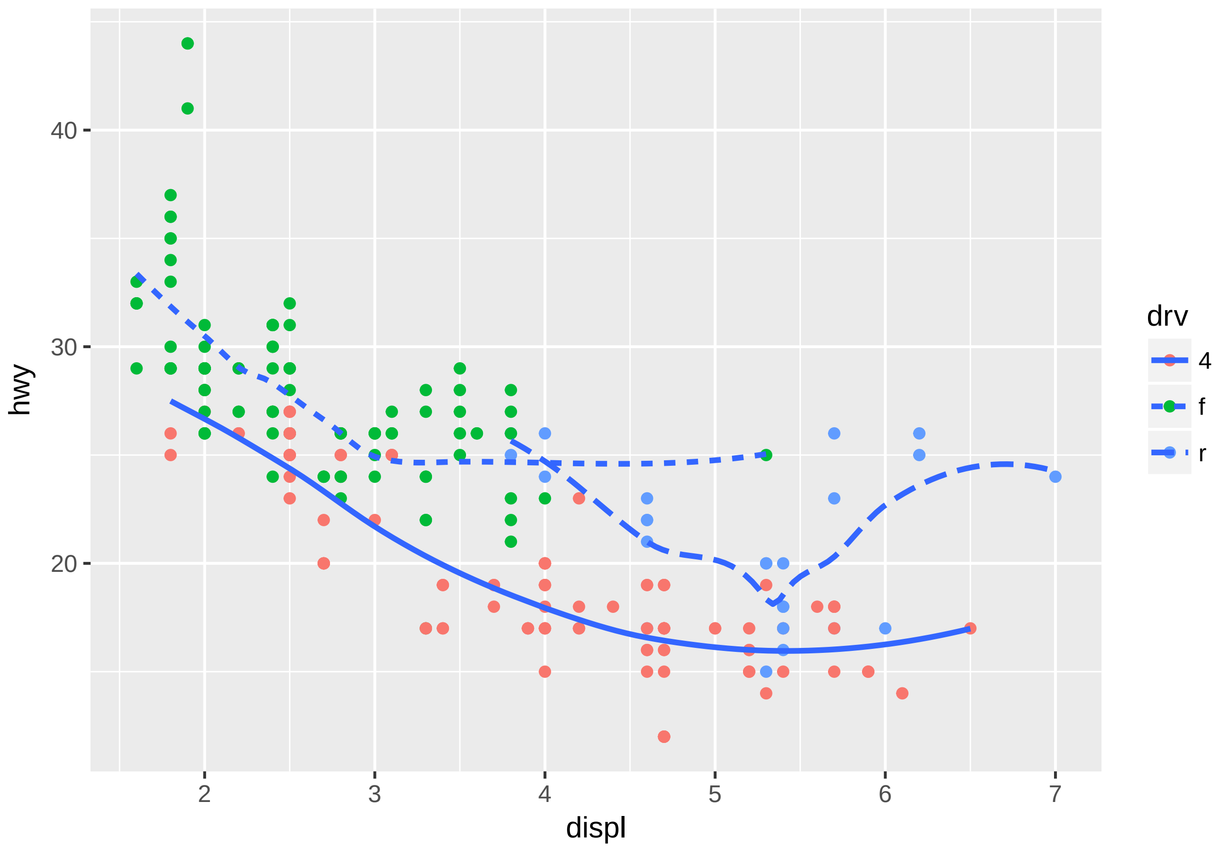
```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +
+   geom_point() +
+   geom_smooth(se = FALSE)
```



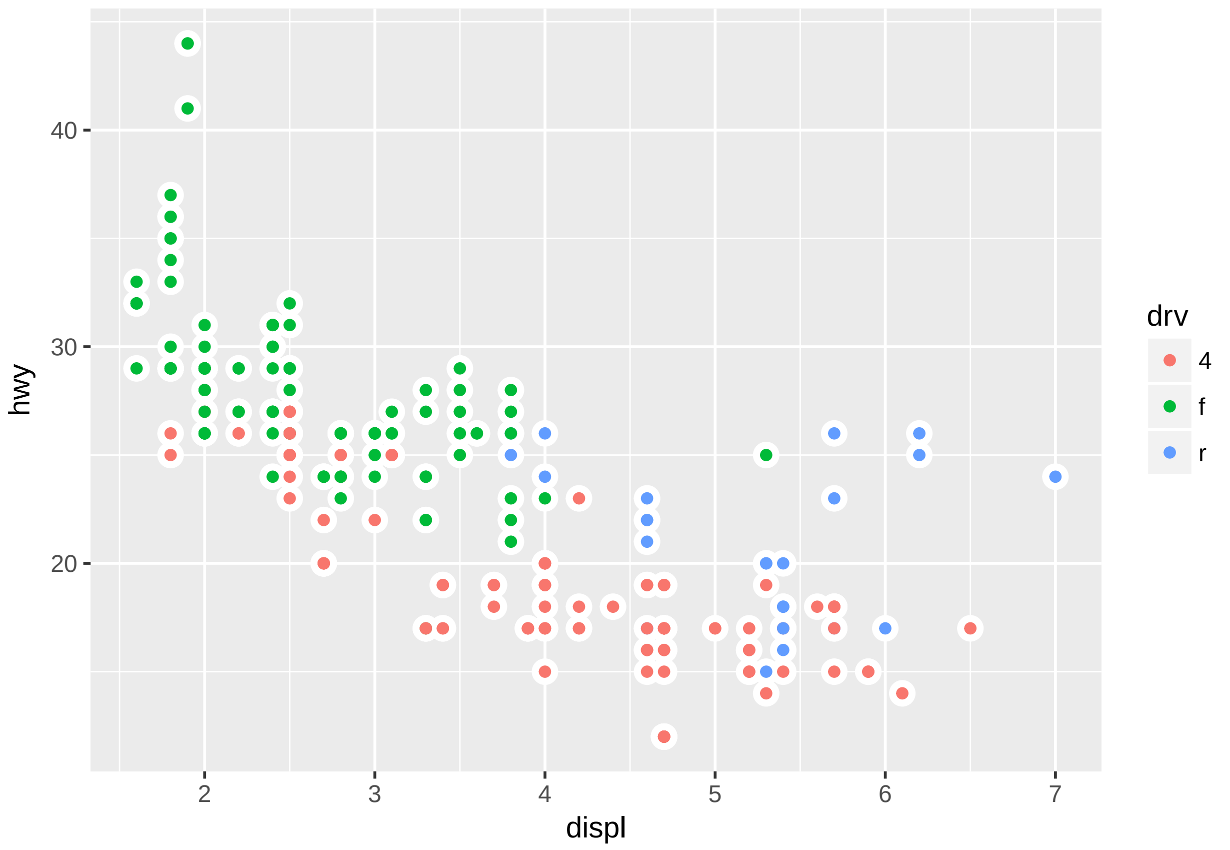
```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
+   geom_point(mapping =  aes(color = drv)) + # mapping is not necessary
+   geom_smooth(se = FALSE)
```



```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
+   geom_point(mapping =  aes(color = drv)) + # mapping is not necessary
+   geom_smooth(mapping =  aes(linetype = drv), se = FALSE) # mapping is not necessary
```



```
> ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +
+   geom_point(size = 4, color = "white") +
+   geom_point(aes(colour = drv))
```



#

3.7 Statistical transformation

3.7.1 Example bar chart (geom_bar)

```
> diamonds
```

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

```
> class(diamonds)
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

```
> head(diamonds)
```

```
# A tibble: 6 × 10
```

```
  carat       cut color clarity depth table price     x     y     z
  <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23     Ideal     E     SI2  61.5    55   326  3.95  3.98  2.43
2  0.21   Premium     E     SI1  59.8    61   326  3.89  3.84  2.31
3  0.23      Good     E     VS1  56.9    65   327  4.05  4.07  2.31
4  0.29   Premium     I     VS2  62.4    58   334  4.20  4.23  2.63
5  0.31      Good     J     SI2  63.3    58   335  4.34  4.35  2.75
6  0.24 Very Good     J    VVS2  62.8    57   336  3.94  3.96  2.48
```

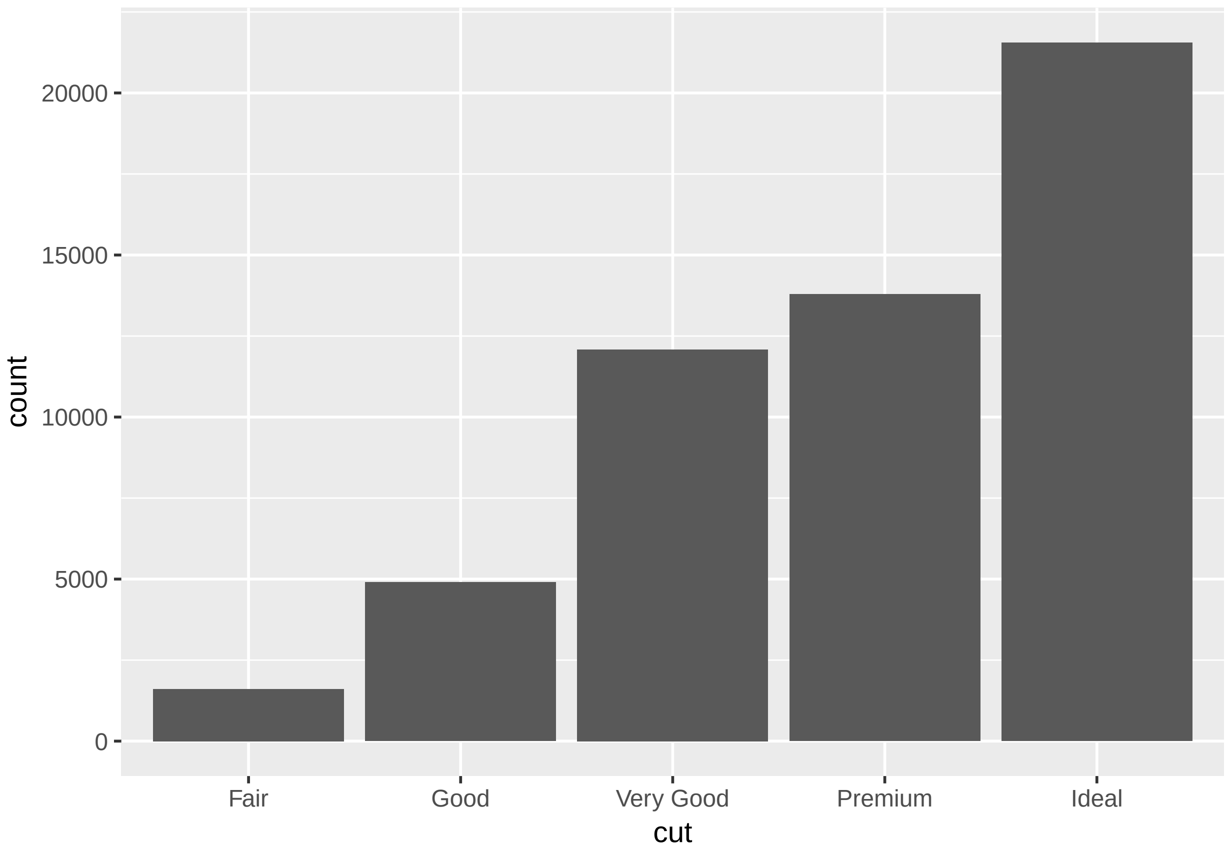
```
> tail(diamonds)
```

```
# A tibble: 6 × 10
  carat       cut color clarity depth table price     x     y     z
  <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.72   Premium     D     SI1  62.7    59  2757  5.69  5.73  3.58
2  0.72     Ideal     D     SI1  60.8    57  2757  5.75  5.76  3.50
3  0.72      Good     D     SI1  63.1    55  2757  5.69  5.75  3.61
4  0.70 Very Good     D     SI1  62.8    60  2757  5.66  5.68  3.56
5  0.86   Premium     H     SI2  61.0    58  2757  6.15  6.12  3.74
6  0.75     Ideal     D     SI2  62.2    55  2757  5.83  5.87  3.64
```

```
> str(diamonds)
```

```
Classes 'tbl_df', 'tbl' and 'data.frame':   53940 obs. of  10 variables:
 $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
 $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
 $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
 $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
 $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
 $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
 $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
 $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
 $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
 $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

```
> ggplot(data = diamonds) +
+   geom_bar(mapping = aes(x = cut))
```



Many graphs, like scatterplots, plot the raw values of your dataset. Other graphs, like bar charts, calculate new values to plot:

- bar charts, histograms, and frequency polygons bin your data and then plot bin counts, the number of points that fall in each bin.
- smoothers fit a model to your data and then plot predictions from the model.
- boxplots compute a robust summary of the distribution and then display a specially formatted box.

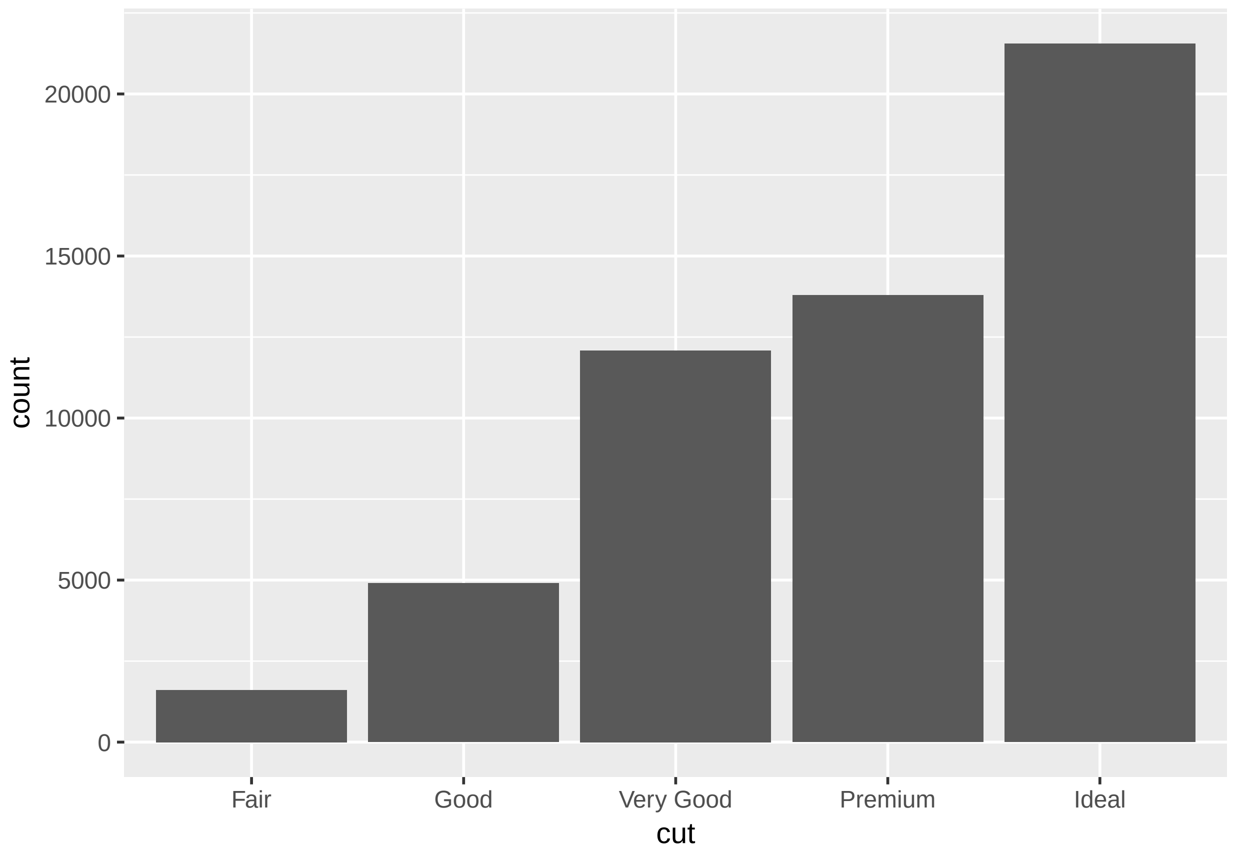
The algorithm used to calculate new values for a graph is called a **stat**, short for statistical transformation. You can learn which stat a geom uses by inspecting the default value for the `stat` argument.

3.7.2 stat and geom

You can learn which stat a geom uses by inspecting the default value for the stat argument. For example, `?geom_bar shows the default value for`stat`is "count", which means that`geom_bar()`uses`stat_count()`.`stat_count(`documented on the same page as`geom_bar()`, and if you scroll down you can find a section called "Computed variables". That tells that it computes two new variables:`count`and`prop‘.

You can generally use geoms and stats interchangeably. For example, you can recreate the previous plot using `stat_count()` instead of ‘geom_bar():

```
> ggplot(data = diamonds) +
+   stat_count(mapping = aes(x = cut))
```



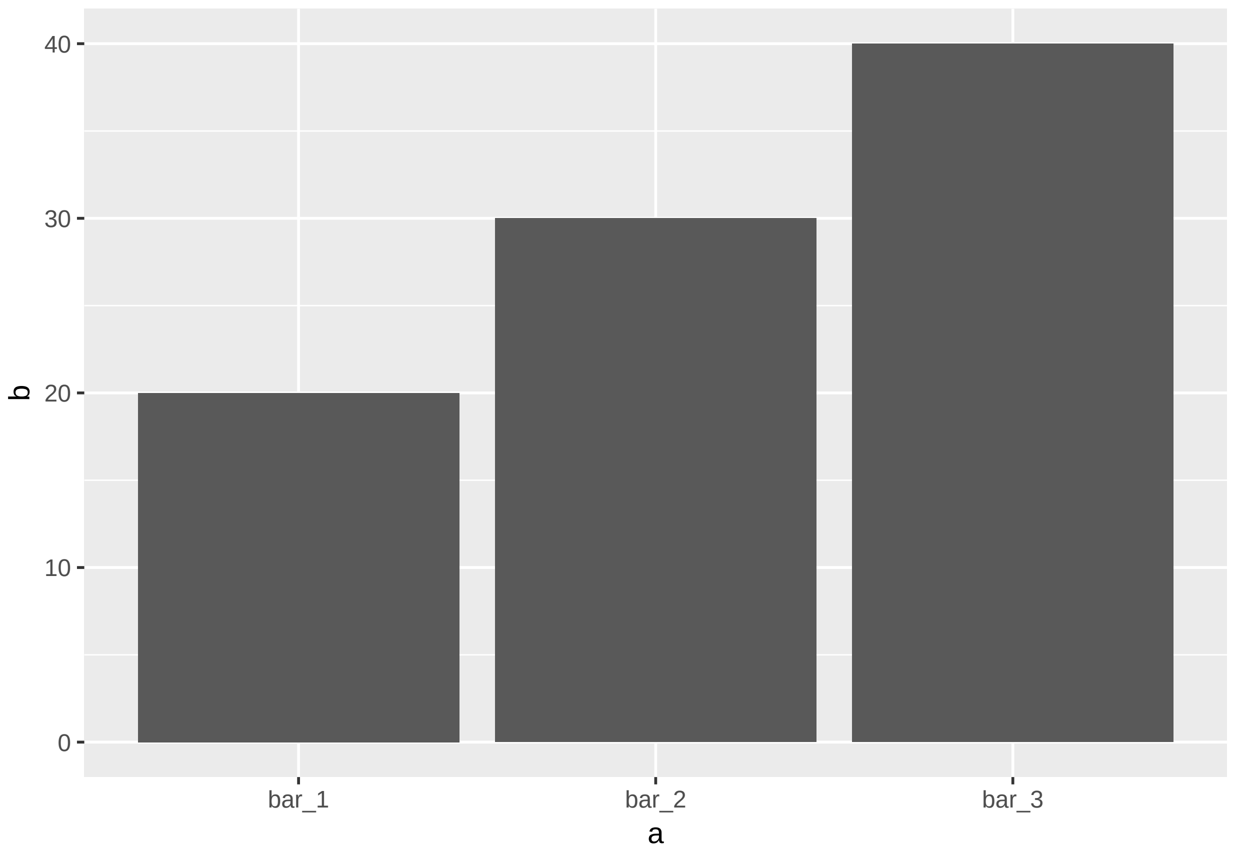
This works because every geom has a default stat; and every stat has a default geom. This means that you can typically use geoms without worrying about the underlying statistical transformation.

3.7.3 Reasons for using stat explicitly

There are three reasons you might need to use a stat explicitly:

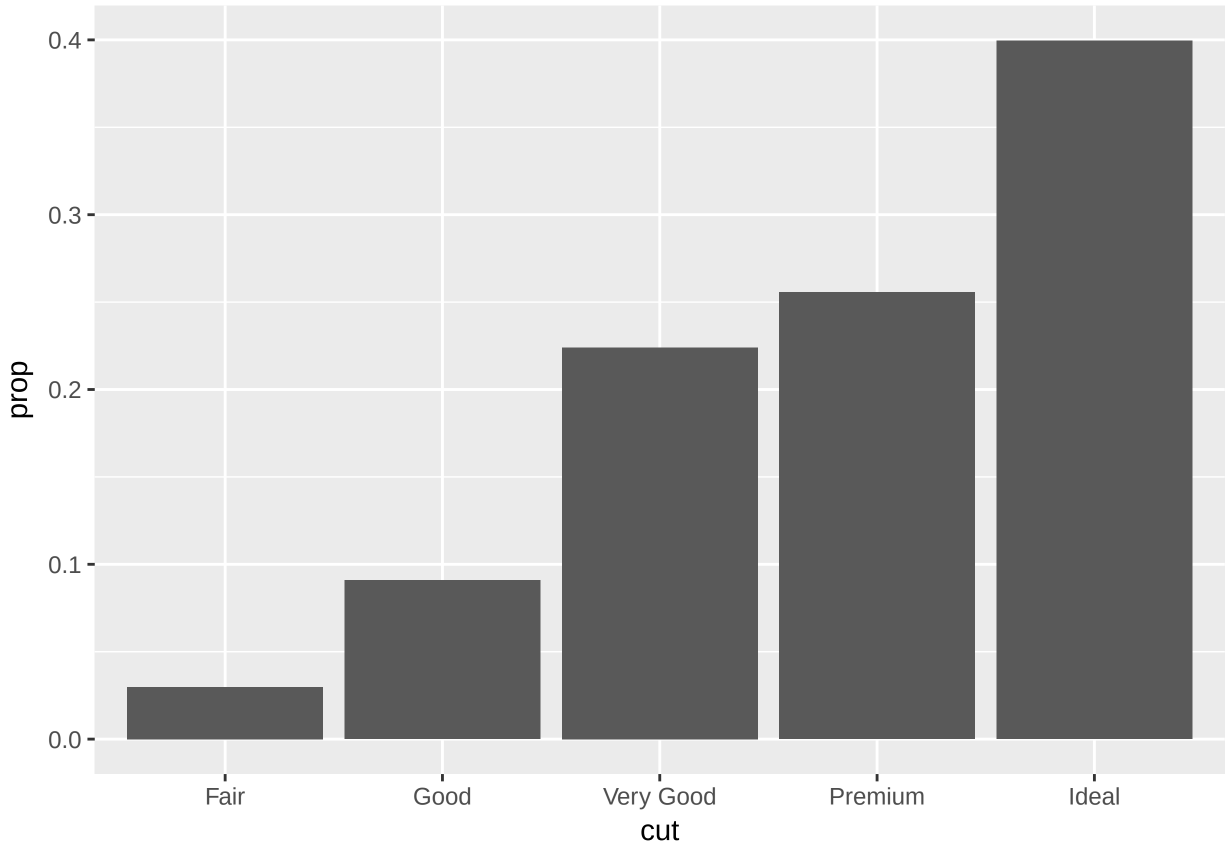
1. You might want to override the default stat. In the code below, I change the stat of `geom_bar()` from `count` (the default) to ‘identity. This lets me map the height of the bars to the raw values of a *y* variable. Unfortunately when people talk about bar charts casually, they might be referring to this type of bar chart, where the height of the bar is already present in the data, or the previous bar chart where the height of the bar is generated by counting rows.

```
> demo <- tribble(
+   ~a,      ~b,
+   "bar_1", 20,
+   "bar_2", 30,
+   "bar_3", 40
+ )
>
> ggplot(data = demo) +
+   geom_bar(mapping = aes(x = a, y = b), stat = "identity")
```



2. You might want to override the default mapping from transformed variables to aesthetics. For example, you might want to display a bar chart of proportion, rather than count:

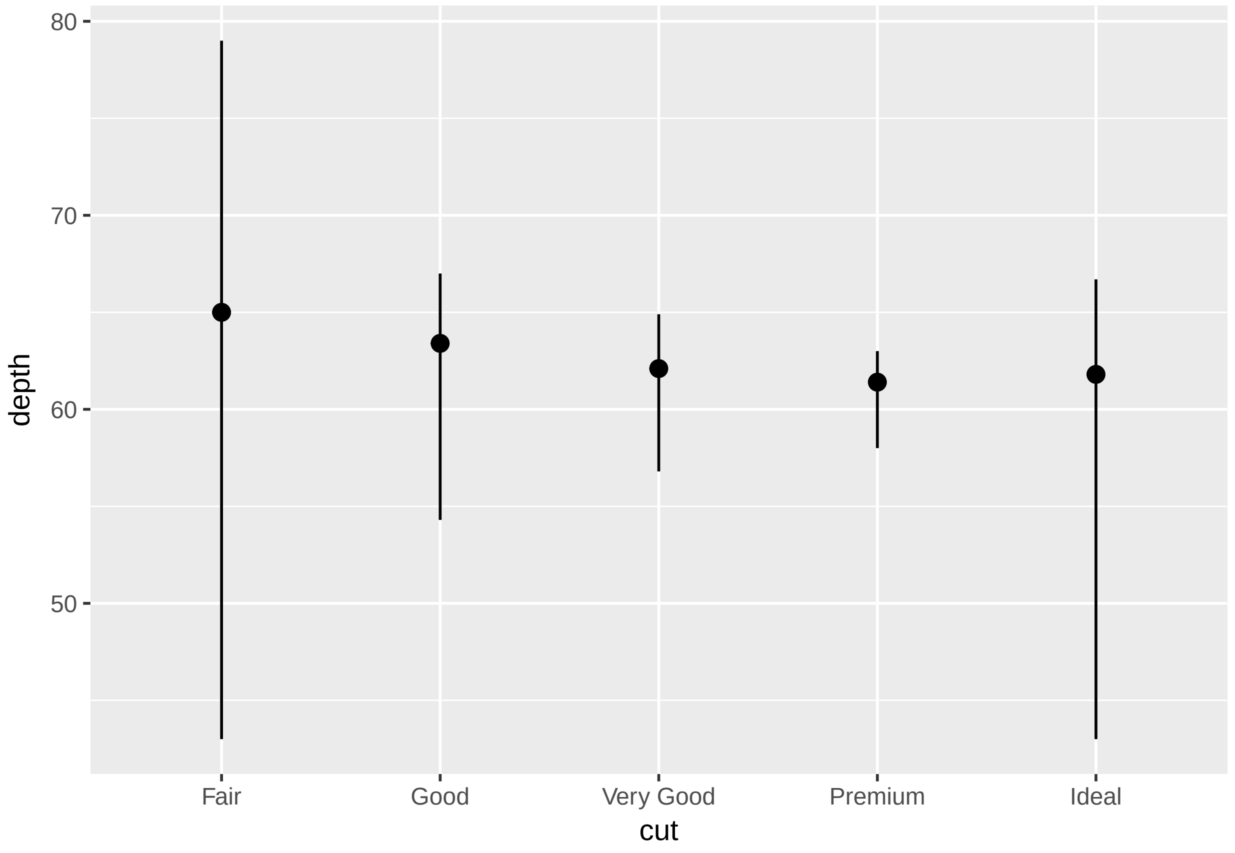
```
> ggplot(data = diamonds) +
+   geom_bar(mapping = aes(x = cut, y = ..prop.., group = 1))
```



```
> ## The double dot is signal for `ggpolot2` to do it's own internal computation of the value rather tha
```

3. You might want to draw greater attention to the statistical transformation in your code. For example, you might use ‘stat_summary(), which summarises the y values for each unique x value, to draw attention to the summary that you’re computing:

```
> ggplot(data = diamonds) +
+   stat_summary(
+     mapping = aes(x = cut, y = depth),
+     fun.ymin = min,
+     fun.ymax = max,
+     fun.y = median
+   )
```



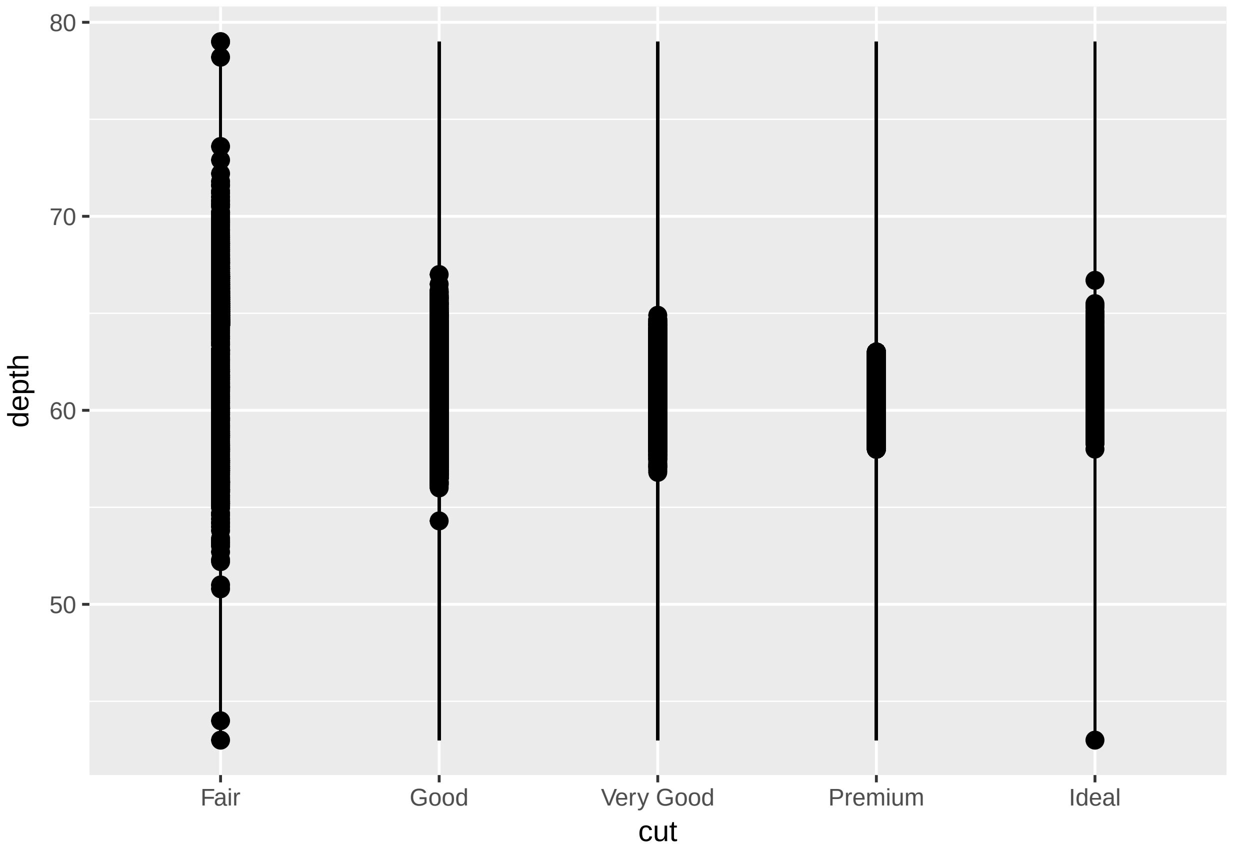
ggplot2 provides over 20 stats for you to use. Each stat is a function, so you can get help in usual way, e.g. `?stat_bin`. To see a complete list of stats, try the ggplot2 cheatsheet.

3.7.4 Exercises

(1) 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?

My Solution: The default geom associated with `stat_summary()` is geom_pointrange (or geom_linerange)?

```
> ggplot(data = diamonds) +
+   geom_pointrange(
+     mapping = aes(x = cut, y = depth, ymin = min(depth), ymax = max(depth)))
```



(2) What does `geom_col()` do? How is it different `to geom_bar()`?

My Solution: "`geom_bar` uses `stat_count` by default: it counts the number of cases at each x position. `geom_col` uses `stat_identity`: it leaves the data as is." In `geom_bar` are the heigths of the bars porpotional to the number of cases in each group, in `geom_col` the height of the bars represent values in the data.

(3) Most geoms and stats come in pairs that are almost always used in concert. Read through the documentation and make a list of all the pairs. What do they have in common?

My Solution:

1. geom_abline = Reference lines: horizontal, vertical, and diagonal
2. geom_bar <-> stat_count = Bars charts
3. geom_bin2d <-> stat_bin_2d = Heatmap of 2d bin counts
4. geom_blank = Draw nothing
5. geom_boxplot <-> stat_boxplot = A box and whiskers plot (in the style of Tukey)
6. geom_contour <-> stat_contour = 2d contours of a 3d surface
7. geom_count <-> stat_sum = Count overlapping points
8. geom_density <-> stat_density = Smoothed density estimates
9. geom_density_2d <-> stat_density_2d = Contours of a 2d density estimate
10. geom_dotplot = Dot plot
11. geom_errorbarh = Horizontal error bars
12. geom_hex <-> stat_bin_hex = Hexagonal heatmap of 2d bin counts
13. geom_histogram/freqpoly <-> stat_bin = Histograms and frequency polygons
14. geom_jitter = Jittered points
15. geom_crossbar = Vertical intervals: lines, crossbars & errorbars
16. geom_map = Polygons from a reference map
17. geom_path = Connect observations
18. geom_point = Points

19. geom_polygon = Polygons
20. geom_qq <-> stat_qq = A quantile-quantile plot
21. geom_quantile <-> tat_quantile = Quantile regression
22. geom_ribbon = Ribbons and area plots
23. geom_rug = Rug plots in the margins
24. geom_segment = Line segments and curves
25. geom_smooth <-> stat_smooth = Smoothed conditional means
26. geom_spoke = Line segments parameterised by location, direction and distance
27. geom_label = Text
28. geom_raster = Rectangles
29. geom_violin <-> stat_ydensity = Violing plot
30. geom_label_repel = Repulsive textual annotations.
31. geom_ref_line = Add a reference line

18a. geom_pointrange <-> stat_summary Summarise y values at unique/binned x

(4) What variables does stat_smooth() compute? What parameters control its behaviour?

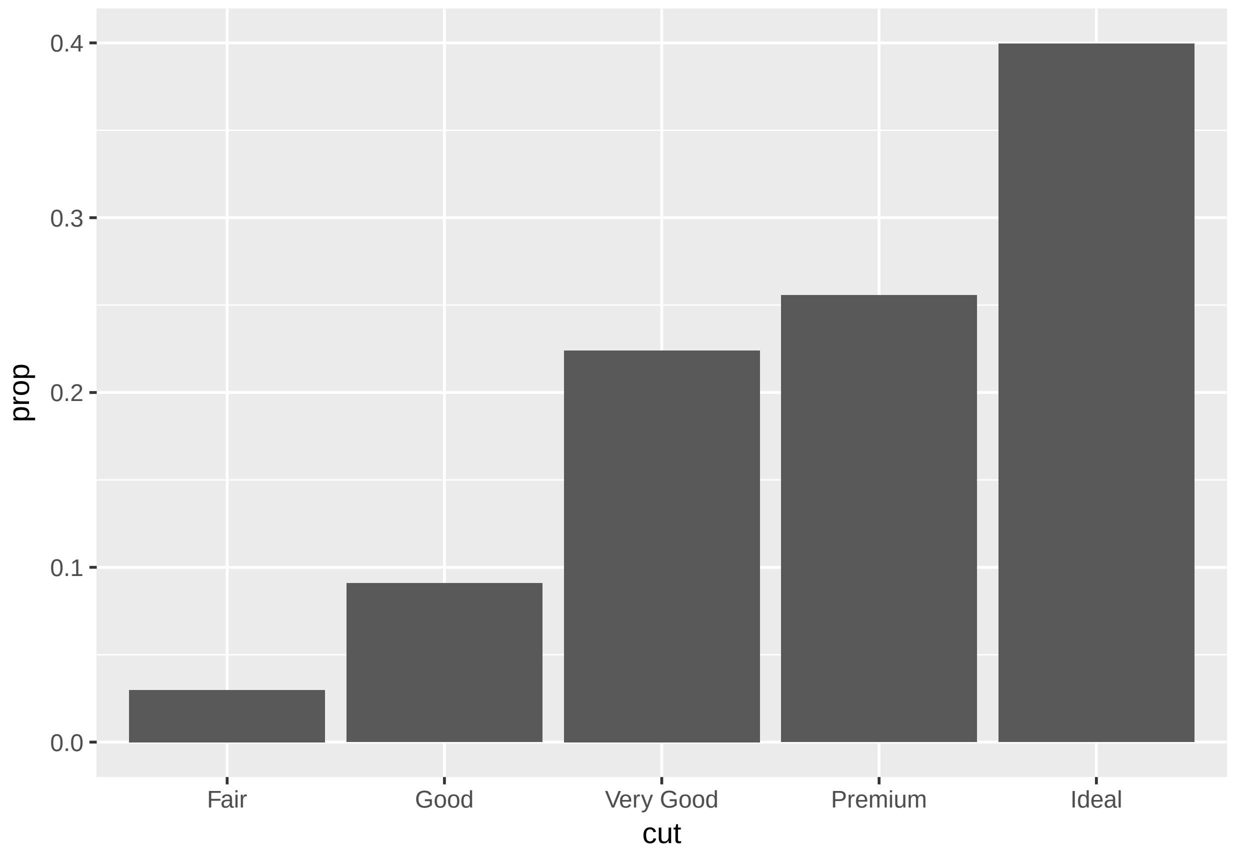
My Solution:

- Computed variables: y (predicted value), ymin (lower pointwise confidence interval around the mean), ymax (upper pointwise confidence interval around the mean), se (standard error).
- position, method, formula, se, n, span, fullrange, level, na.rm, show.legend, inherit.aes

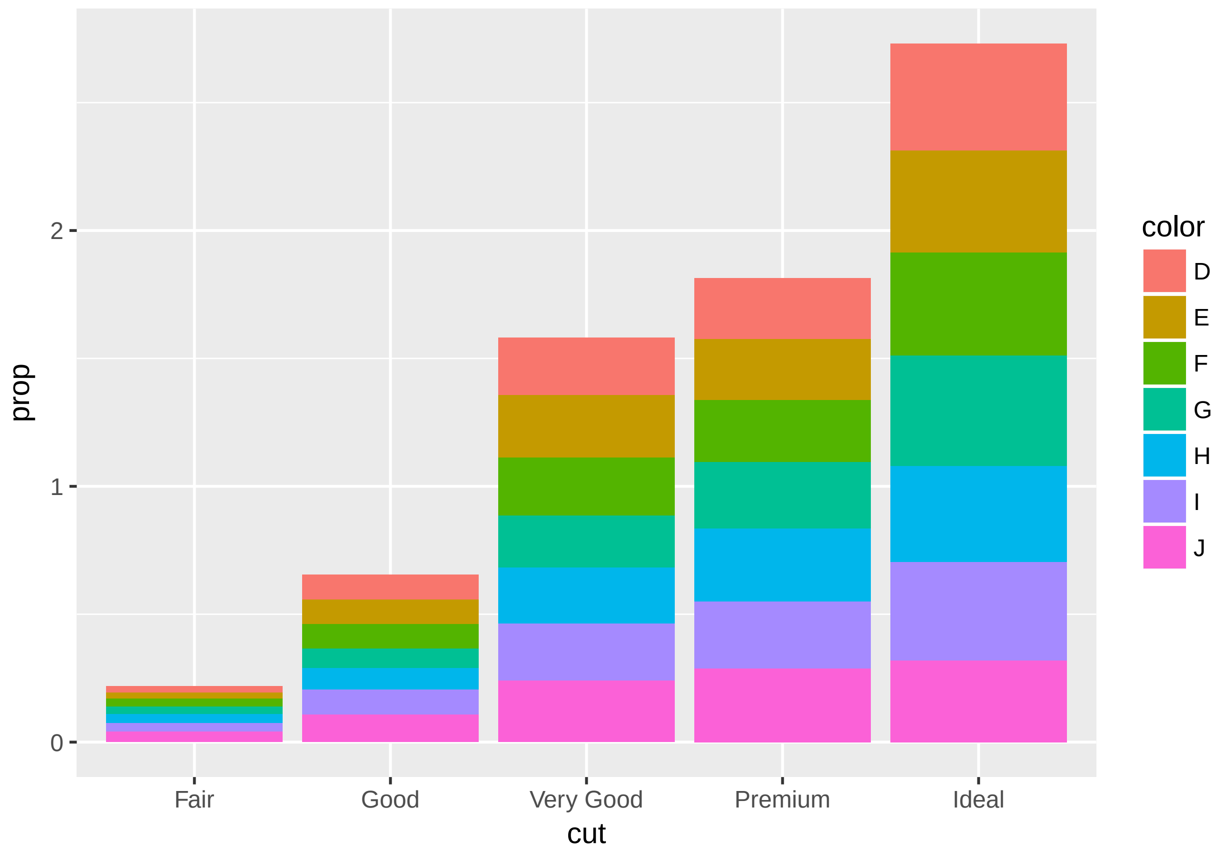
(5) In our proportion bar chart, we need to set group = 1. Why? In other words what is the problem with these two graphs?

My Solution:

```
> ggplot(data = diamonds) +
+   geom_bar(mapping = aes(x = cut, y = ..prop.., group = 1)) # needs group argument
```



```
> ggplot(data = diamonds) +
+   geom_bar(mapping = aes(x = cut, fill = color, y = ..prop.., group = color))
```

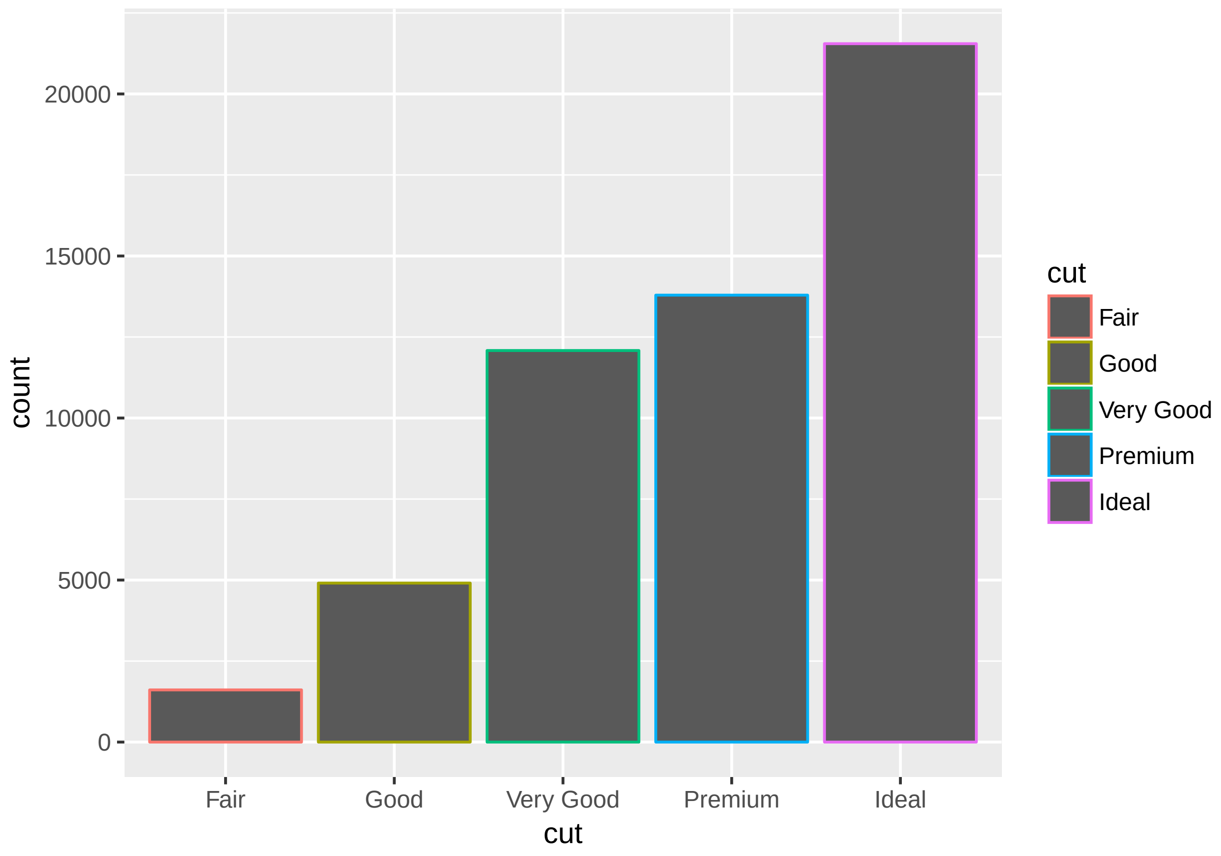


3.8 Position adjustments

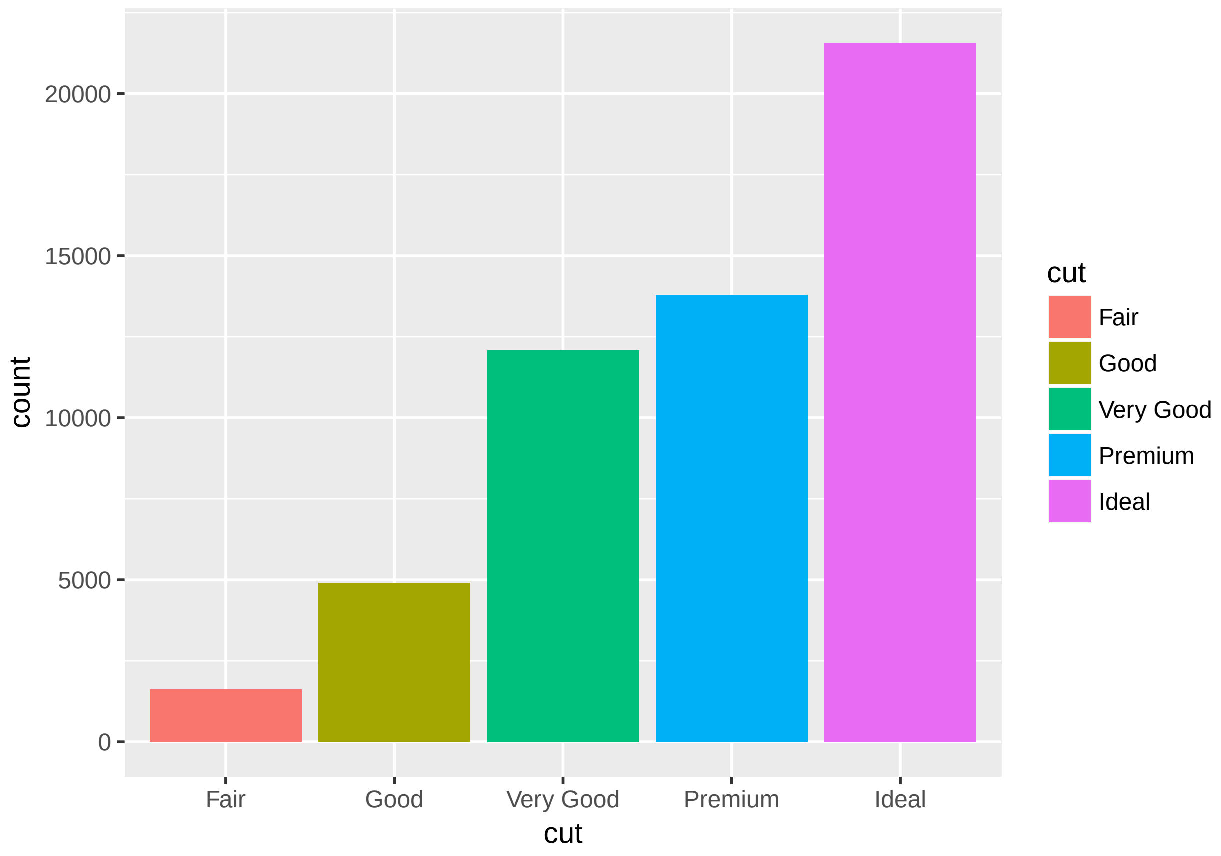
3.8.1 Coloring a bar chart

There's one more piece of magic associated with bar charts. You can colour a bar chart using either the `colour` aesthetic, or more usefully, `fill`:

```
> ggplot(data = diamonds) +
+   geom_bar(mapping = aes(x = cut, colour = cut))
```



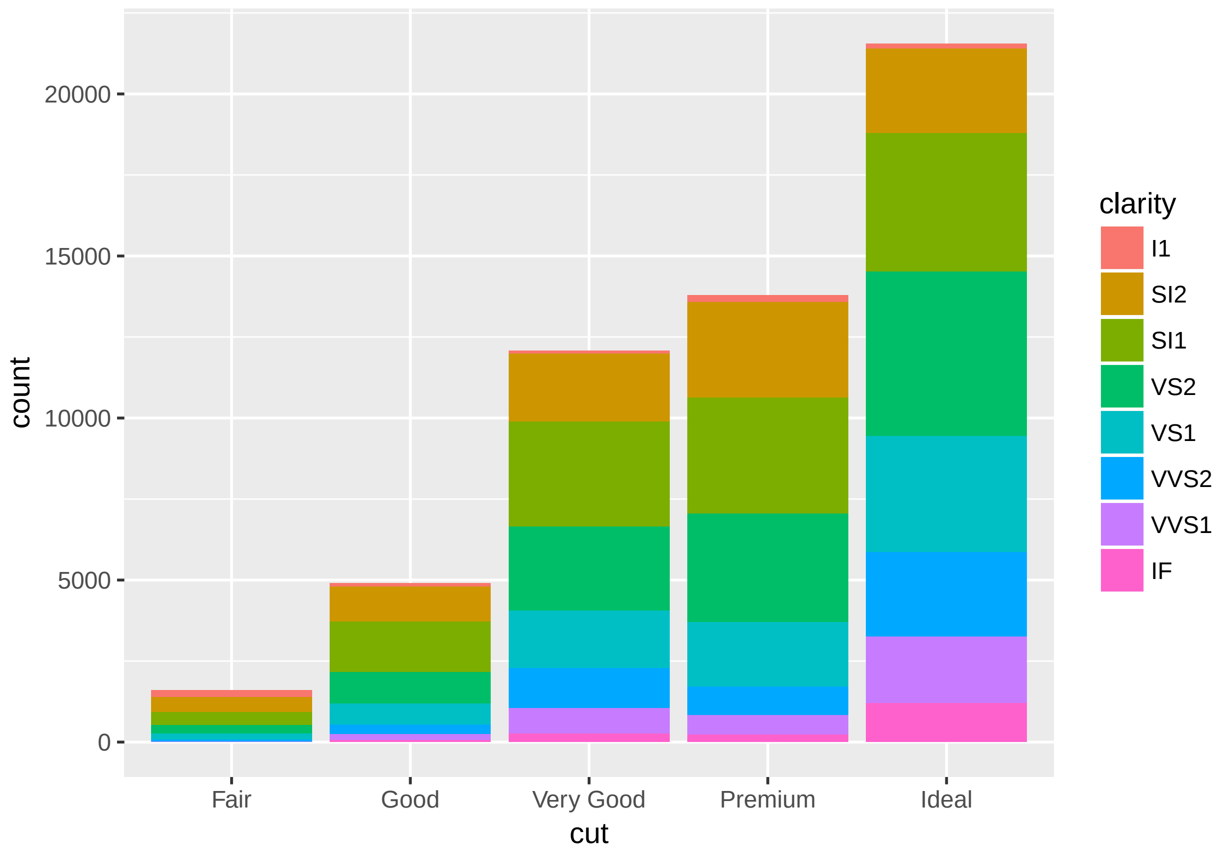
```
> ggplot(data = diamonds) +
+   geom_bar(mapping = aes(x = cut, fill = cut))
```



3.8.2 Map fill aesthetic to another variable

Note what happens if you map the fill aesthetic to another variable, like clarity: the bars are automatically stacked. Each colored rectangle represents a combination of cut and clarity.

```
> ggplot(data = diamonds) +
+   geom_bar(mapping = aes(x = cut, fill = clarity))
```

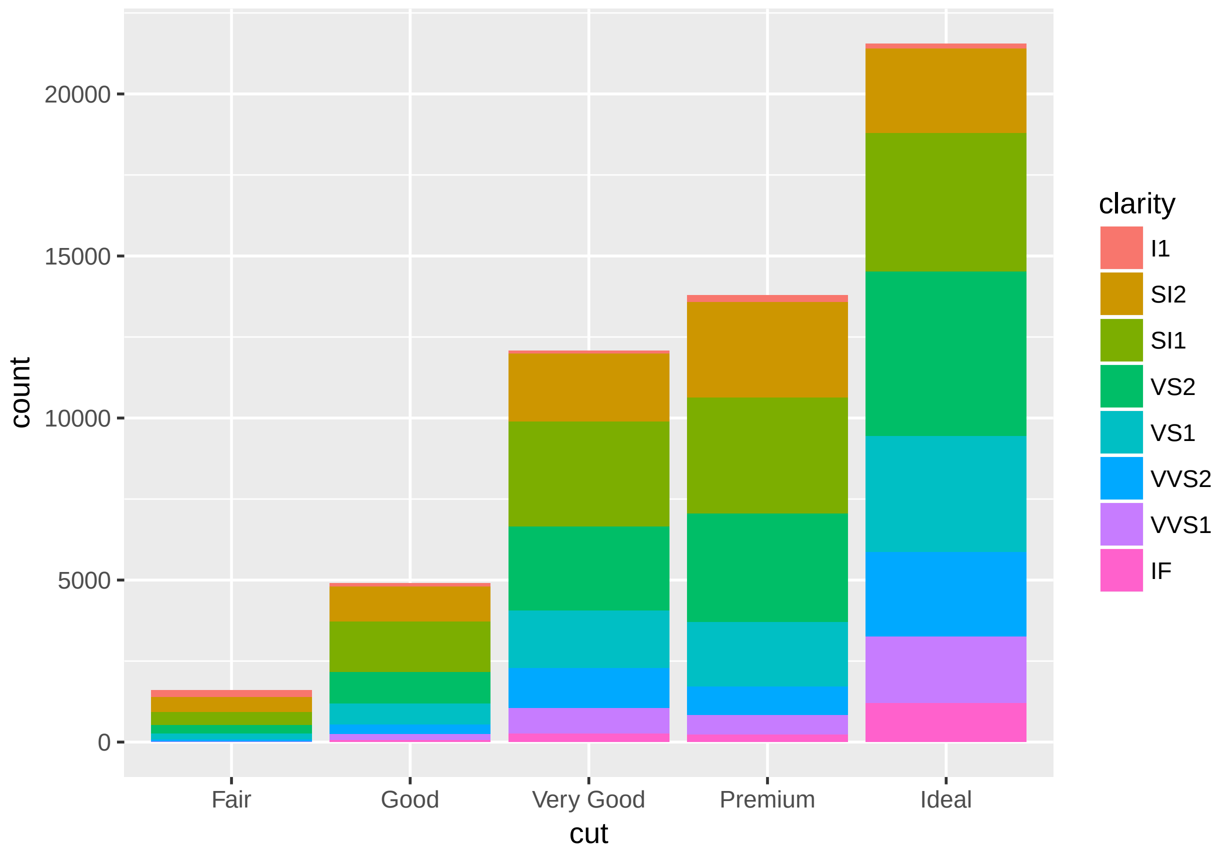


3.8.3 Four options for position argument

The stacking is performed automatically by the position adjustment specified by the position argument (default is position = "stack"). If you don't want a stacked bar chart, you can use one of three other options: `"identity"`, `"dodge"` or "'fill"

3.8.3.1 Position = "stack"

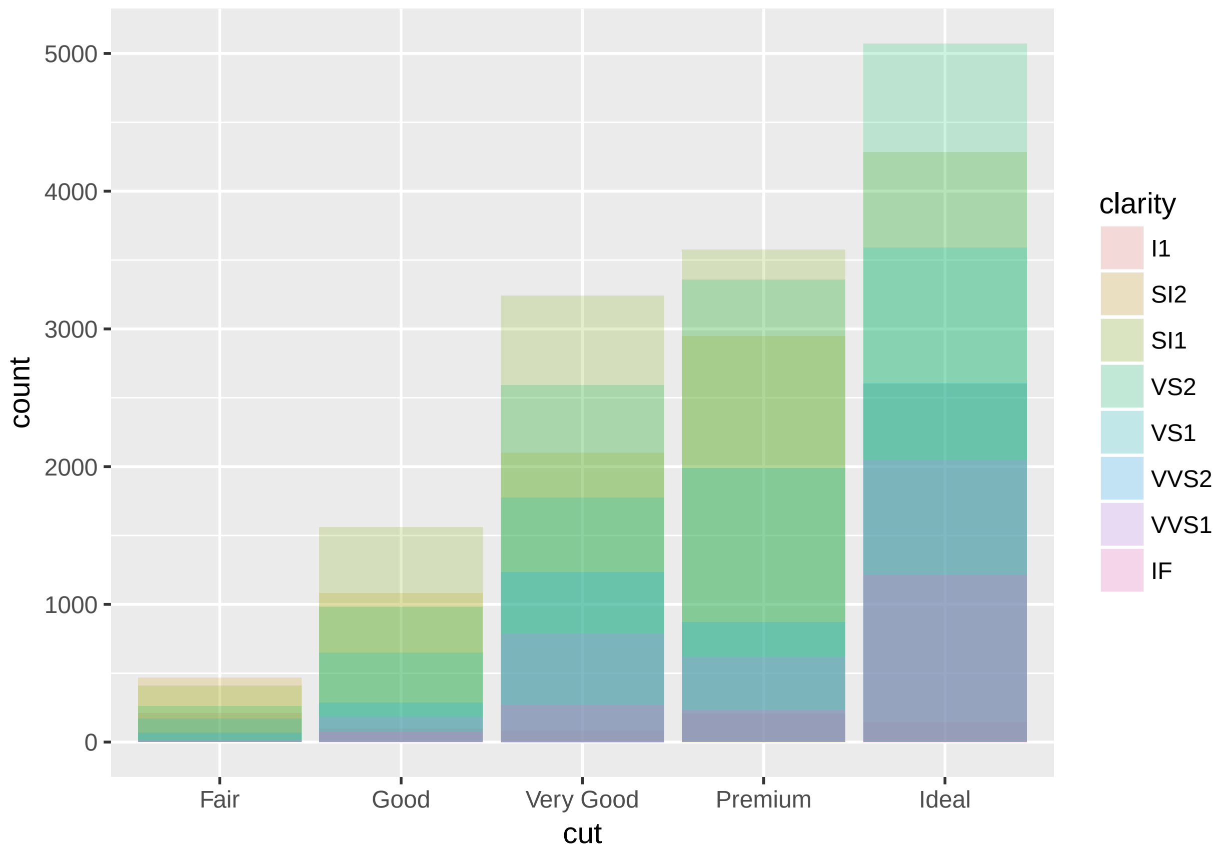
```
> ggplot(data = diamonds) +
+   geom_bar(mapping = aes(x = cut, fill = clarity), position = "stack")
```



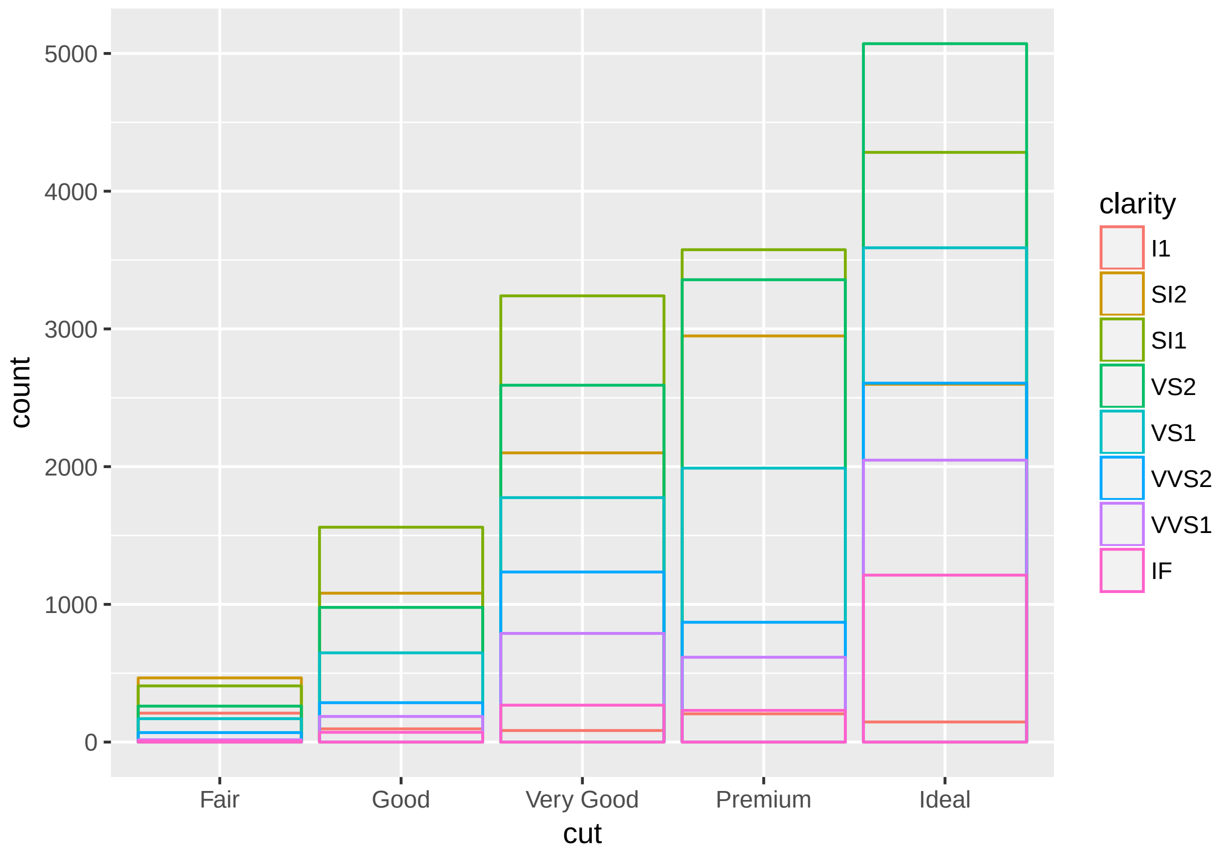
3.8.3.2 Position = "identity" 'position = "identity" will place each object exactly where it falls in the context of the graph. This is not very useful for bars, because it overlaps them. The identity position adjustment is more useful for 2d geoms, like points, where it is the default.

To see that overlapping we either need to make the bars slightly transparent by setting alpha to a small value, or completely transparent by setting `fill = NA`.

```
> ggplot(data = diamonds, mapping = aes(x = cut, fill = clarity)) +
+   geom_bar(alpha = 1/5, position = "identity")
```

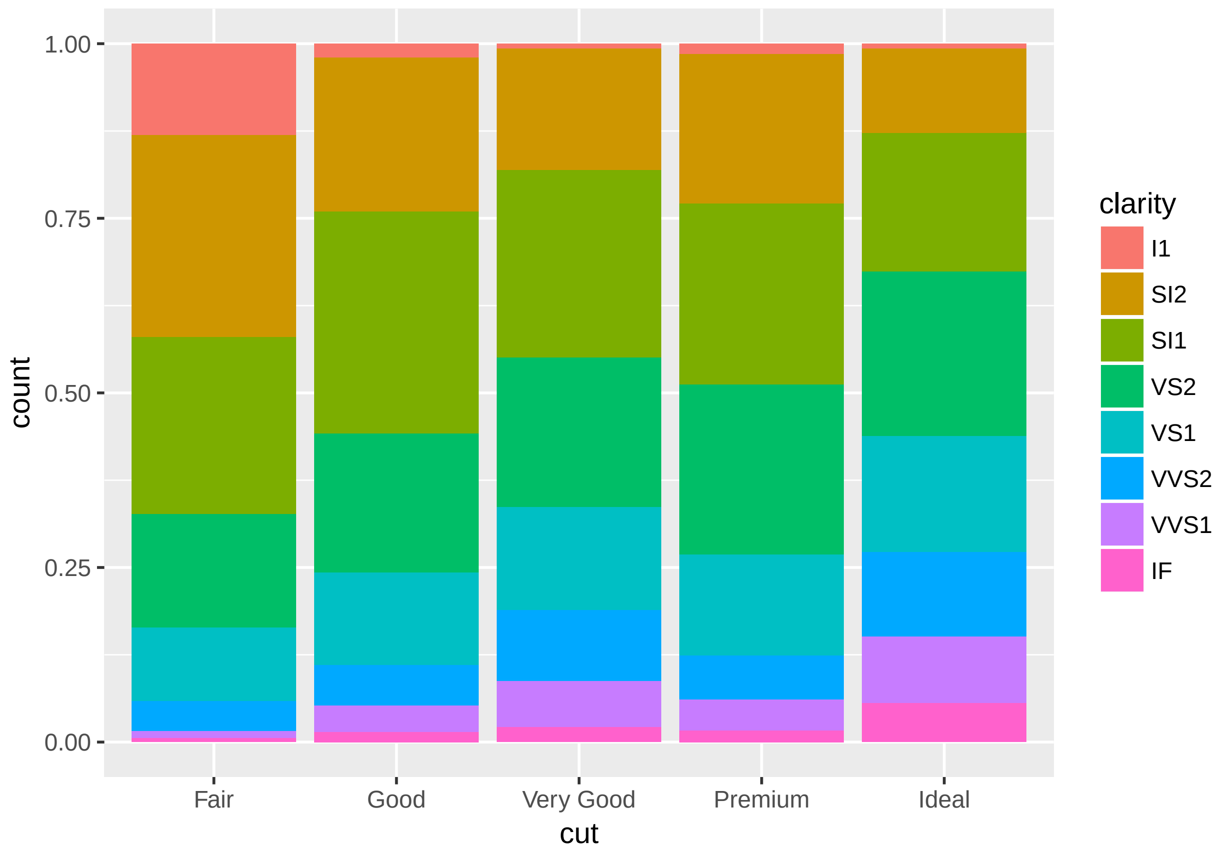


```
> ggplot(data = diamonds, mapping = aes(x = cut, colour = clarity)) +
+   geom_bar(fill = NA, position = "identity")
```



3.8.3.3 Position = "fill" `position = "fill"` works like stacking, but makes each set of stacked bars the same height. This makes it easier to compare proportions across groups.

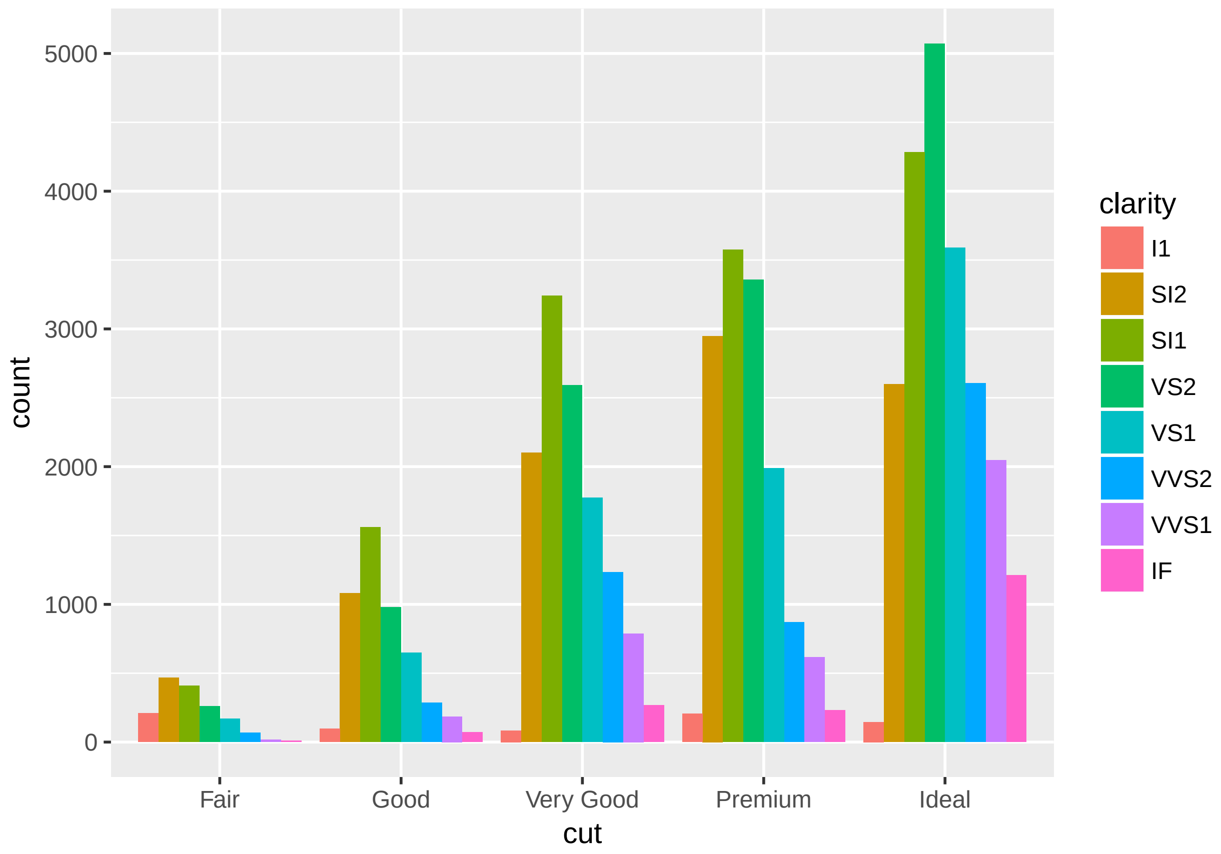
```
> ggplot(data = diamonds) +
+   geom_bar(mapping = aes(x = cut, fill = clarity), position = "fill")
```



3.8.3.4 Position = "dodge"

`position = "dodge"` places overlapping objects directly beside one another. This makes it easier to compare individual values.

```
> ggplot(data = diamonds) +
+   geom_bar(mapping = aes(x = cut, fill = clarity), position = "dodge")
```



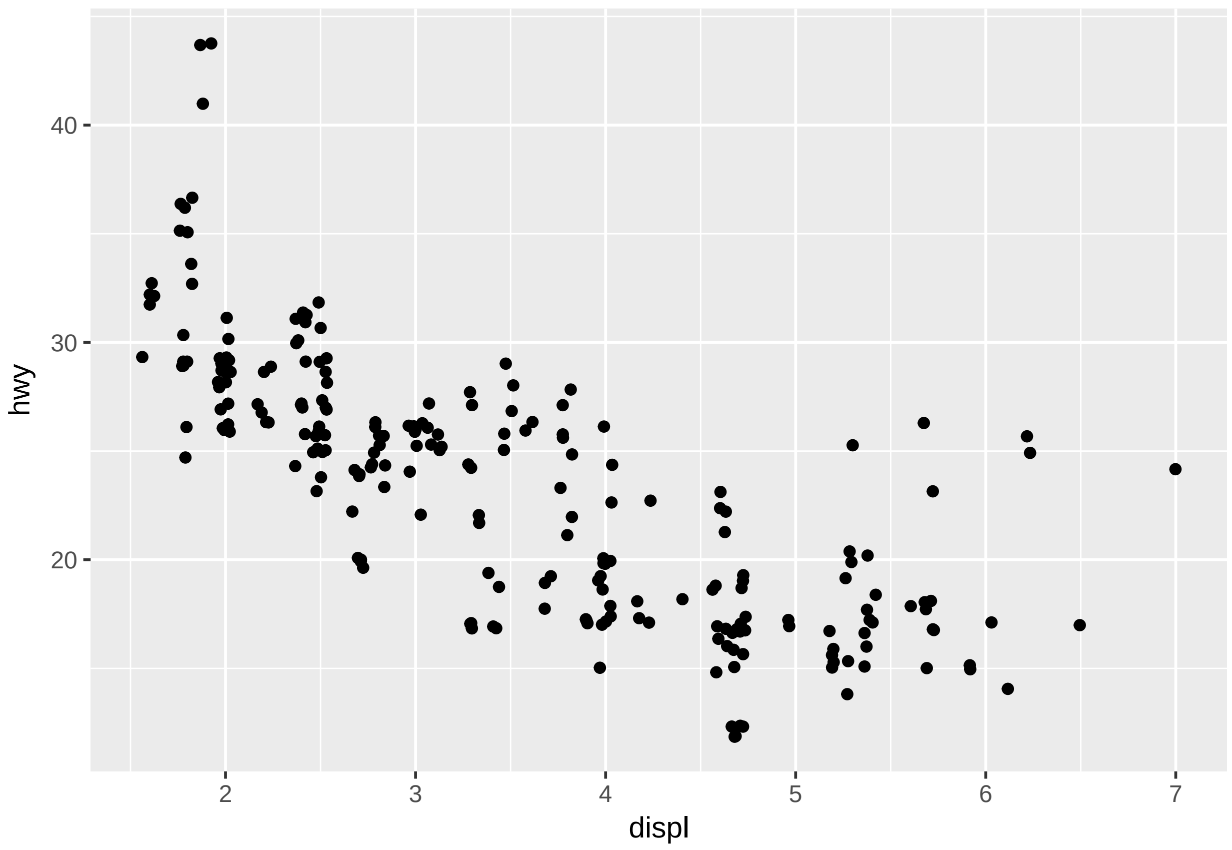
3.8.3.5 Position = "jitter"

There's one other type of adjustment that's not useful for bar charts, but it can be very useful for scatterplots. Recall our first scatterplot. Did you notice that the plot displays only 126 points, even though there are 234 observations in the dataset?

The values of hwy and displ are rounded so the points appear on a grid and many points overlap each other. This problem is known as **overplotting**. This arrangement makes it hard to see where the mass of the data is. Are the data points spread equally throughout the graph, or is there one special combination of hwy and displ that contains 109 values?

You can avoid this gridding by setting the position adjustment to "jitter". 'position = "jitter" adds a small amount of random noise to each point. This spreads the points out because no two points are likely to receive the same amount of random noise.

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

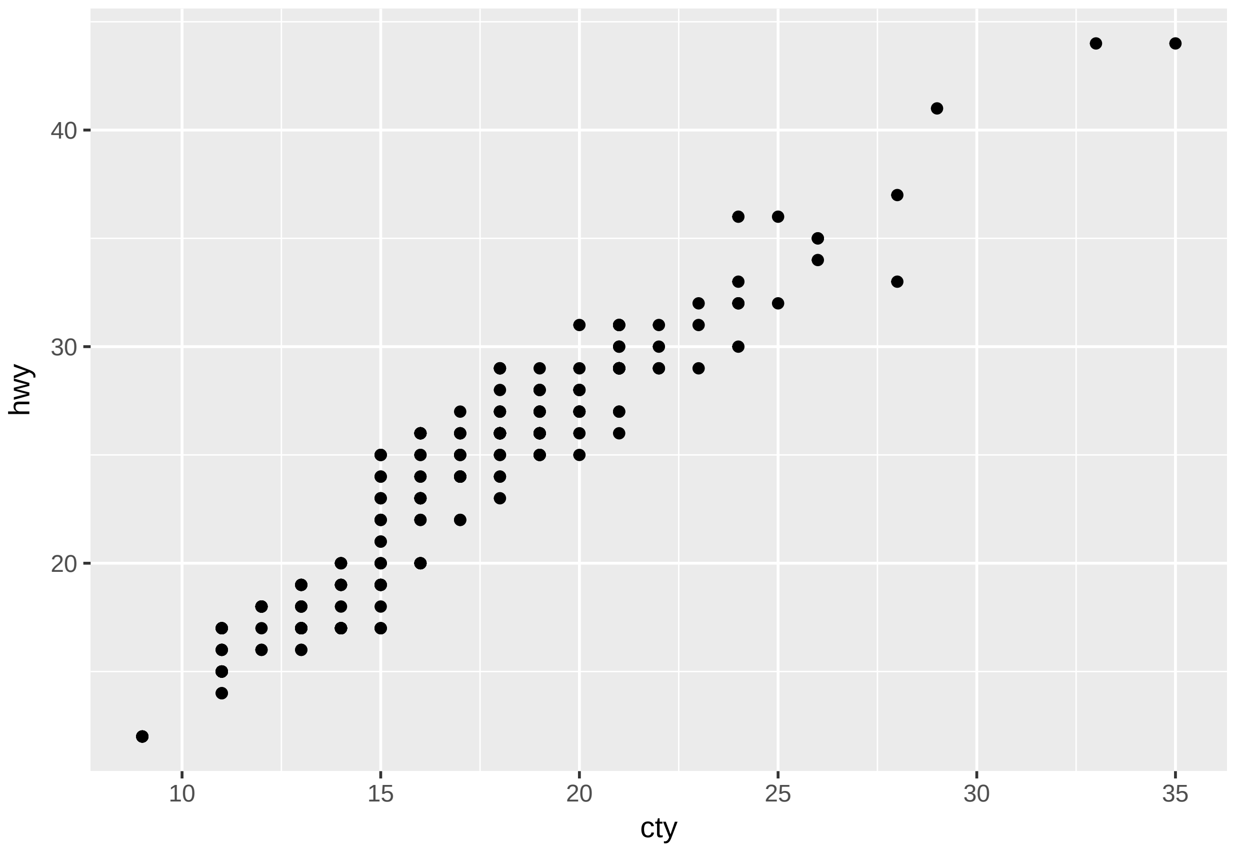


Adding randomness seems like a strange way to improve your plot, but while it makes your graph less accurate at small scales, it makes your graph more revealing at large scales. Because this is such a useful operation, ggplot2 comes with a shorthand for `geom_point(position = "jitter")`: `geom_jitter()`.

3.8.4 Exercises

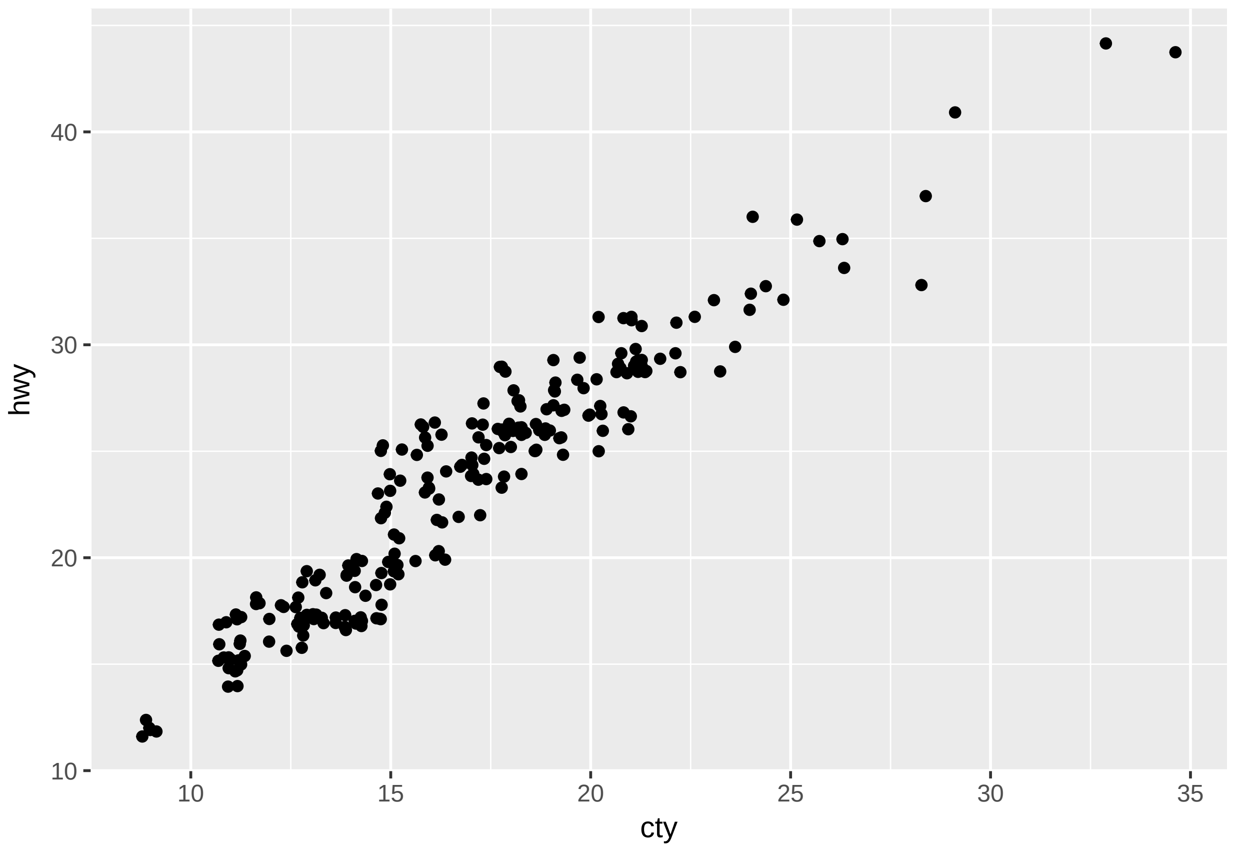
0. Learn more about a position adjustment: look up the help page associated with each adjustment: `?position_dodge`, `?position_fill`, `?position_identity`, `?position_jitter`, and `?position_stack`.
1. What is the problem with this plot? How could you improve it?

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



My Solution: Many data points overlay each other (=overplotting). Using `position = "jitter"` show the whole spectrum of data. By adding a small amount of random variation "overplotting" is prevented.

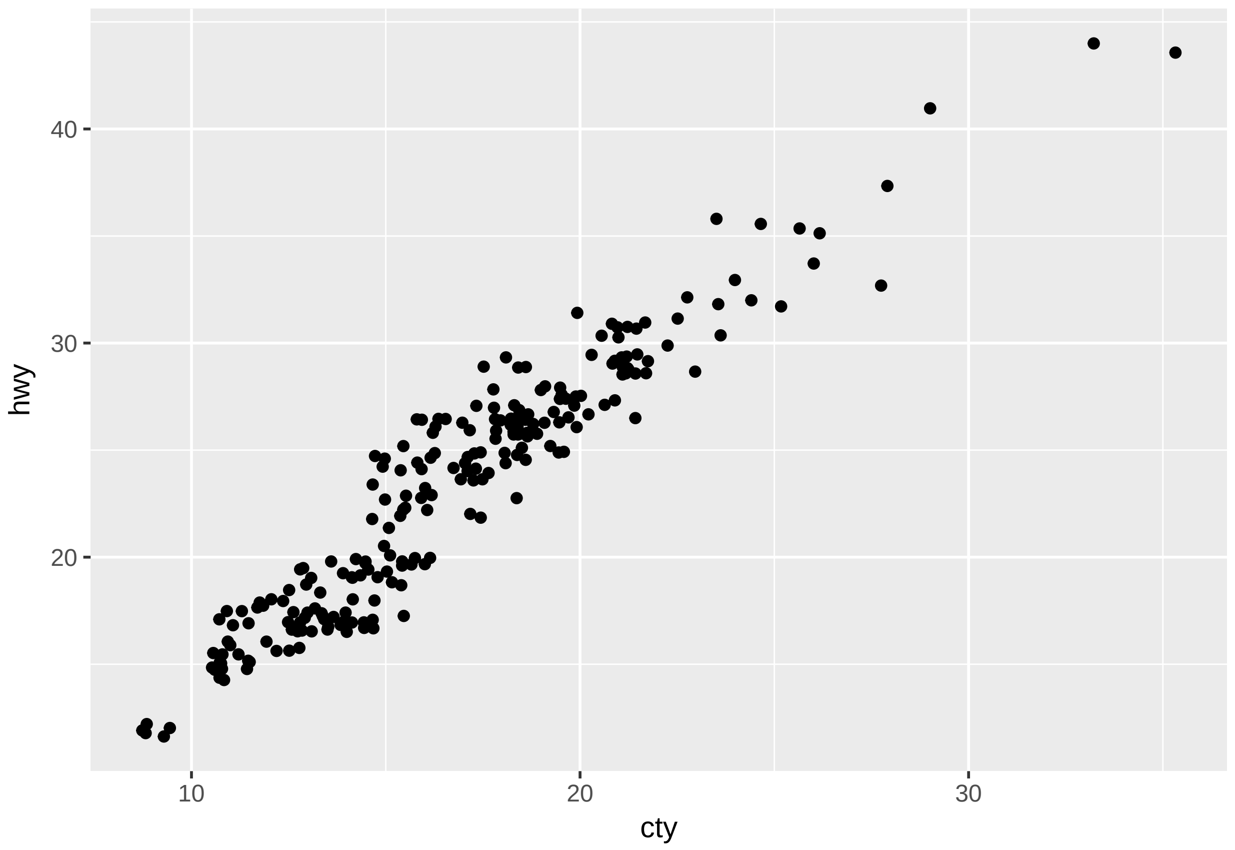
```
> ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
+   geom_point(position = "jitter")
```



2\.
What parameters to `geom_jitter()` control the amount of jittering? What parameters to `geom_jitter()` `control` the amount of jittering?

My Solution: width and height From the help file: "Amount of vertical and horizontal jitter. The jitter is added in both positive and negative directions, so the total spread is twice the value specified here. If omitted, defaults to 40% of the resolution of the data: this means the jitter values will occupy 80% of the implied bins. Categorical data is aligned on the integers, so a width or height of 0.5 will spread the data so it's not possible to see the distinction between the categories."

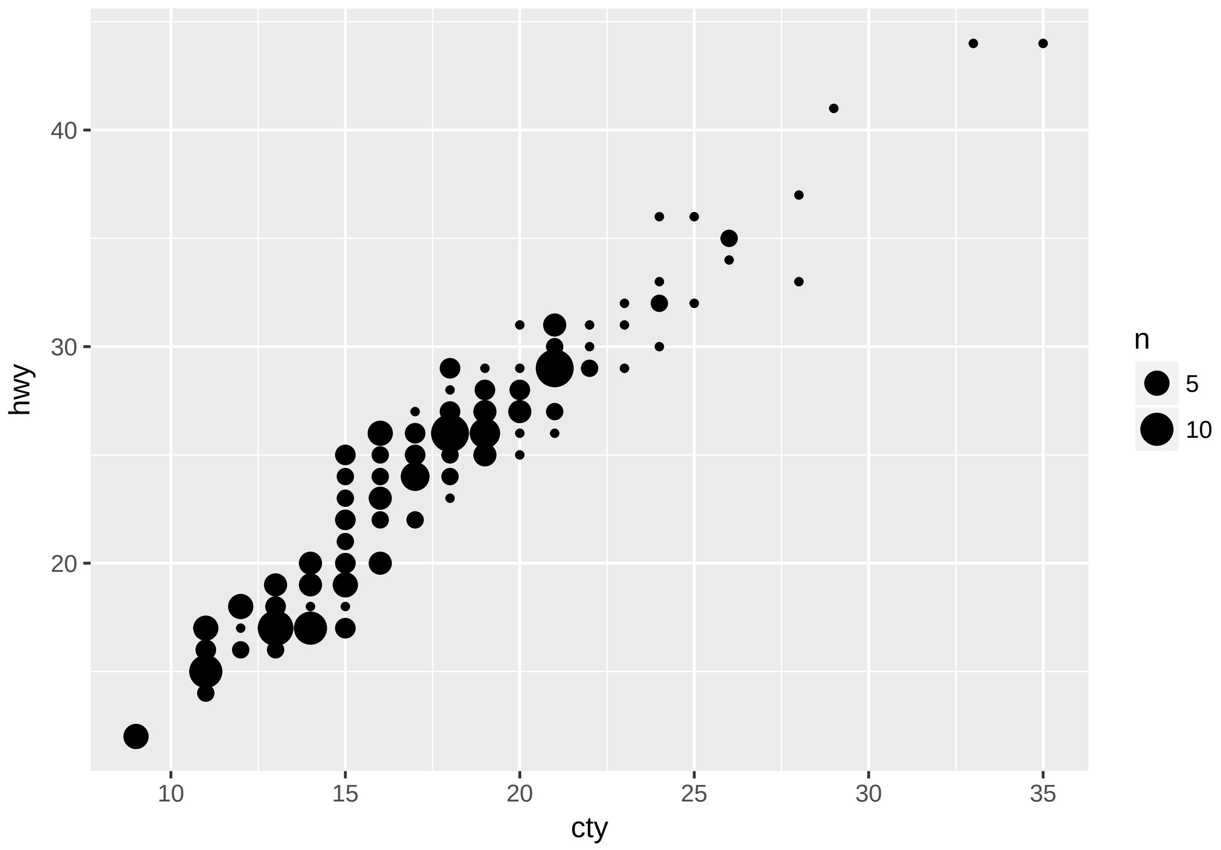
```
> ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
+   geom_jitter(width = 0.5, height = 0.5)
```



3.

Compare and contrast `geom_jitter()` with `geom_count()`

```
> ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
+   geom_count()
```

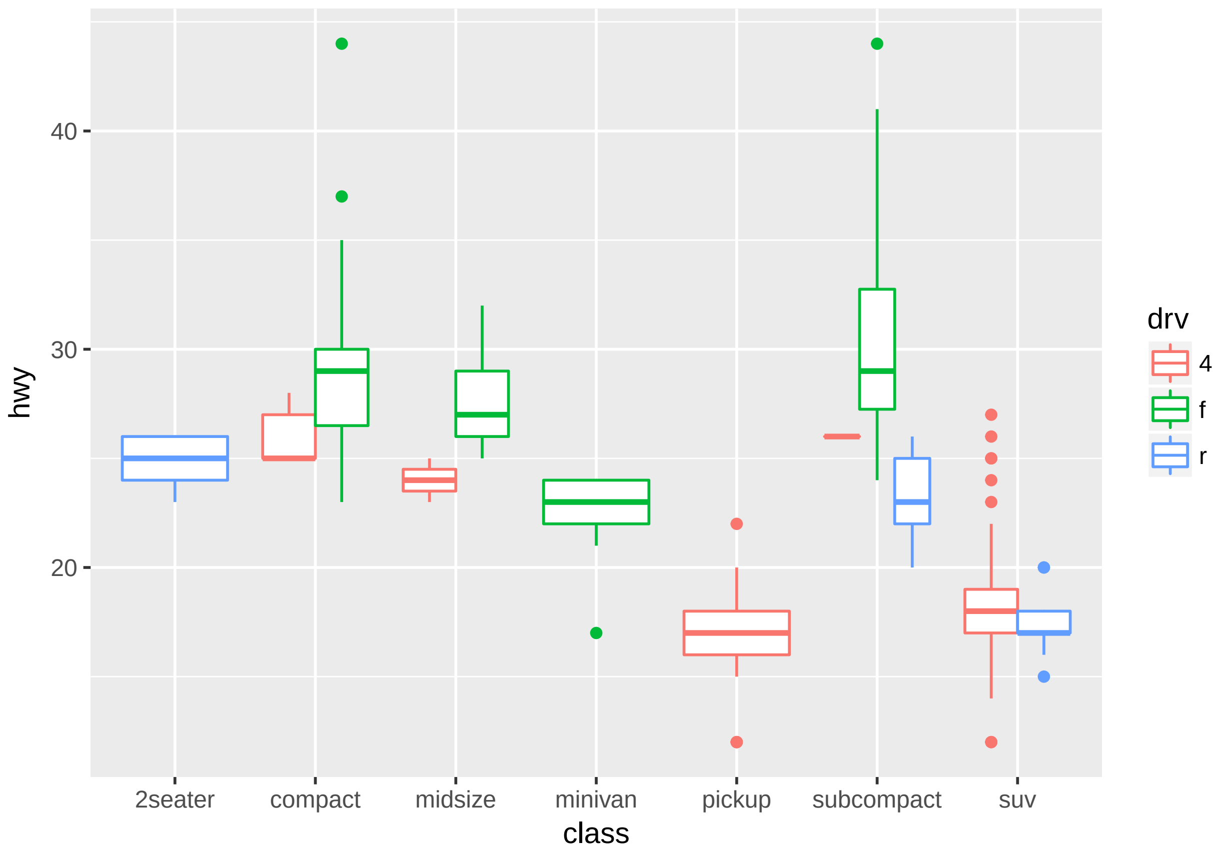


My Solution: geom_count displays overplotting with bigger points, geom_jitter adds saome random variance. For me geom_jitter looks more natural and gives me a better impression of "crowded" places than the somewhat articifal bigger bubbles in geom_count. But with jitter one has to set parameteres which has big effects of the overall look of the graphic.

(4) What's the default position adjustment for `geom_boxplot()`? Create a visualisation of the `mpg` dataset that demonstrates it.

My Solution: The default position adjustment for `geom_boxplot()` is "dodge". How to create a visualisation of the `mpg` dataset that demonstrates it? Quote and example are from the online help (cf. http://docs.ggplot2.org/current/geom_boxplot.html) "Boxplots are automatically dodged when any aesthetic is a factor."

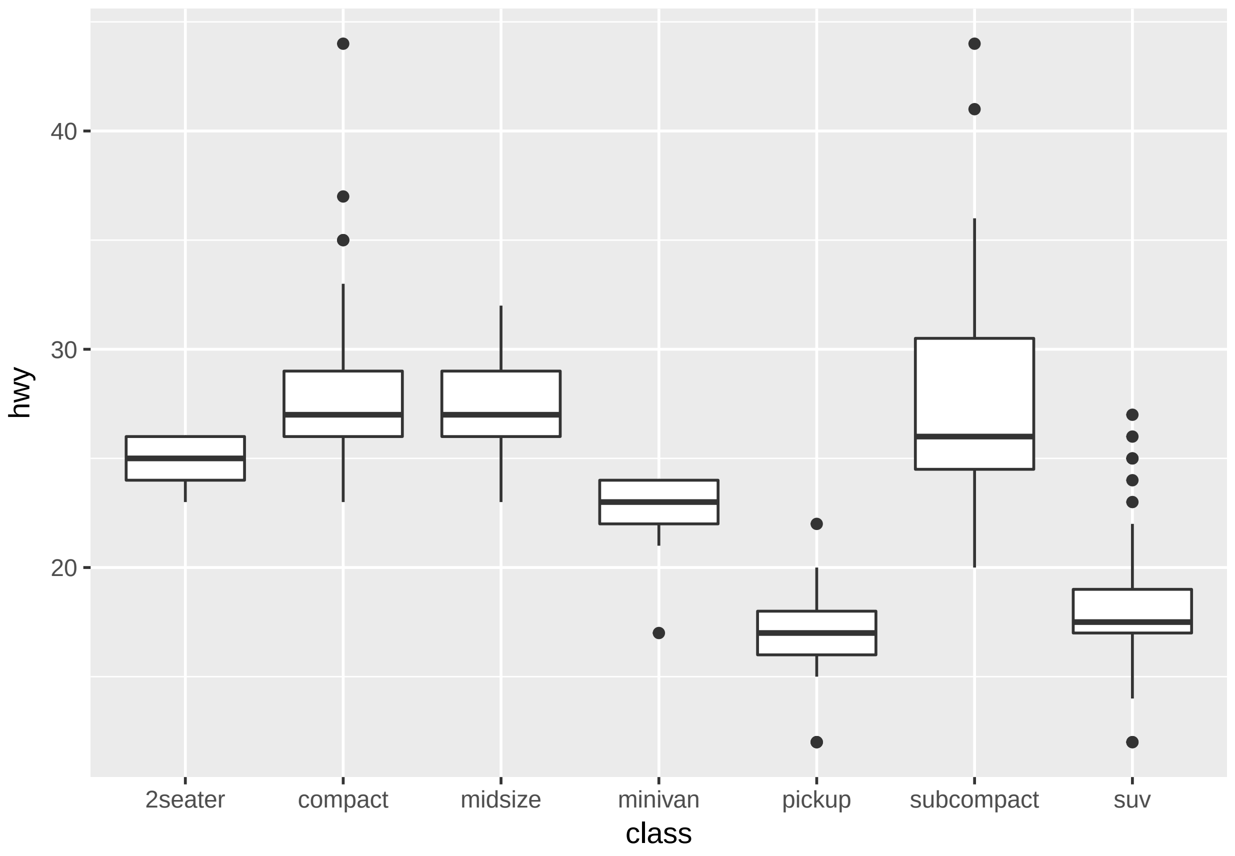
```
> ggplot(mpg, aes(class, hwy)) +
+         geom_boxplot((aes(color = drv)))
```



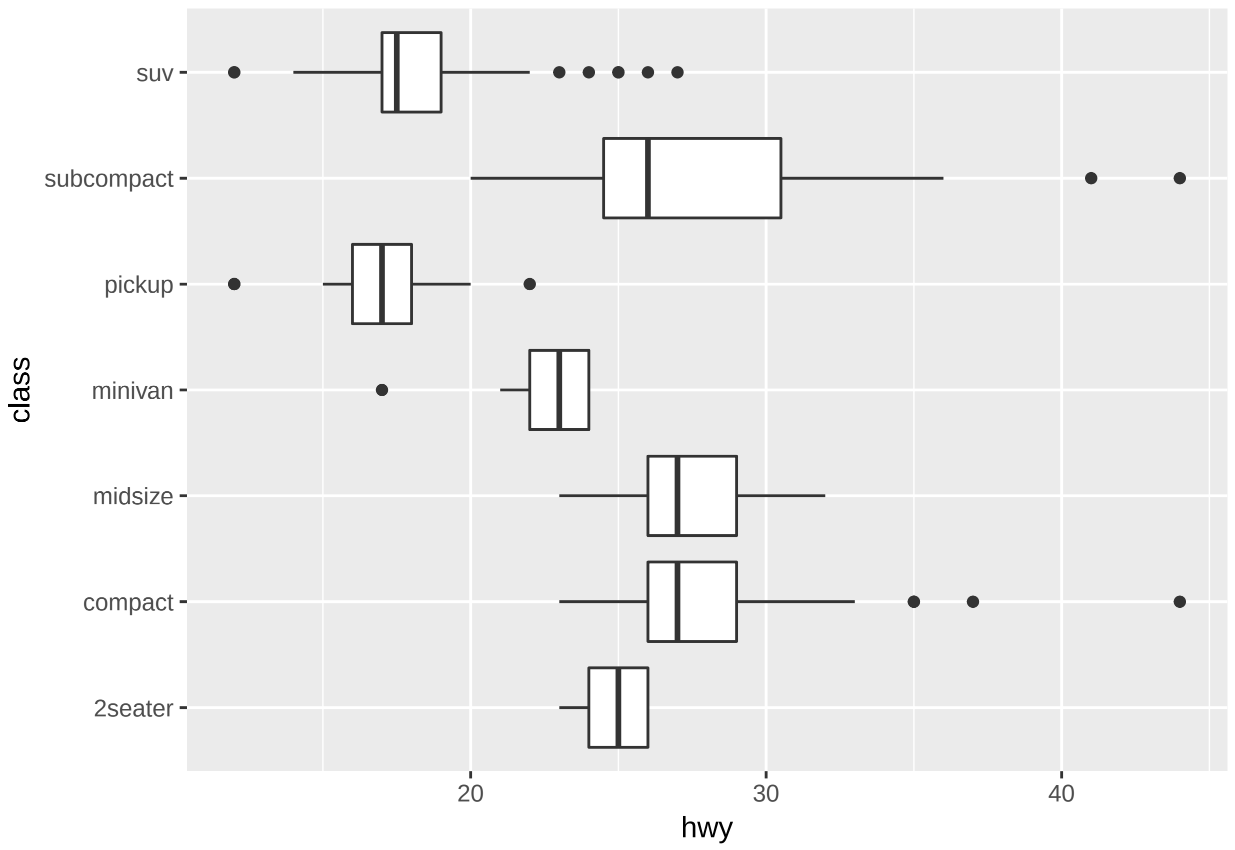
#
3.9 Coordinate systems ## 3.9.1 Switching x and y axes: coord_flip()

`coord_flip()` switches the x and y axes. This is useful (for example), if you want horizontal boxplots. It's also useful for long labels: it's hard to get them to fit without overlapping on the x-axis.

```
> ggplot(data = mpg, mapping = aes(x = class, y = hwy)) +
+   geom_boxplot()
```



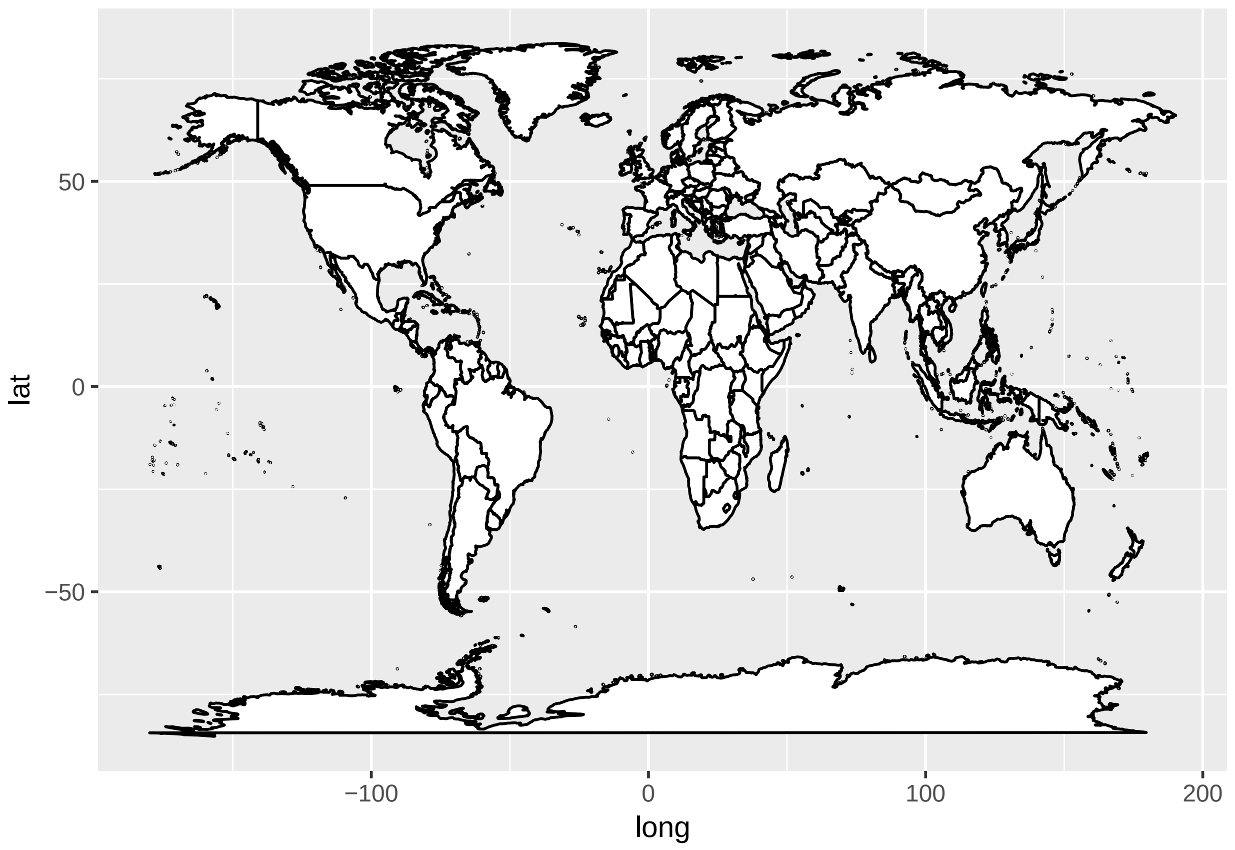
```
> ggplot(data = mpg, mapping = aes(x = class, y = hwy)) +
+   geom_boxplot() +
+   coord_flip()
```



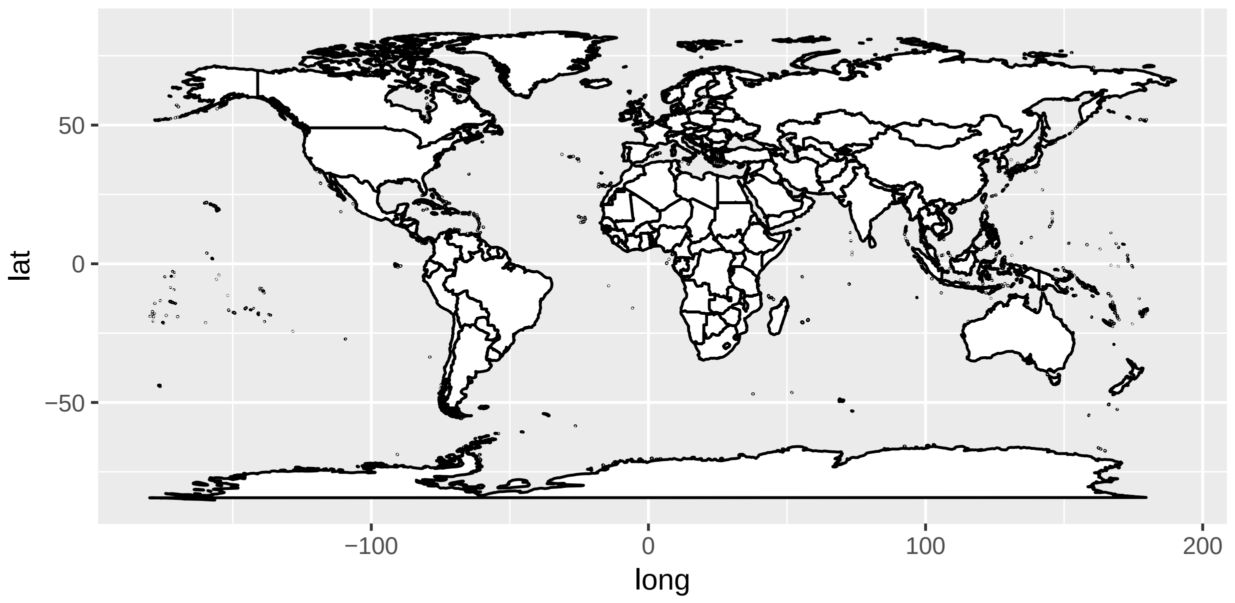
3.9.2. Setting aspect ratio for maps correctly: coord_quickmap()

`coord_quickmap()` sets the aspect ratio correctly for maps. This is very important if you're plotting spatial data with `ggplot2` (which unfortunately we don't have the space to cover in this book).

```
> world <- map_data("world")
>
> ggplot(world, aes(long, lat, group = group)) +
+   geom_polygon(fill = "white", colour = "black")
```

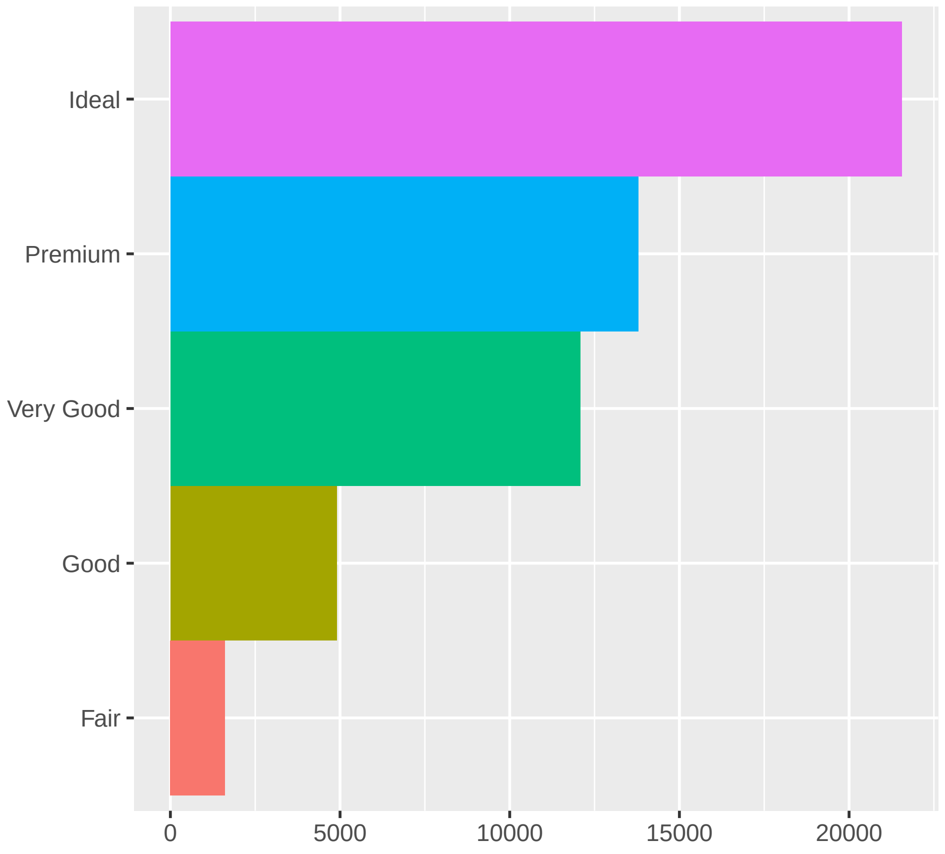


```
> ggplot(world, aes(long, lat, group = group)) +
+   geom_polygon(fill = "white", colour = "black") +
+   coord_quickmap()
```

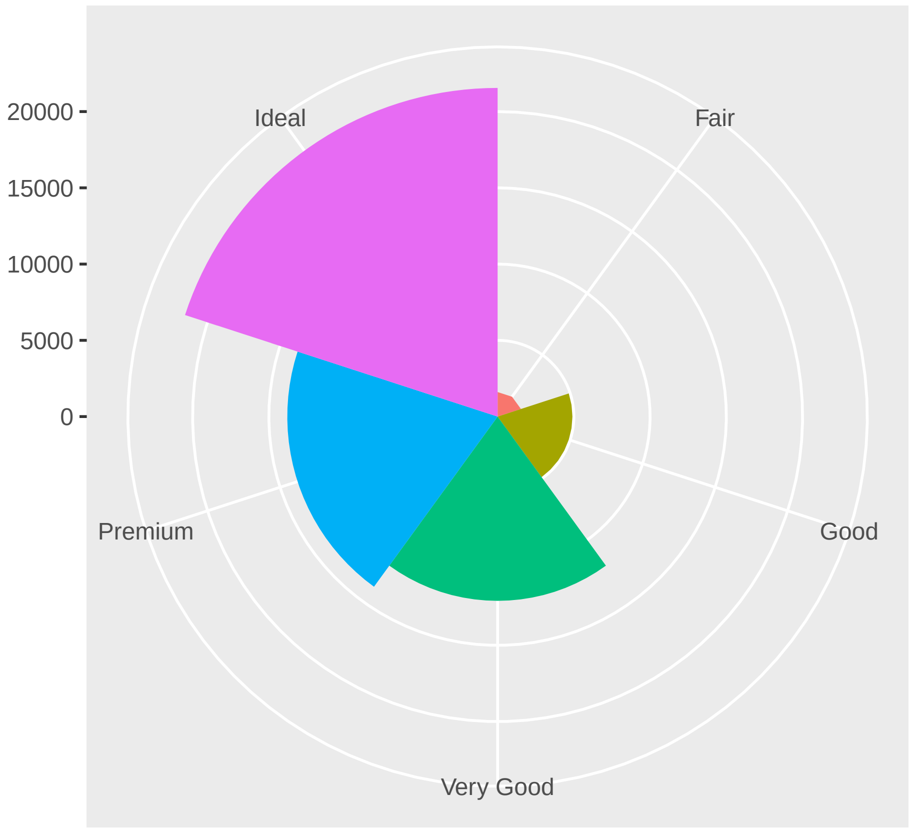


3.9.3 Using polar coordinates: coord_polar() ‘coord_polar() uses polar coordinates. Polar coordinates reveal an interesting connection between a bar chart and a Coxcomb chart.

```
> bar <- ggplot(data = diamonds) +
+   geom_bar(
+     mapping = aes(x = cut, fill = cut),
+     show.legend = FALSE,
+     width = 1
+   ) +
+   theme(aspect.ratio = 1) +
+   labs(x = NULL, y = NULL)
>
> bar + coord_flip()
```



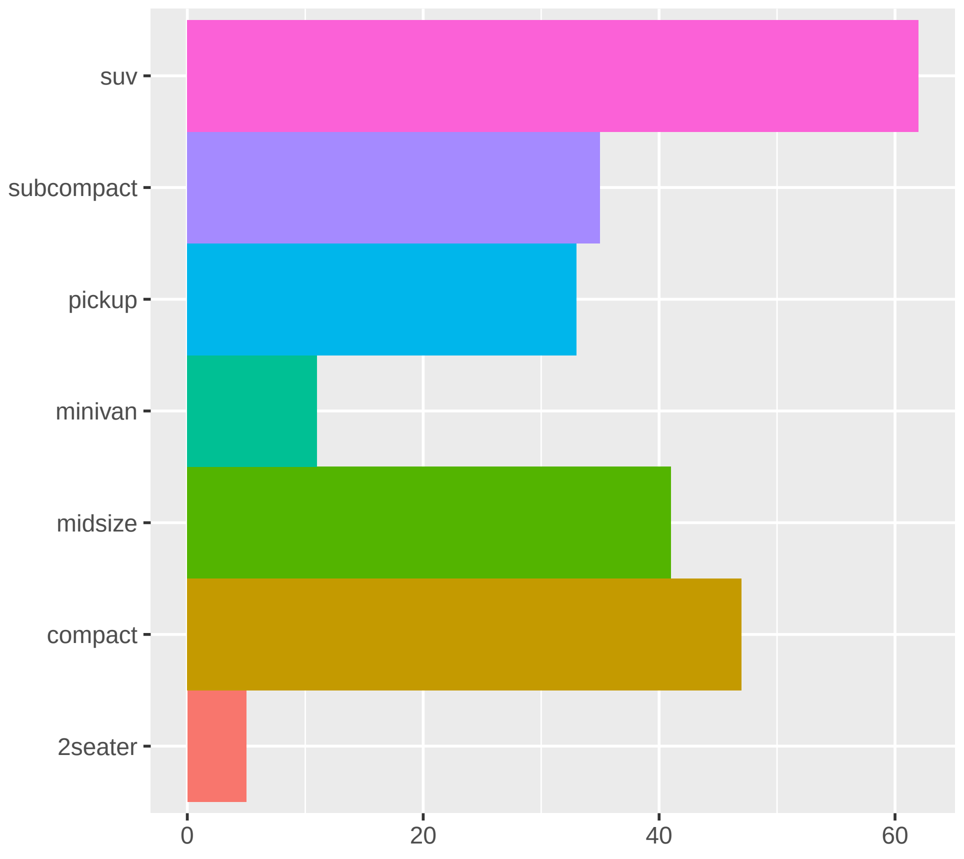
```
> bar + coord_polar()
```



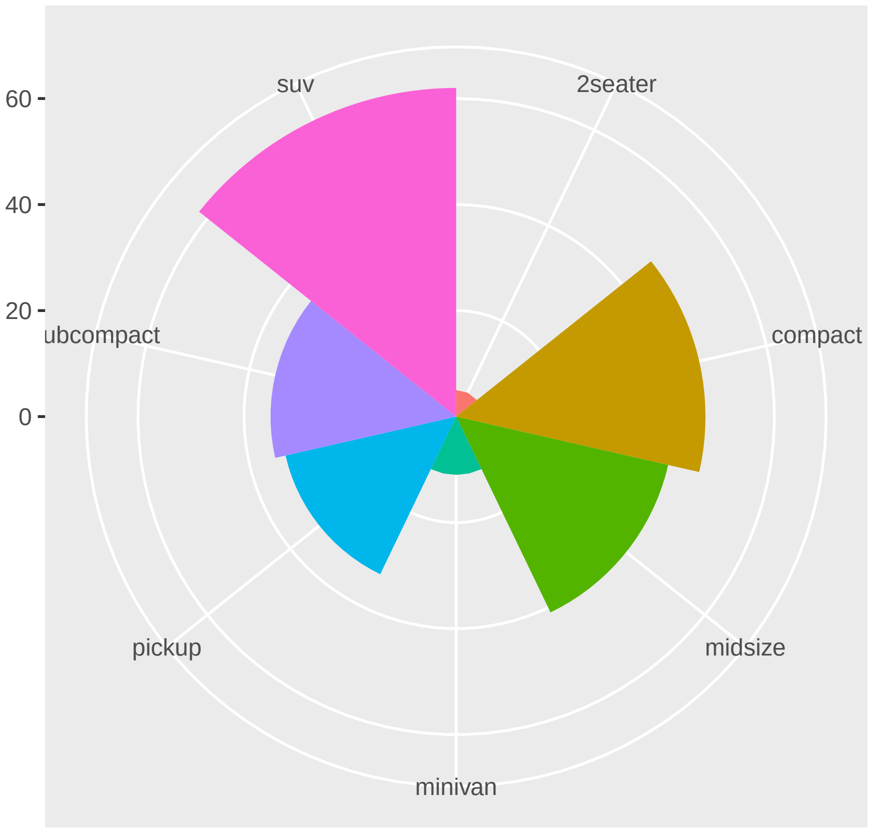
3.9.4 Exercises (1) Turn a stacked bar chart into a pie chart using `coord_polar()`.

My Solution:

```
> bar <- ggplot(data = mpg) +
+   geom_bar(
+     mapping = aes(x = class, fill = class),
+     show.legend = FALSE,
+     width = 1
+   ) +
+   theme(aspect.ratio = 1) +
+   labs(x = NULL, y = NULL)
>
> bar + coord_flip()
```



```
> bar + coord_polar()
```



(2) What does labs() do? Read the documentation.

My Solution: `labs()` modifies axis, legend, and plot labels

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

My Solution: From the online help: `coord_map` projects a portion of the earth, which is approximately spherical, onto a flat 2D plane using any projection defined by the `mapproj` package. Map projections do not, in general, preserve straight lines, so this requires considerable computation. `coord_quickmap` is a quick approximation that does preserve straight lines. It works best for smaller areas closer to the equator.

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

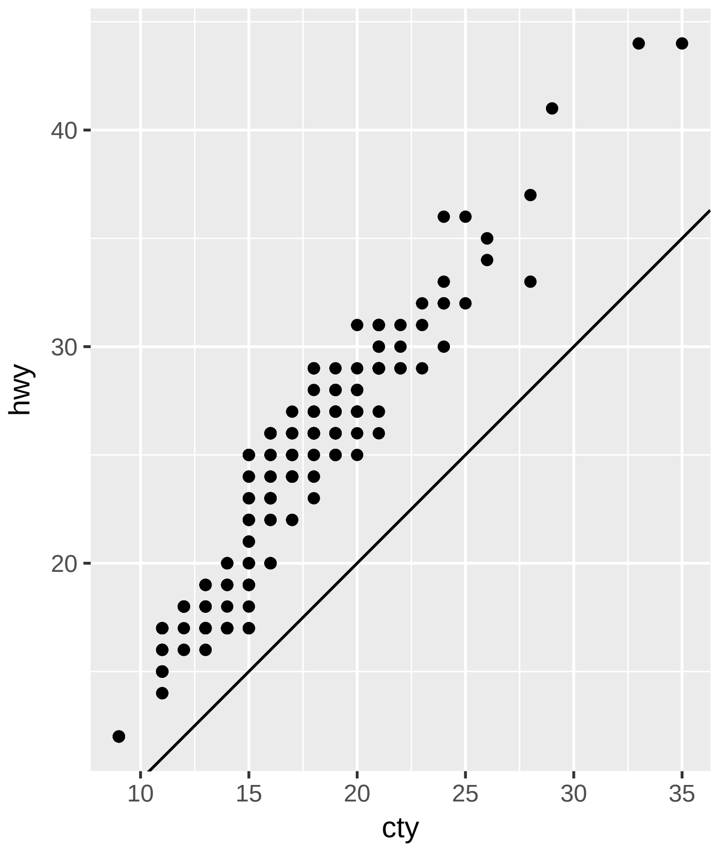
My Solution:

(4a) The comsumption in generally on the highway lower than in the city. Cars with a high rate of miles per gallons in the city also have a relative high rate on the highway. There are some outliers to the positive linear trend.

(4b) Why is `coord_fixed()` important? Form the online Manual: A fixed scale coordinate system forces a specified ratio between the physical representation of data units on the axes. The ratio represents the number of units on the y-axis equivalent to one unit on the x-axis. The default, ratio = 1, ensures that one unit on the x-axis is the same length as one unit on the y-axis. Ratios higher than one make units on the y axis longer than units on the x-axis, and vice versa.

(4c) What does `geom_abline()` do? It adds a reference line to the graph, which is useful for annoting plots.

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



3.10 The layered grammar of graphics

We started with the following simple template:

```
ggplot(data = <DATA>) +
        <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

After this chapter we can add position adjustments, stats, coordinate systems, and faceting to the code template

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