Introduction to R Programming

Data Visualization with ggplot2

Pedro Fonseca

21 Abril 2020


ggplot2:
Build a data
MASTERPIECE
THEMES
GEOMS
aes
fill
color
ggANIMATE
plotly
HORST '18

Introduction

- In lecture you will learn how to visualize data with the ggplot2 package.
- ggplot2 is one of the most elegant and versatile systems for making graphs.
- ggplot2 implements the grammar of graphics (Wickham 2010), a coherent system for describing and building graphs.

Prerequisites

To access the datasets, help pages, and functions that we will use, load the ggplot2 package.

```
library(ggplot2)
```

The mpg dataset

The mpg dataset contains information about 38 models of cars. Among the variables in mpg are:

- displ: engine size, in liters
- cyl: number of cylinders
- cty: city miles per gallon
- hwy: fuel efficiency on the highway, in miles per gallon (mpg).
- class: type of car

```
View(mpg)
```

For additional information see ?mpg.

My first `ggplot()`

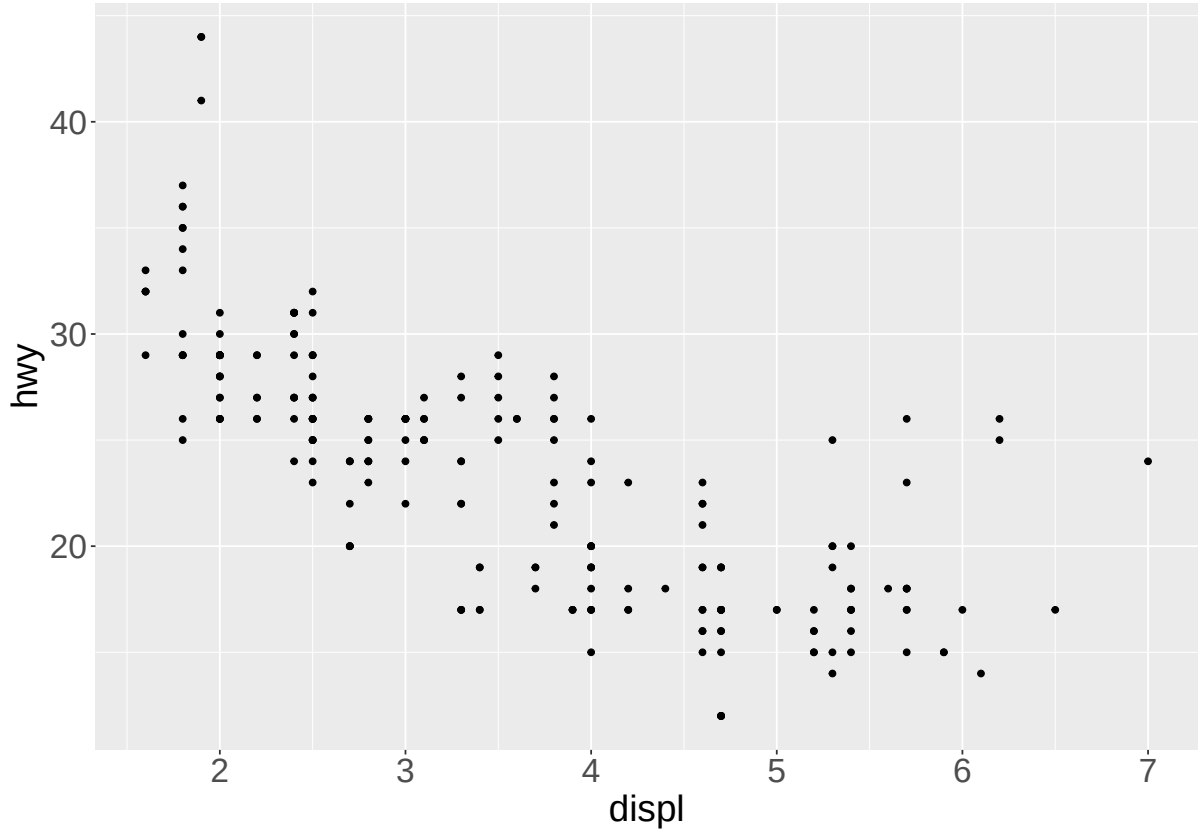
Let's build a graph to answer the following questions:

- Do cars with big engines use more fuel than cars with small engines?
- What does the relationship between engine size and fuel efficiency look like? Is it positive? Negative? Linear? Nonlinear?

We start by plotting engine size (`displ`) versus fuel efficiency (`hwy`).

My first ggplot()

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



My first `ggplot()`

- The plot shows a negative relationship between engine size and fuel efficiency.
- In other words: on average, cars with big engines use more fuel.

A simple ggplot() template

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

- The first argument of ggplot() is a dataset
- We complete the graph by adding layers to ggplot()
- geom_point() adds a layer of points, creating scatterplot.
- ggplot2 has many geom functions that each add a different type of layer to a plot.

A simple `ggplot()` template

- Each geom function takes a `mapping` argument.
- The `mapping` argument is always paired with `aes()`.
- The `x` and `y` arguments of `aes()` specify which variables to map to the `x` and `y` axes.

Geometrical objects

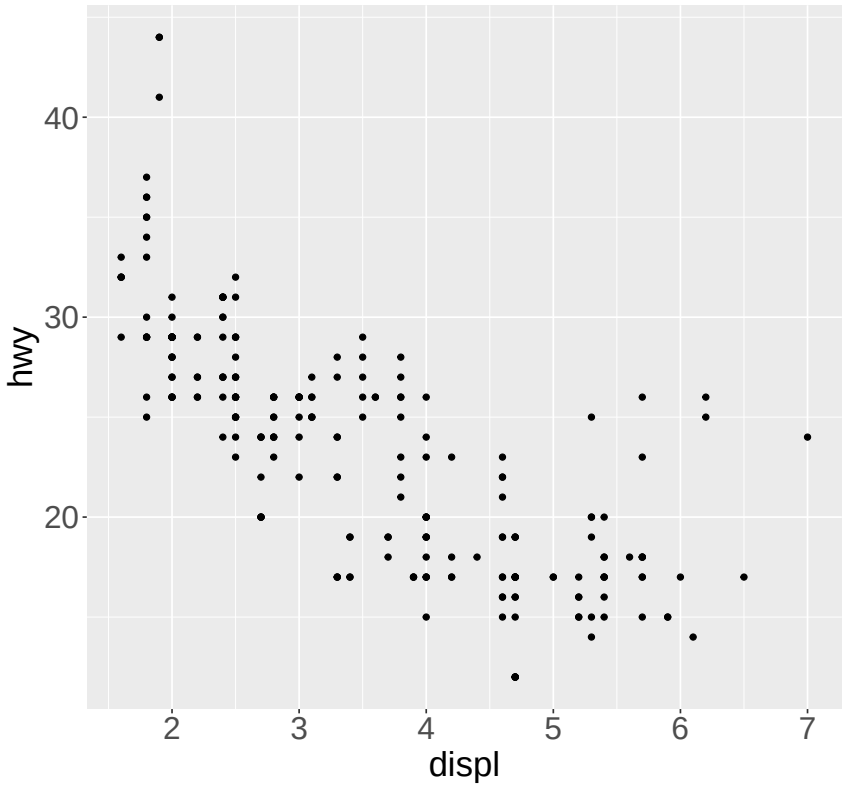
- A **geom** is the geometrical object that a plot uses to represent data (e.g. points, bars, lines. . .).
- People often describe plots by the type of **geom** that it uses:
  - Bar charts use **bar** geoms
  - Line charts use **line** geoms
  - Boxplots use **boxplot** geom
  - . . .
- Scatterplots break the trend: they use the **point** geom.
- In ggplot2, we add geoms to a plot with **geom functions**.

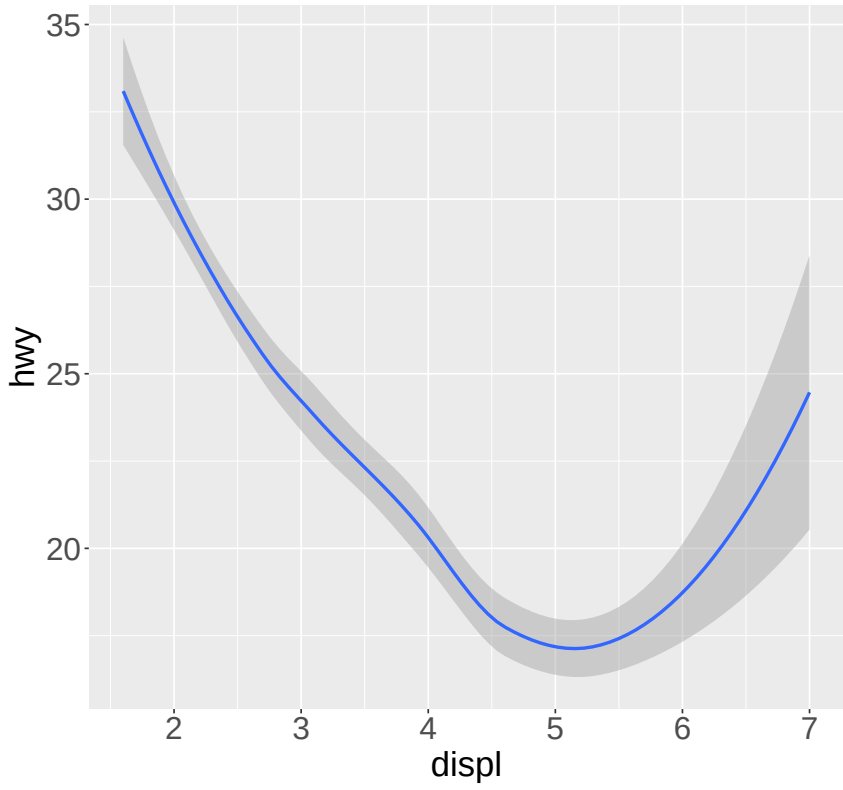
Geometrical objects

Compare the next two plots. How are they similar?

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

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

Geometrical objects





Geometrical objects

Both plots:

- use the same data
- have the same `x` variable
- have the same `y` variable

But they are not identical:

- Each plot uses a different **geom**.
- The plot on the left uses the **point** geom.
- The plot on the right uses the **smooth** geom, a smooth line fitted to the data.

The ggplot2 cheat sheet

- ggplot2 provides over 40 geoms
- The best way to get a comprehensive overview is the ggplot2 cheatsheet.

Aesthetics

- **Aesthetics** are visual properties of the **geoms**.
- **Aesthetics** include things like the size, shape, and color of the points in a scatterplot.
- `aes()` builds **aesthetic mappings** that define how the variables in the dataset are mapped to aesthetics of the geoms.
- To map an aesthetic to a variable, associate the name of the aesthetic to the name of the variable inside `aes()`.
- Every geom function needs an aesthetic mapping, but not every aesthetic works with every geom, for example:
  - We can set the `shape` of a point, but not of a line.
  - We can set the `linetype` of a line, but not of a point.

My first `ggplot()` revisited

One group of points (highlighted in red) seems to fall outside the linear trend. How can we explain these cars?

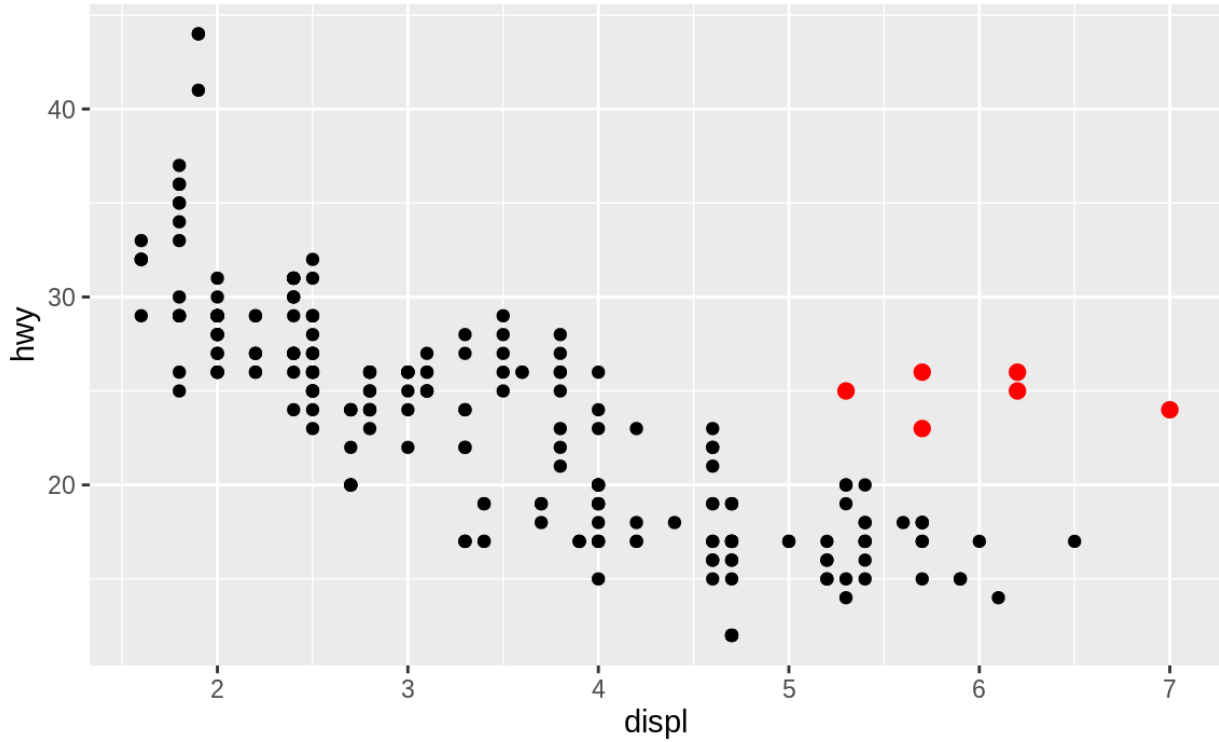


Figure 1: Some cars have a higher mileage than we might expect

My first `ggplot()` revisited

- The highlighted cars may have some common characteristic with respect to some other variable (e.g. they might all be hybrids).
- Lets try `class`, a variable that classifies cars into groups such as compact, midsize, and SUV.
- We can add a third variable to a two dimensional scatterplot by mapping it to an aesthetic.

The color aesthetic

For example, we can map the color of the points to the `class` variable, so that the graph reveals the class of each car:

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

The color aesthetic

The color aesthetic

Now we know that all but one of the highlighted cars are two-seater cars! Let's find their model and manufacturer:

```
subset(mpg, class == "2seater",
       select = c("manufacturer", "model", "year"))
```

```
## # A tibble: 5 x 3
##   manufacturer model      year
##   <chr>        <chr>     <int>
## 1 chevrolet    corvette   1999
## 2 chevrolet    corvette   1999
## 3 chevrolet    corvette   2008
## 4 chevrolet    corvette   2008
## 5 chevrolet    corvette   2008
```

The color aesthetic

- These cars are, in fact, sports cars!
- Sports cars have large engines like SUVs and pickup trucks, but small bodies like midsize and compact cars, which improves their gas mileage.

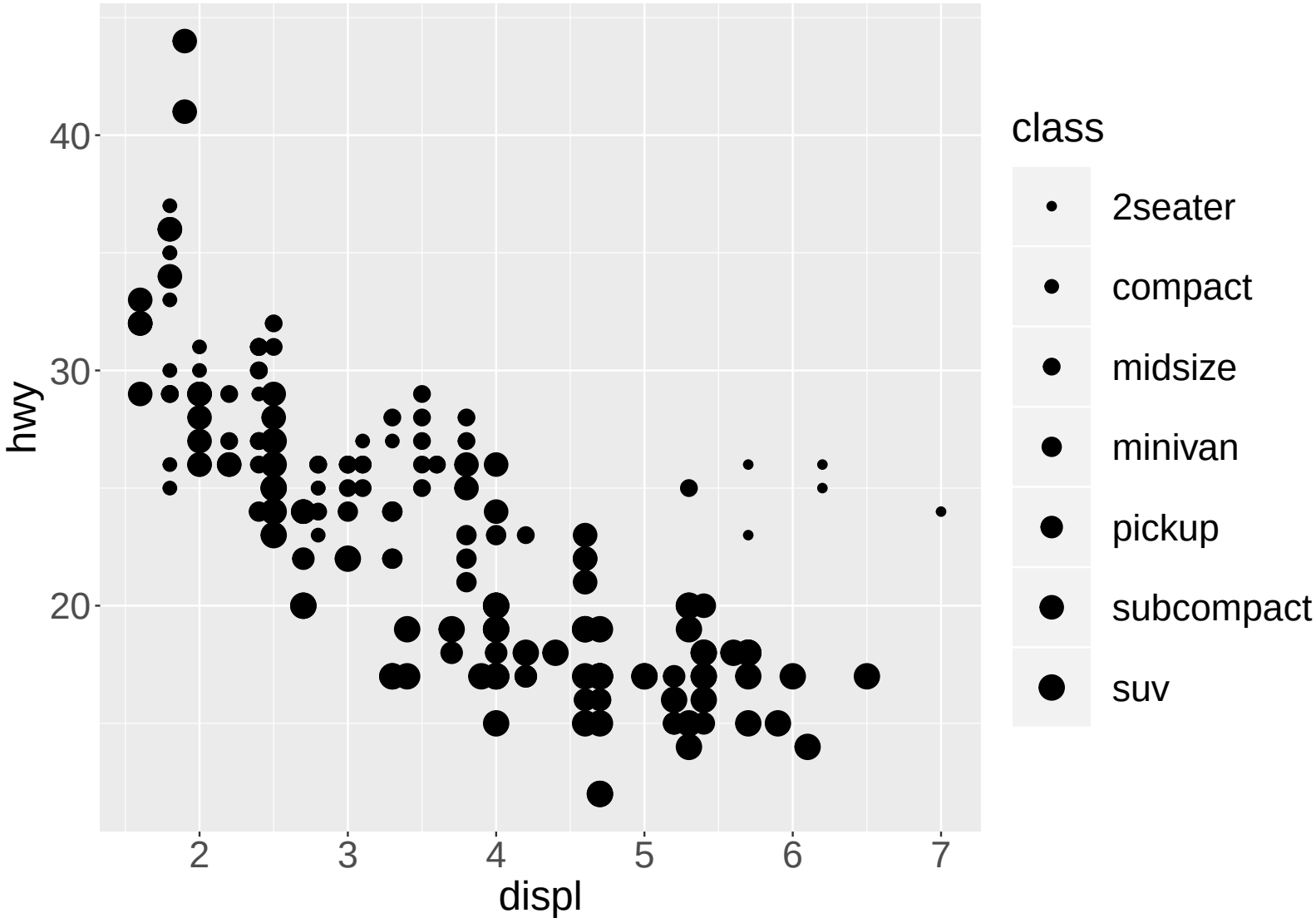
Figure 2: 2008 Chevrolet Corvette

The size aesthetic

Now let´s map `class` to the size aesthetic:

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

The size aesthetic



The size aesthetic

- Each level of the `class` variable is assigned to a different size.
- This plot, however, came with a warning message:

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

- This is because mapping an unordered variable (`class`) to ordered aesthetic (`size`) is not a good idea.
- Ordered aesthetics, like `size` and `alpha`, are more appropriate for continuous variables.

The alpha aesthetic

The `alpha` aesthetic changes the transparency of the points:

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

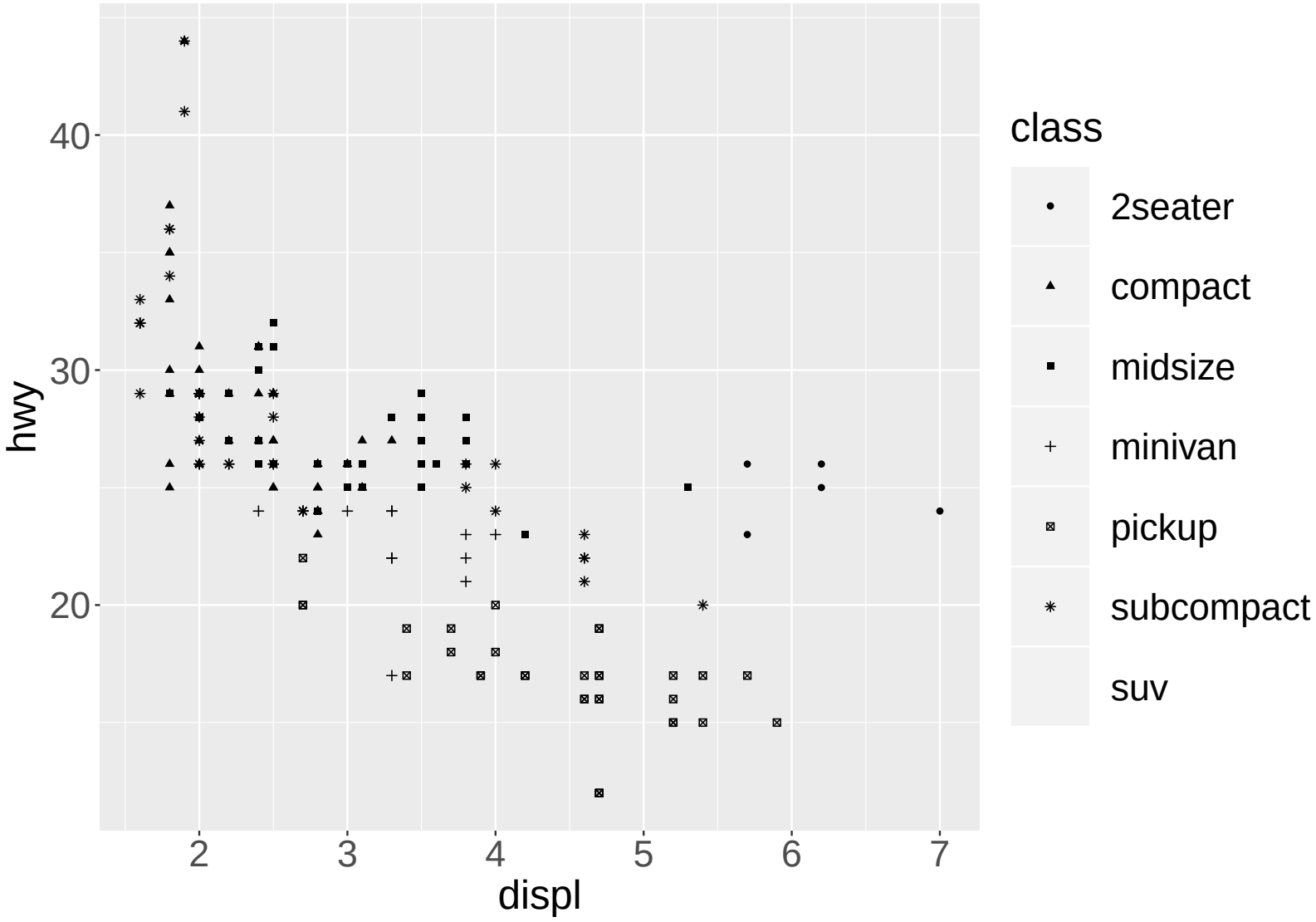
The alpha aesthetic

The shape aesthetic

The `shape` aesthetic changes the shape of the points:

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

The shape aesthetic



The shape aesthetic

- What happened to the SUVs?
- By default ggplot2 uses only up to six shapes at a time.
- This is because points become difficult to discriminate if we use too many shapes.
- We can, however, use more than six shapes if we set them "manually" with `scale_shape_manual()`.

The shape aesthetic

We can choose the following shapes of points by their number:

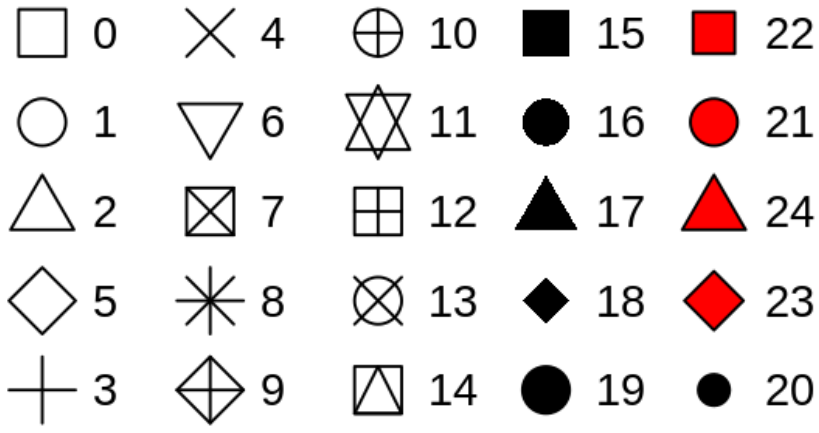


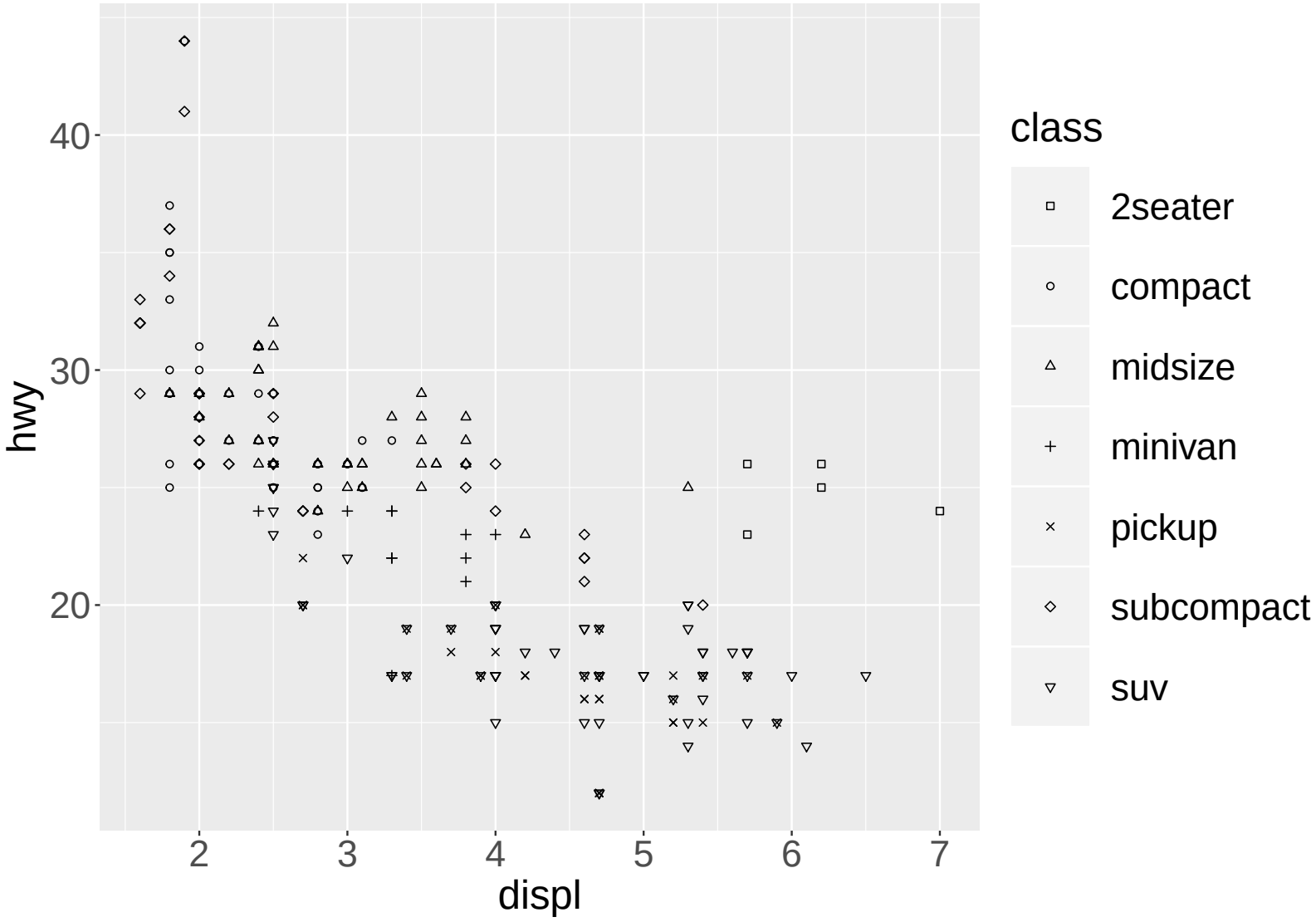
Figure 3: R Built in shapes

The shape aesthetic

Let's choose the first 7 shapes:

```
ggplot(data = mpg) +
  geom_point(aes(x = displ, y = hwy, shape = class)) +
  scale_shape_manual(values = 0:6)
```

The shape aesthetic

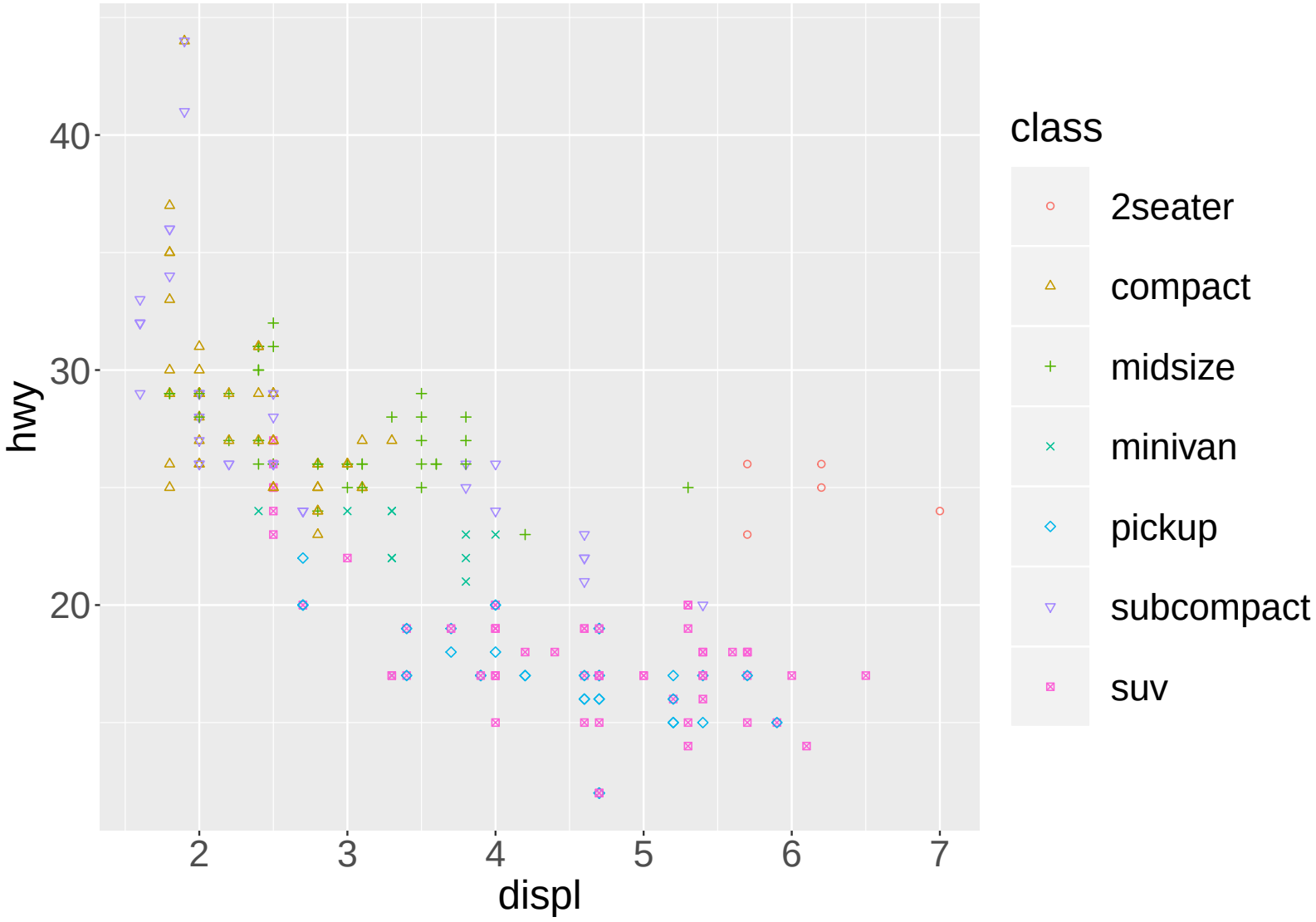


Mapping one variable to two aesthetics

- Mapping a single variable to multiple aesthetics is redundant, and should be avoided in most cases.
- In this case, however, since we have many categories, it can improving visual discrimination.
- Let´s map `class` to `color` and `shape`:

```
ggplot(data = mpg) +
  geom_point(aes(x = displ, y = hwy,
                 shape = class, color = class)) +
  scale_shape_manual(values = 1:7)
```

Mapping one variable to two aesthetics



Mapping continuous variables to aesthetics

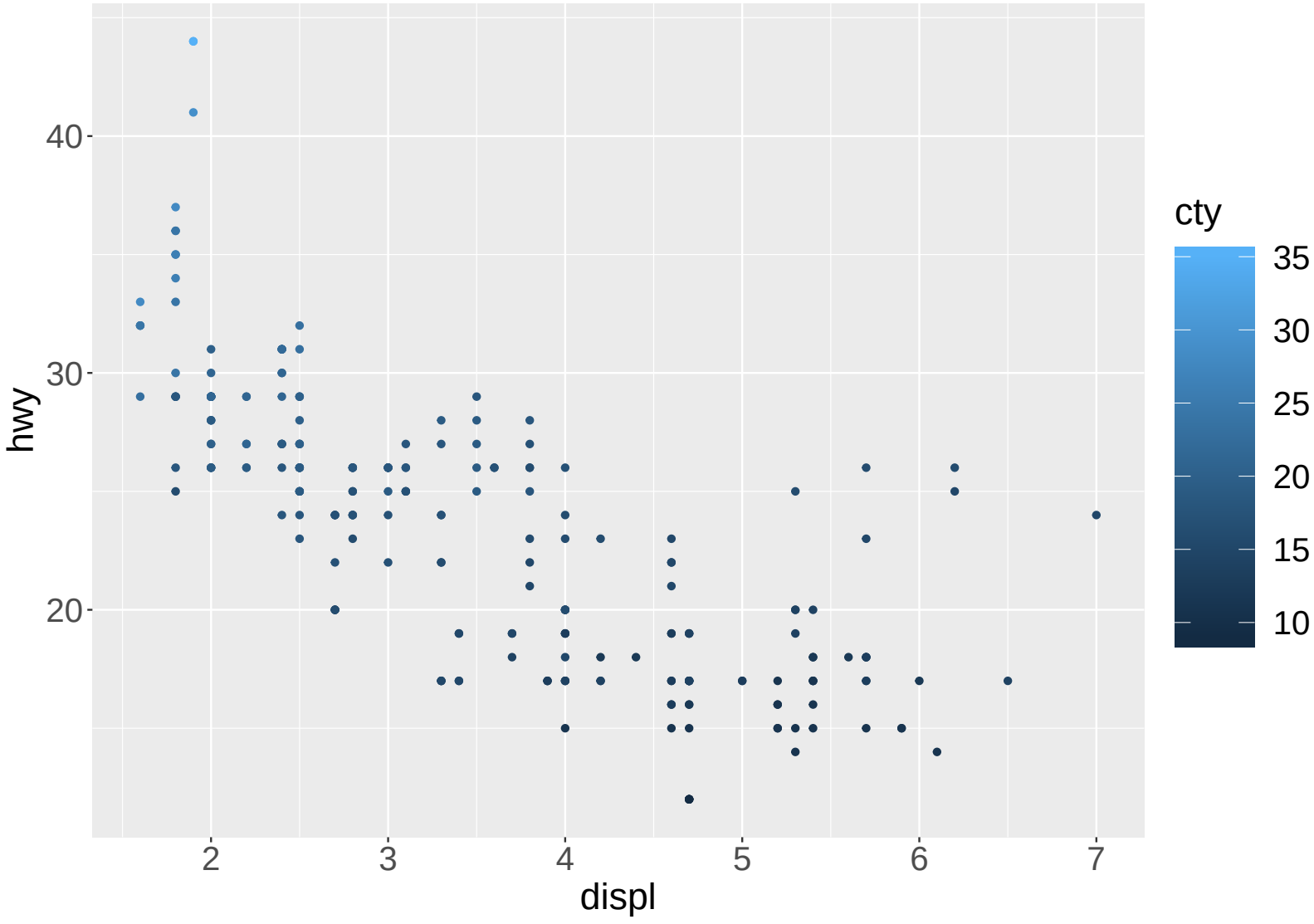
- So far we mapped `class`, a categorical variable, to the `color`, `size`, `shape` and `alpha` aesthetics of `geom_point()`.
- Now let´s map continuous variables to these aesthetics and see how they behaves differently for categorical versus continuous variables.
- The variable `cty` (city miles per gallon) is a continuous variable.

Mapping continuous variables to aesthetics

Let's map `cty` to the `color` aesthetic:

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

Mapping continuous variables to aesthetics

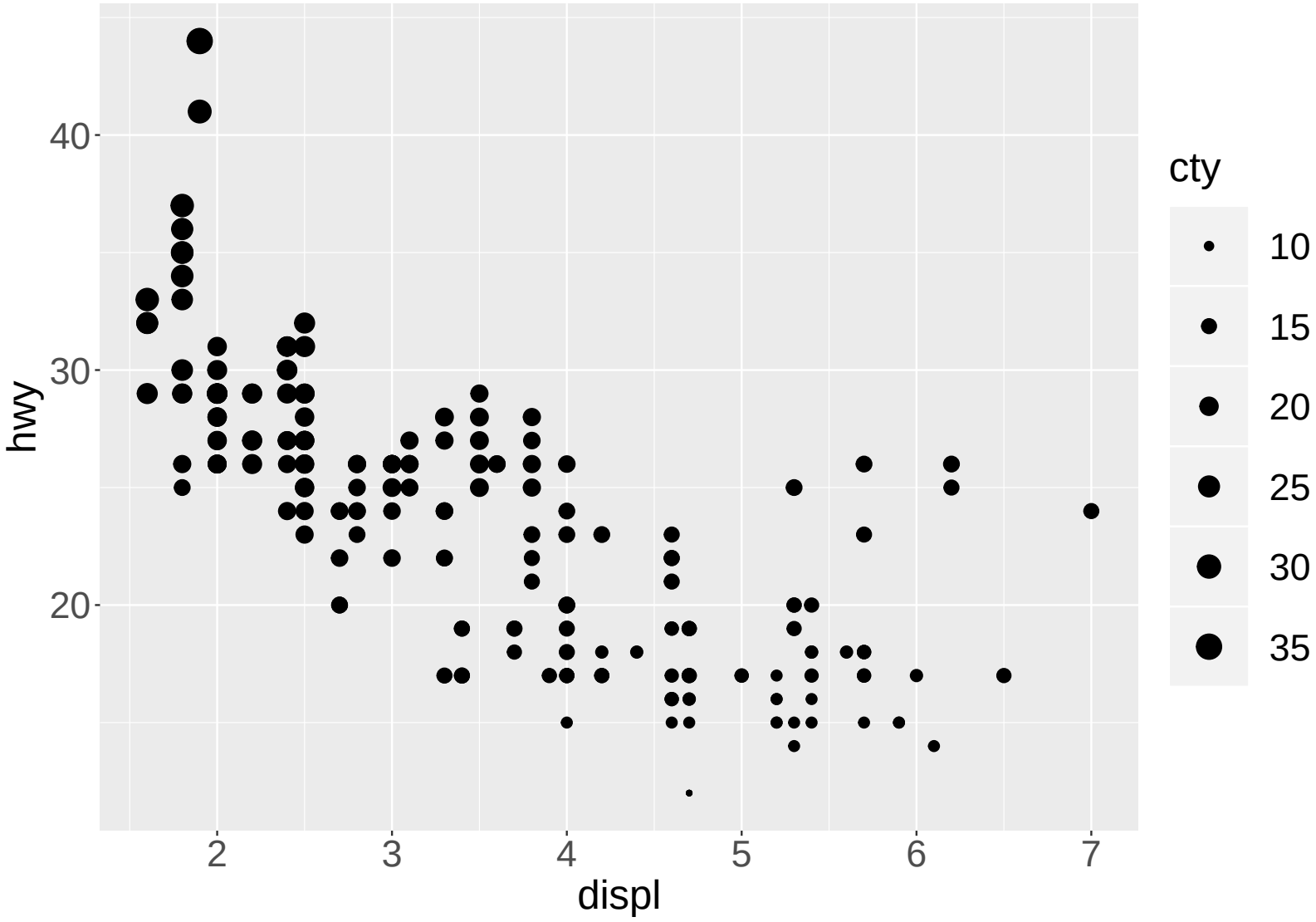


Mapping continuous variables to aesthetics

Now let's map `cty` to the `size` aesthetic:

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

Mapping continuous variables to aesthetics

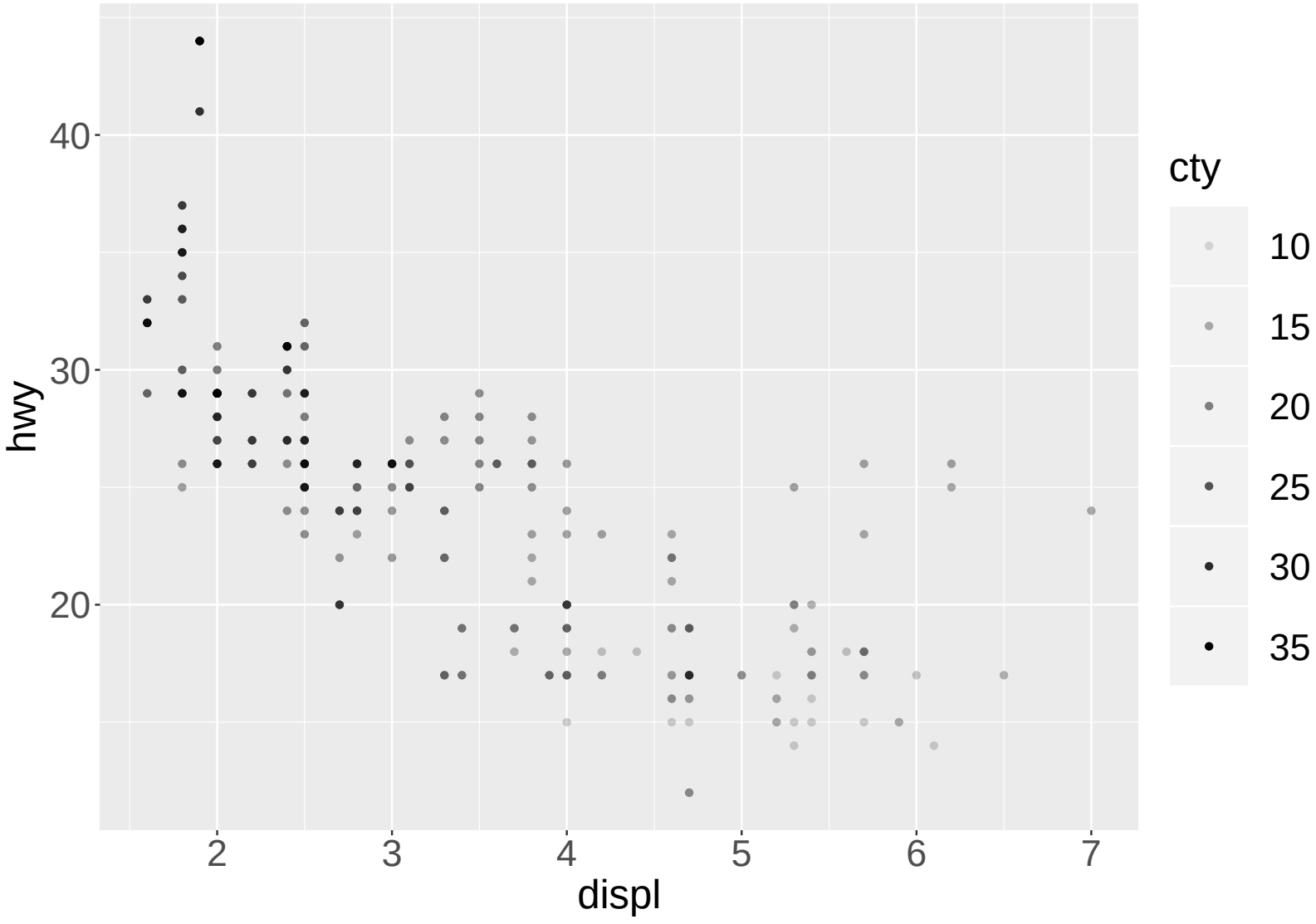


Mapping continuous variables to aesthetics

Now let's map `cty` to the `alpha` aesthetic:

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

Mapping continuous variables to aesthetics



Mapping continuous variables to aesthetics

And finally let's map `cty` to the `shape` aesthetic:

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

#> Error: A continuous variable can not be mapped to
#> shape
```

Aesthetic mappings: categorical vs continuous variables

Color and **alpha** aesthetics:

- Each level of a categorical variable is assigned to a different (discrete) color.
- The color of the points vary continuously from light to dark as a function of the values of a continuous variables.

Size aesthetic:

- Each level of a categorical variable is assigned to a different (discrete) size.
- The size of the points vary continuously as a function of the values of a continuous variables.

Aesthetic mappings: categorical vs continuous variables

Shape aesthetic:

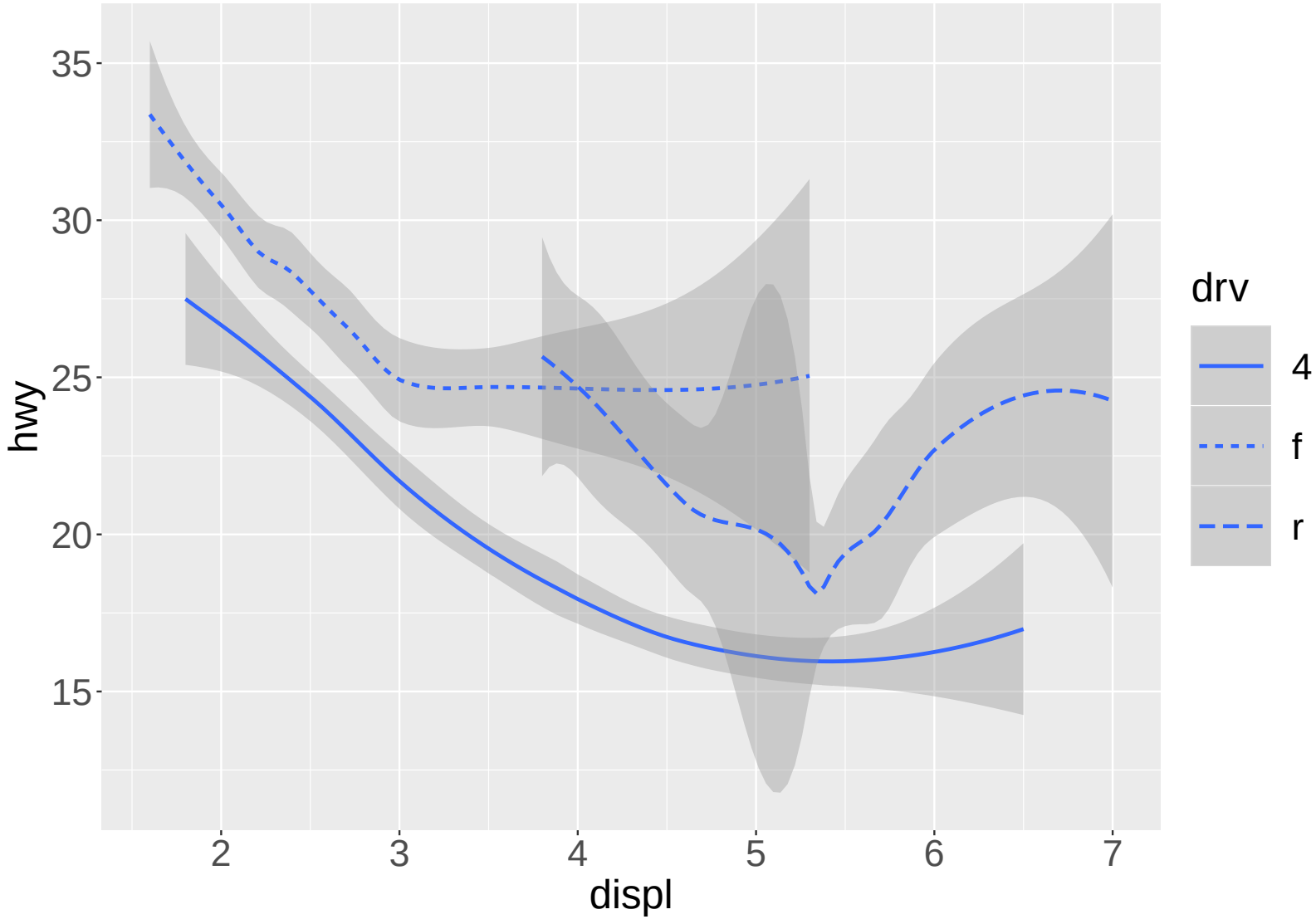
- A different shape is assigned to each level of a categorical value.
- The shape aesthetic can not be mapped to continuous variables.

The `linetype` aesthetic

`geom_smooth()` draws a different type of line for each level of the variable that is mapped to the `linetype` aesthetic:

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

The `linetype` aesthetic



The `linetype` aesthetic

- The `linetype` aesthetic automatically adds a legend.
- The `linetype` aesthetic can not be mapped to continuous variables.

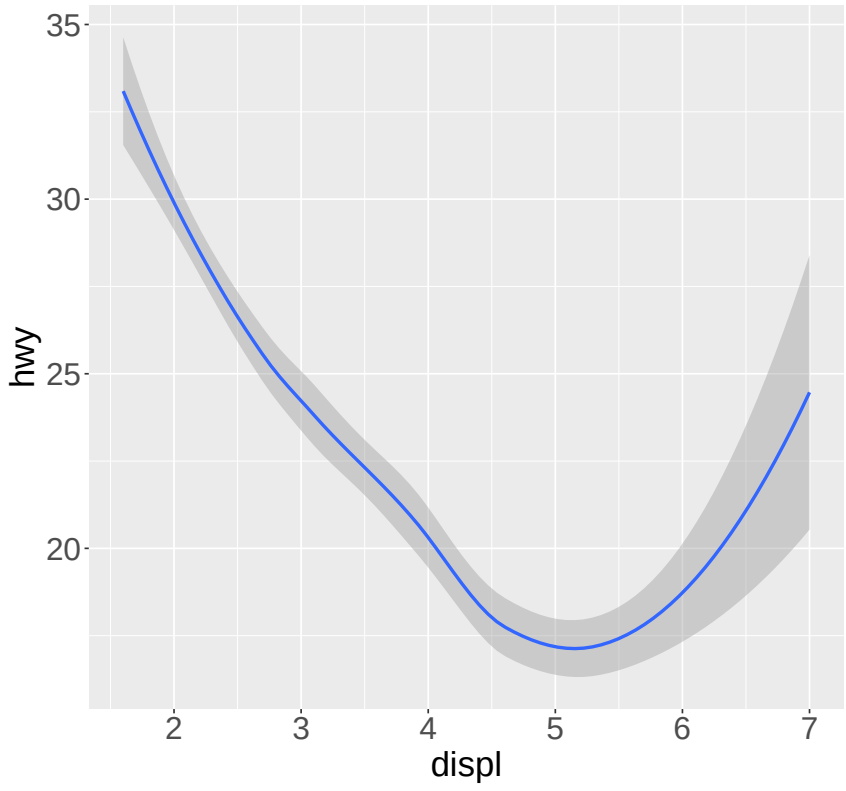
The `group` aesthetic

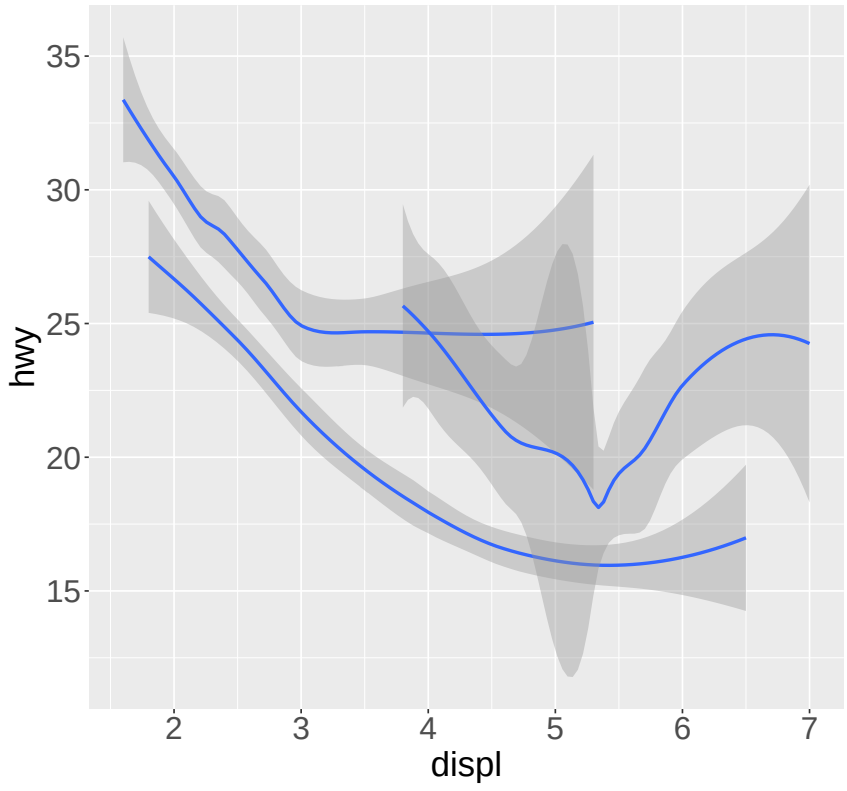
- The `group` aesthetic also draws a separate object for each level of a discrete variable.
- The `group` aesthetic does not add a legend or any other distinguishing feature.

The group aesthetic

```
p1 <- ggplot(data = mpg) +
  geom_smooth(aes(x = displ, y = hwy))

p2 <- ggplot(data = mpg) +
  geom_smooth(aes(x = displ, y = hwy, group = drv))
```

The group aesthetic

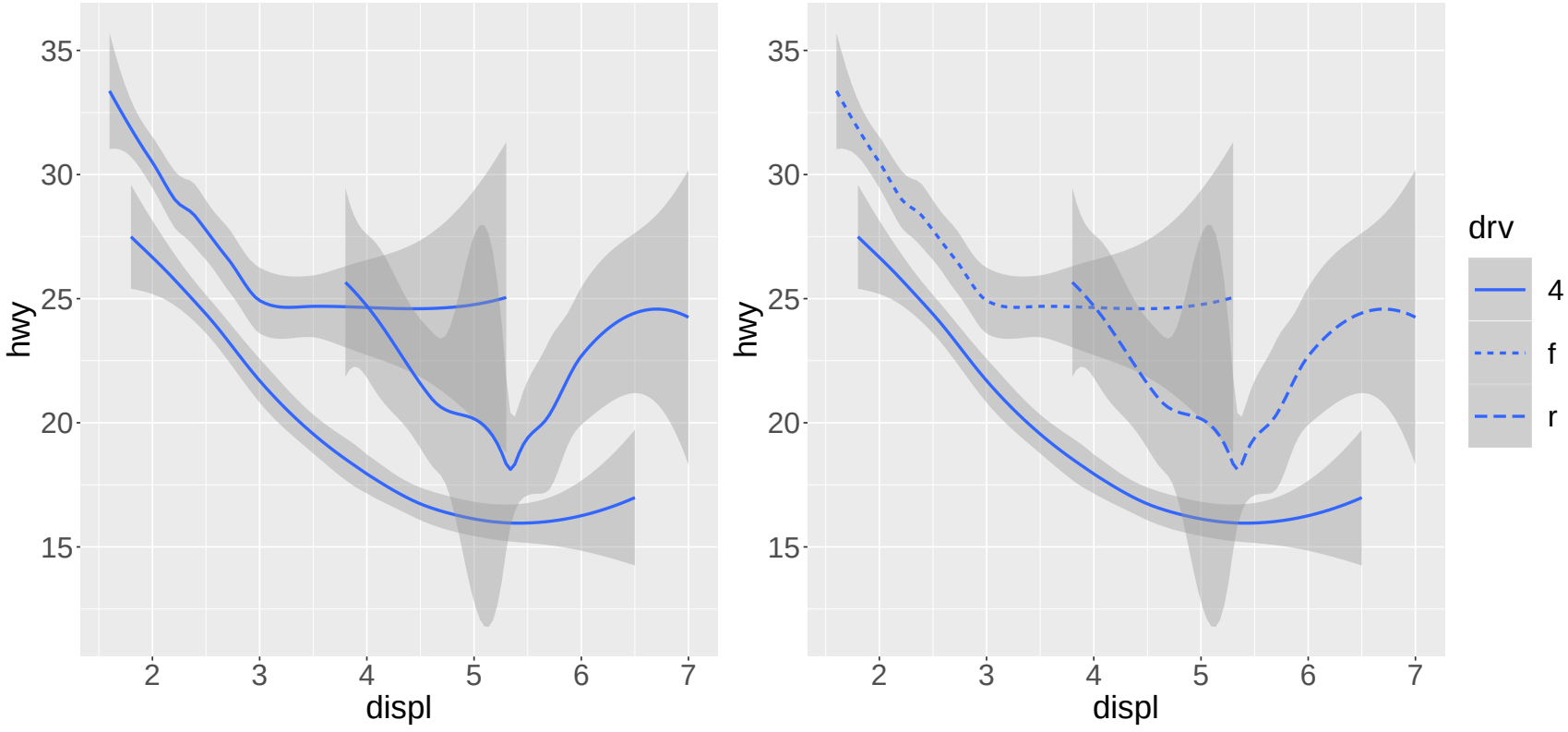




group vs linetype

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

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

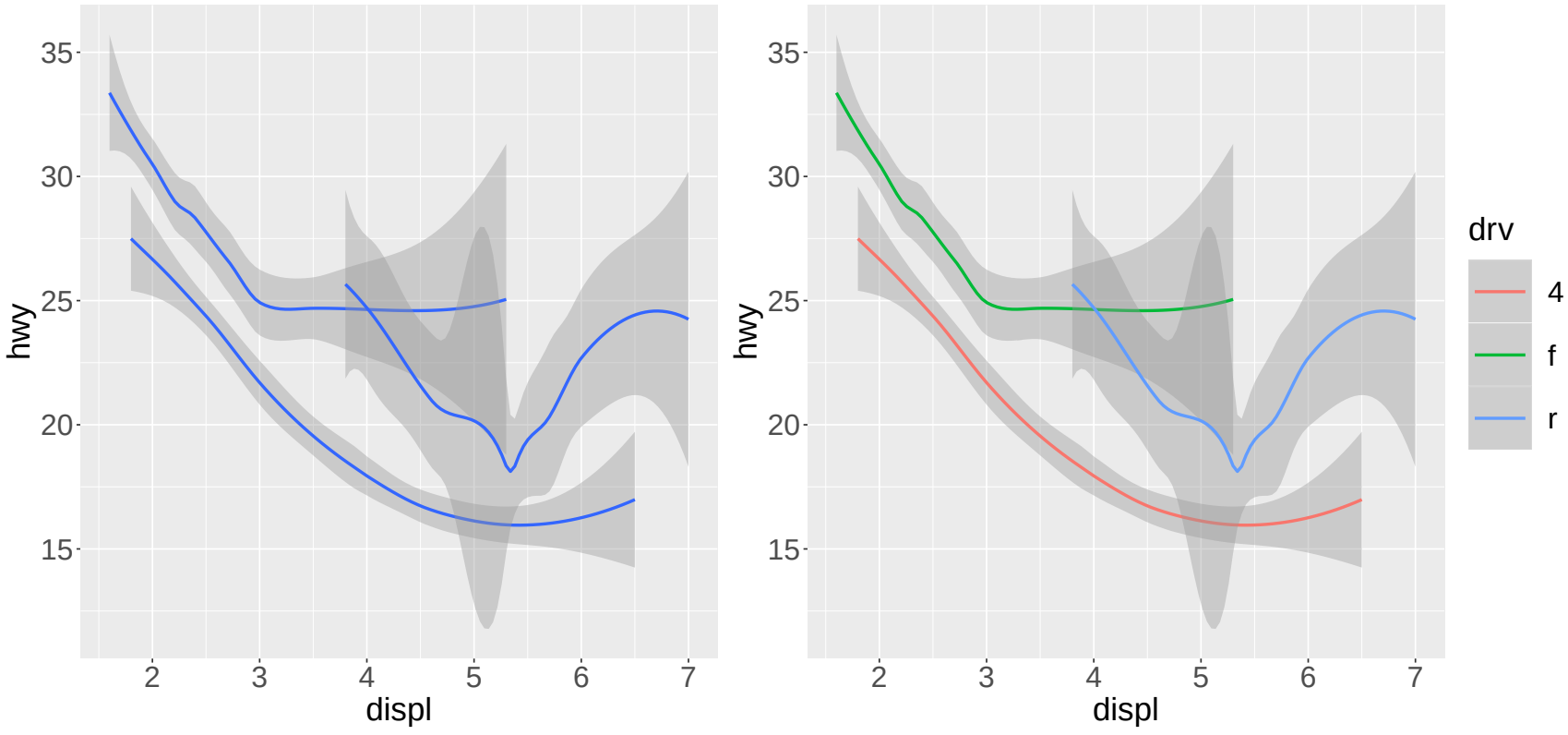
group vs linetype



group vs color

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

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

group vs color



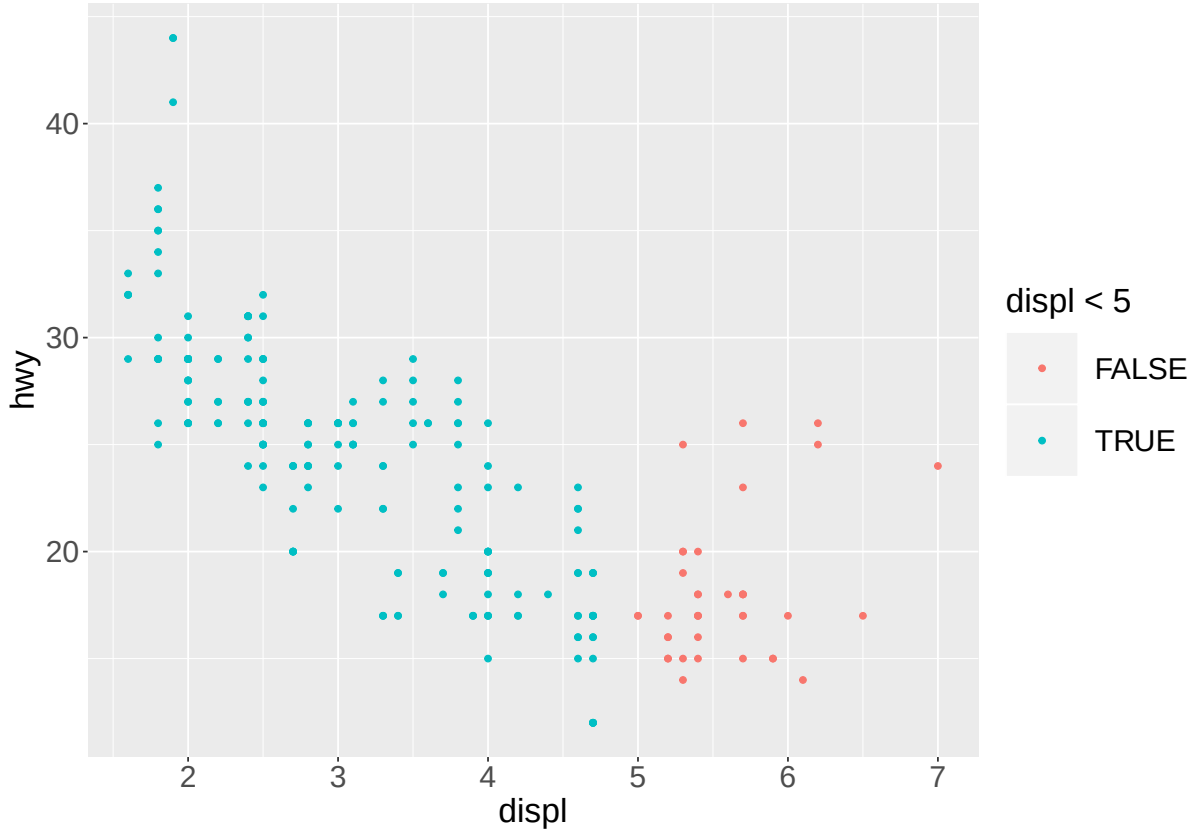
Aesthetics and logical conditions

Aesthetics can be mapped to logical expressions. For example, if we map an aesthetic to `displ < 5`:

- `ggplot()` creates a temporary variable with values equal to the result `displ < 5`.
- The result of `displ < 5` is a logical variable.
- `ggplot()` then maps the aesthetic to the temporary variable.

Aesthetics and logical conditions

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



Aesthetics and logical conditions

```
ggplot(mpg) +
  geom_point(aes(x = displ, y = hwy,
                 color = class == "2seater"))
```

Changing default aesthetic properties of geoms

- To change a default aesthetic property, set the aesthetic by name as an argument of the geom function.
  - Names of colors should be indicated as character strings.
  - Size of points should be indicated in mm.
  - Shapes are identified by the numbers in figure 3.

Changing default aesthetic properties of geoms

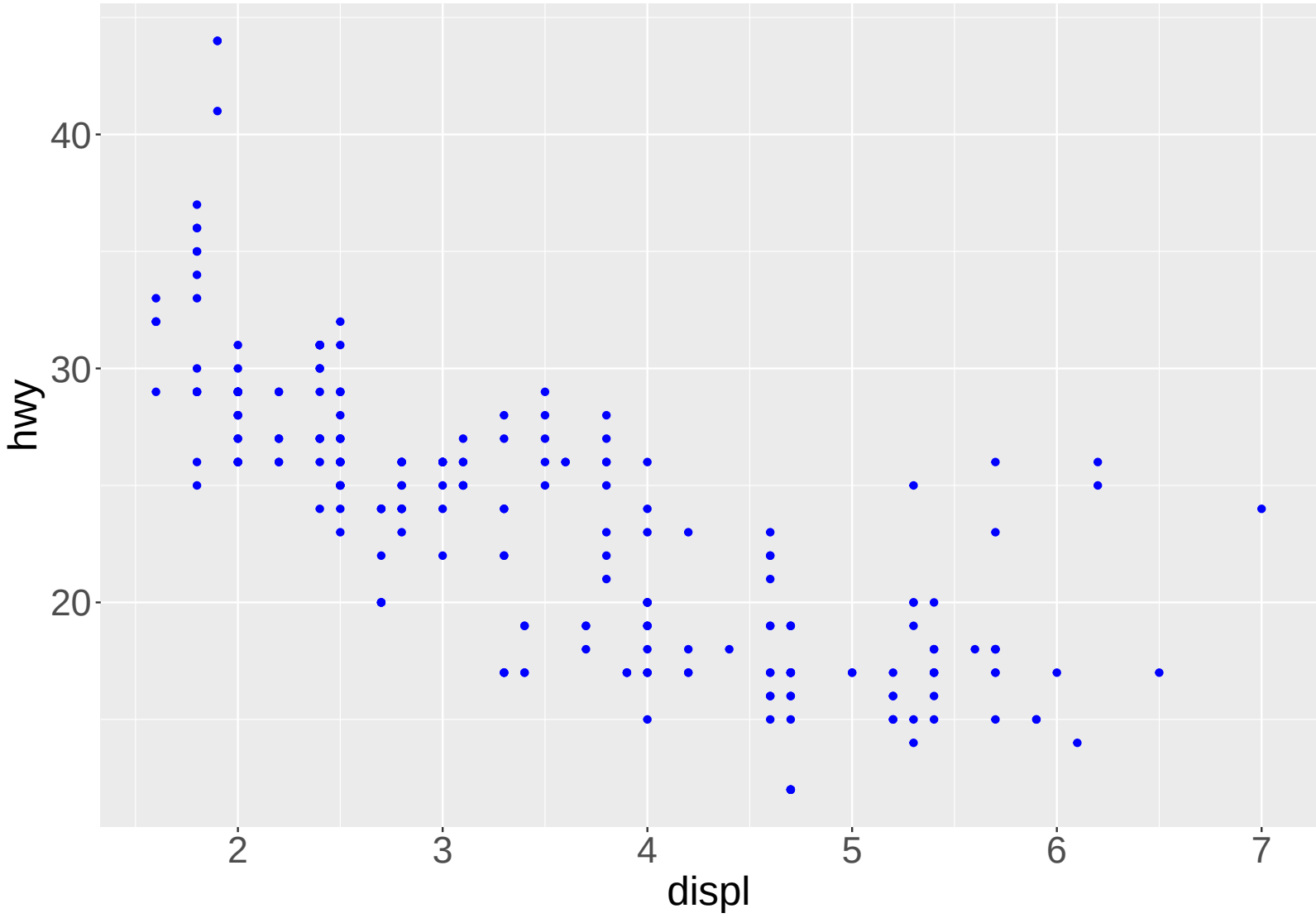
- There are some seeming duplicate shapes (e.g. 0, 15, and 22 are all squares).
- The difference comes from the interaction of the `color` and `fill` aesthetics:
  - The hollow shapes (0–14) have a border determined by the `color` aesthetic
  - The solid shapes (15–18) are filled with the color `aesthetic`
  - The filled shapes (21–24) have a border set by the `color` aesthetic and are filled with the `fill` aesthetic

Changing default aesthetic properties of geoms

For example, we can make all of the points blue:

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

Changing default aesthetic properties of geoms

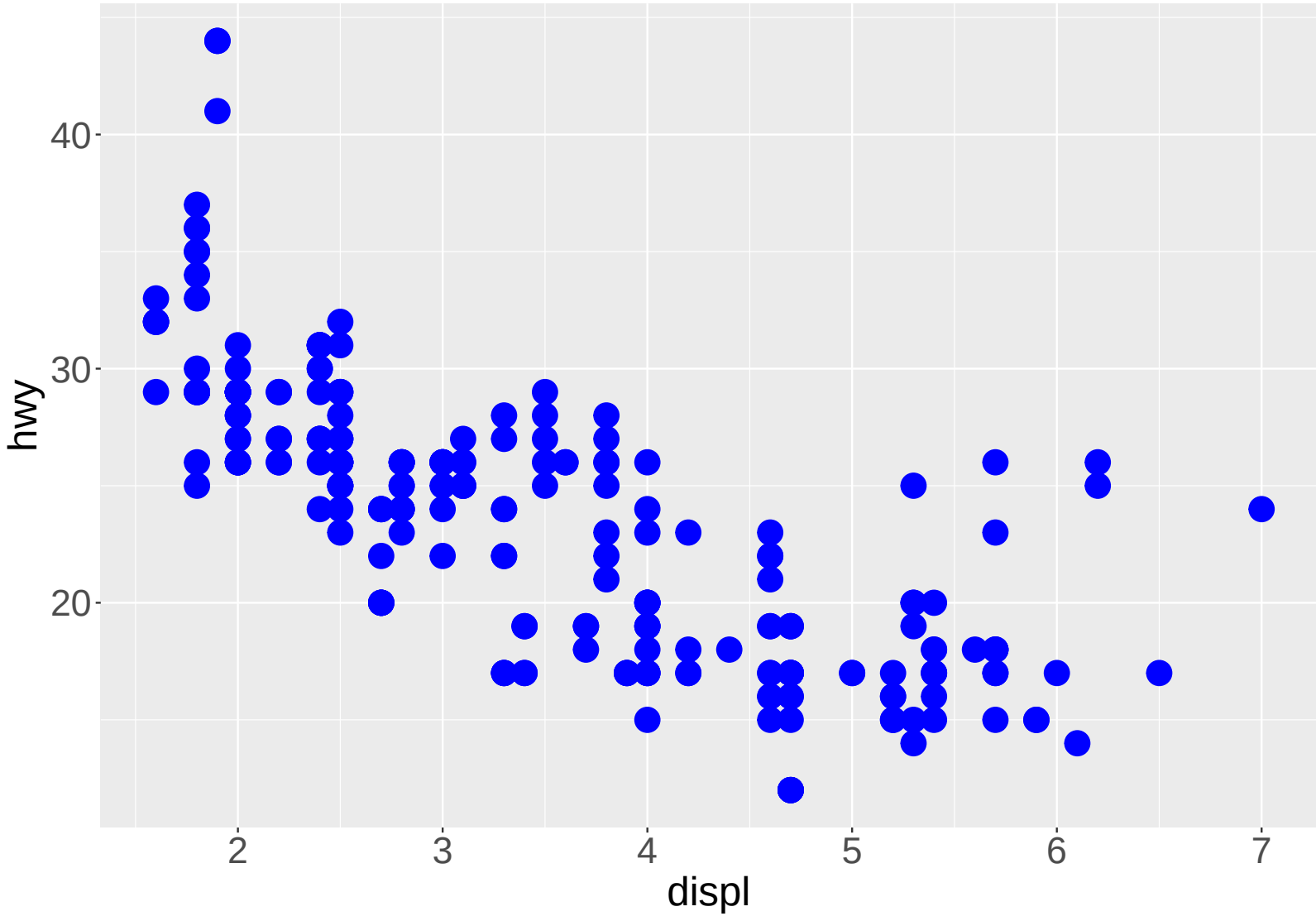


Changing default aesthetic properties of geoms

Now let's change the size of the points:

```
ggplot(data = mpg) +
  geom_point(aes(x = displ, y = hwy),
    color = "blue",
    size = 6)
```

Changing default aesthetic properties of geoms

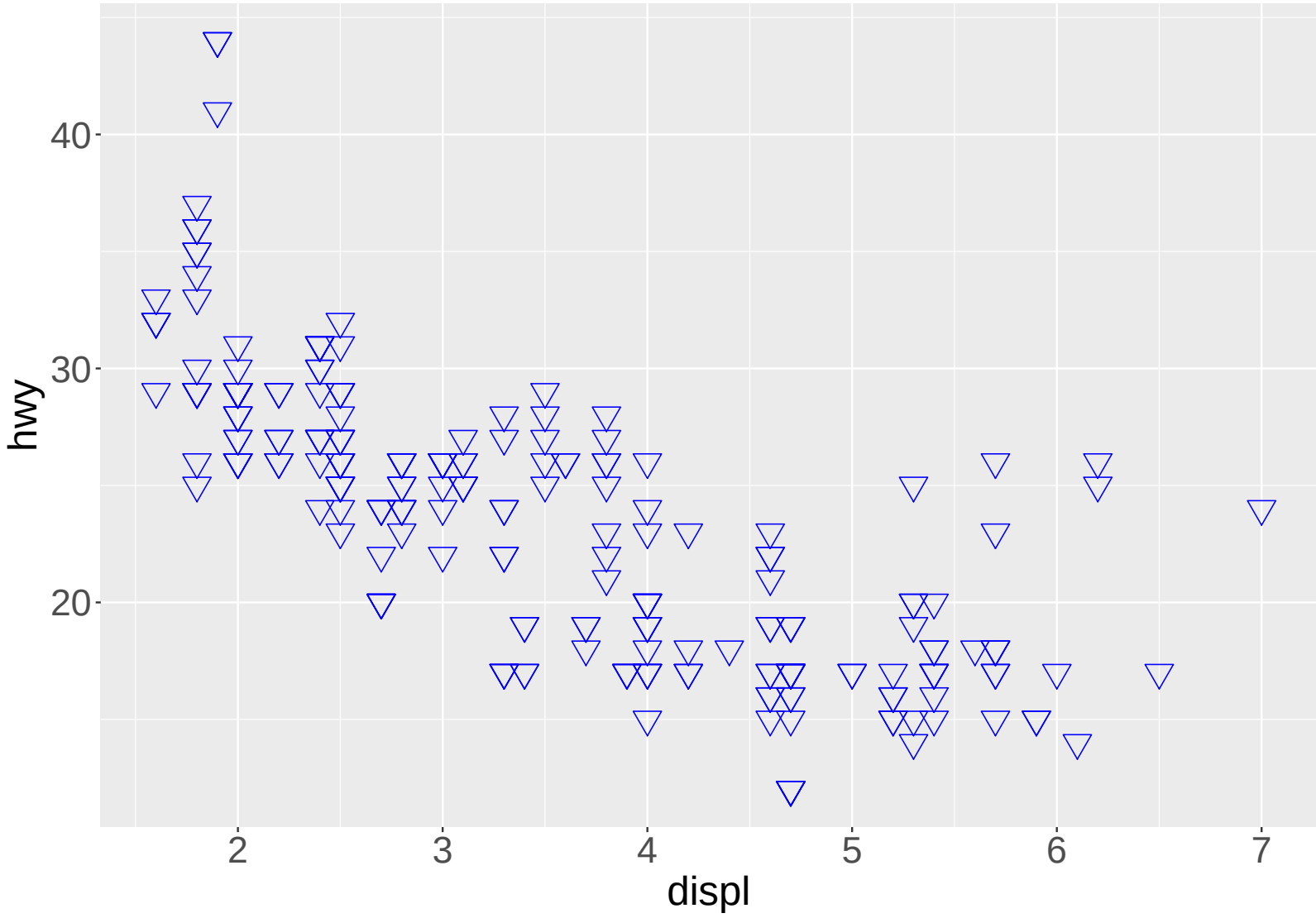


Changing default aesthetic properties of geoms

We can also change the default shape of the points:

```
ggplot(data = mpg) +
  geom_point(aes(x = displ, y = hwy),
    color = "blue",
    size = 5,
    shape = 6)
```

Changing default aesthetic properties of geoms

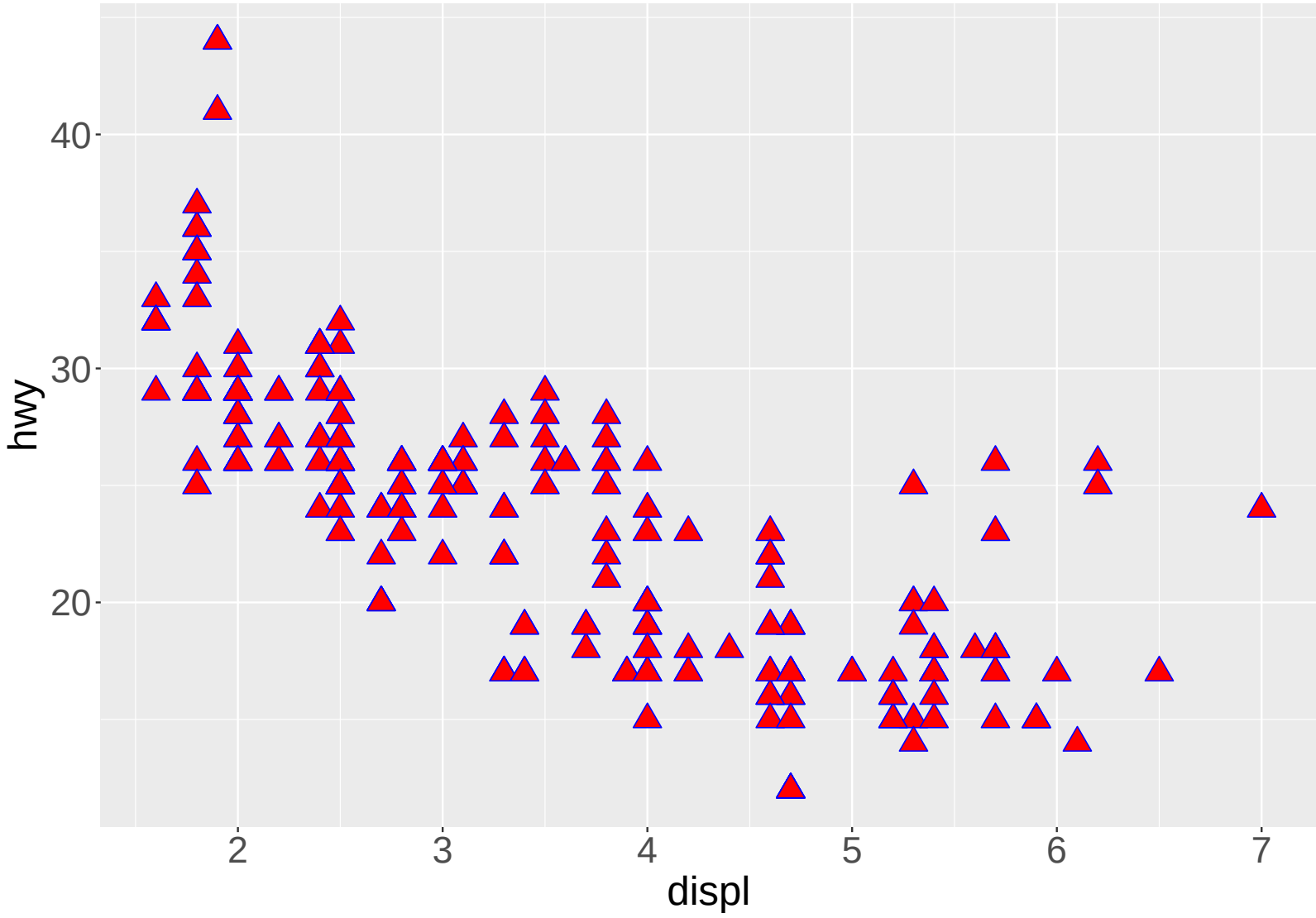


Changing default aesthetic properties of geoms

- Shape 6 is a hollow shape, it interacts only with the `color` aesthetic.
- If we want triangles filled with color, we must use shape 24, which interacts both with the color and `fill` aesthetics:

```
ggplot(data = mpg) +
  geom_point(
    mapping = aes(x = displ, y = hwy),
    color = "blue",
    size = 5,
    shape = 24,
    fill = "red"
  )
```

Changing default aesthetic properties of geoms

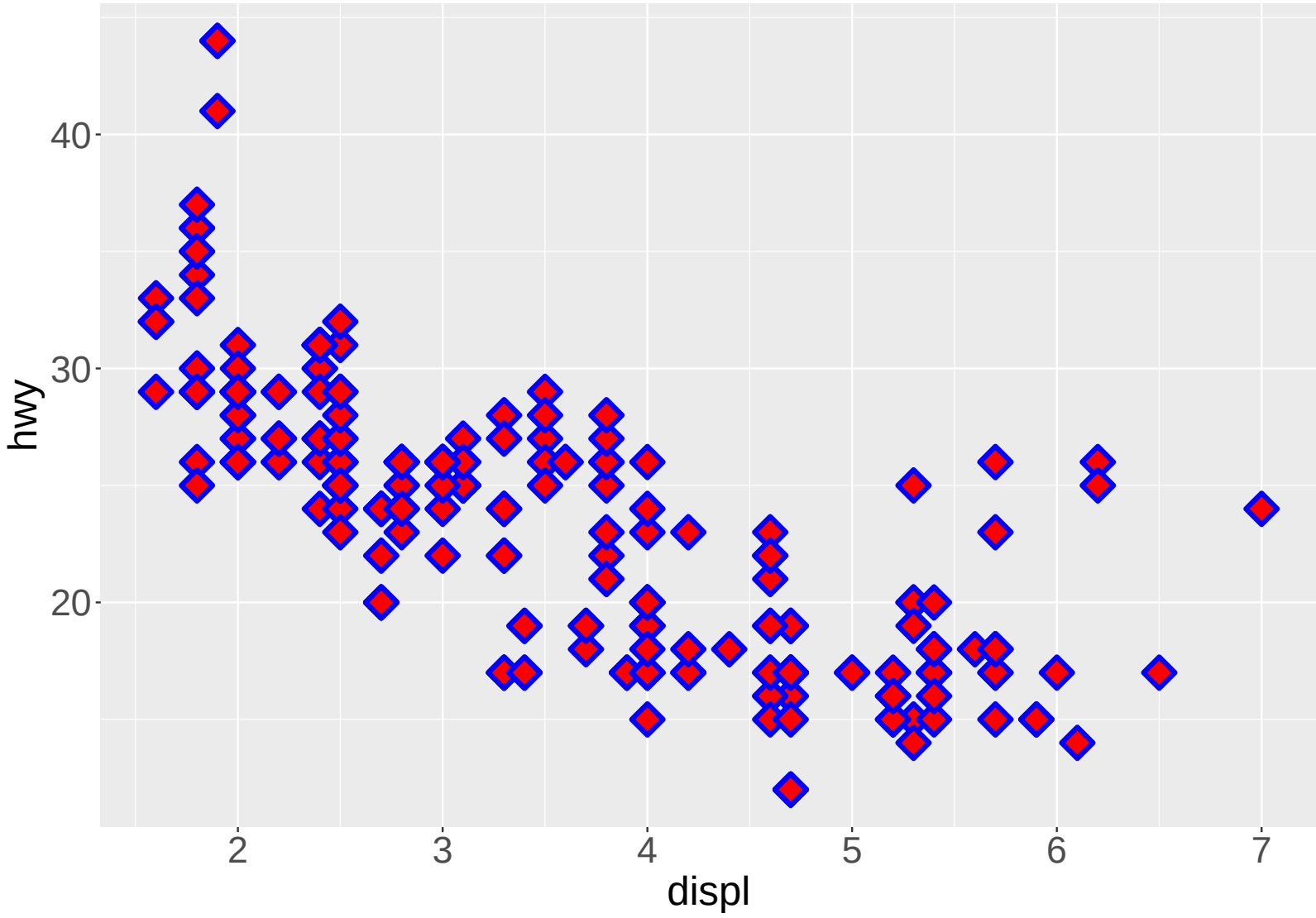


Changing default aesthetic properties of geoms

The `stroke` aesthetic changes the thickness of the border of the points:

```
ggplot(data = mpg) +
  geom_point(aes(x = displ, y = hwy),
    color = "blue",
    size = 5,
    shape = 23,
    fill = "red",
    stroke = 2
  )
```

Changing default aesthetic properties of geoms

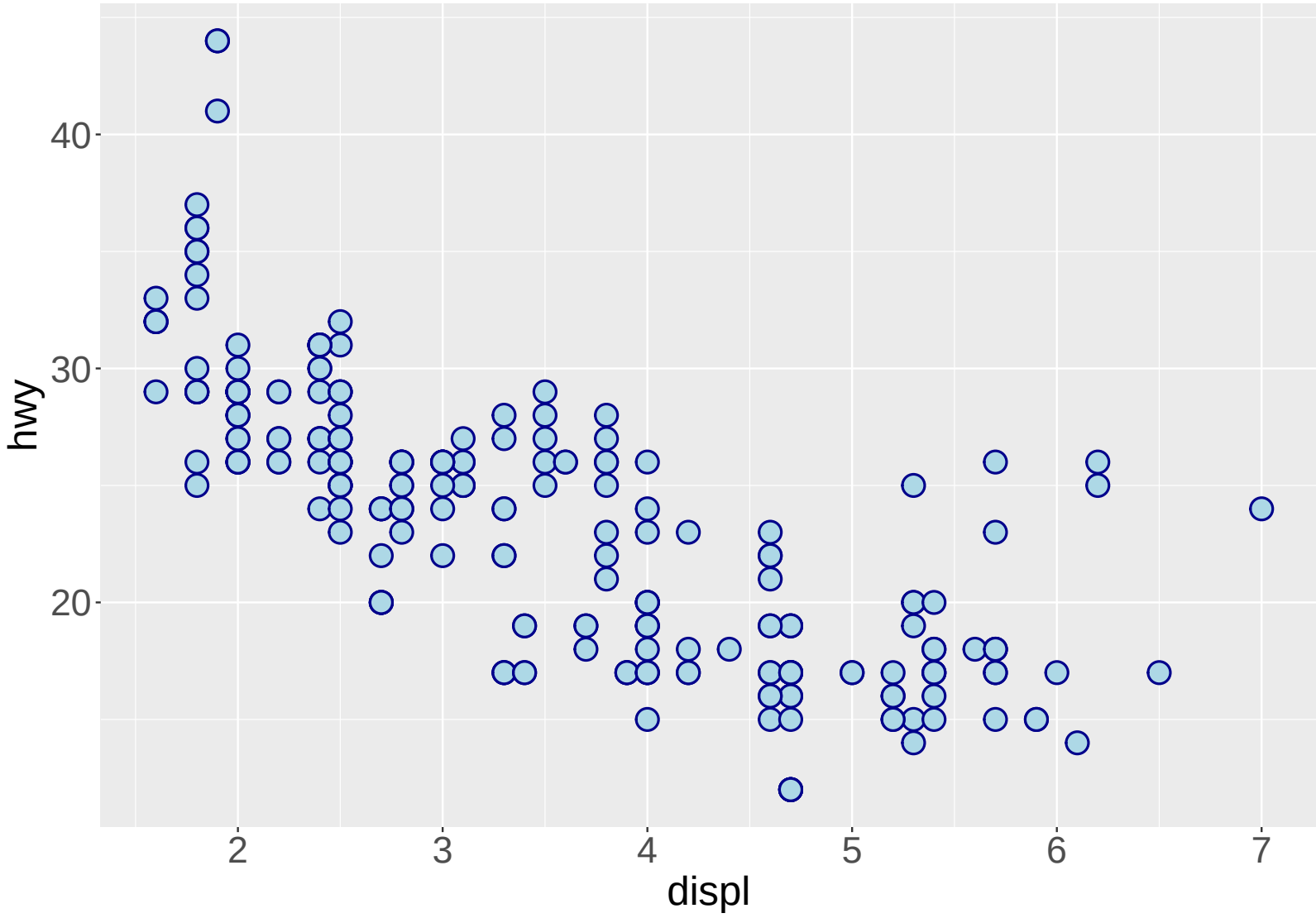


Changing default aesthetic properties of geoms

We can also use colors like dark blue and light blue:

```
ggplot(data = mpg) +
  geom_point(aes(x = displ, y = hwy),
    color = "darkblue",
    fill = "lightblue",
    shape = 21,
    size = 5,
    stroke = 1
  )
```

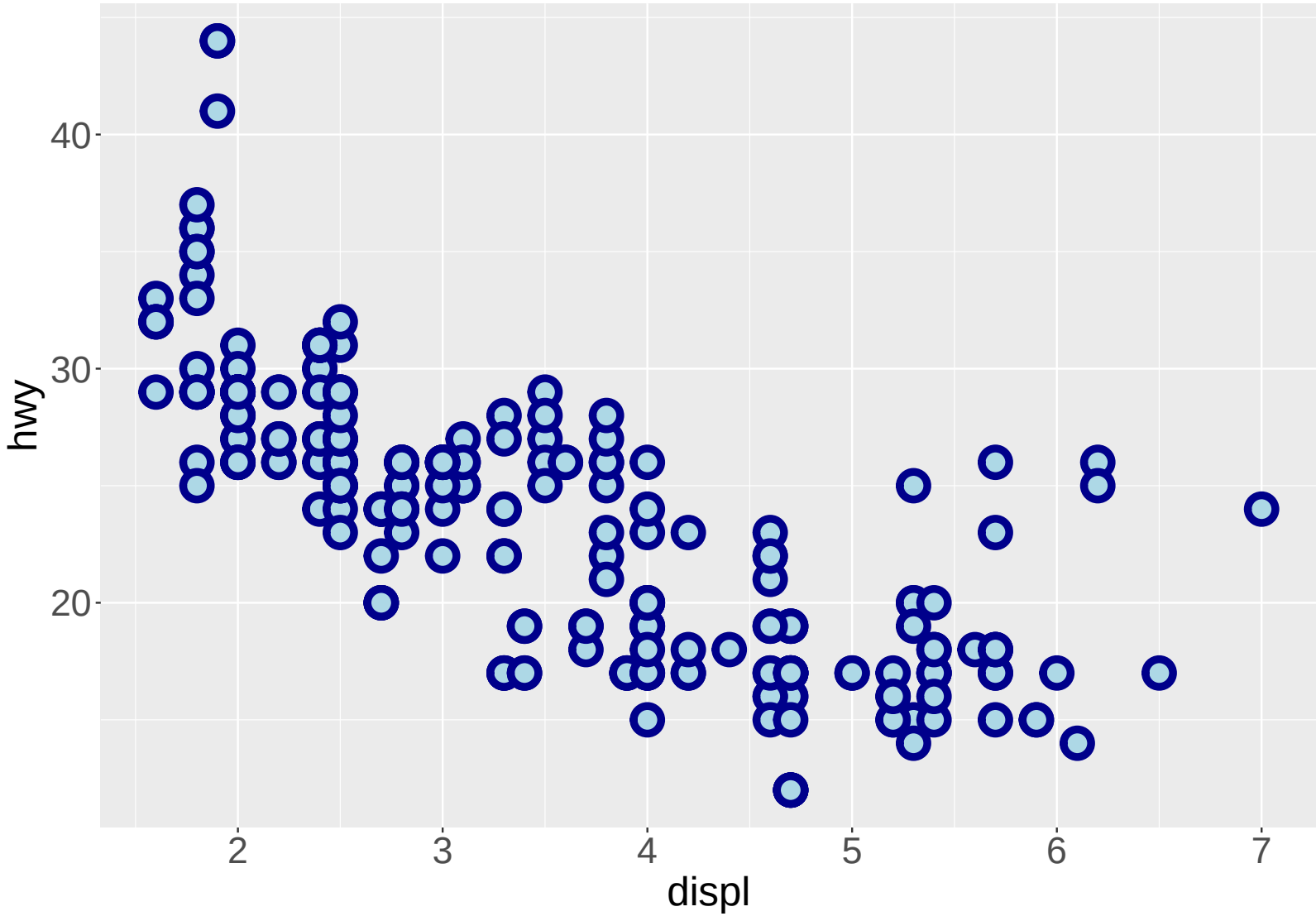
Changing default aesthetic properties of geoms



Changing default aesthetic properties of geoms

```
ggplot(data = mpg) +
  geom_point(aes(x = displ, y = hwy),
    color = "darkblue",
    fill = "lightblue",
    shape = 21,
    size = 5,
    stroke = 3
  )
```

Changing default aesthetic properties of geoms



Changing default aesthetic properties of geoms

- For a list of available colors and their names see: http://sape.inf.usi.ch/quick-reference/ggplot2/color

Changing default aesthetic properties of geoms

- Alternatively, we can use RGB codes.
- The RGB model reproduces a broad array of colors by mixing together red, green, and blue in different proportions.
- We can specify colors like in HTML/CSS, using the hexadecimal values (00 to FF) for red, green, and blue, concatenated into a string, prefixed with a "#".
- A pure red color this is represented with "#FF0000".

See:

- https://www.rapidtables.com/web/color/RGB_Color.html
- https://htmlcolorcodes.com

Changing default aesthetic properties of geoms

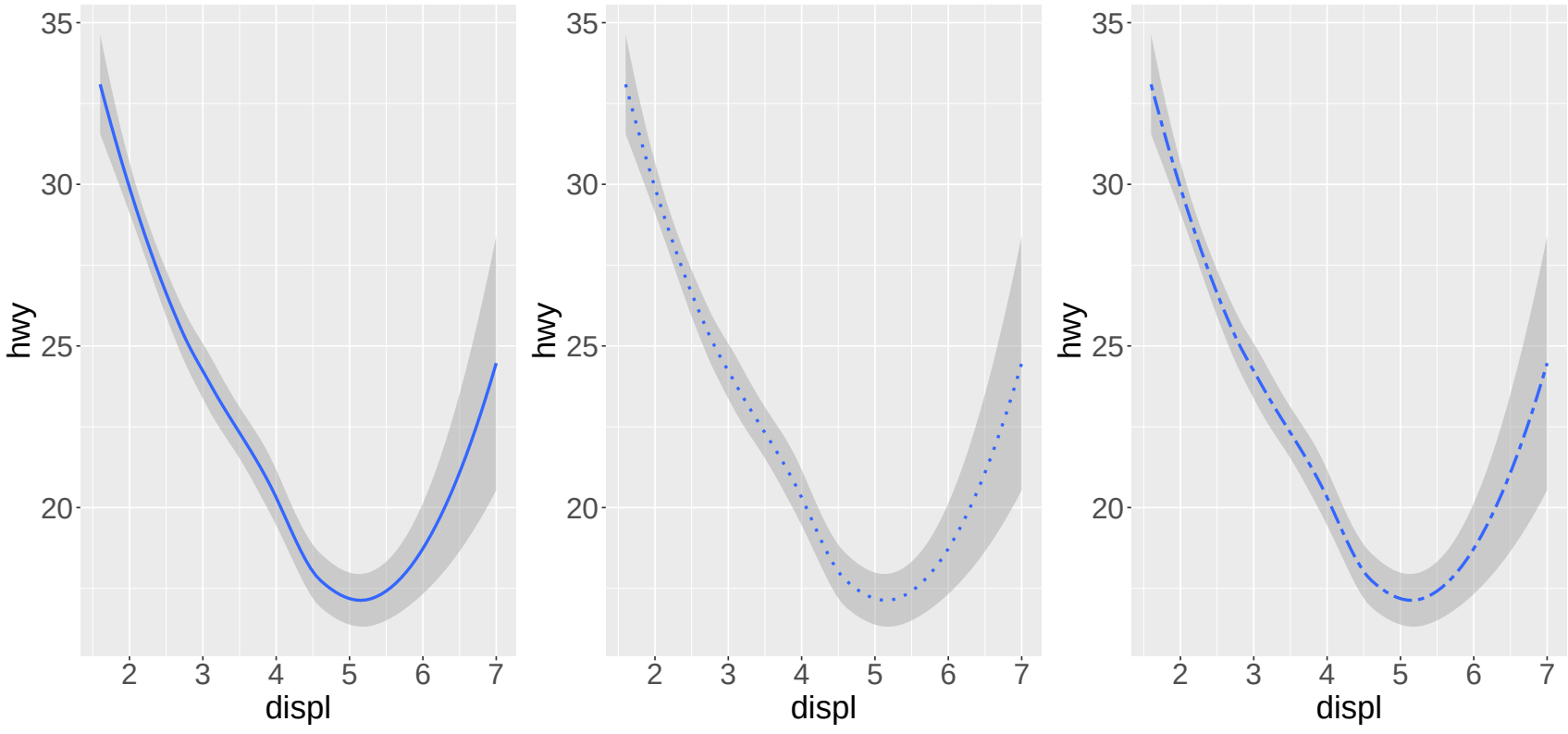
We can change the default line type of geom functions like `geom_smooth()`.

Figure 4: Built in line types

Changing default aesthetic properties of geoms

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

ggplot(data = mpg) +
  geom_smooth(aes(x = displ, y = hwy),
    linetype = "dotted")

ggplot(data = mpg) +
  geom_smooth(aes(x = displ, y = hwy),
    linetype = "twodash")
```

Changing default aesthetic properties of geoms

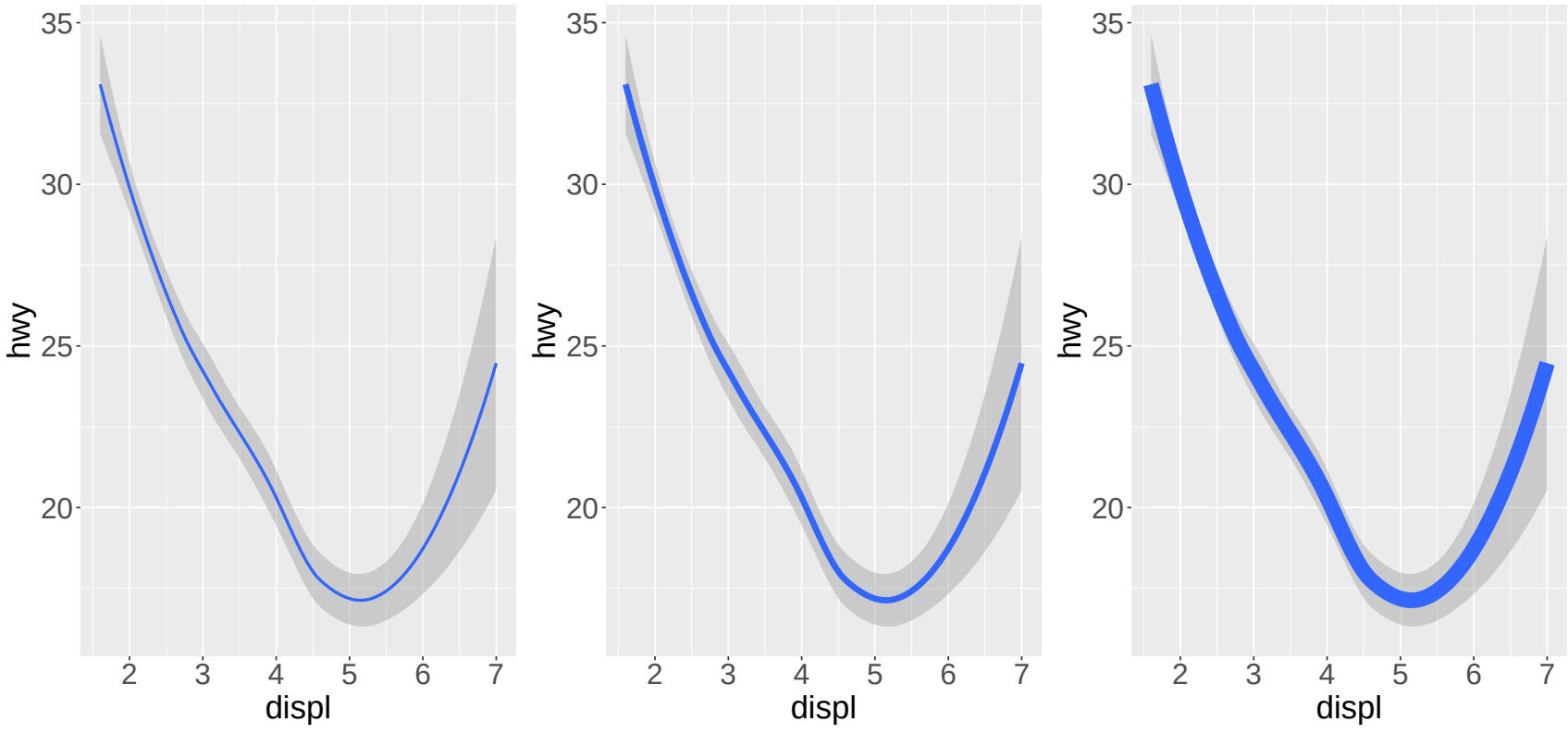


Changing default aesthetic properties of geoms

We can also change the thickness of the lines:

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

ggplot(data = mpg) +
  geom_smooth(aes(x = displ, y = hwy), size = 2)

ggplot(data = mpg) +
  geom_smooth(aes(x = displ, y = hwy), size = 5)
```

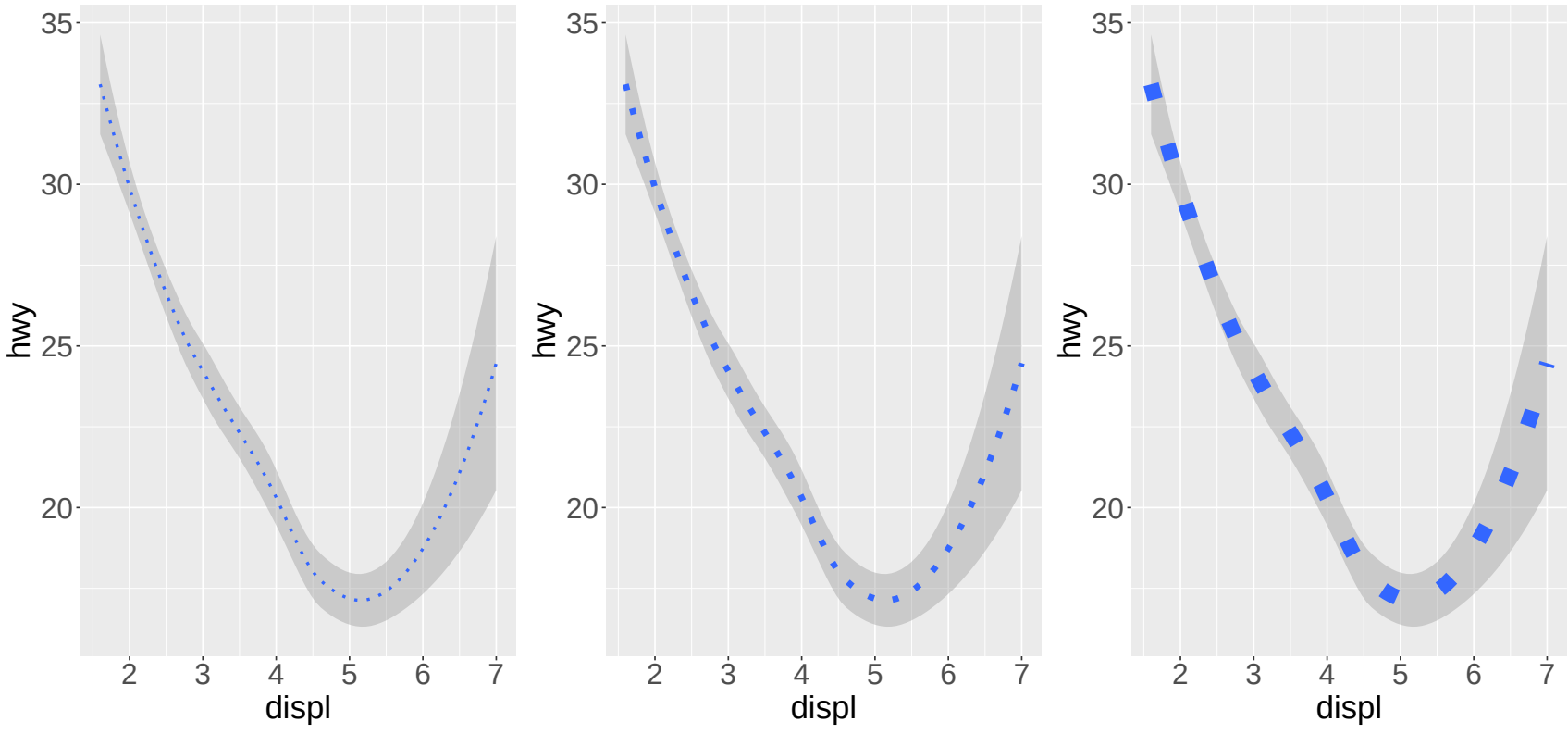
Changing default aesthetic properties of geoms



Changing default aesthetic properties of geoms

```
ggplot(data = mpg) +
  geom_smooth(aes(x = displ, y = hwy),
              linetype = "dotted")

ggplot(data = mpg) +
  geom_smooth(aes(x = displ, y = hwy),
              linetype = "dotted", size = 2)

ggplot(data = mpg) +
  geom_smooth(aes(x = displ, y = hwy),
              linetype = "dotted", size = 5)
```

Changing default aesthetic properties of geoms

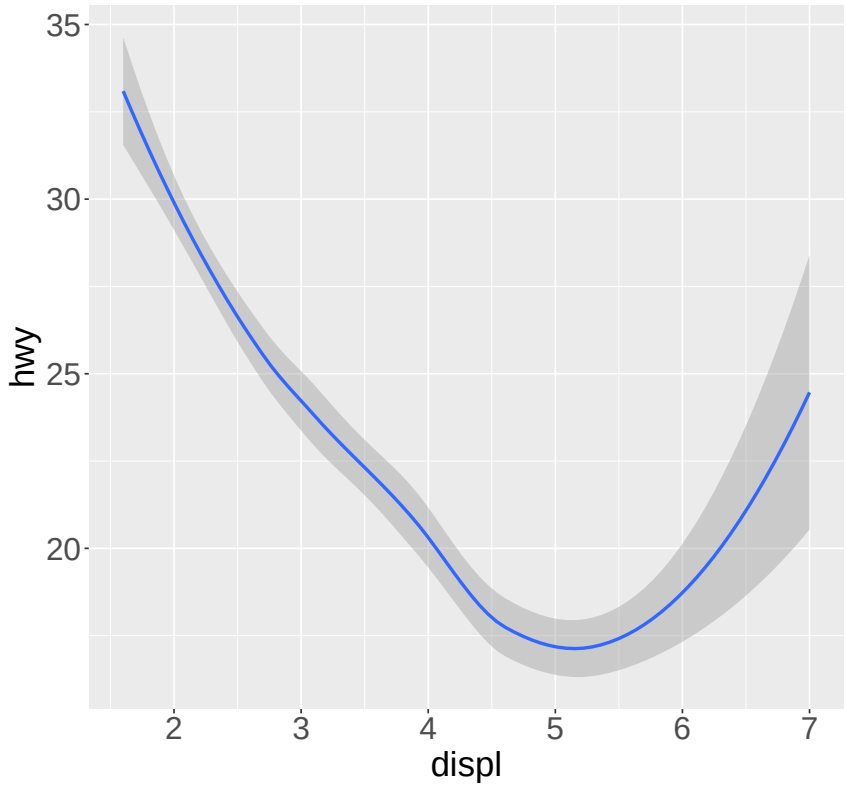


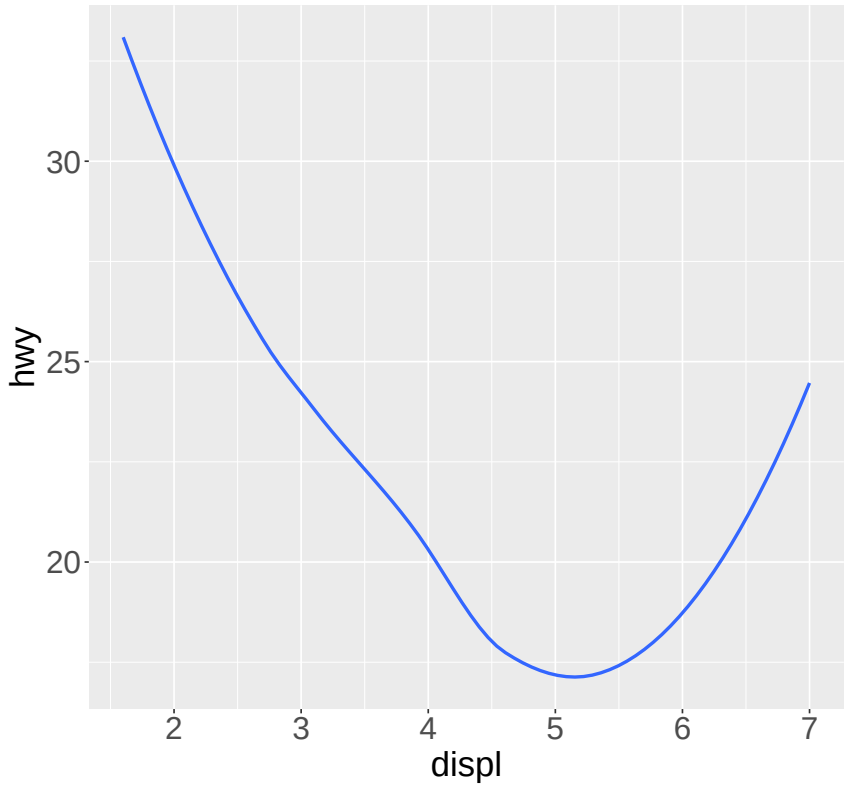
Changing default aesthetic properties of geoms

- By default, `geom_smooth()` displays confidence intervals around the smoothed line.
- Confidence intervals may be disabled by setting the `se` (standard error) aesthetic to `FALSE`:

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

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

Changing default aesthetic properties of geoms

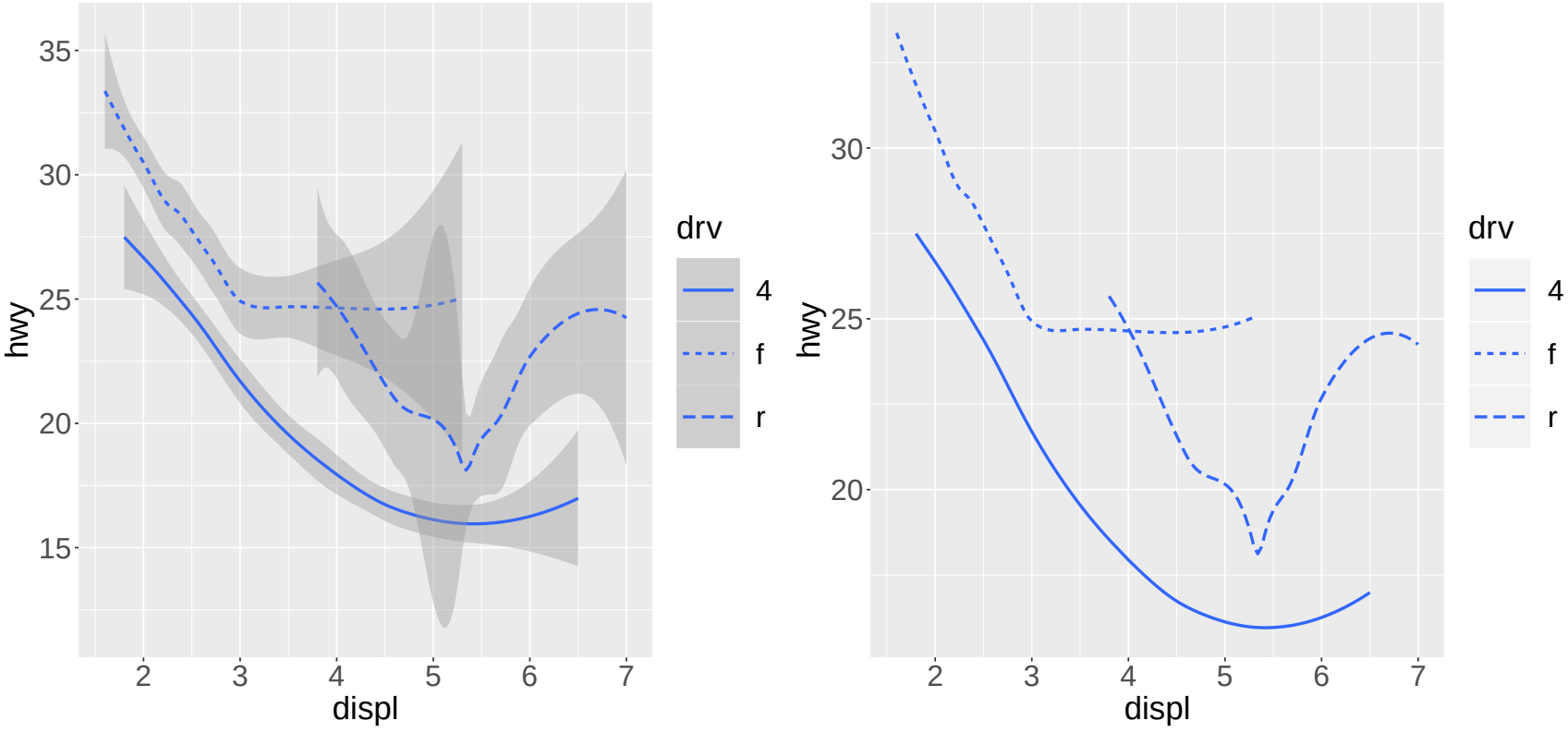




Changing default aesthetic properties of geoms

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

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

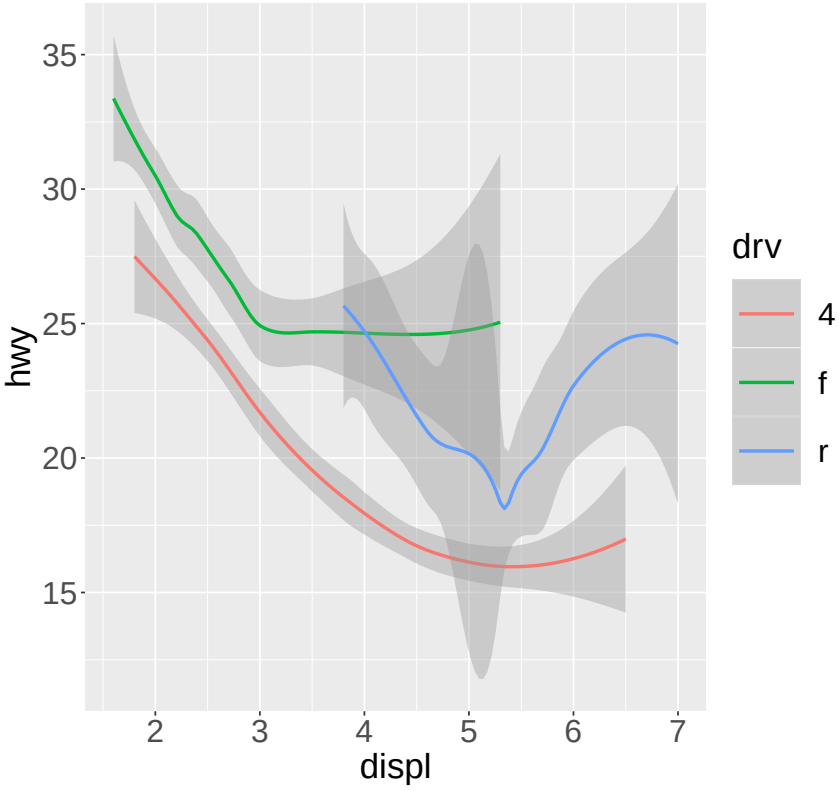
Changing default aesthetic properties of geoms

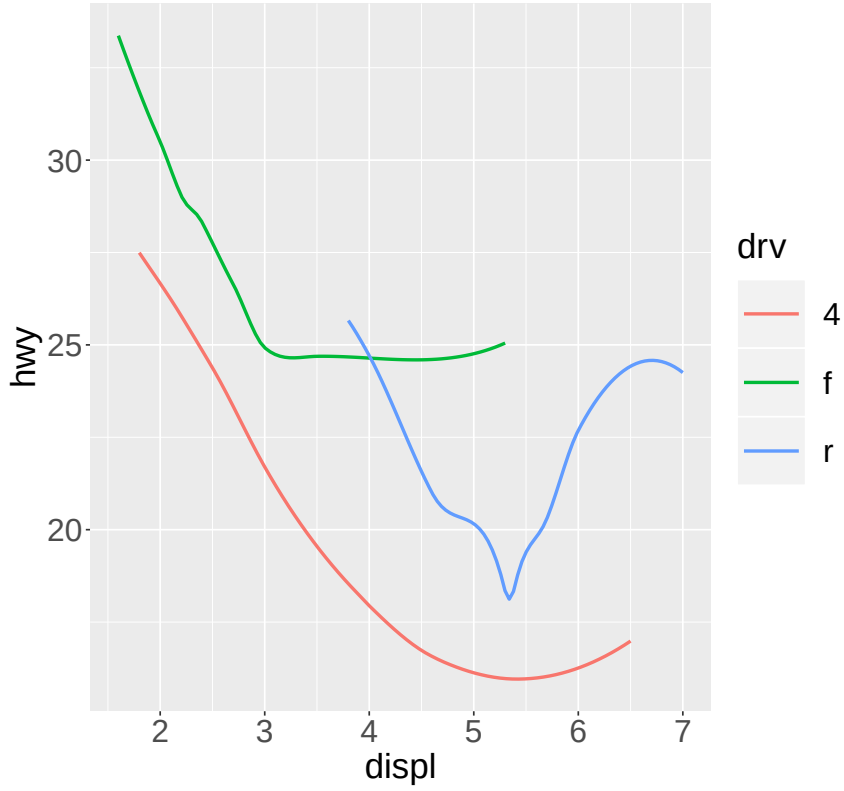


Changing default aesthetic properties of geoms

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

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

Changing default aesthetic properties of geoms





Changing default aesthetic properties of geoms

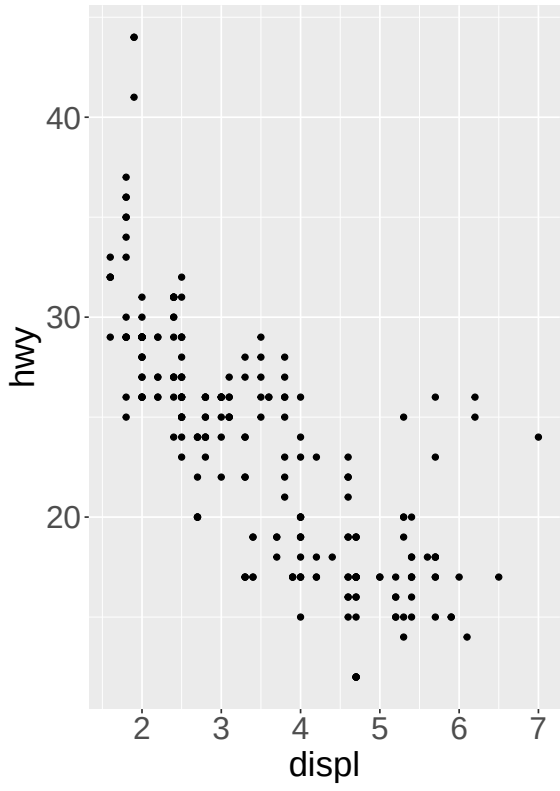
- The default smoothing method of `geom_smooth` can be changed.
- The default method (`"auto"`) depends on the size of the largest group.
- For less than 1000 observations the method is `"loess"` (a local polynomial regression).
- Admissible methods are: `"auto"`, `"lm"`, `"glm"`, `"gam"` and `"loess"`.

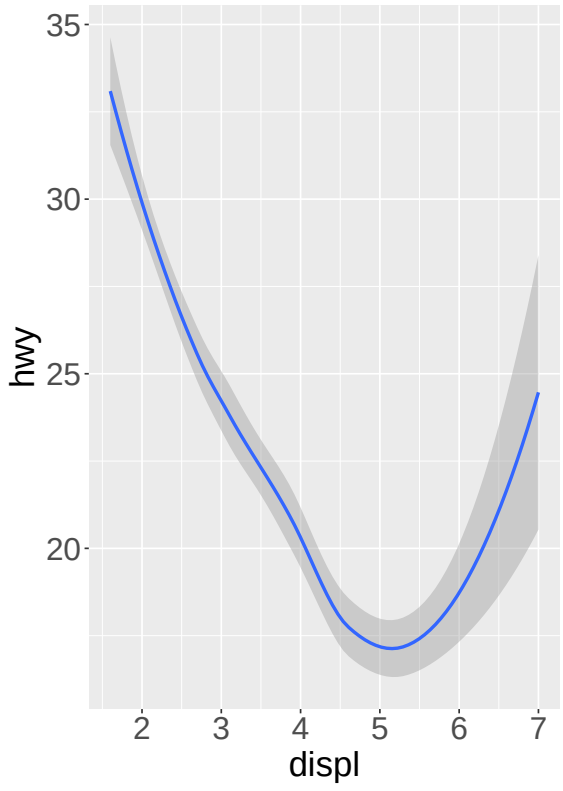
Changing default aesthetic properties of geoms

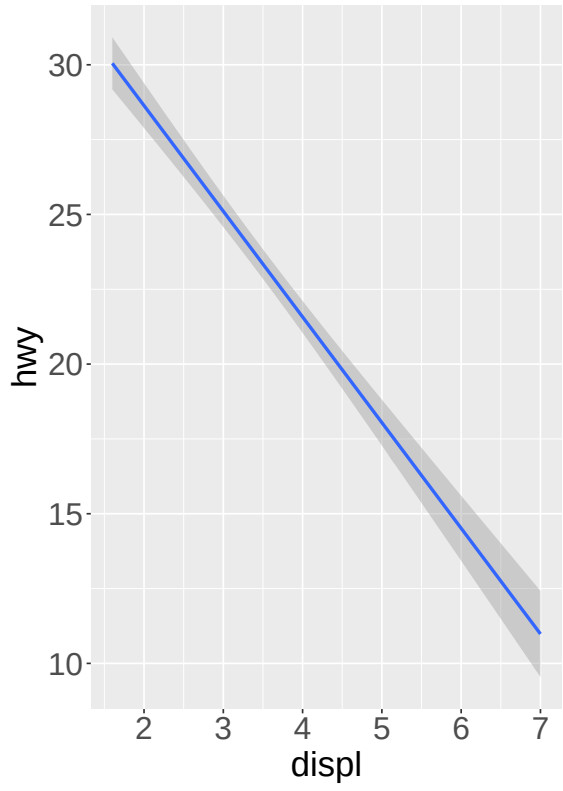
To add a linear regression line use `method = "lm"`:

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

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

ggplot(data = mpg) +
  geom_smooth(aes(x = displ, y = hwy), method = "lm")
```

Changing default aesthetic properties of geoms





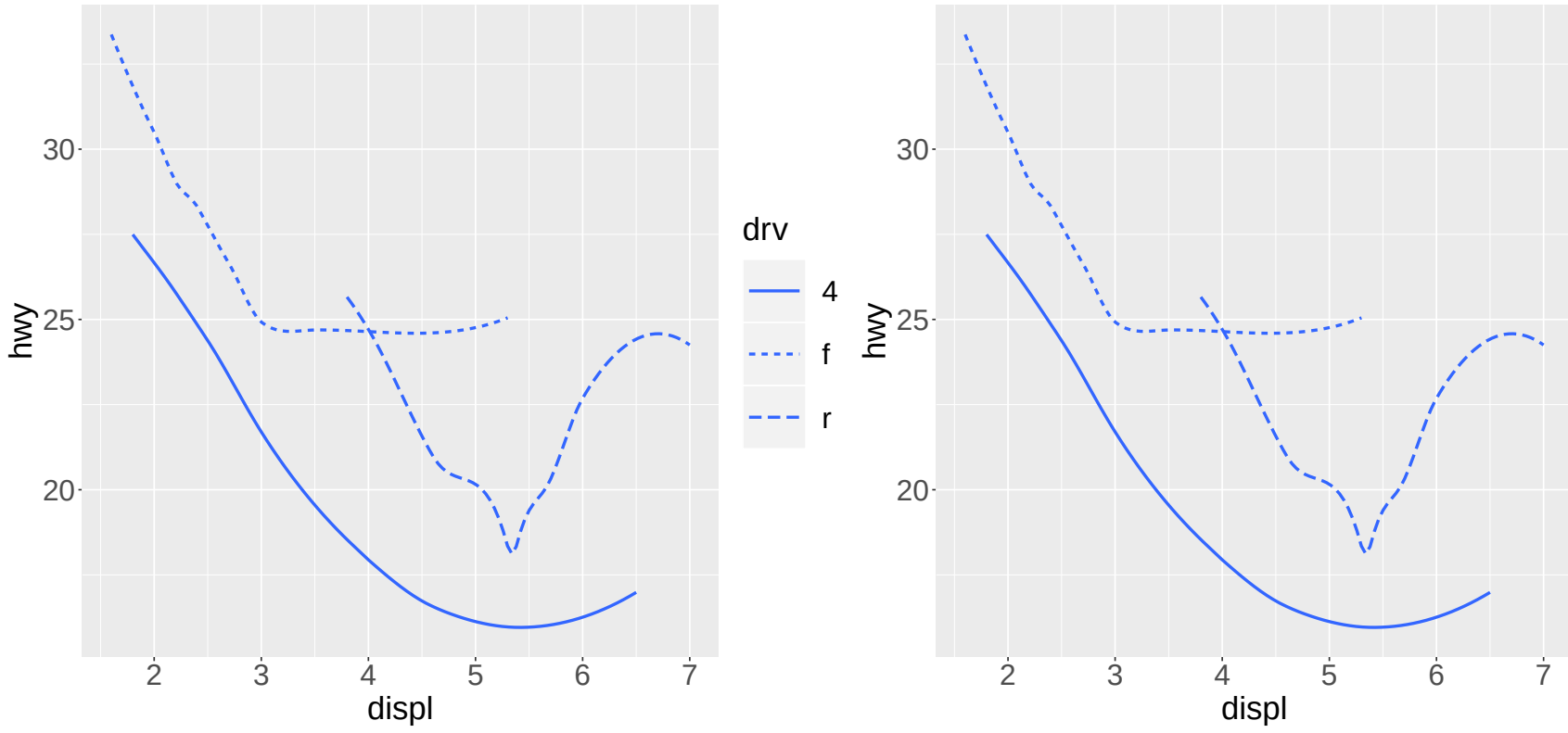


Changing default aesthetic properties of geoms

Legends can also be disabled:

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

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

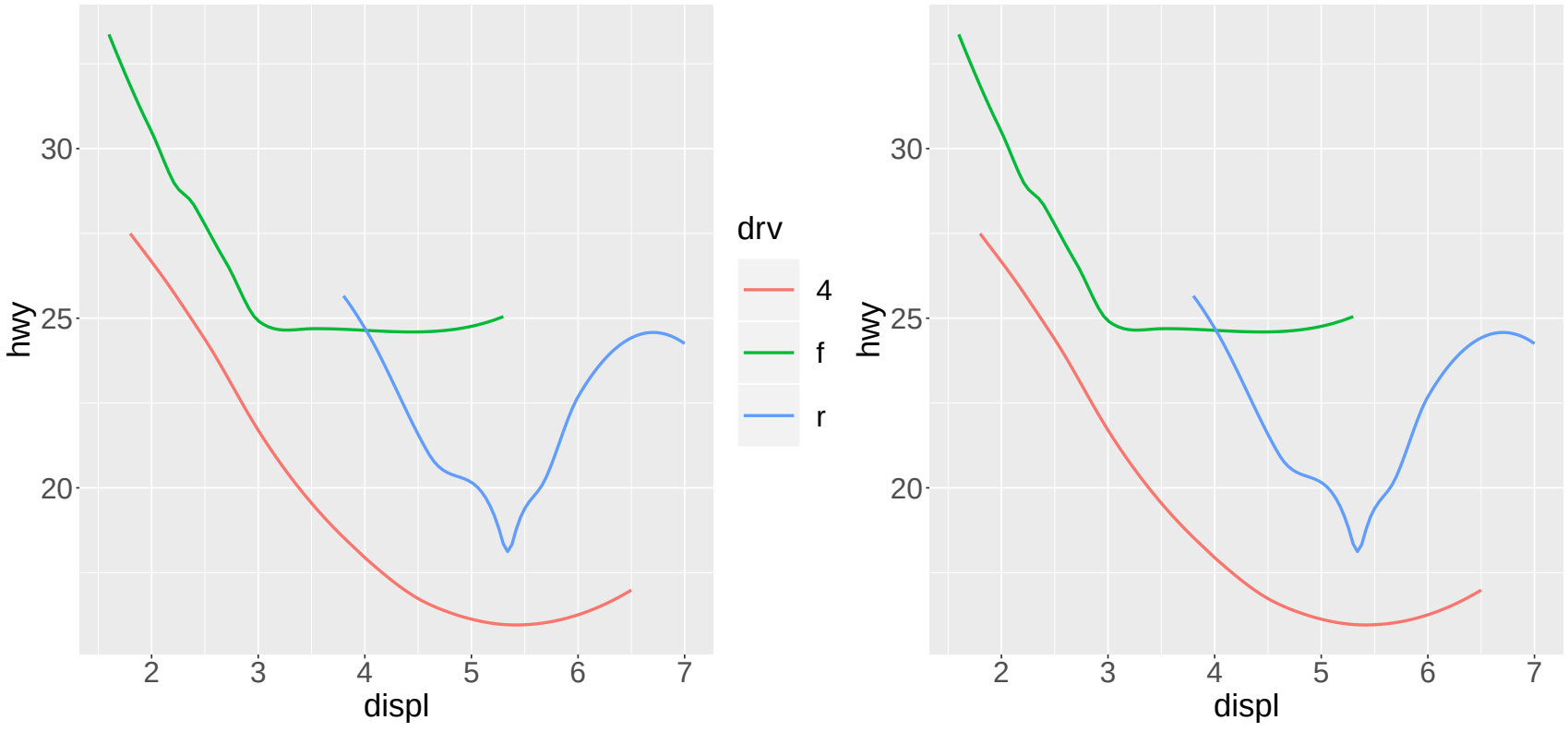
Changing default aesthetic properties of geoms



Changing default aesthetic properties of geoms

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

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

Changing default aesthetic properties of geoms



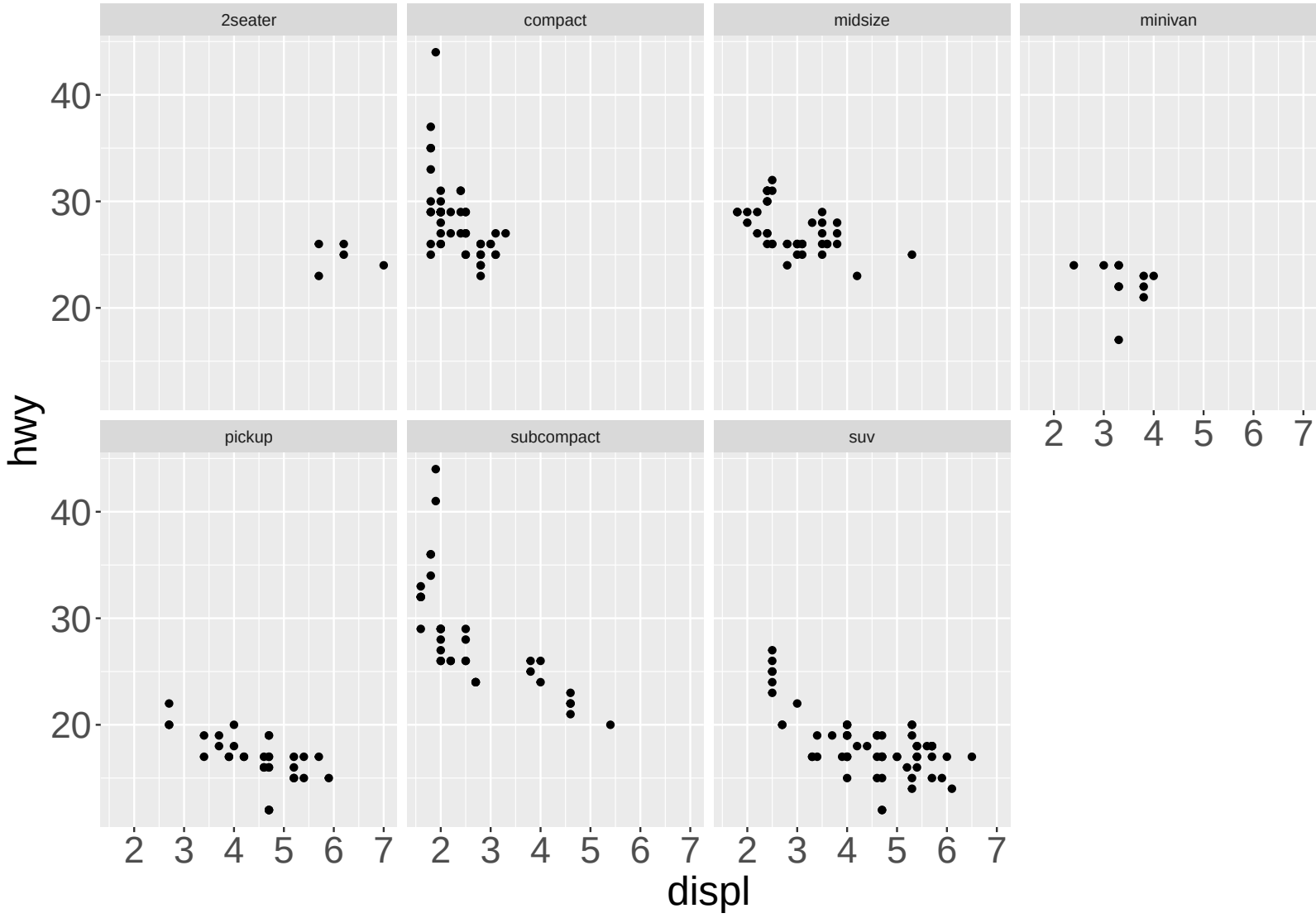
Facets

- One way to add additional variables to a plot is with **aesthetics**.
- Another way, particularly useful for categorical variables, is to split the plot into **facets**, subplots that each display one subset of the data.
- To facet a plot by a single variable, use `facet_wrap()`.
- The first argument of `facet_wrap()` should be `"~"` followed the name of a variable.

Facets

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

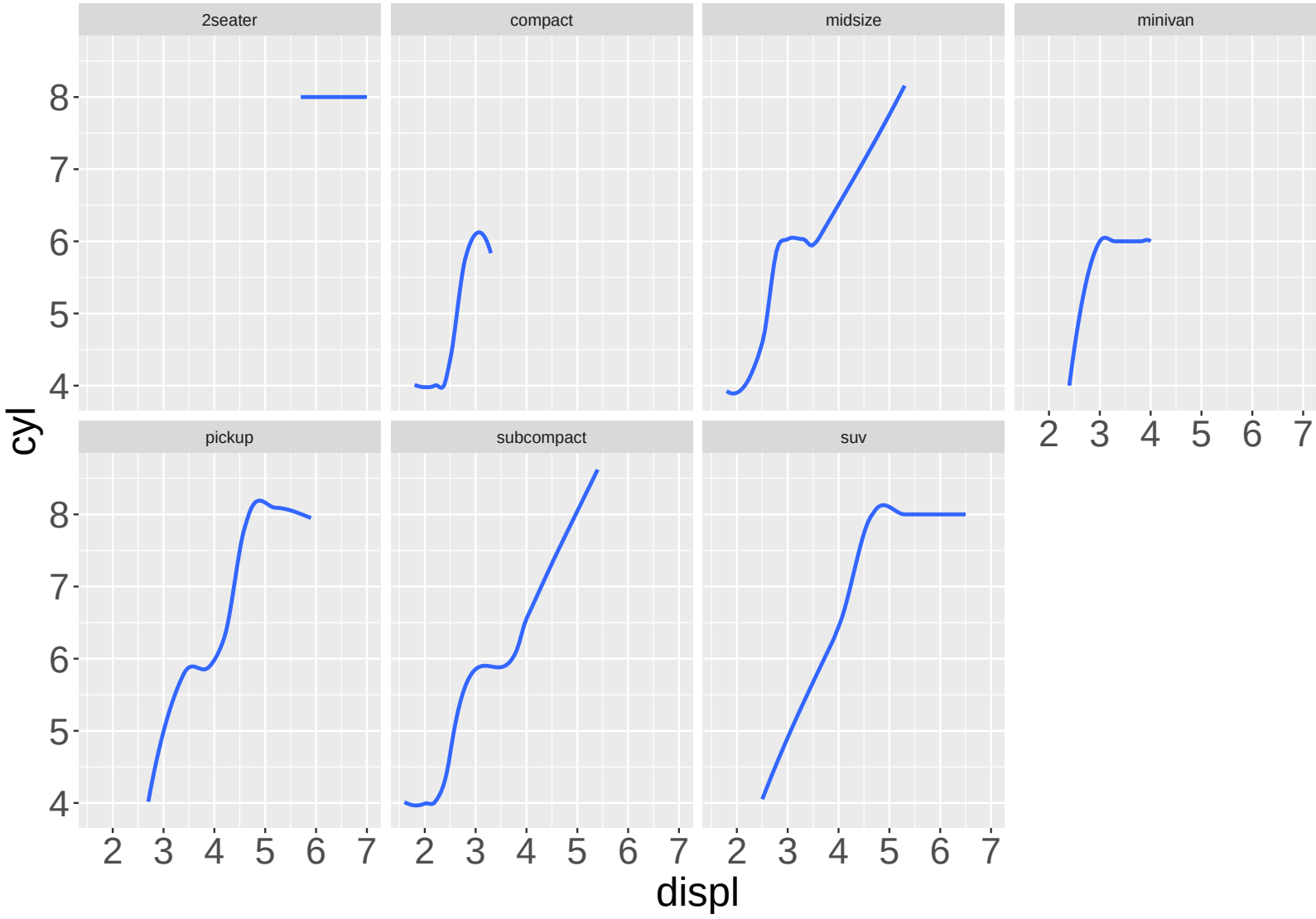
Facets


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

Facets

```
ggplot(data = mpg) +
  geom_smooth(aes(x = displ, y = cyl), se = FALSE) +
  facet_wrap(~class, nrow = 2)
```

Facets

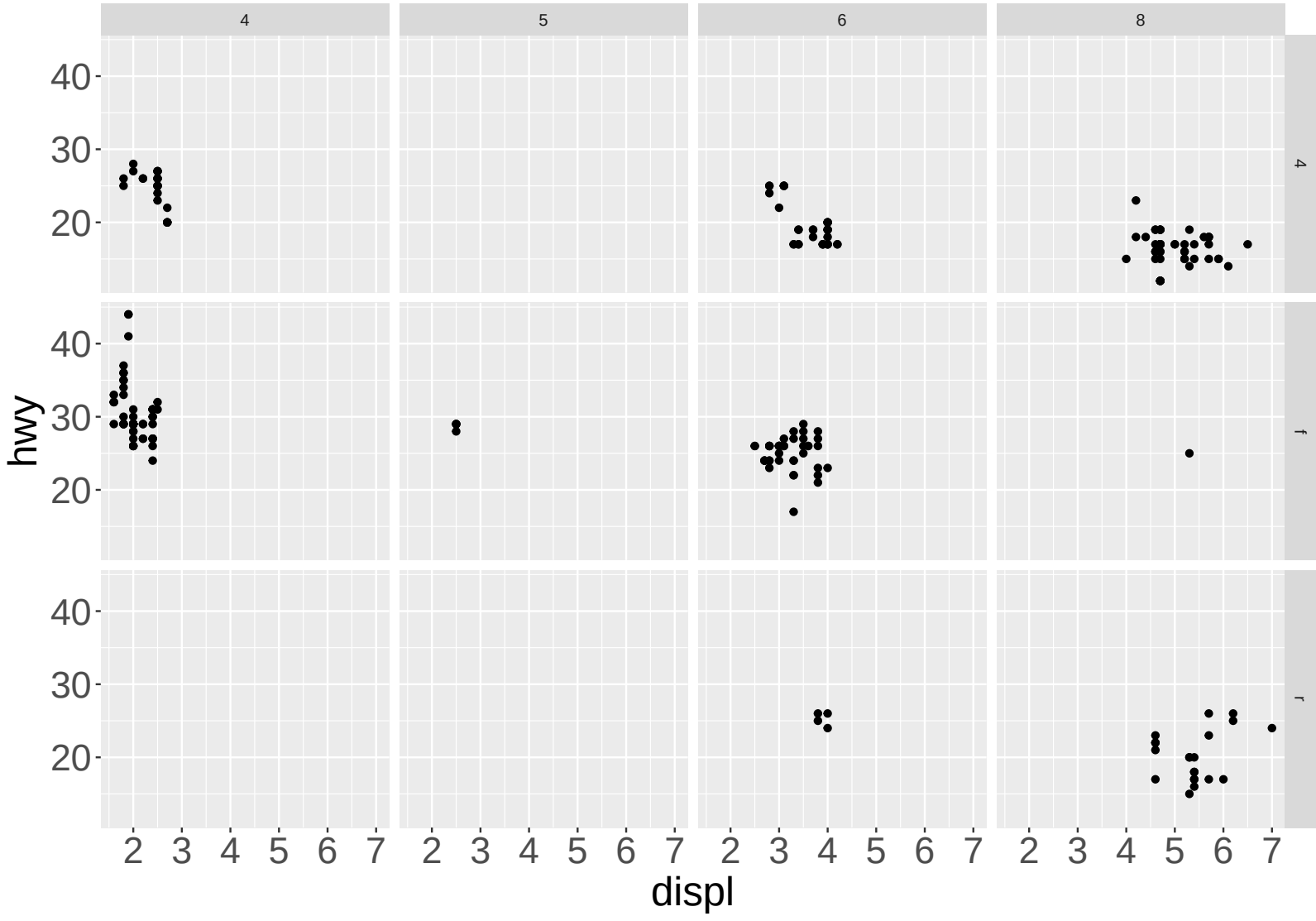

2seater
compact
midsize
minivan
pickup
subcompact
suv
cyl
displ
8
7
6
5
4
2
3
4
5
6
7

Facets

- To facet a plot on the combination of two variables, use `facet_grid()`.
- The first argument of `facet_grid()` should contain the two variable names separated by `"~"`.

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

Facets

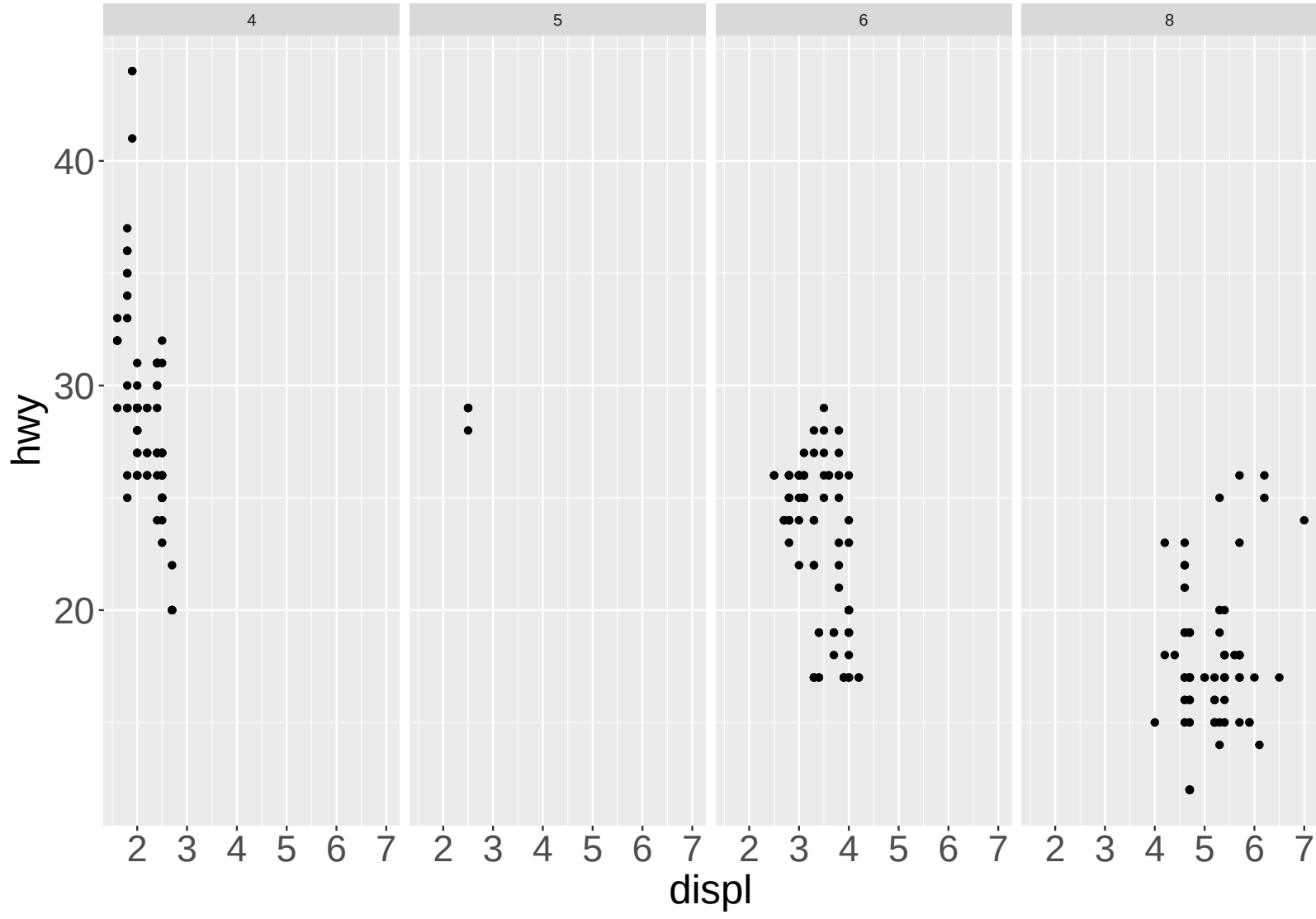


Facets

If we prefer to not facet in the rows (or columns), we can use a "." instead of a variable name:

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

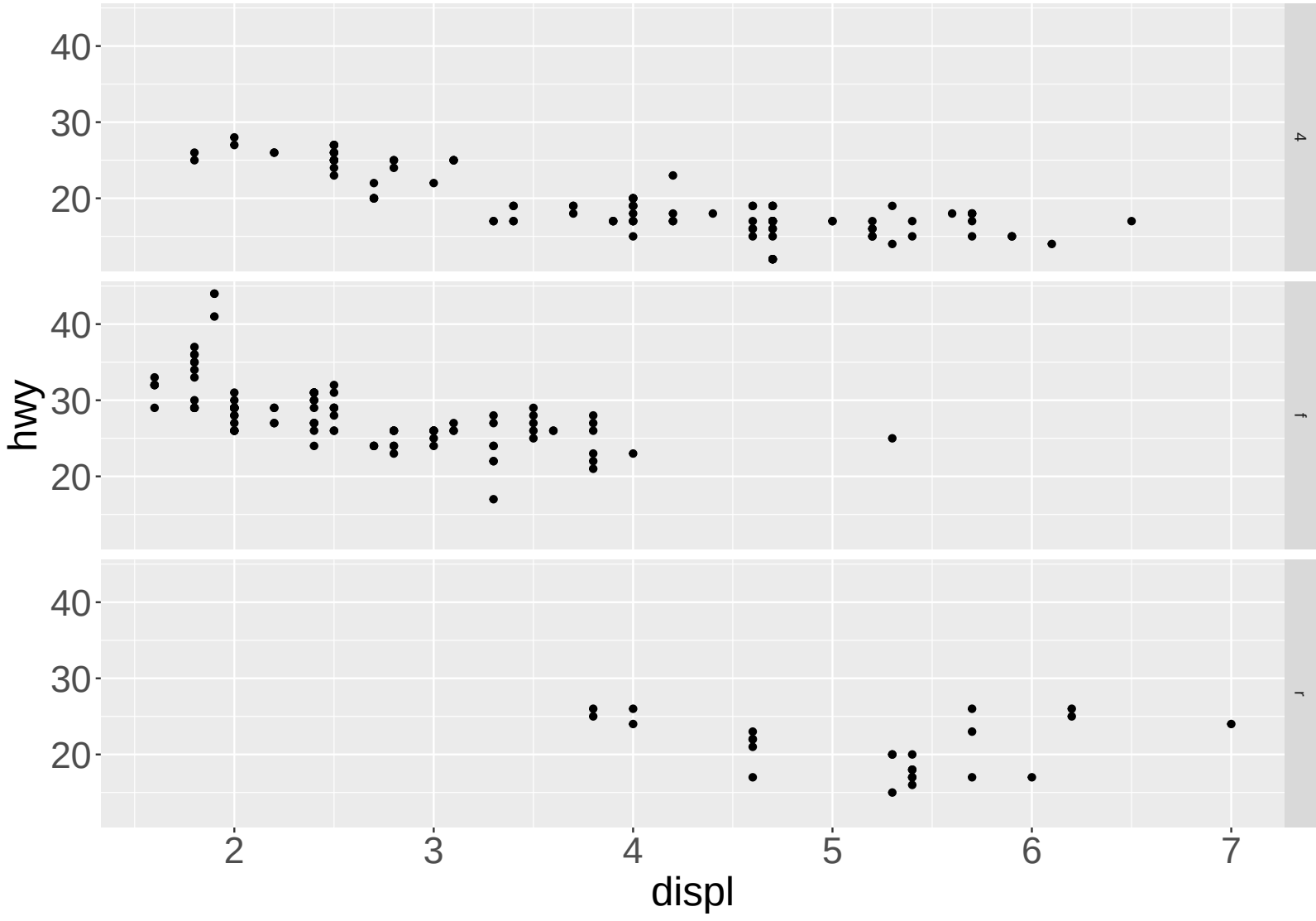
Facets



Facets

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

Facets

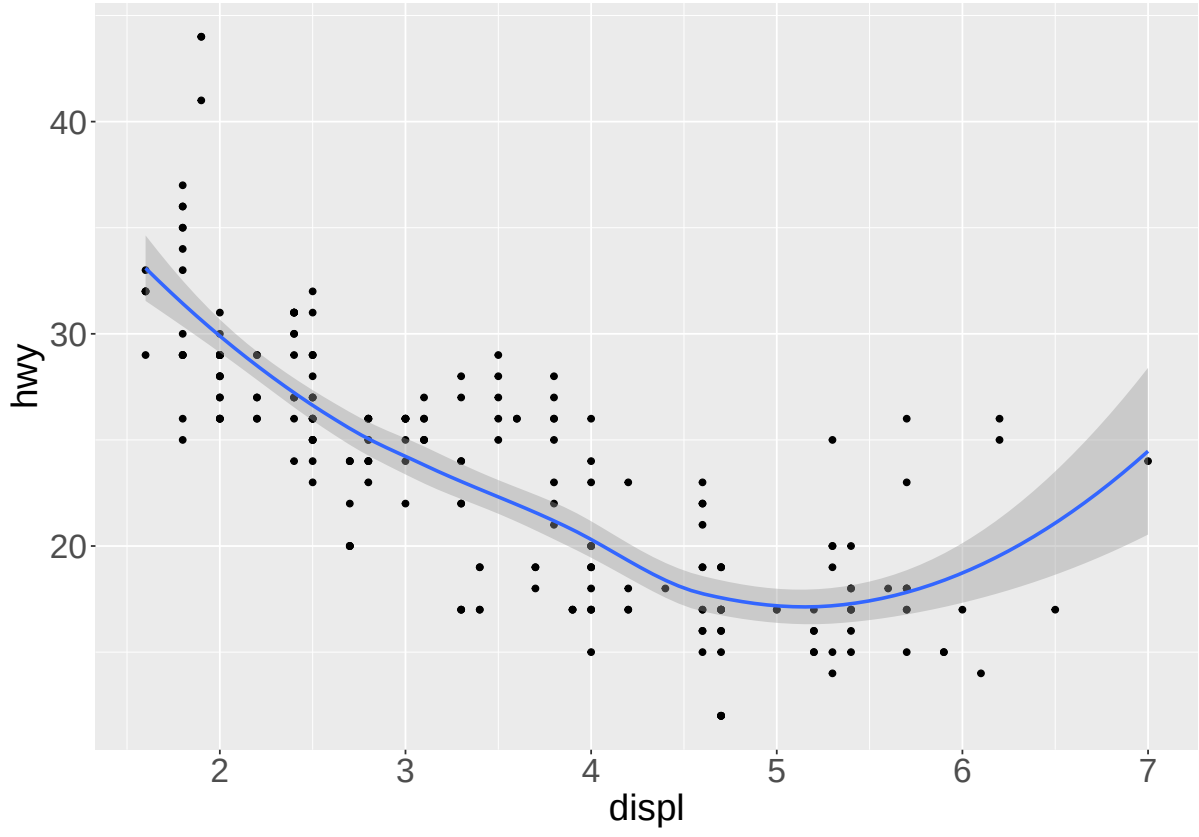


Multiple geoms

- We can add multiple geoms to the same graph.
- For example, we can overlap points and lines.
- This is archived by adding multiple geom functions to `ggplot()`.

Multiple geoms

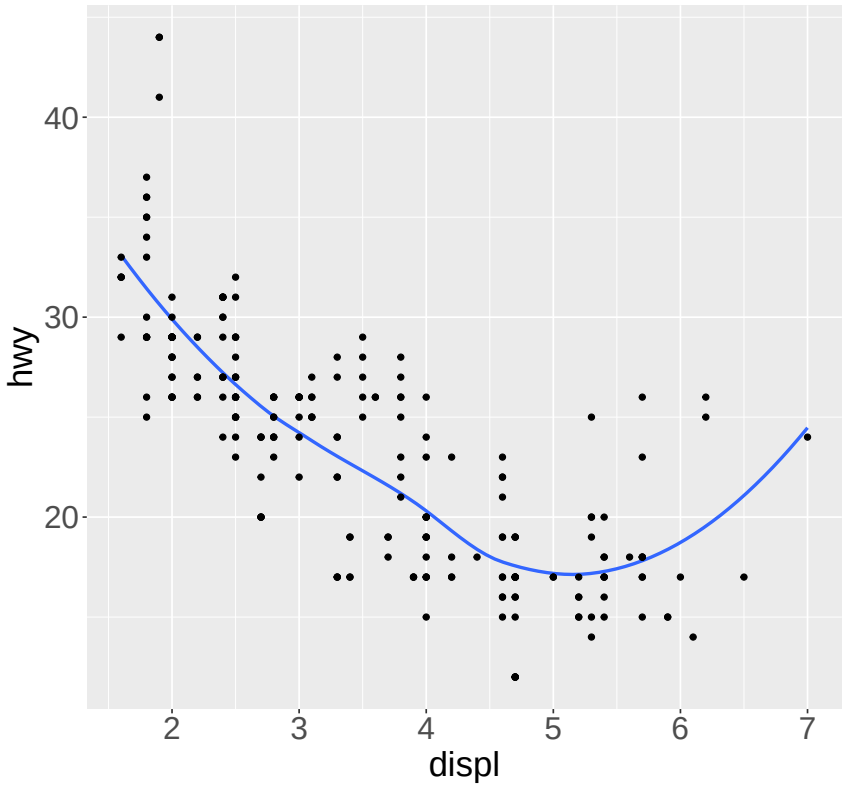
```
ggplot(data = mpg) +
  geom_point(aes(x = displ, y = hwy)) +
  geom_smooth(aes(x = displ, y = hwy))
```

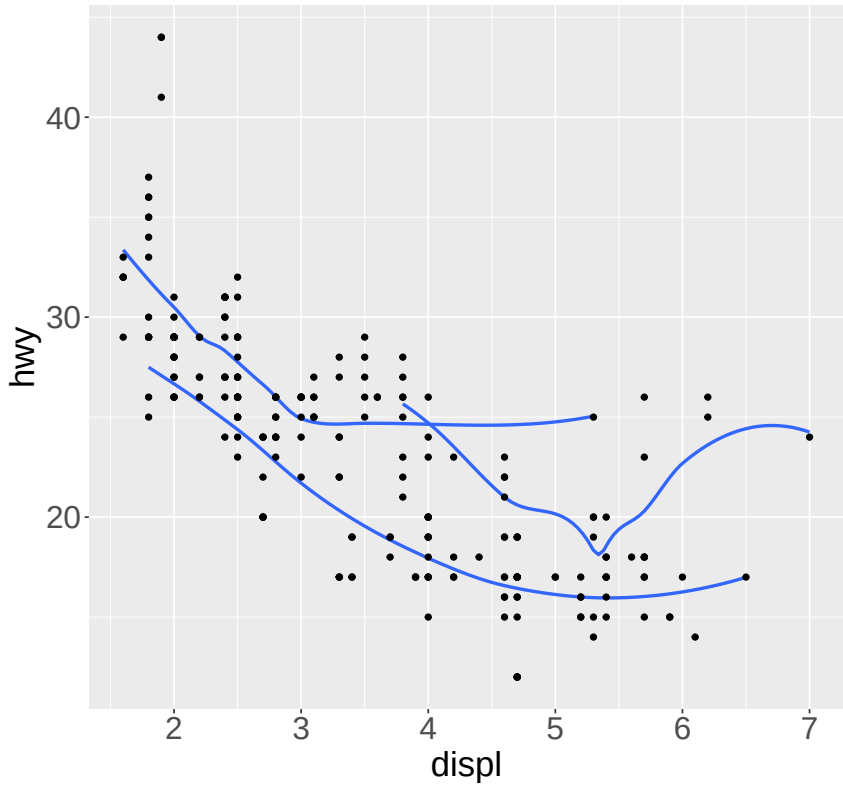


Multiple geoms

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

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

Multiple geoms





Multiple geoms

- This, however, induces duplication in our code.
- We built the same aesthetic mapping twice in each graph.
- We 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.

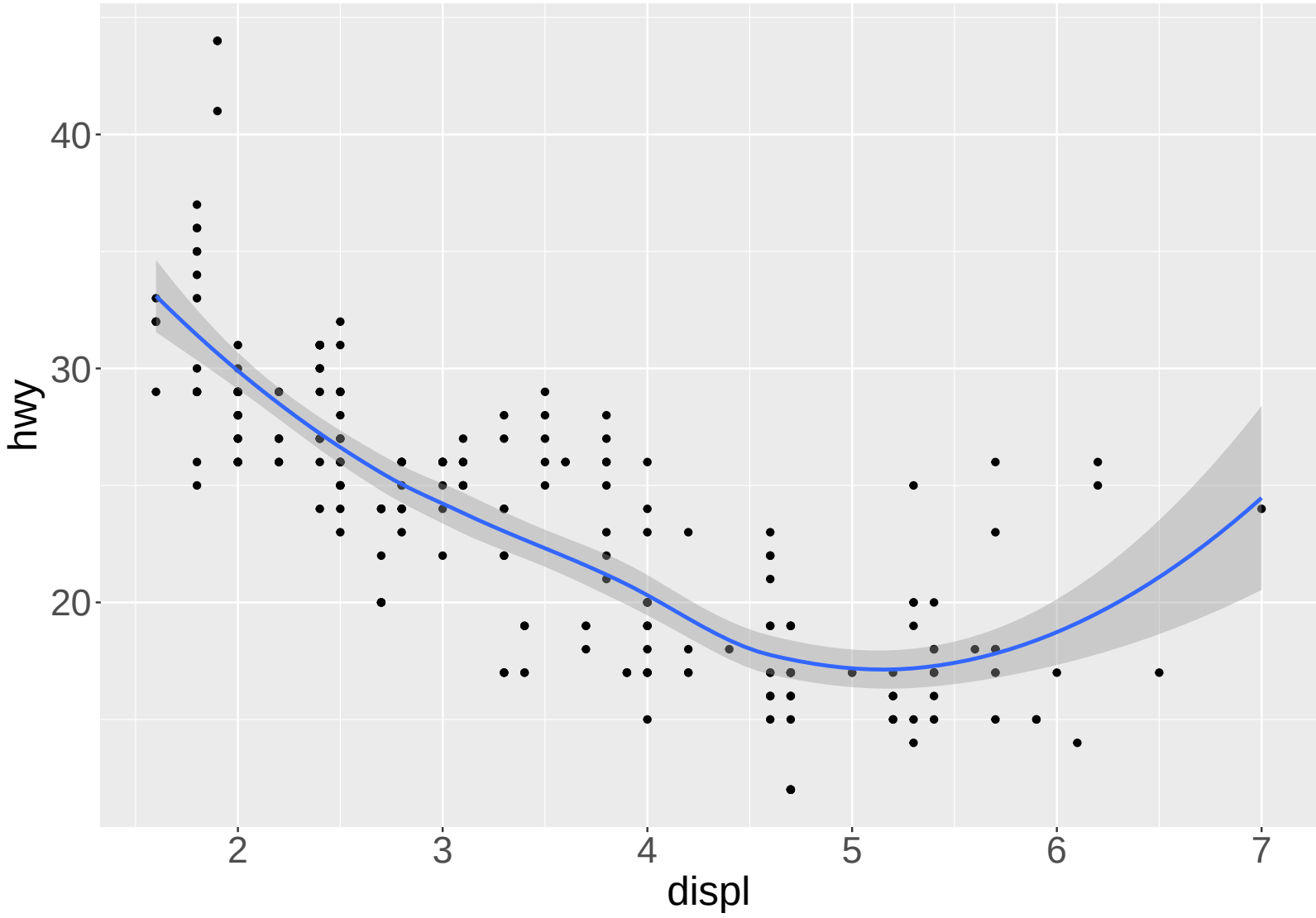
Multiple geoms

These two chunks of code result in the same plot:

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

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

Multiple geoms



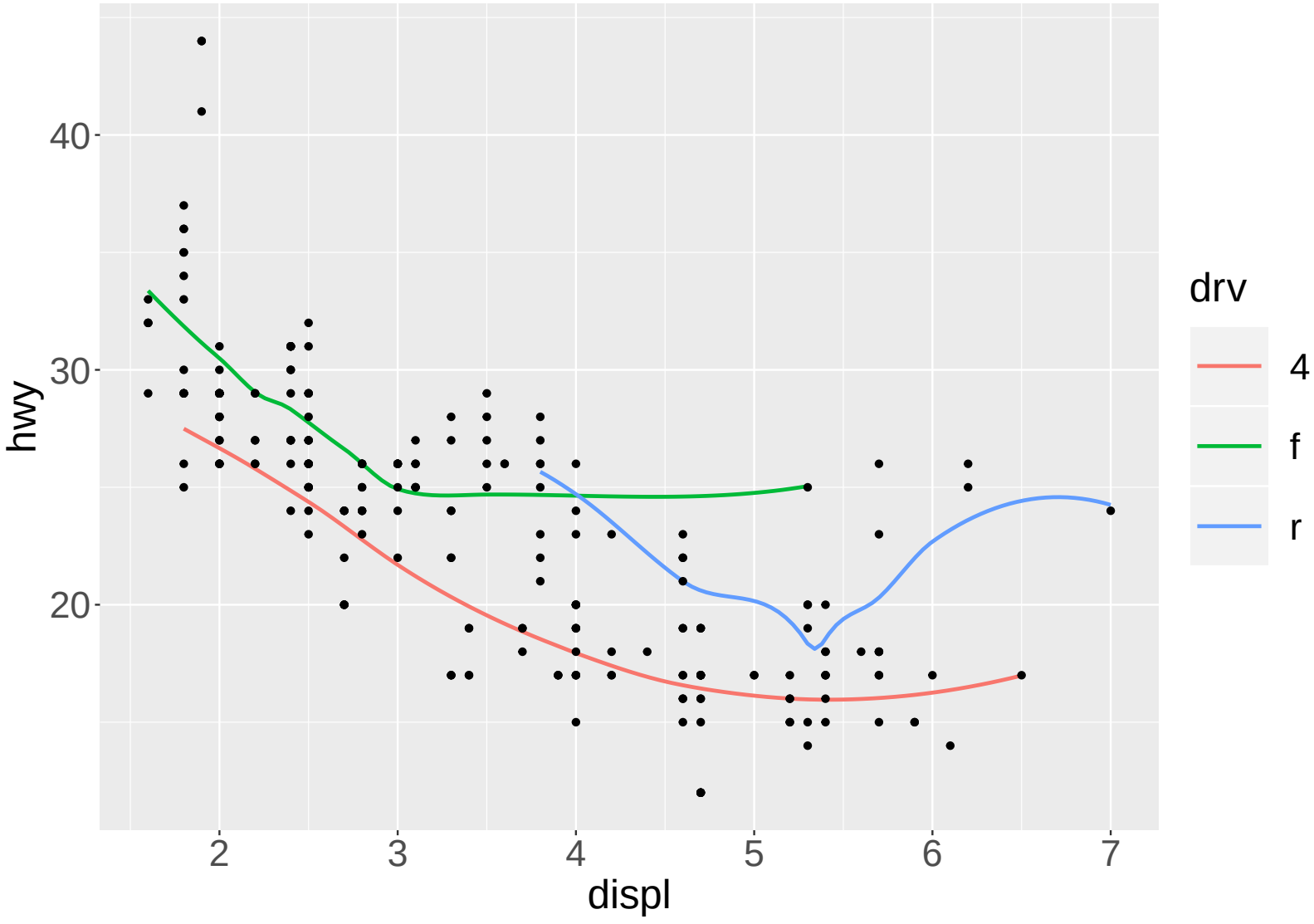
Multiple geoms

- If we place mappings in a geom function, ggplot2 will treat them as local mappings for that 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.

Multiple geoms

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

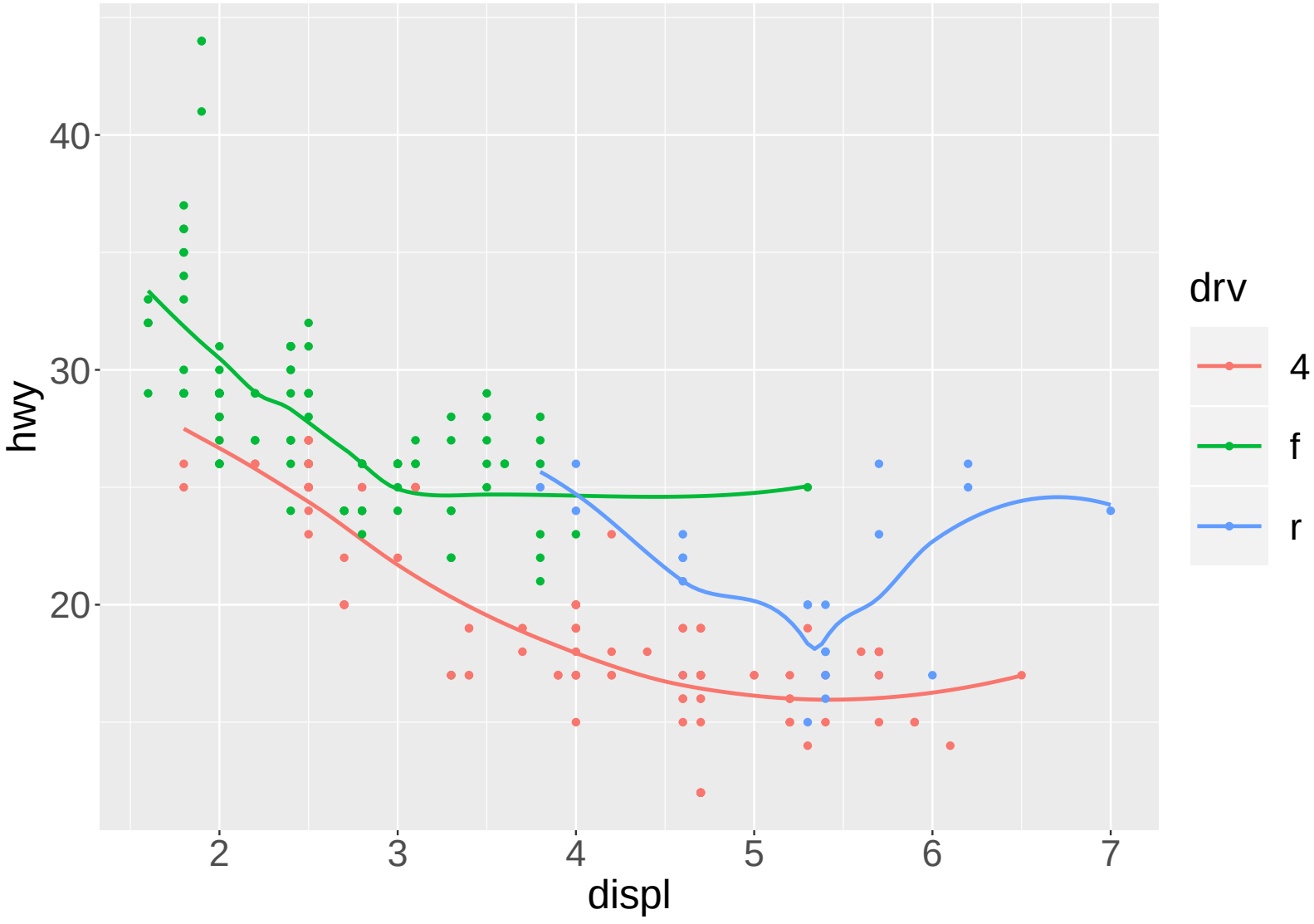
Multiple geoms



Multiple geoms

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

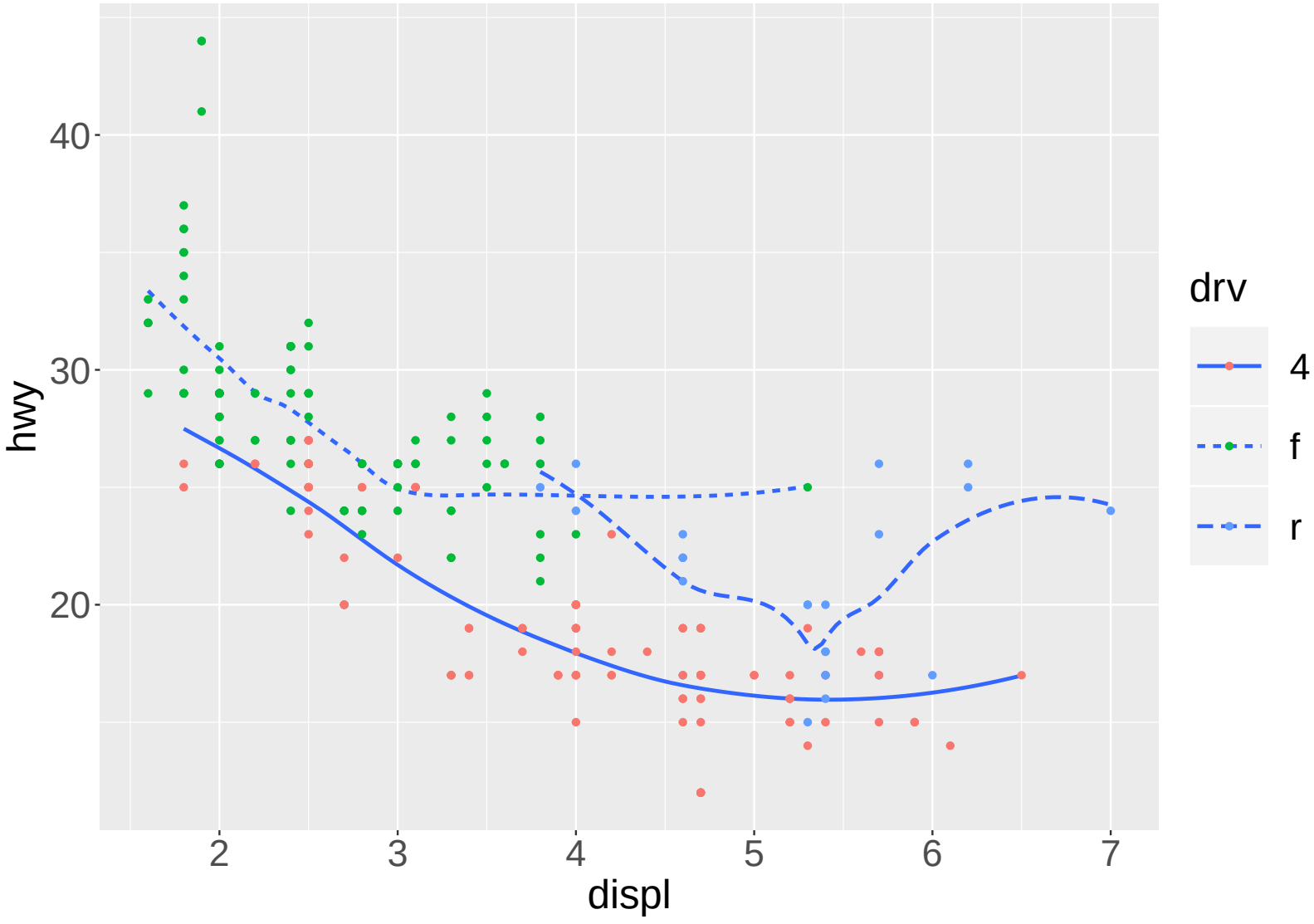
Multiple geoms



Multiple geoms

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

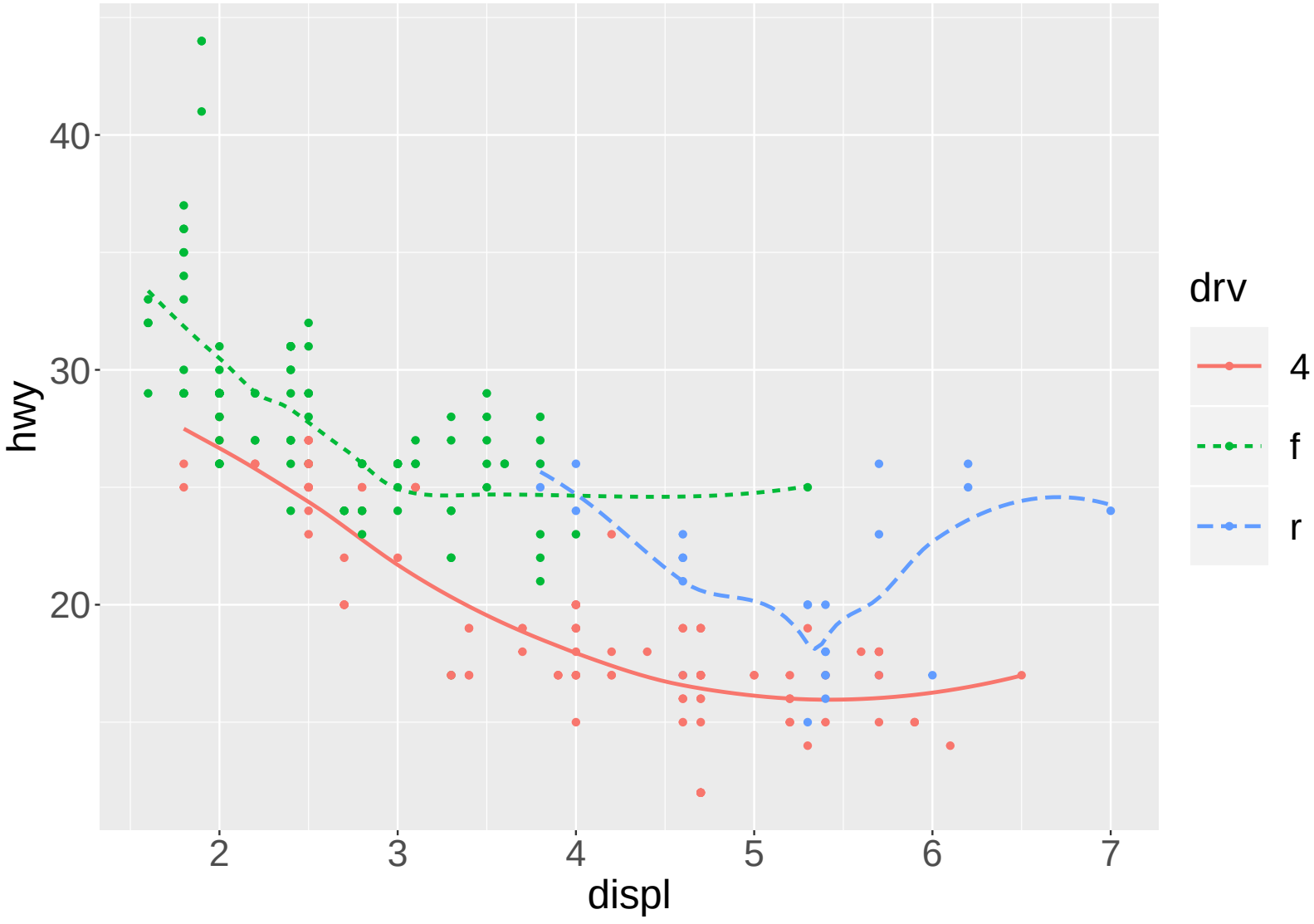
Multiple geoms



Multiple geoms

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

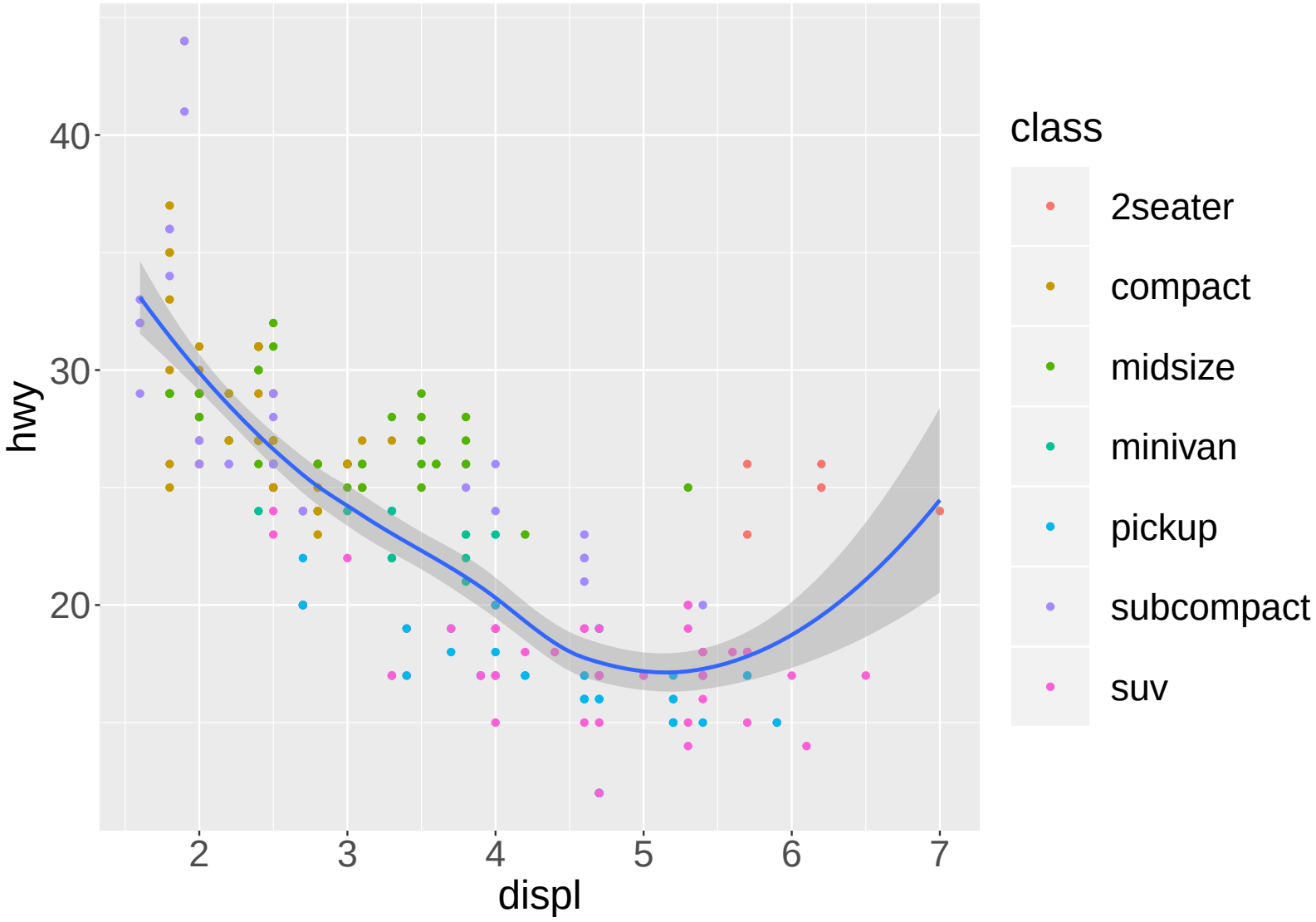
Multiple geoms



Multiple geoms

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

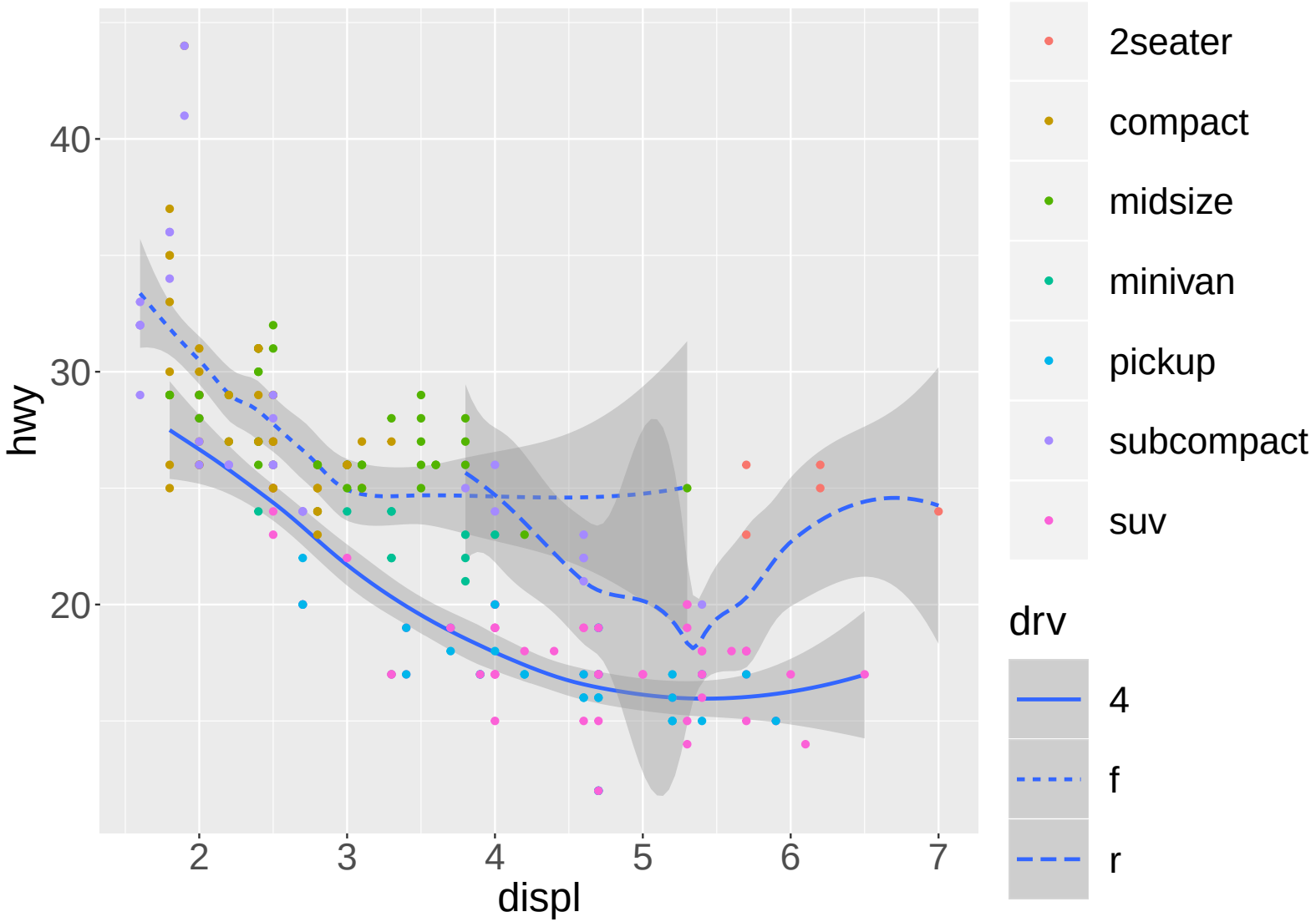
Multiple geoms



Multiple geoms

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

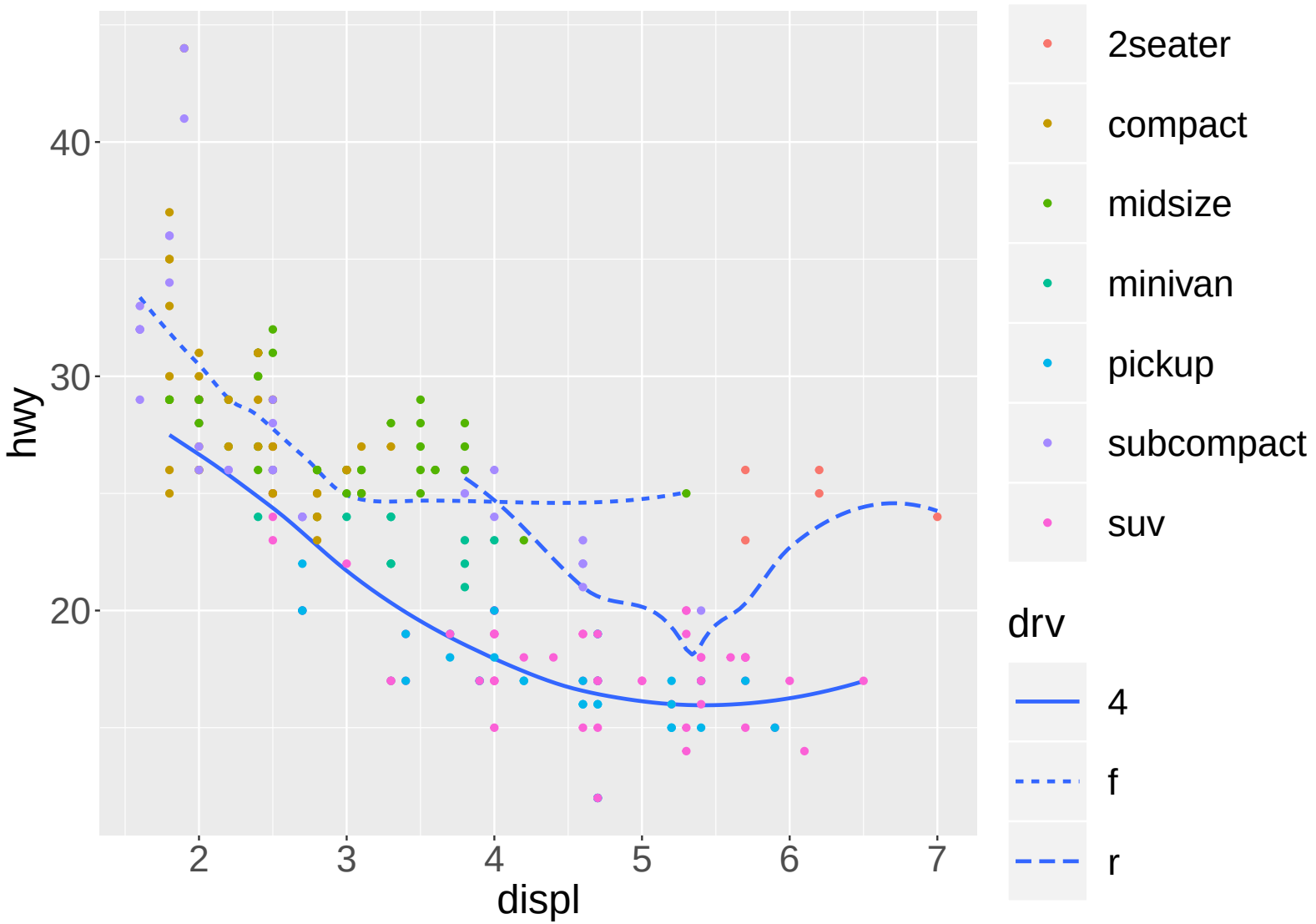
Multiple geoms



Multiple geoms

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

Multiple geoms

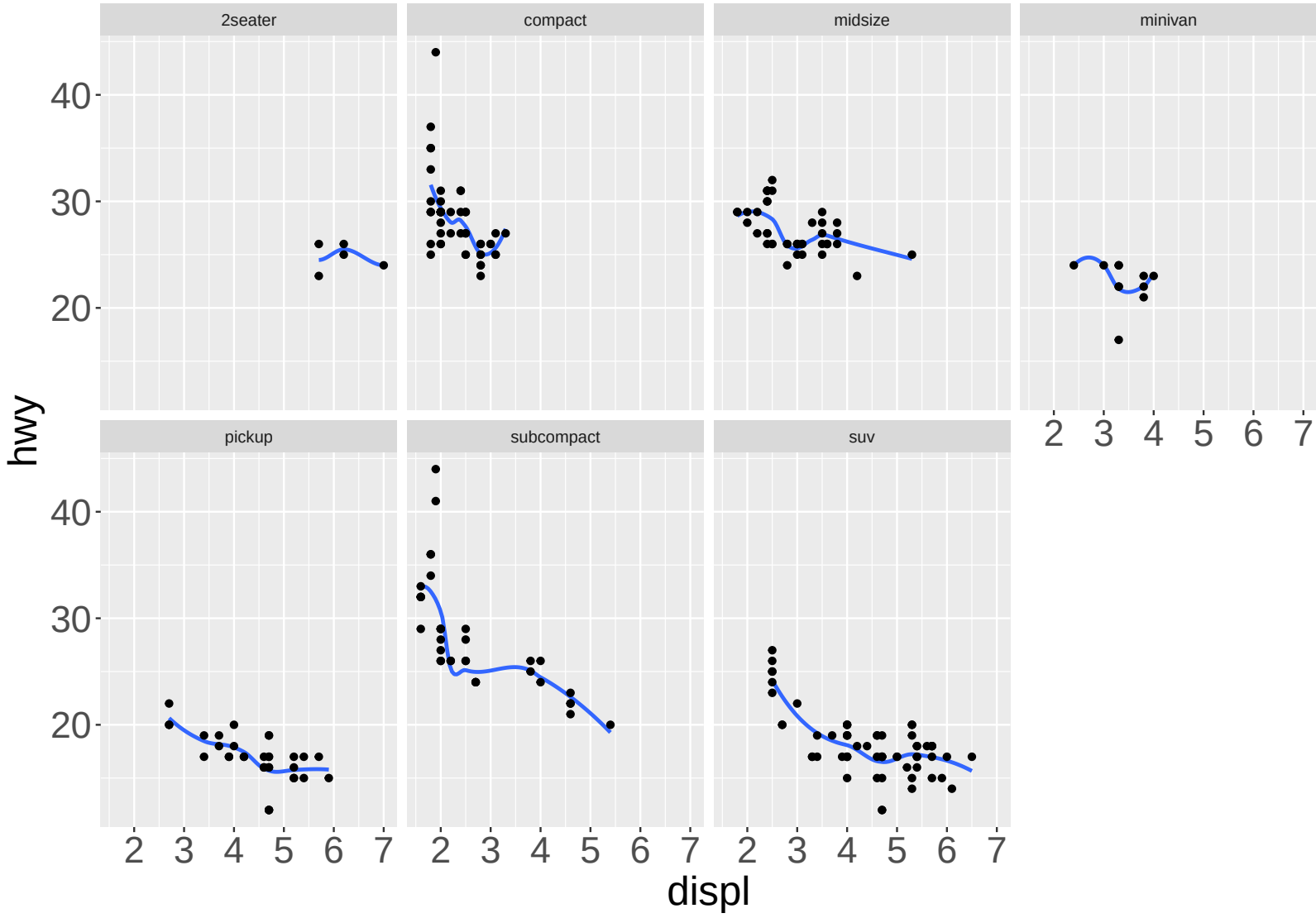


Multiple geoms

We can also overlap geoms in grids:

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

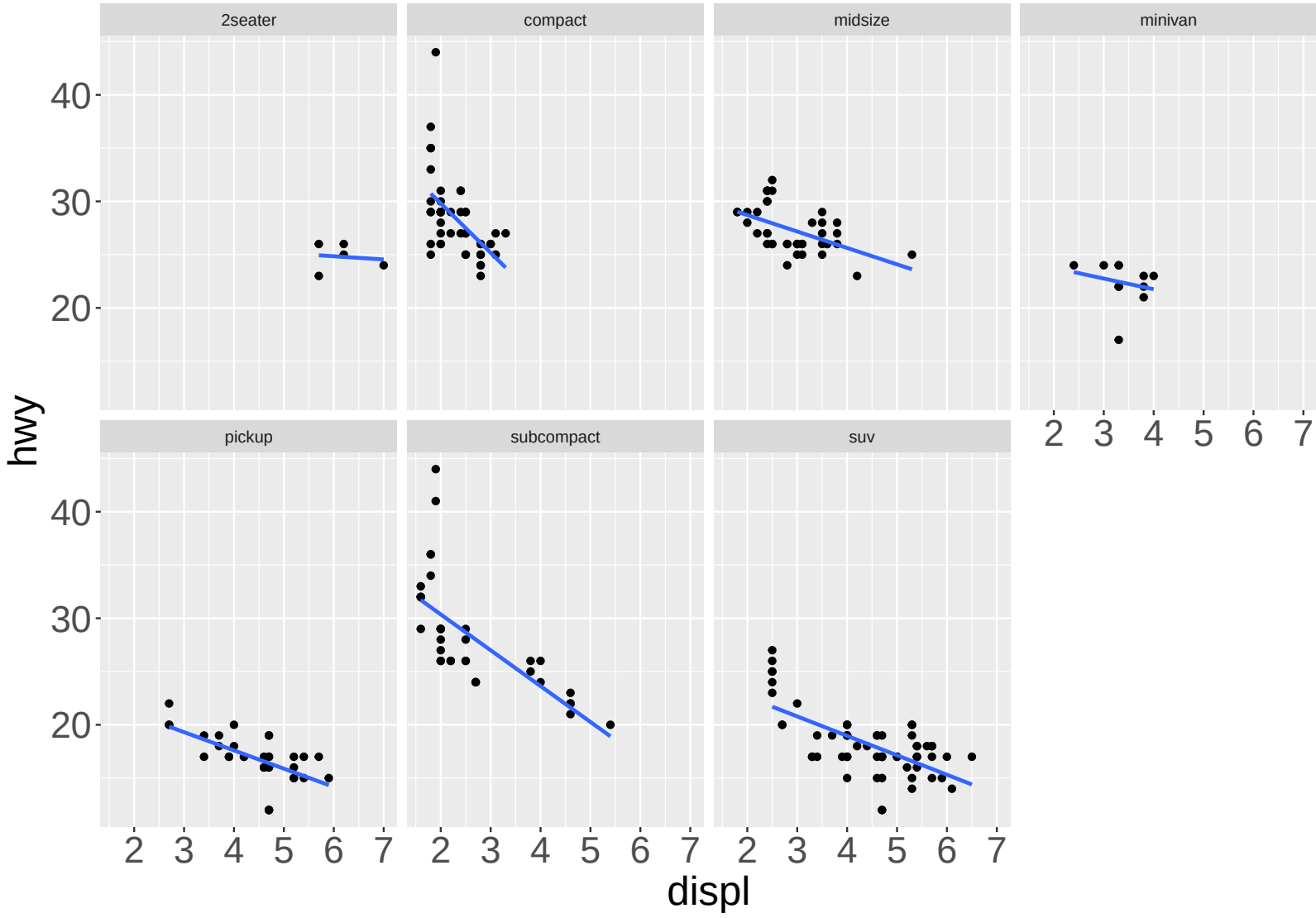
Multiple geoms



Multiple geoms

```
ggplot(data = mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~class, nrow = 2)
```

Multiple geoms



Multiple geoms

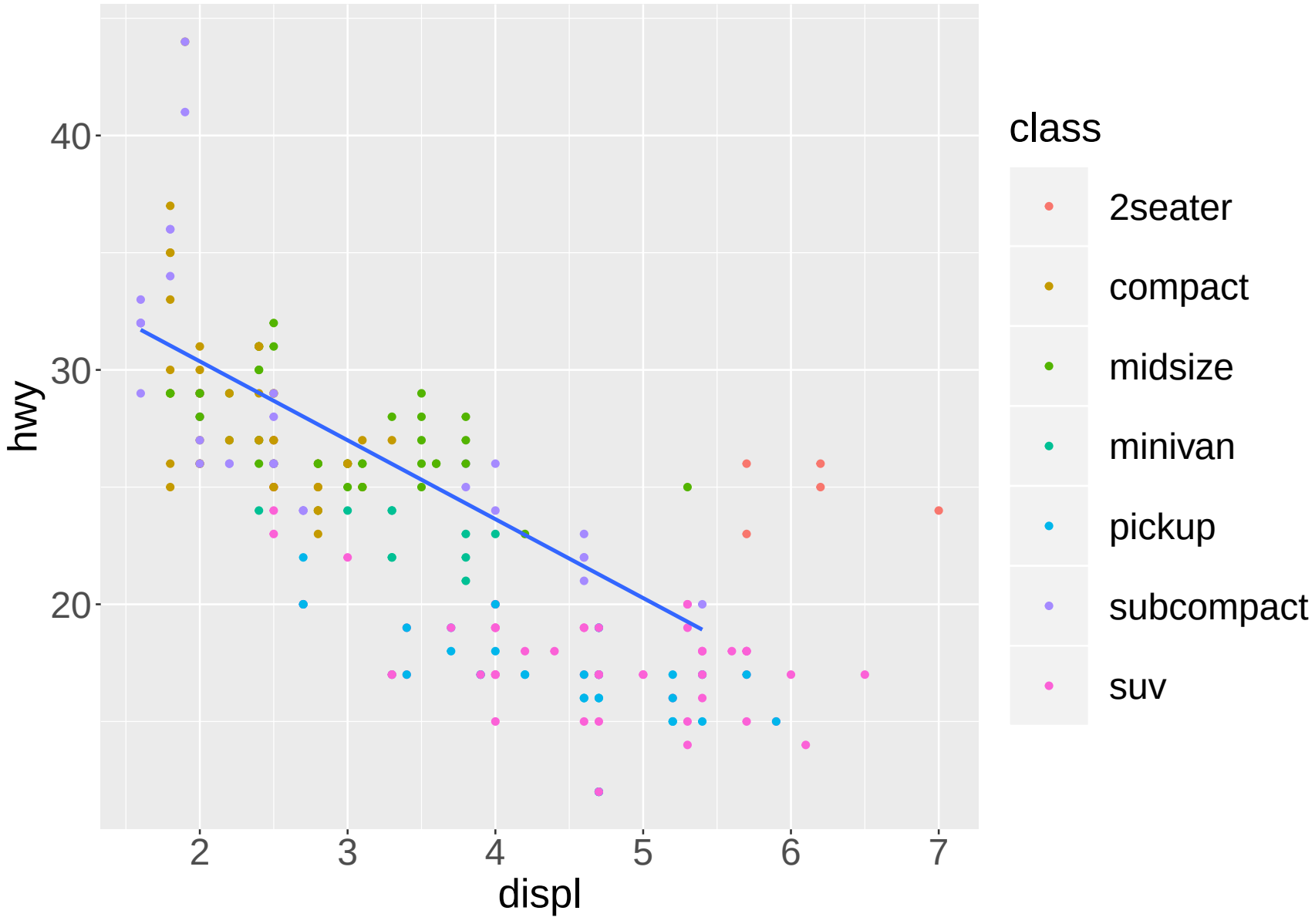
- We can use the same logic to specify a different dataset for each layer.

Multiple geoms

```
subcompact <- subset(mpg, class == "subcompact")

ggplot(data = mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(data = subcompact, se = FALSE,
              method = "lm")
```

- The straight 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.

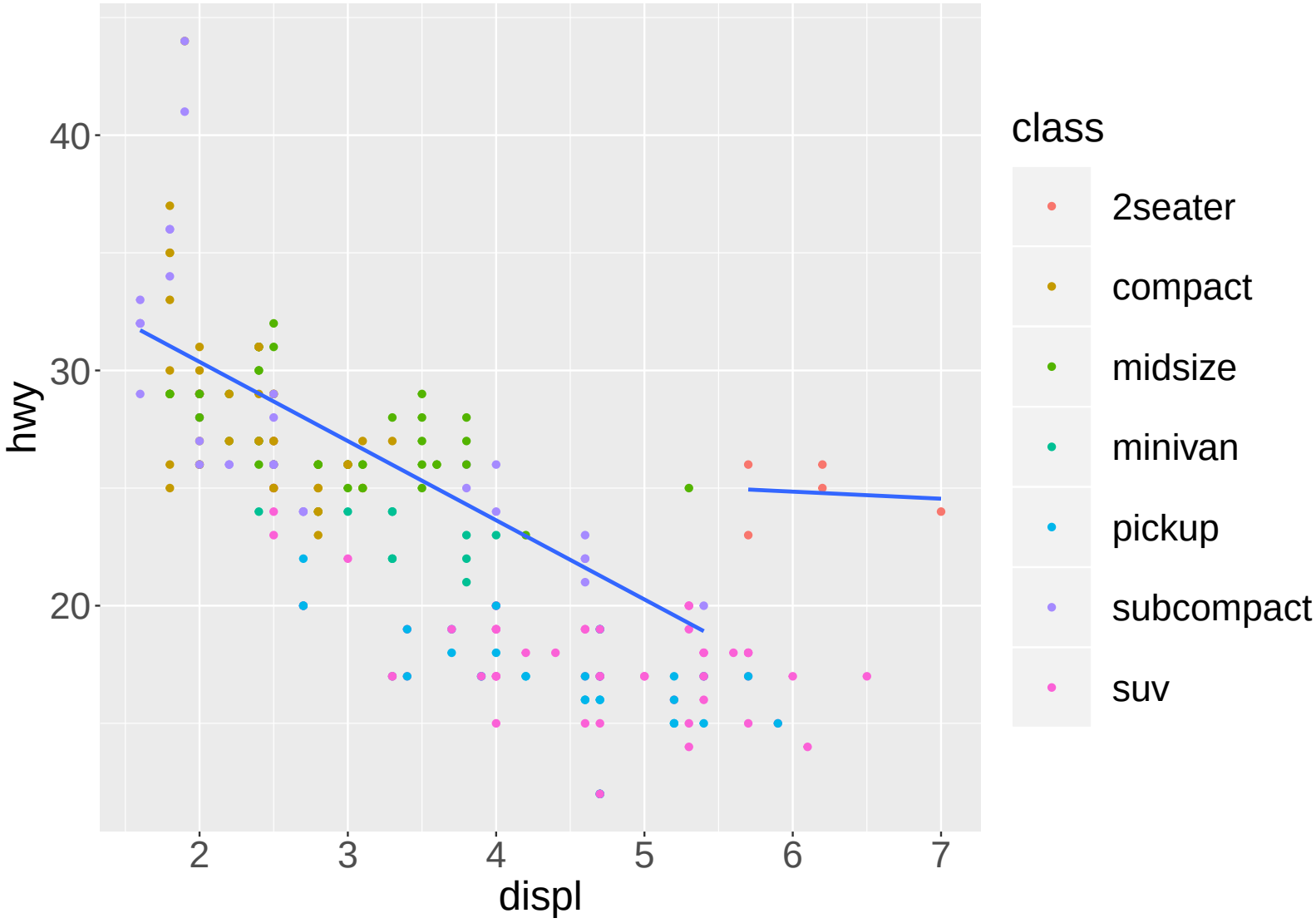
Multiple geoms



Multiple geoms

```
subcompact <- subset(mpg, class == "subcompact")
two_seater <- subset(mpg, class == "2seater")

ggplot(data = mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(data = subcompact, se = FALSE,
              method = "lm") +
  geom_smooth(data = two_seater, se = FALSE,
              method = "lm")
```

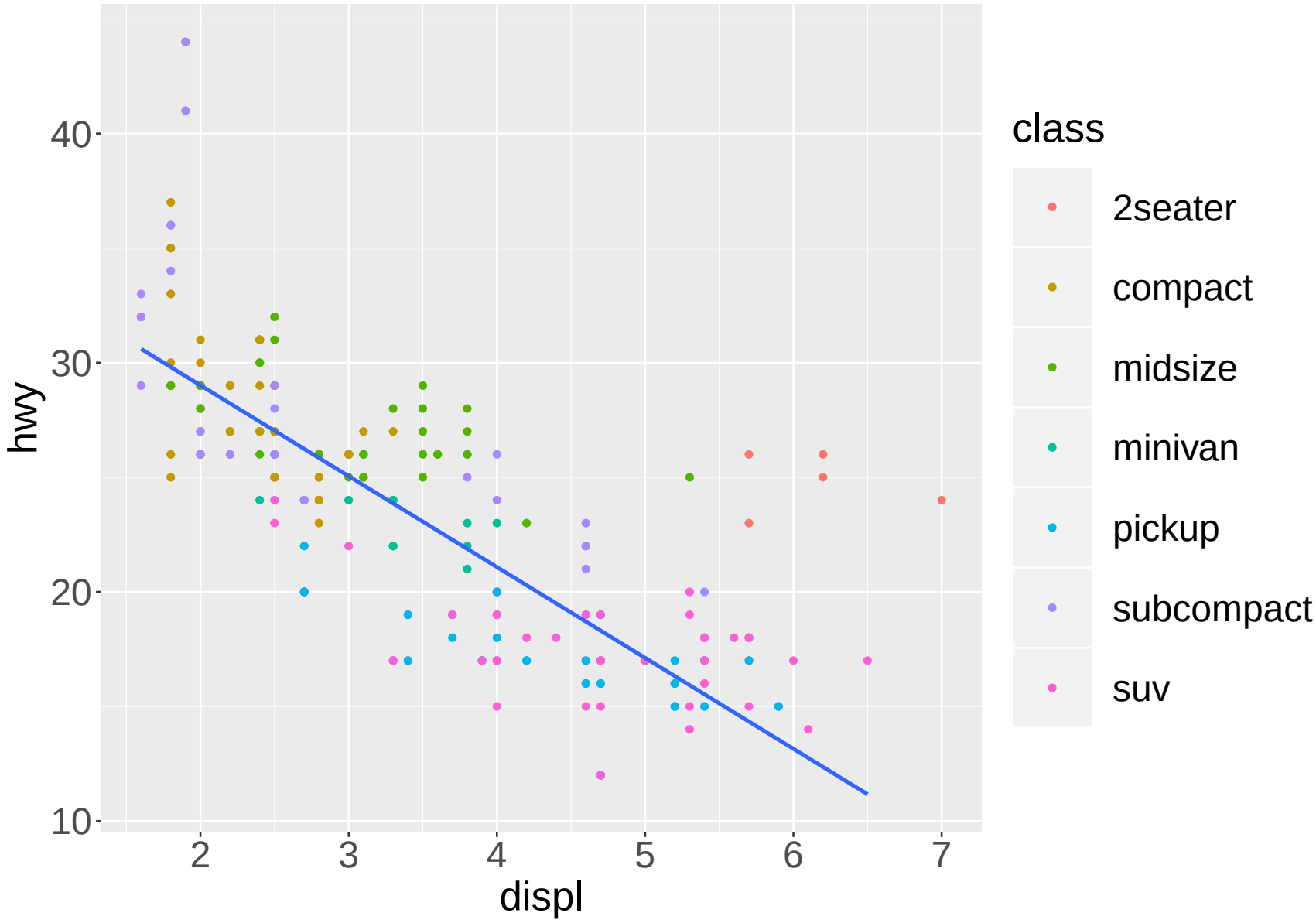
Multiple geoms



Multiple geoms

```
ggplot(data = mpg, aes(x = displ, y = hwy)) +
       geom_point(aes(color = class)) +
       geom_smooth(
         data = subset(mpg, class != "2seater"),
         se = FALSE,
         method = "lm")
```

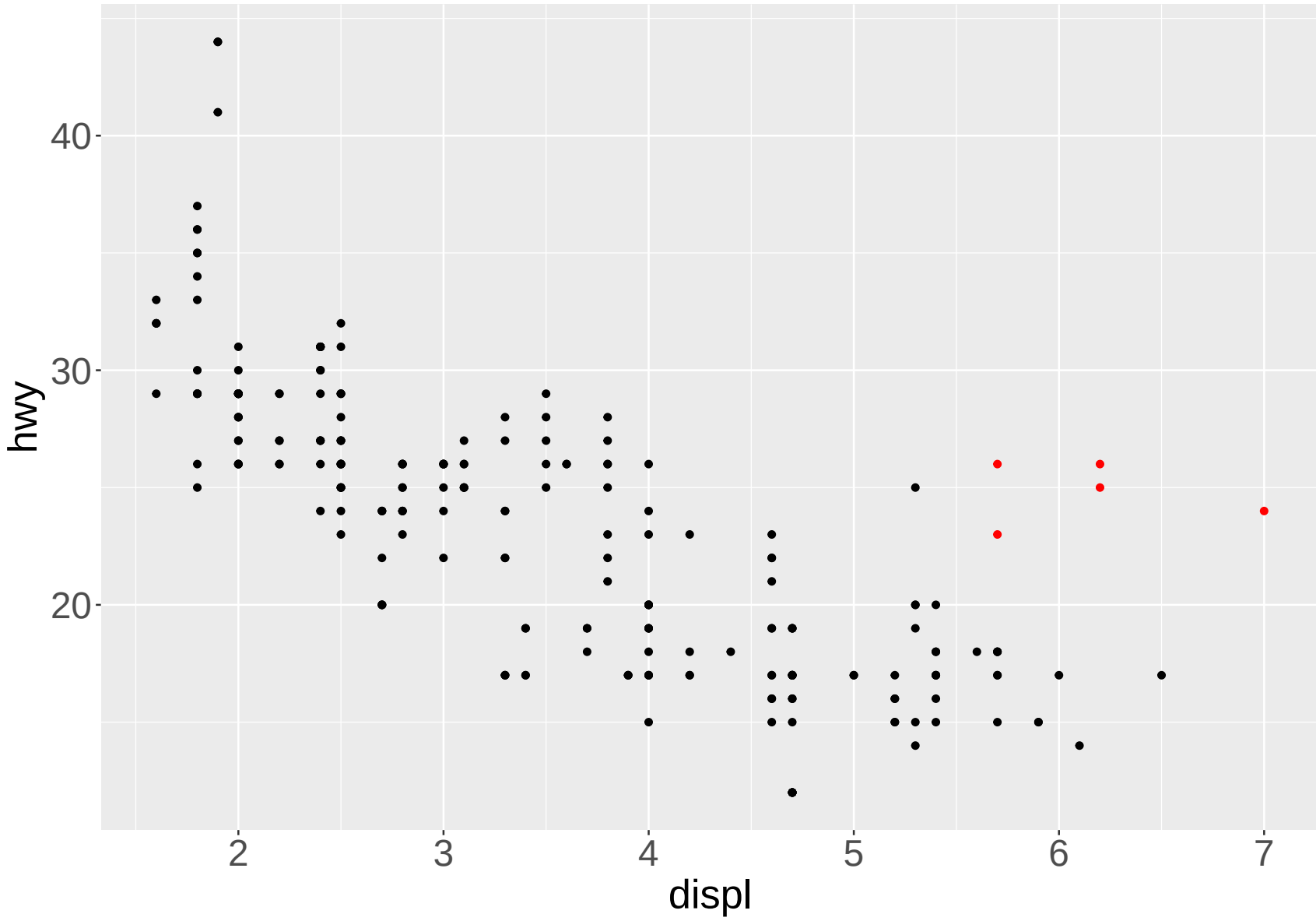
Multiple geoms



Multiple geoms

```
corvette <- subset(mpg, model == "corvette")
not_corvette <- subset(mpg, model != "corvette")

ggplot(mapping = aes(x = displ, y = hwy)) +
  geom_point(data = not_corvette) +
  geom_point(data = corvette,  color = "Red")
```

Multiple geoms

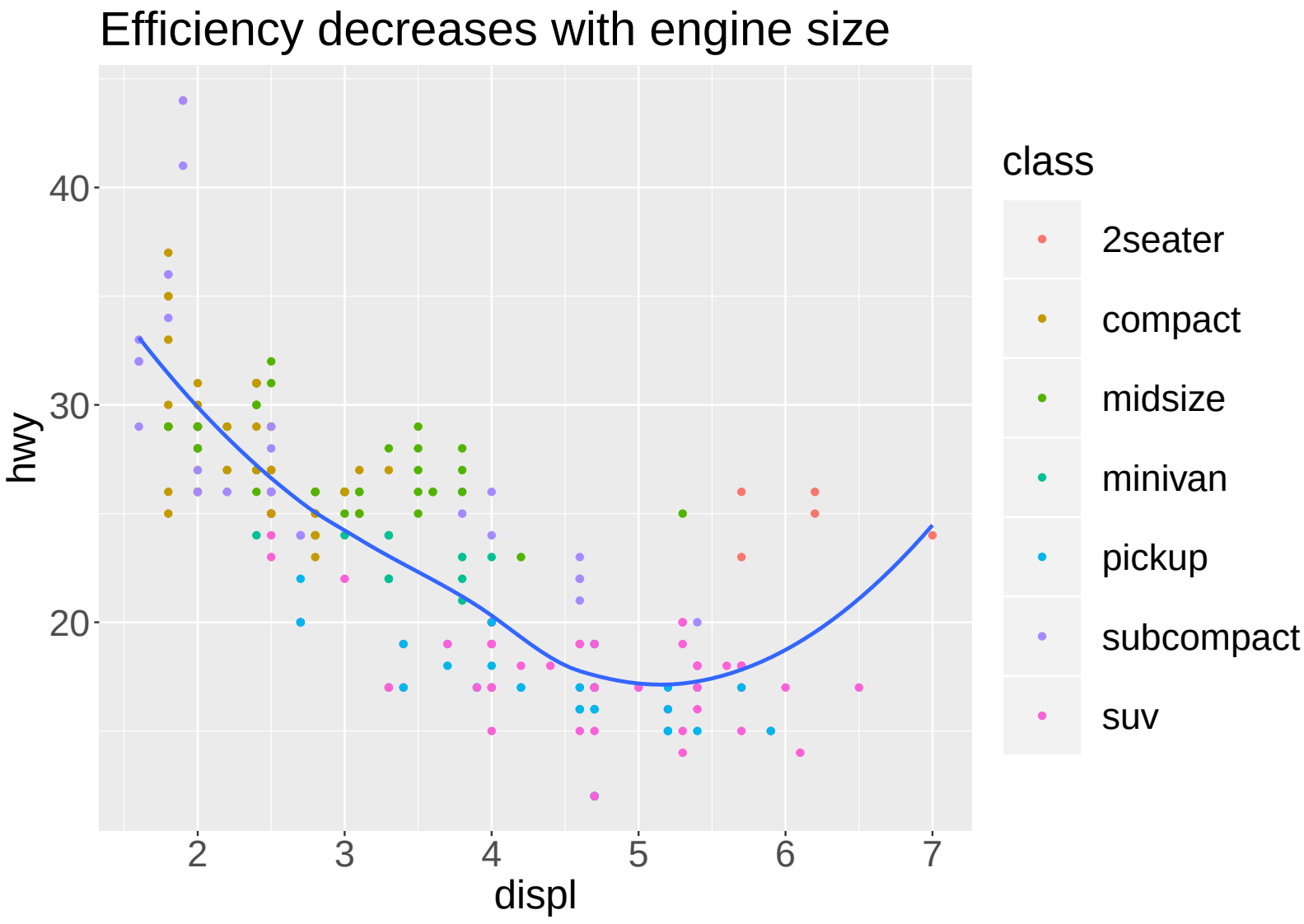


Labels

We can add labels to a graph with the `labs()` function:

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  labs(
    title = "Efficiency decreases with engine size")
```

Labels

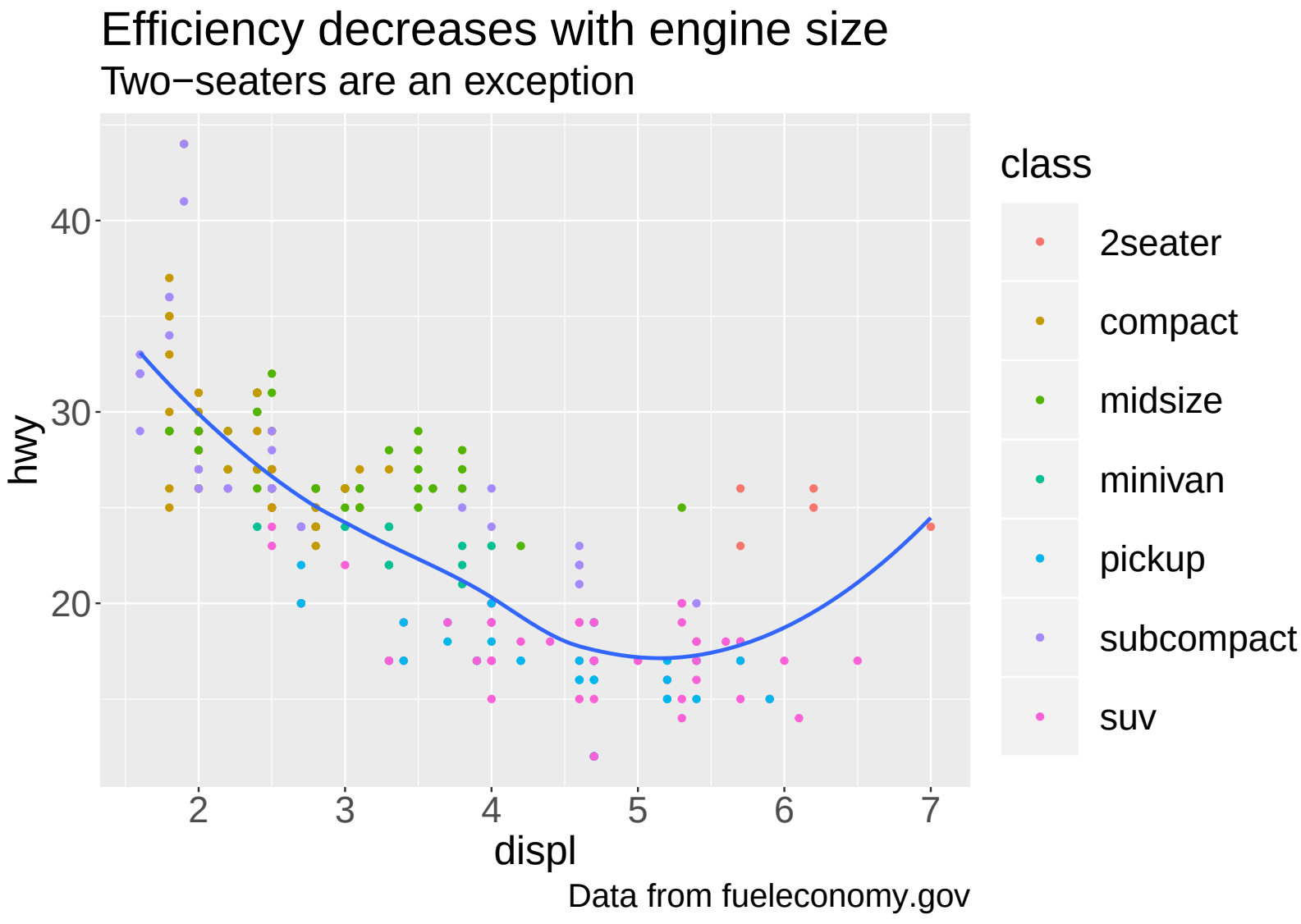

Efficiency decreases with engine size
hwy
40
30
20
2
3
4
5
6
7
displ
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

Labels

The `labs()` function can also be used to add a subtitle and/or a caption:

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  labs(
    title = "Efficiency decreases with engine size",
    subtitle = "Two-seaters are an exception",
    caption = "Data from fueleconomy.gov"
  )
```

Labels


Efficiency decreases with engine size
Two−seaters are an exception
hwy
40
30
20
2
3
4
5
6
7
displ
Data from fueleconomy.gov
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

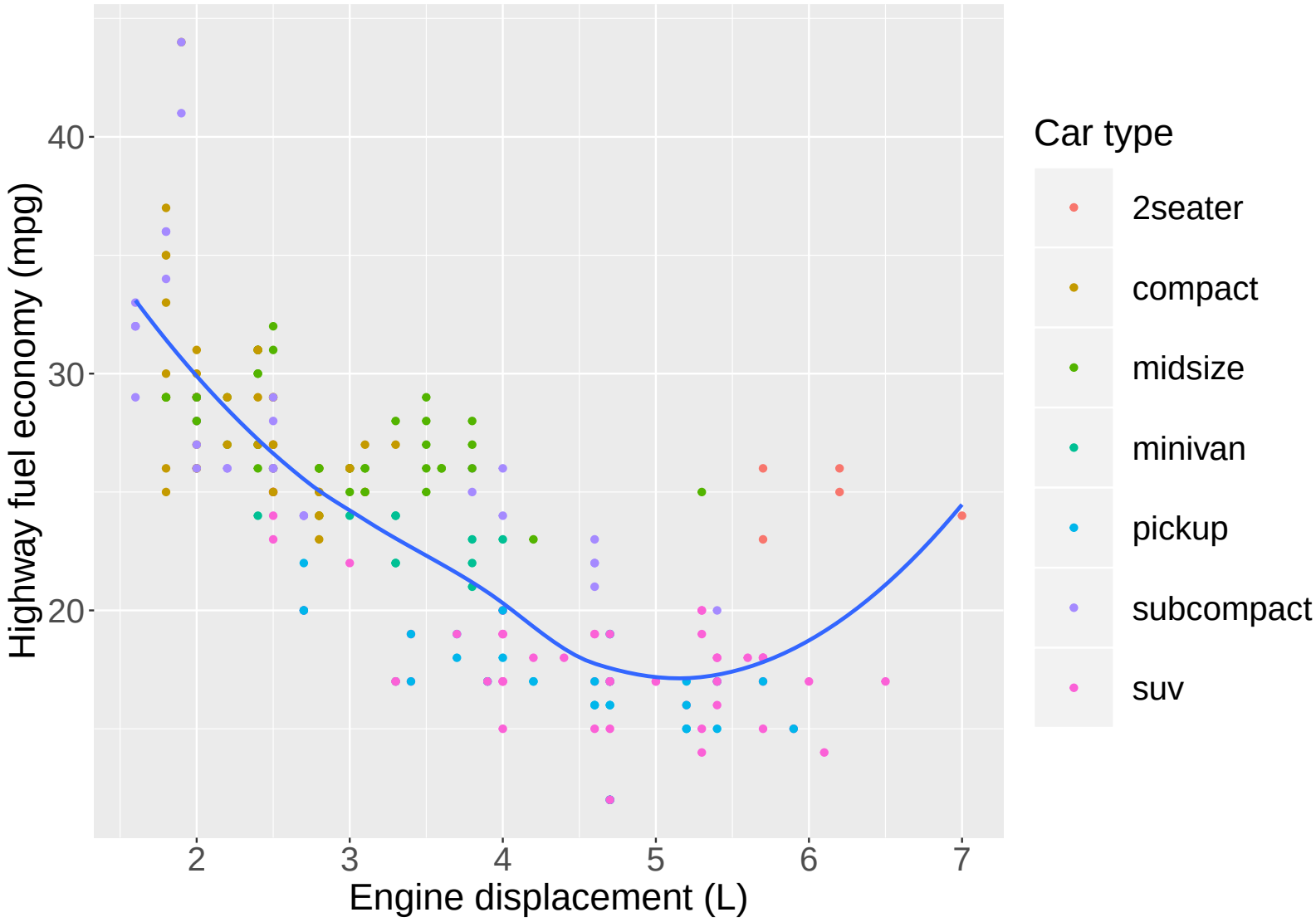
Labels

- We can also use labs() to replace the axis and legend titles.
- It's usually a good idea to replace short variable names with more detailed descriptions, and to include the units.

Labels

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  labs(
    x = "Engine displacement (L)",
    y = "Highway fuel economy (mpg)",
    color = "Car type"
```

Labels

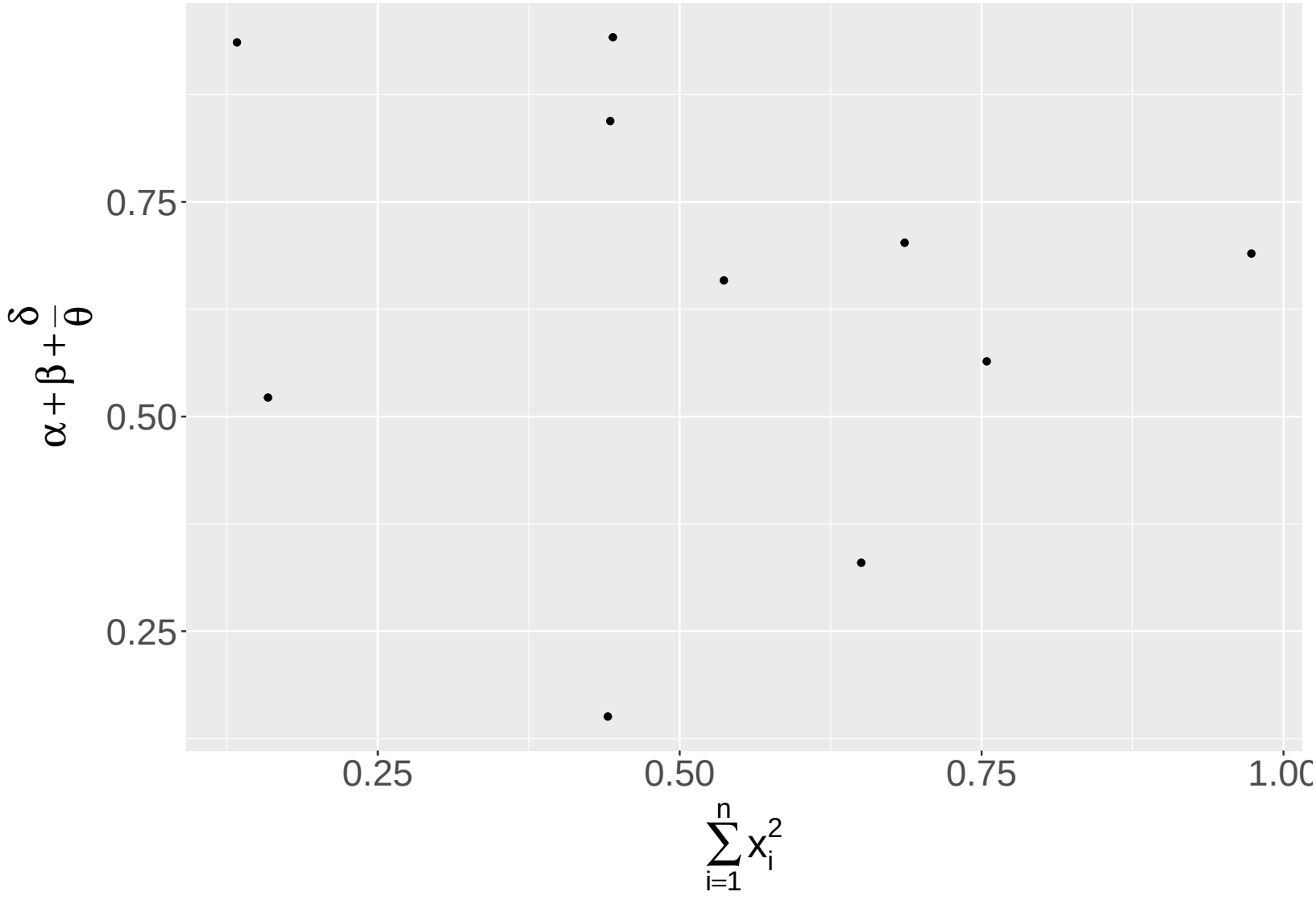

Highway fuel economy (mpg)
40
30
20
2
3
4
5
6
7
Engine displacement (L)
Car type
2seater
compact
midsize
minivan
pickup
subcompact
suv

Labels

It's possible to use mathematical equations instead of text strings:

```
df <- tibble(
  x = runif(10),
  y = runif(10))

ggplot(df, aes(x, y)) +
  geom_point() +
  labs(
    x = quote(sum(x[i] ^ 2, i == 1, n)),
    y = quote(alpha + beta + frac(delta, theta))
  )
```

Labels


0.75
0.50
0.25
$\alpha+\beta+\frac{\delta}{\theta}$
0.25
0.50
0.75
1.00
$\sum_{i=1}^{n} x_i^2$

Labels

To learn more about the mathematical syntax available see:

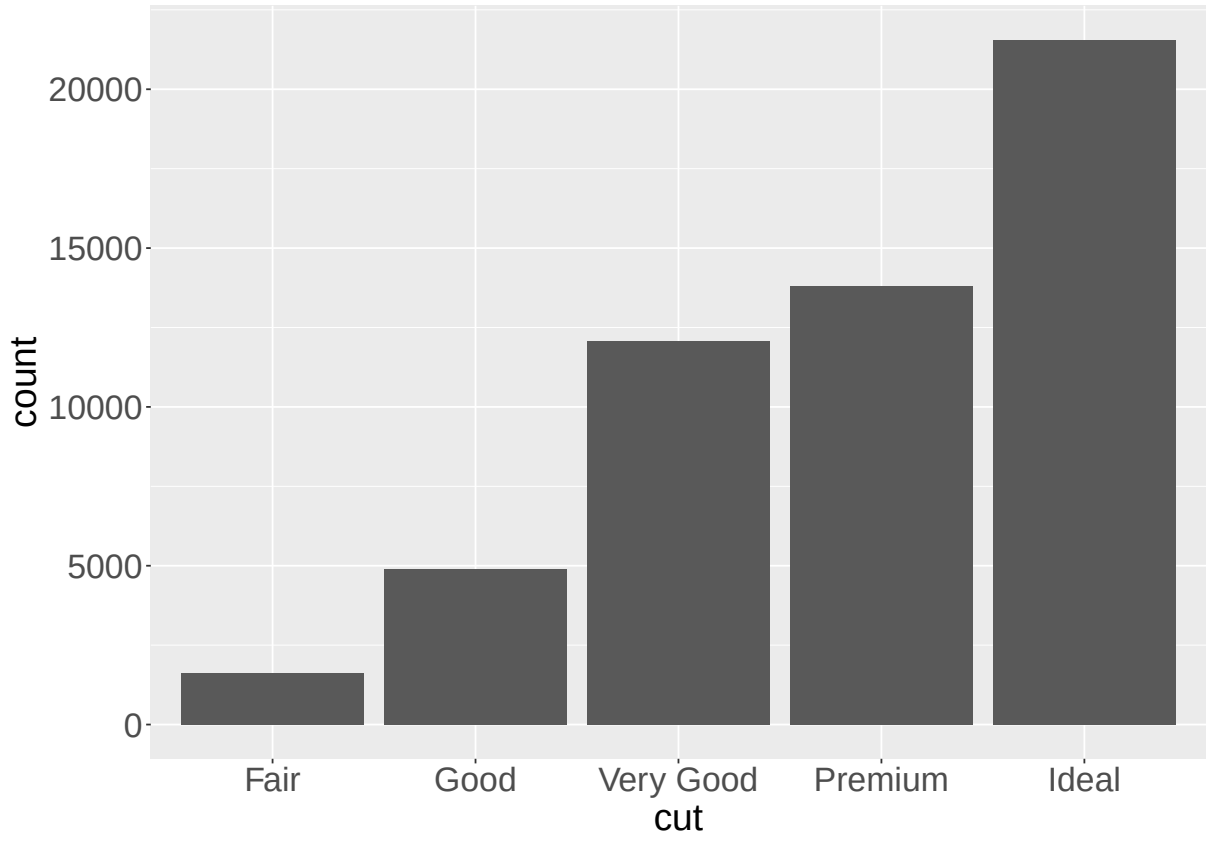
```
?plotmath
```

The `diamonds` dataset

- Diamonds is a dataset included in the ggplot2 package.
- Contains attributes of almost 54000 diamonds.
- The variables include price, carat, quality of the cut, color and clarity

Bar charts

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



Statistical transformations

- On the y-axis, bar charts displays counts.
- But counts are not a variable in diamonds dataset!
- Many graphs, like scatterplots, plot the raw values the dataset.
- Other graphs, like bar charts, calculate new values to plot.
- The algorithm used to calculate new values for a graph is called a **stat**, short for statistical transformation.

Statistical transformations

- Bar charts, histograms, and frequency polygons bin the data and then plot bin counts.
- Smoothers fit a model to the data and then plot predictions from the model.
- Boxplots compute summary statistics and then display them on specially formatted box.

Statistical transformations

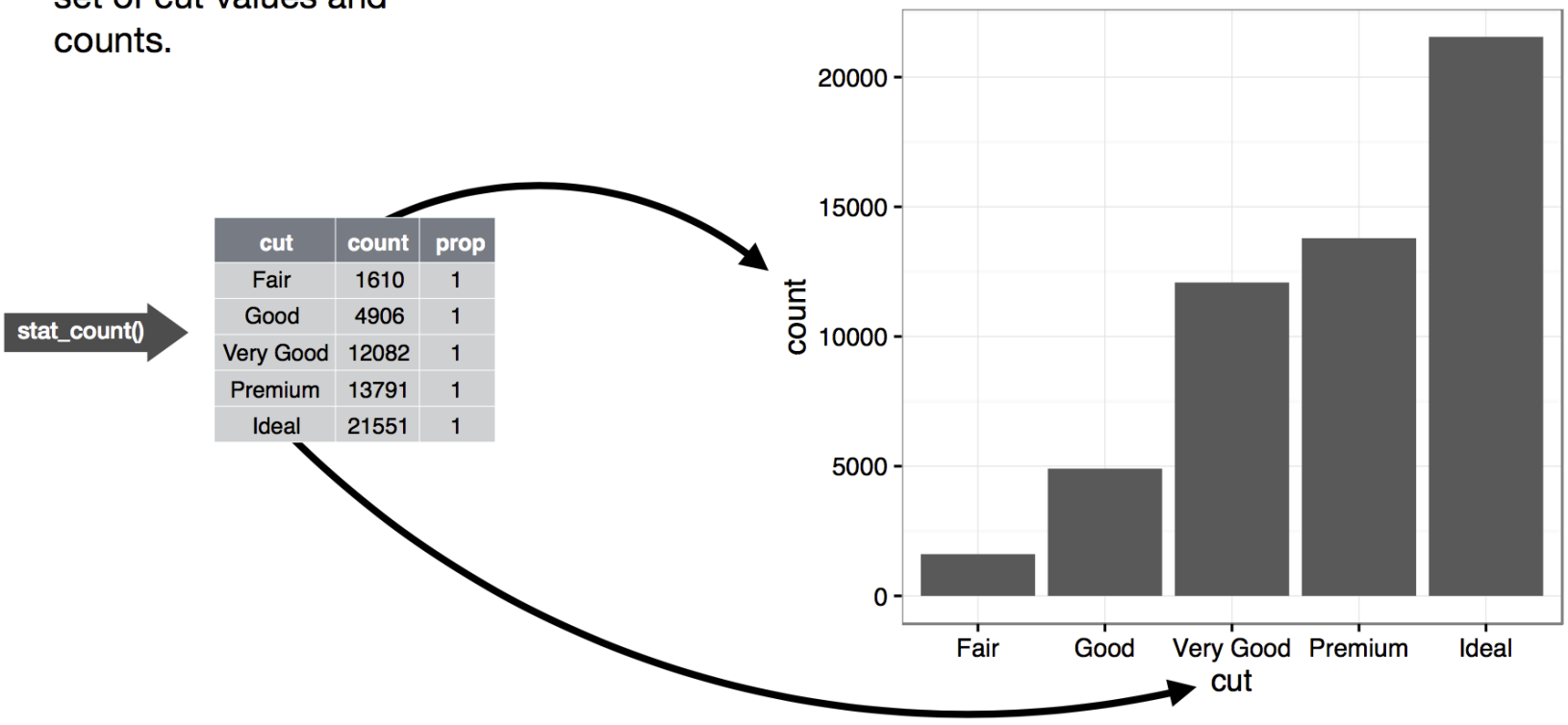
1. **`geom_bar()`** begins with the **`diamonds`** data set

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...

Statistical transformations

2. **`geom_bar()`** transforms the data with the "count" stat, which returns a data set of cut values and counts.

3. **`geom_bar()`** uses the transformed data to build the plot. cut is mapped to the x axis, count is mapped to the y axis.

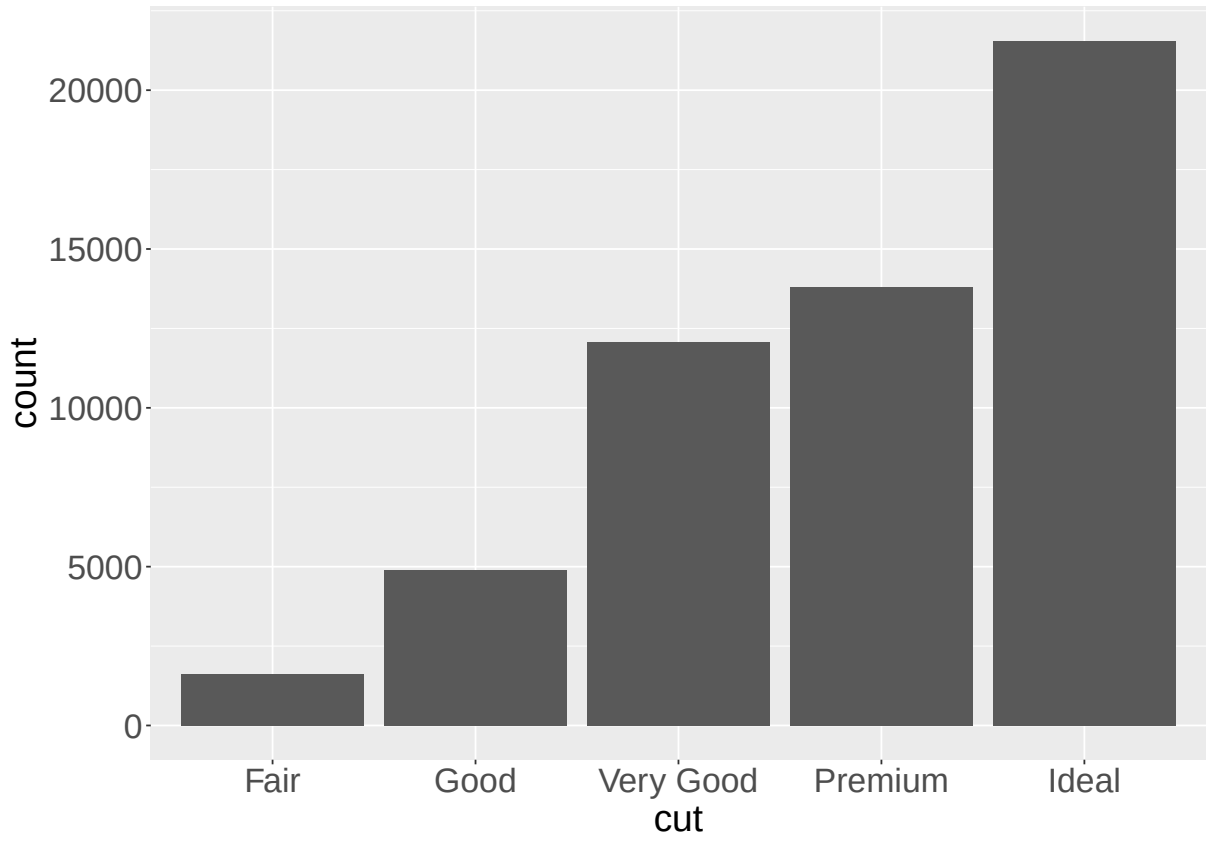


Statistical transformations

- We can learn which stat a geom function uses by inspecting the default value of the `stat` argument.
- For example, with `?geom_bar` we learn that the default `stat` of `geom_bar()` is `count`.
- This means that `geom_bar()` uses `stat_count()` as the default statistical transformation.
- We can generally use geoms and stats interchangeably.
- For example, we can recreate the previous plot using `stat_count()` instead of `geom_bar()`.

Statistical transformations

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



Statistical transformations

This works because:

- Every geom has a default stat, and every stat has a default geom.
- The default `stat` of `geom_bar()` is `count`.
- The default `geom` of `stat_count()` is `bar`.

Overwriting the default stat

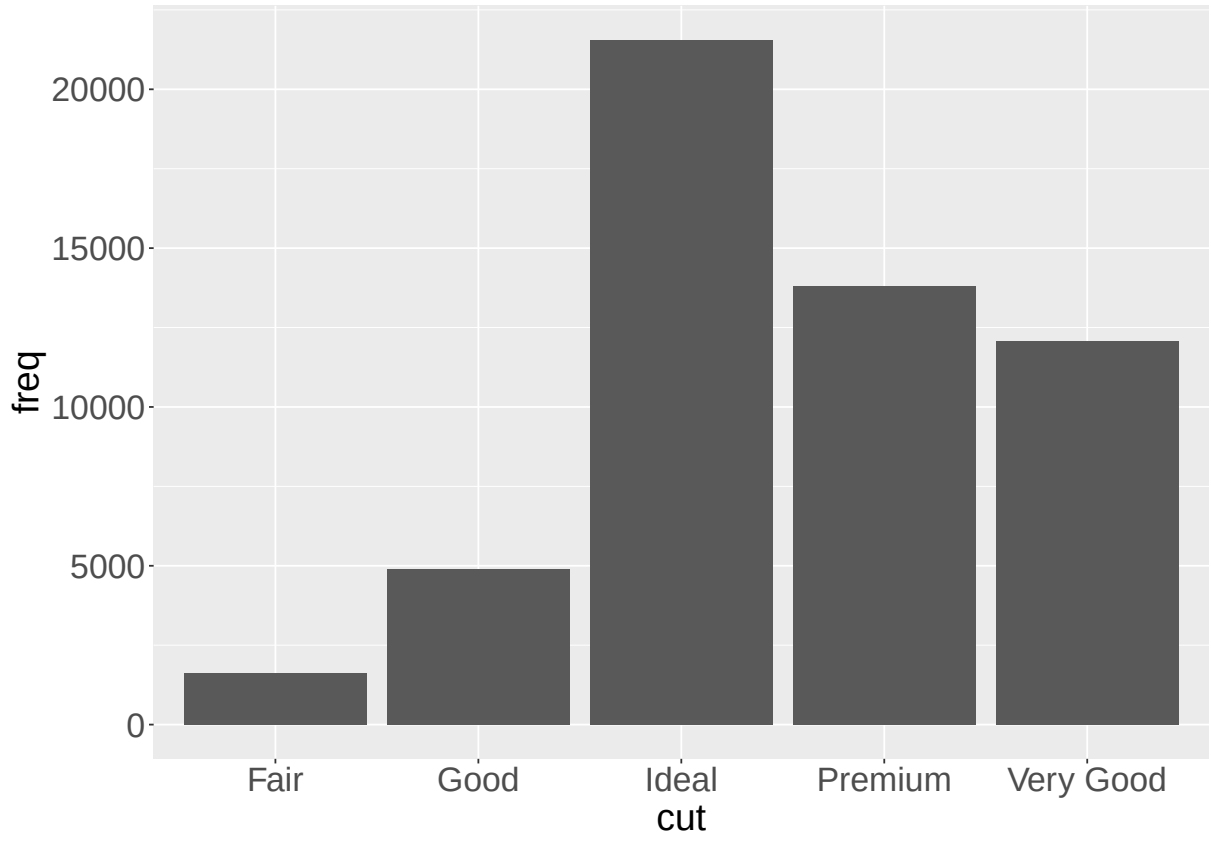
What if we want a bar chart that plot data as is?

- We may have a column with column heights.
- In that case, we need to change the default statistical transformation.

```
tib <- tribble(
  ~cut,          ~freq,
  "Fair",        1610,
  "Good",        4906,
  "Very Good",   12082,
  "Premium",     13791,
  "Ideal",       21551
)
```

Overriding the default stat

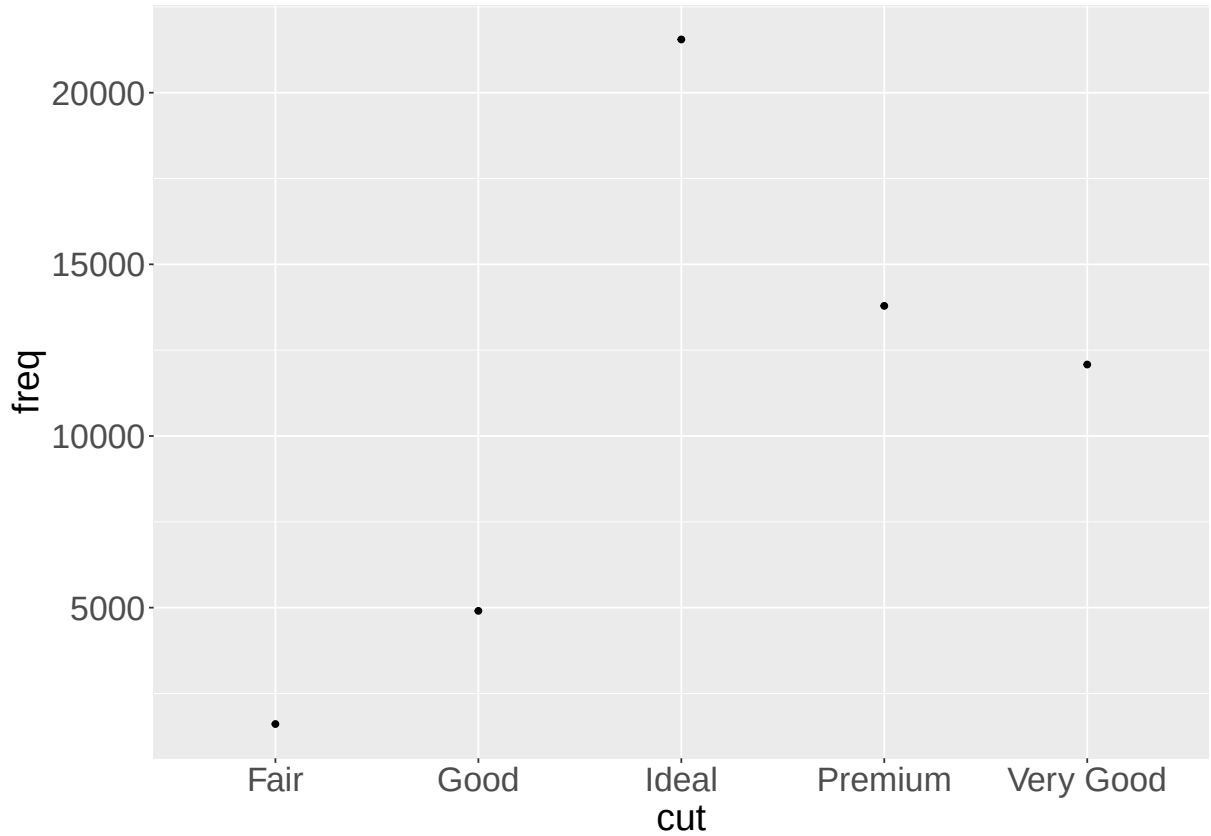
```
ggplot(data = tib) +
  geom_bar(aes(x = cut, y = freq),
    stat = "identity")
```



Overriding the default stat

The default stat of `stat_identity()` is `point`, not `bar`:

```
ggplot(data = tib) +
  stat_identity(aes(x = cut, y = freq))
```

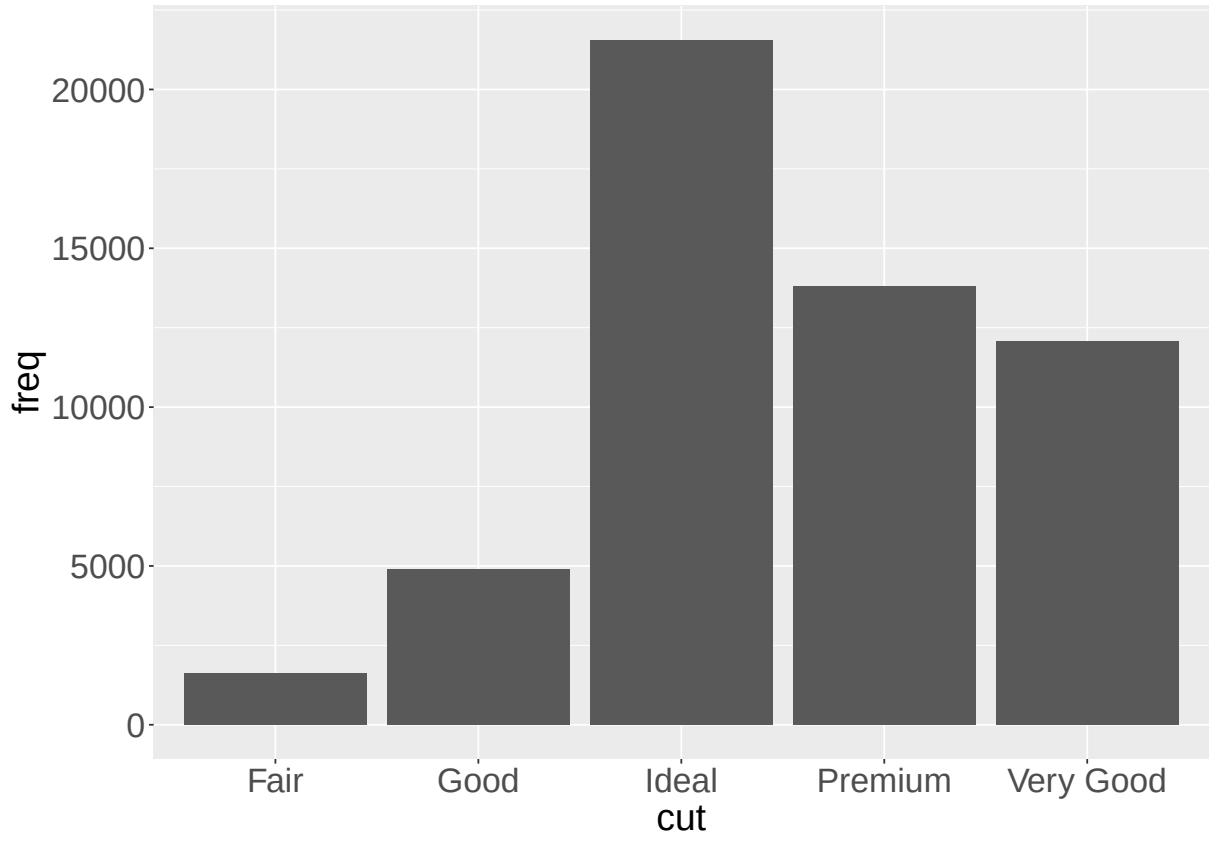


`geom_col()`

- To plot data as is, we can use `geom_col()`.
- The default stat of `geom_col()` is `stat_identity()`.
- `geom_col()` expects a column of y values with bar heights.

geom_col()

```
ggplot(data = tib) +
  geom_col(aes(x = cut, y = freq))
```



Statistical transformations

- Stat functions calculate more variables than the ones that end up being displayed.
- To find the variables computed by a stat, look for the help section titled "computed variables".
- From `?stat_count` we learn that the computed variables are counts and proportions.
- `ggplot_build()` let's us see every value that is calculated in the process of building a graph.

Statistical transformations

```
plt_1 <- ggplot(data = diamonds) +
  geom_bar(aes(x = cut))

plt_1 <- ggplot_build(plt_1)
plt_1$data[[1]][, 1:8]
```

```
##       y count prop x PANEL group ymin  ymax
## 1  1610  1610    1 1     1     1    0  1610
## 2  4906  4906    1 2     1     2    0  4906
## 3 12082 12082    1 3     1     3    0 12082
## 4 13791 13791    1 4     1     4    0 13791
## 5 21551 21551    1 5     1     5    0 21551
```

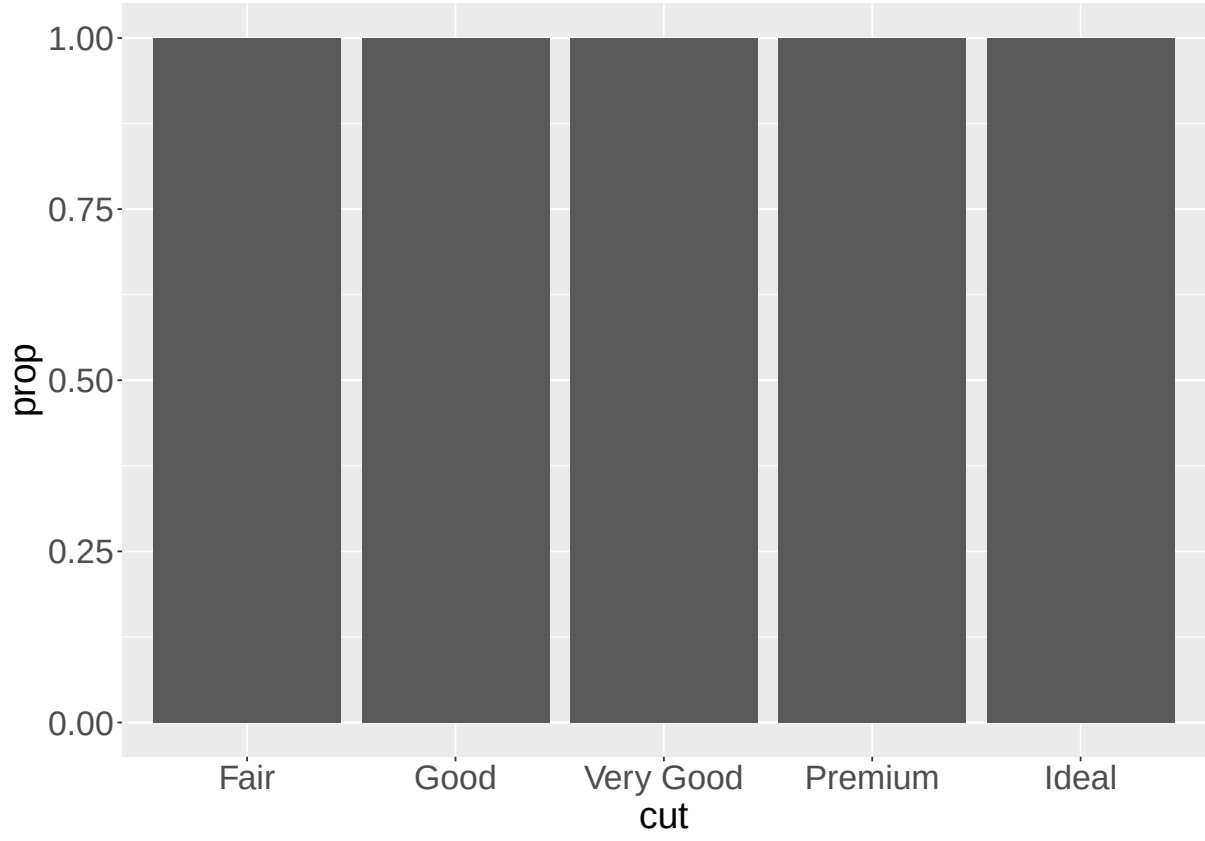
Mapping from transformed variables to aesthetics

- We can override the default mapping from transformed variables to aesthetics.
- For example, we might want to display a bar chart of proportions rather than counts.

Mapping from transformed variables to aesthetics

Let´s map proportions, instead of counts, to the y axis:

```
ggplot(data = diamonds) +
  geom_bar(aes(x = cut, y = stat(prop)))
```



What went wrong?

Mapping from transformed variables to aesthetics

Lets see the computed values:

```
plt_2 <- ggplot(data = diamonds) +
  geom_bar(aes(x = cut, y = stat(prop)))

plt_2 <- ggplot_build(plt_2)
plt_2$data[[1]][, 1:8]
```

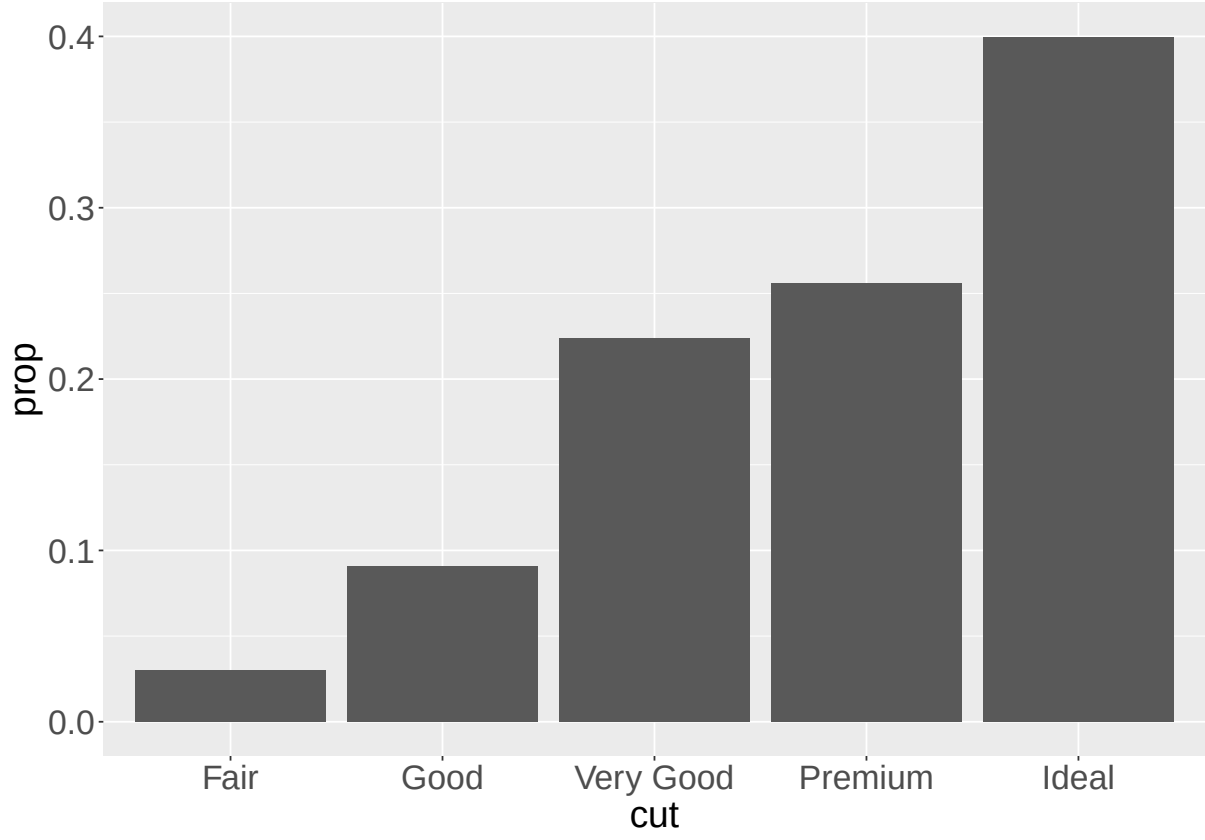
```
##   y count prop x PANEL group ymin ymax
## 1 1  1610    1 1     1     1    0    1
## 2 1  4906    1 2     1     2    0    1
## 3 1 12082    1 3     1     3    0    1
## 4 1 13791    1 4     1     4    0    1
## 5 1 21551    1 5     1     5    0    1
```

Mapping from transformed variables to aesthetics

- The `prop` column is created as `count` divided by the sum of all of the counts that belong to the same group.
- By default, ggplot2 created one group for each level of `x`, so:
  - Each proportion is calculated by dividing the count of each group by itself.
  - All the proportions are set to 1.
- To display proportions instead of counts we have to tell ggplot2 that there is only one group so that it divides the counts by the total number of observations.

Mapping from transformed variables to aesthetics

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



Mapping from transformed variables to aesthetics

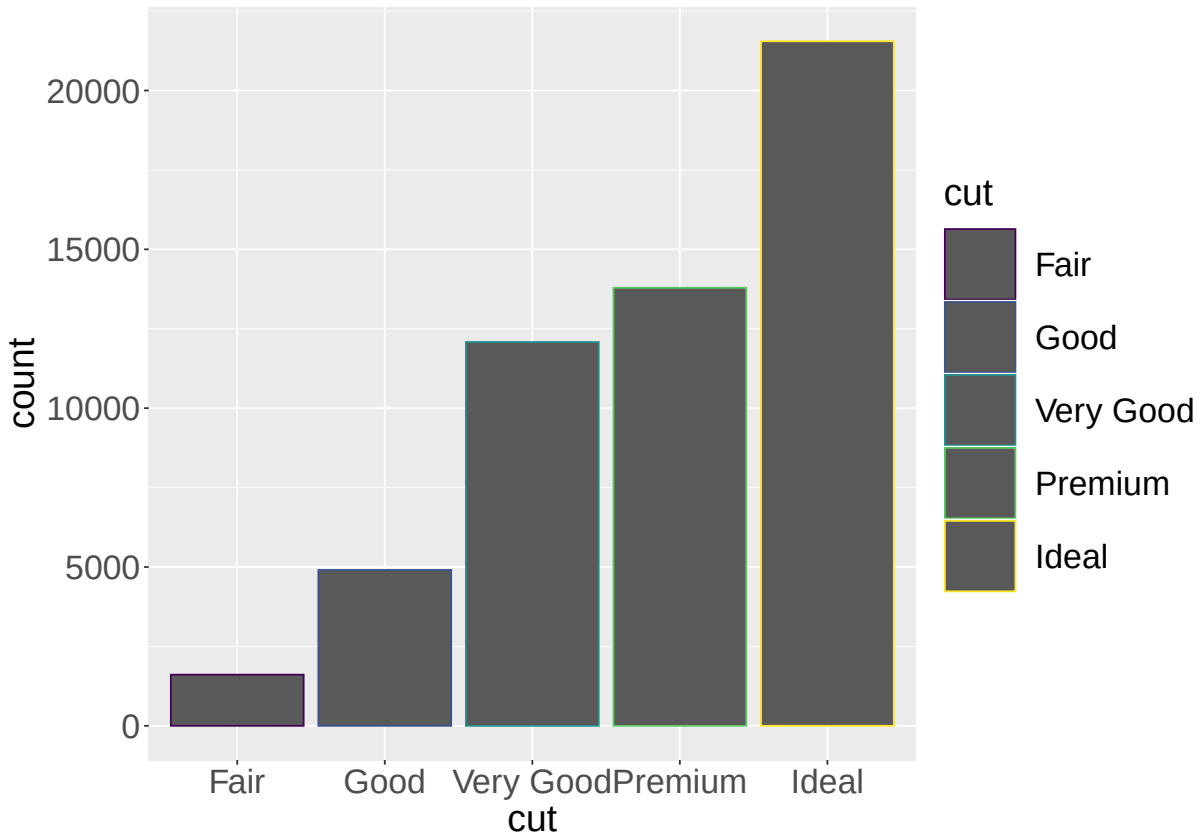
```
plt_3 <- ggplot(data = diamonds) +
  geom_bar(aes(x = cut, y = stat(prop), group = 1))
plt_3 <- ggplot_build(plt_3)

plt_3$data[[1]][, 1:5]
```

```
##             y count       prop x group
## 1 0.02984798  1610 0.02984798 1     1
## 2 0.09095291  4906 0.09095291 2     1
## 3 0.22398962 12082 0.22398962 3     1
## 4 0.25567297 13791 0.25567297 4     1
## 5 0.39953652 21551 0.39953652 5     1
```

Aesthetics of bar charts

We can map the color aesthetic to a grouping variable:

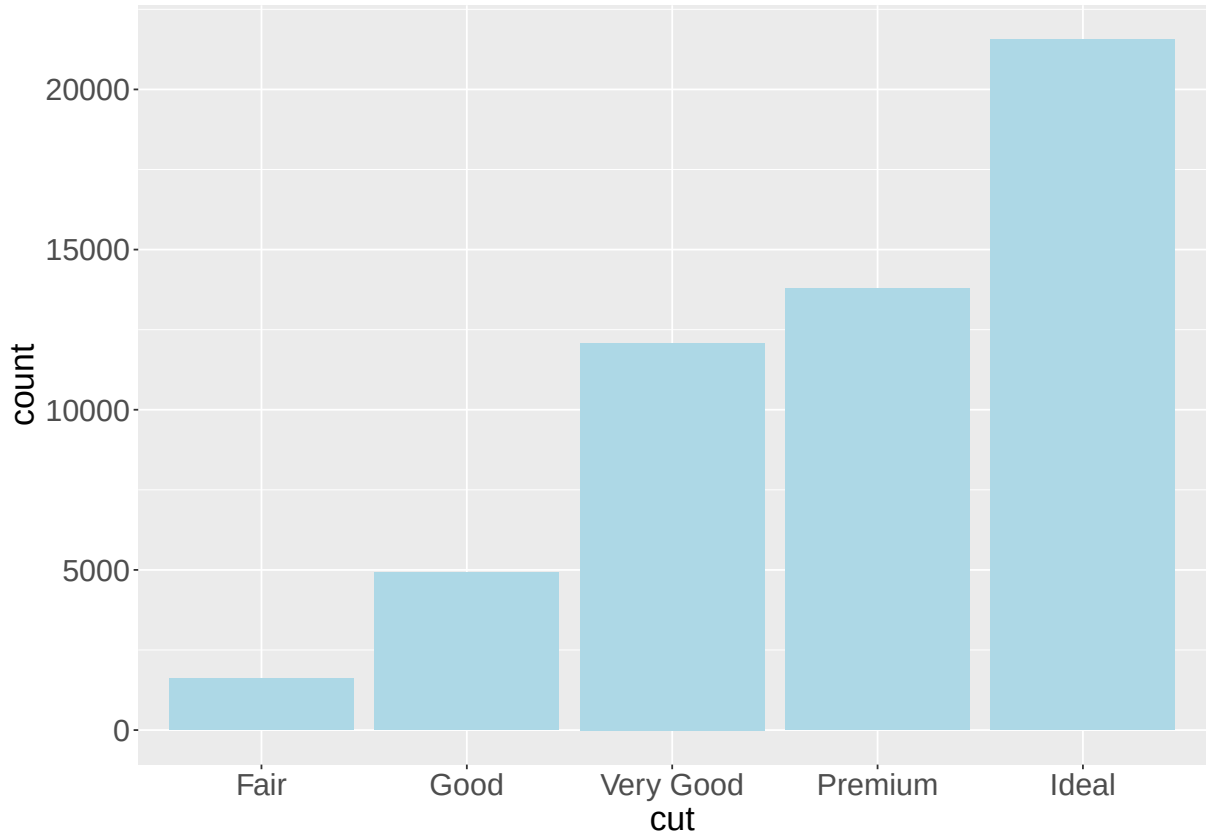
```
ggplot(data = diamonds) +
  geom_bar(aes(x = cut, color = cut))
```



Aesthetics of bar charts

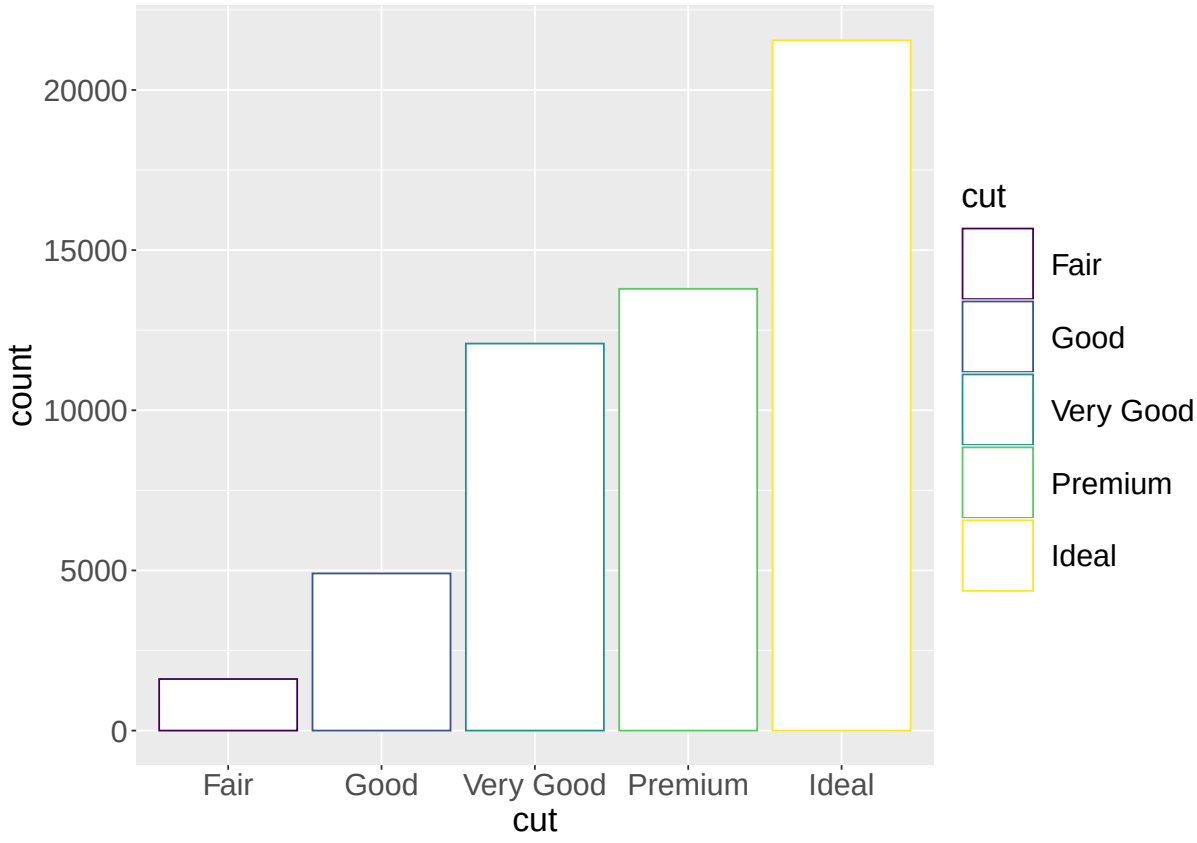
We can also change the default `fill` color of `geom_bar()`:

```
ggplot(data = diamonds) +
  geom_bar(aes(x = cut), fill = "lightblue")
```



Aesthetics of bar charts

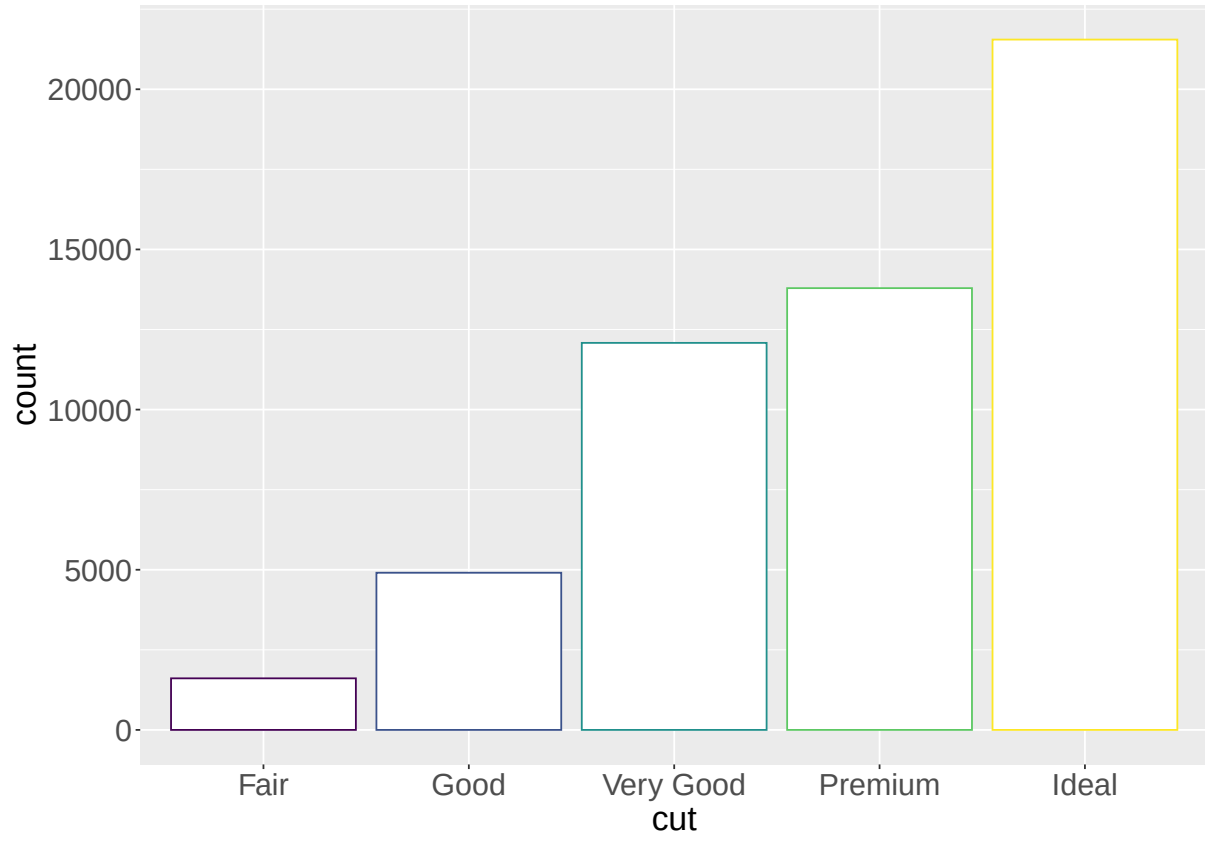
```
ggplot(data = diamonds) +
  geom_bar(aes(x = cut, color = cut), fill = "white")
```



Aesthetics of bar charts

And disable the legend:

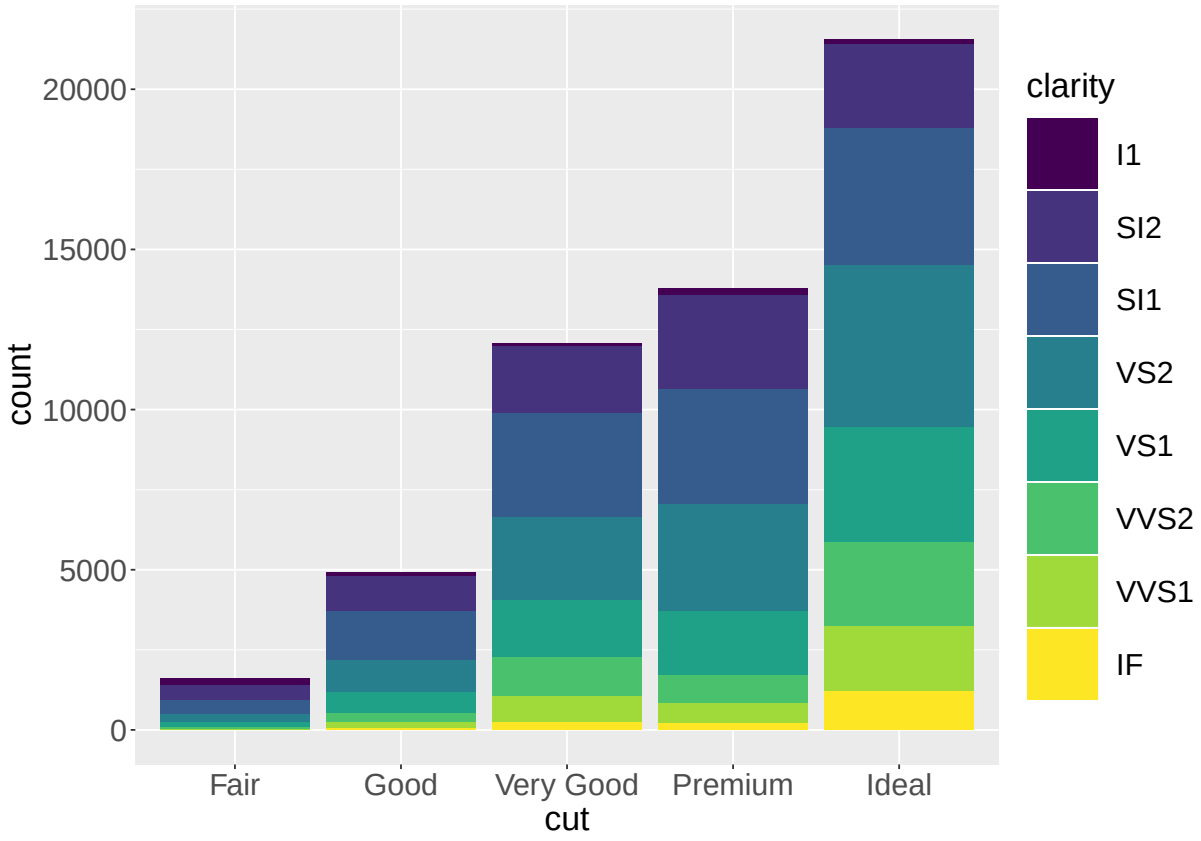
```
ggplot(data = diamonds) +
  geom_bar(aes(x = cut, color = cut),
           fill = "white", show.legend = FALSE)
```



Stacked bar charts

The fill aesthetic can be mapped to variables other than x to add a dimension to the plot:

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



Position adjustments

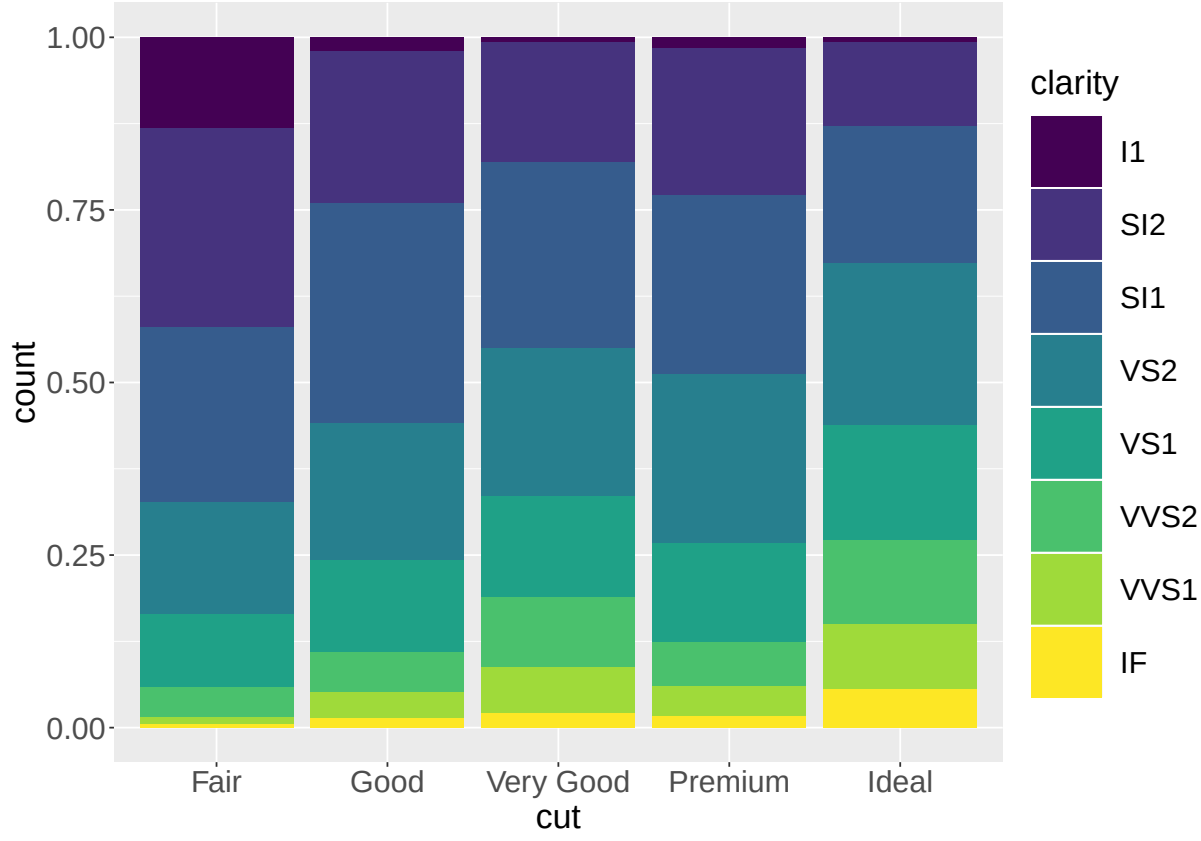
- Each colored rectangle represents a combination of cut and clarity.
- The stacking is performed automatically by the position adjustment specified in the `position` argument of the geom function.
- The default value is `stack`.

Position adjustments

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

Position adjustments

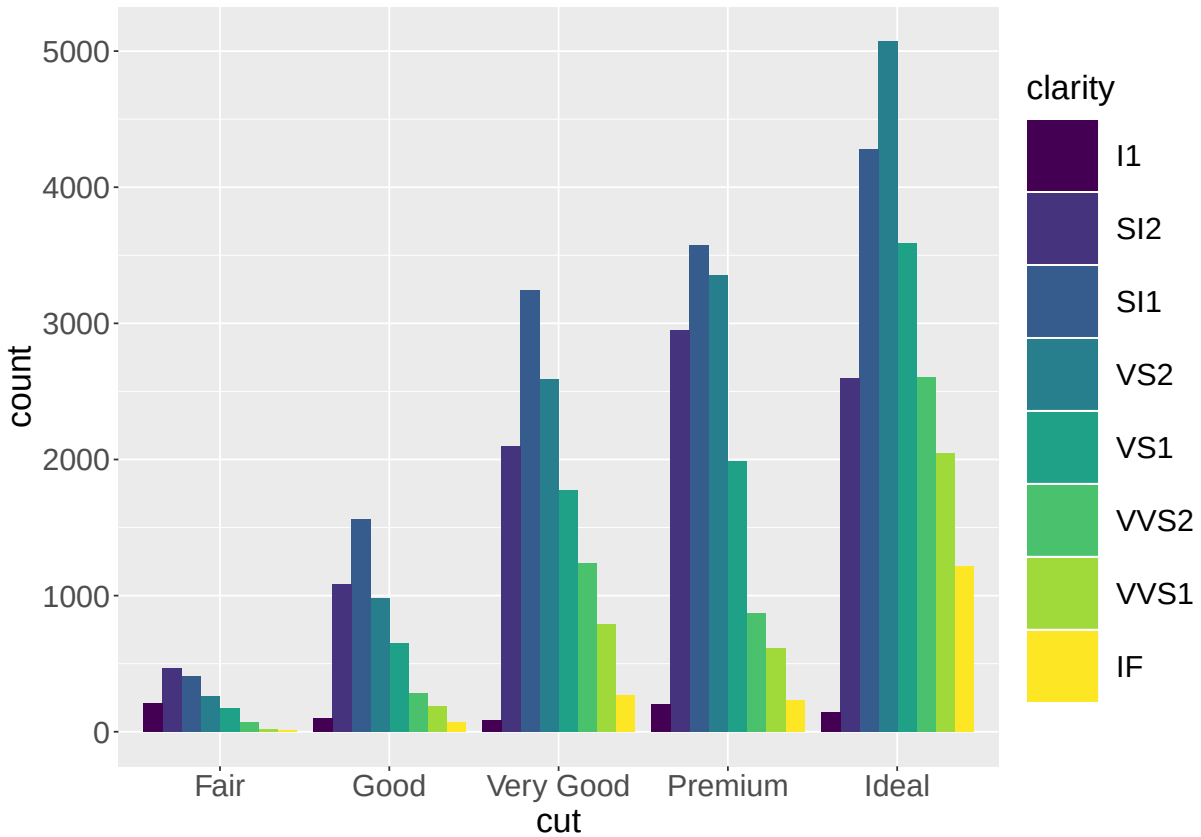
```
ggplot(data = diamonds) +
  geom_bar(aes(x = cut, fill = clarity),
           position = "fill")
```



Position adjustments

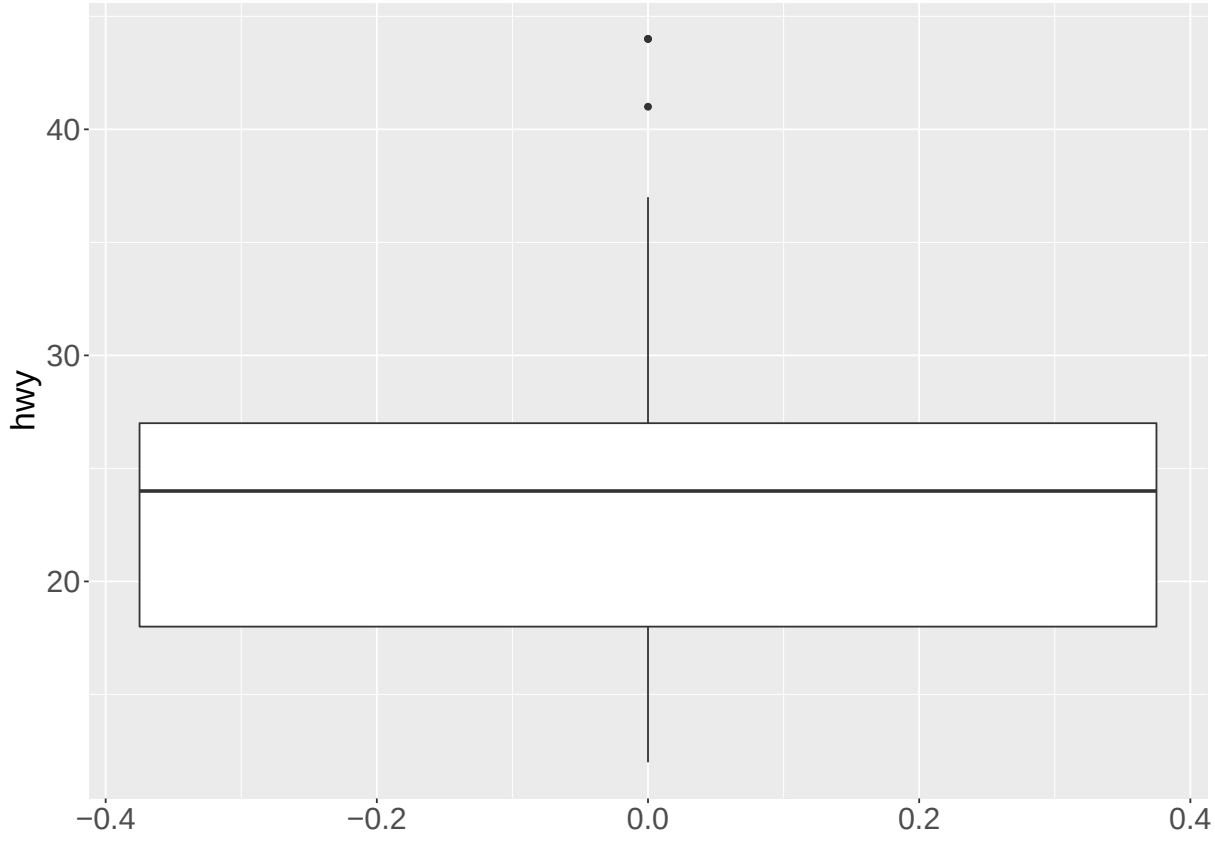
Setting `position = "dodge"` places overlapping objects directly beside one another:

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



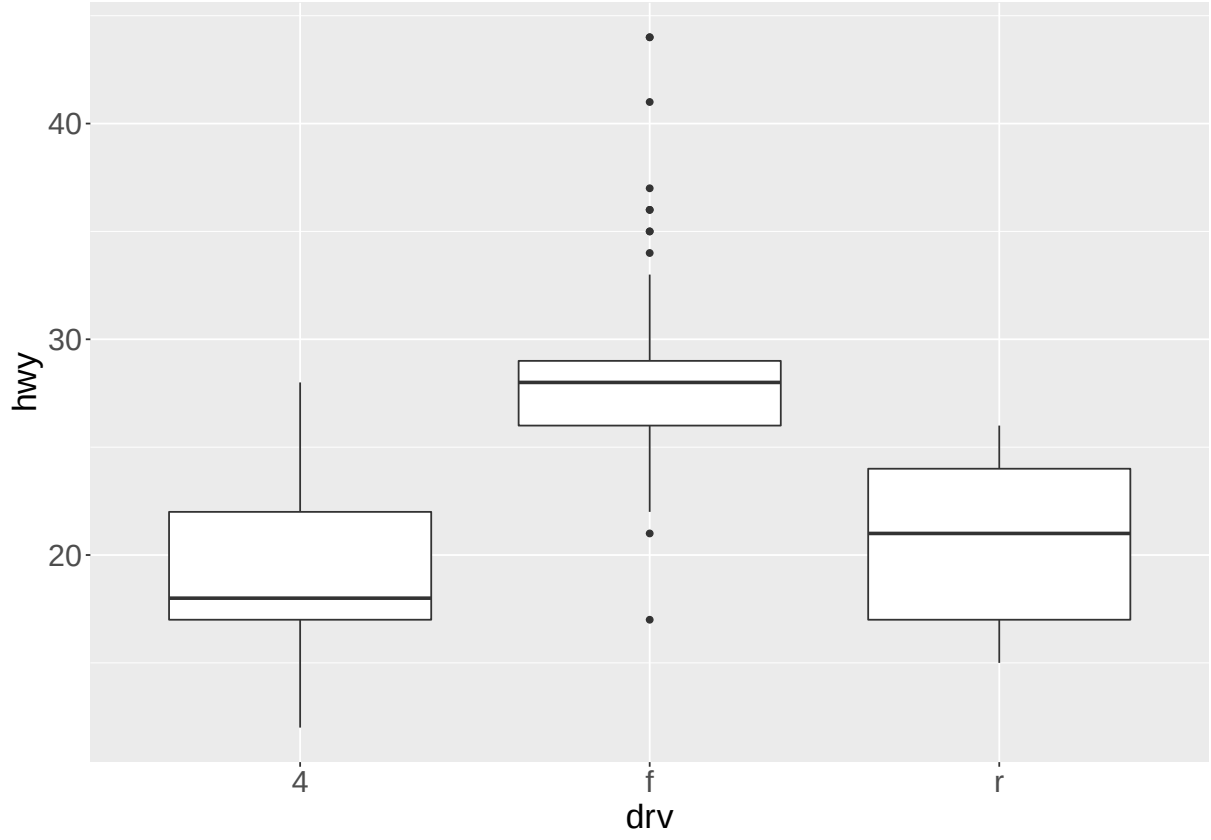
Boxplots

```
ggplot(data = mpg, aes(y = hwy)) +
  geom_boxplot()
```



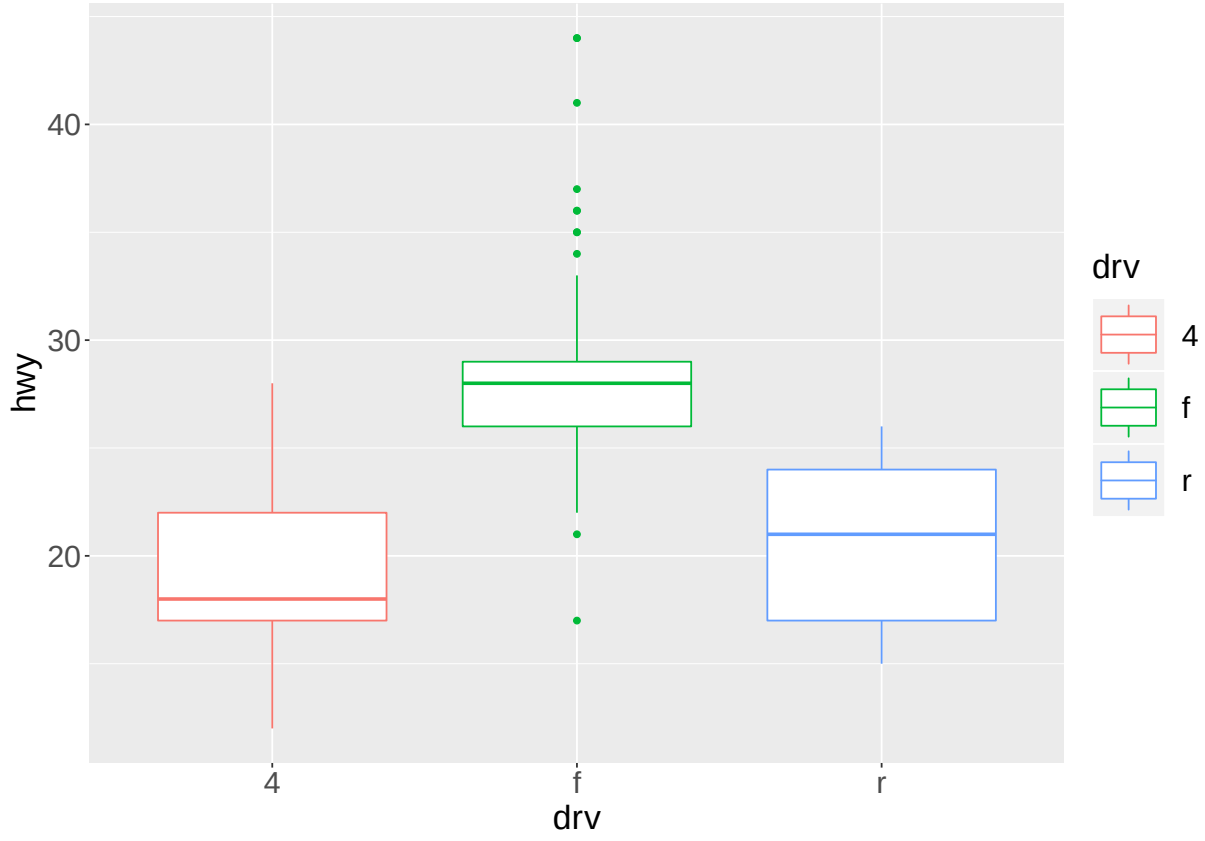
Boxplots

```
ggplot(data = mpg, aes(x = drv, y = hwy)) +
  geom_boxplot()
```



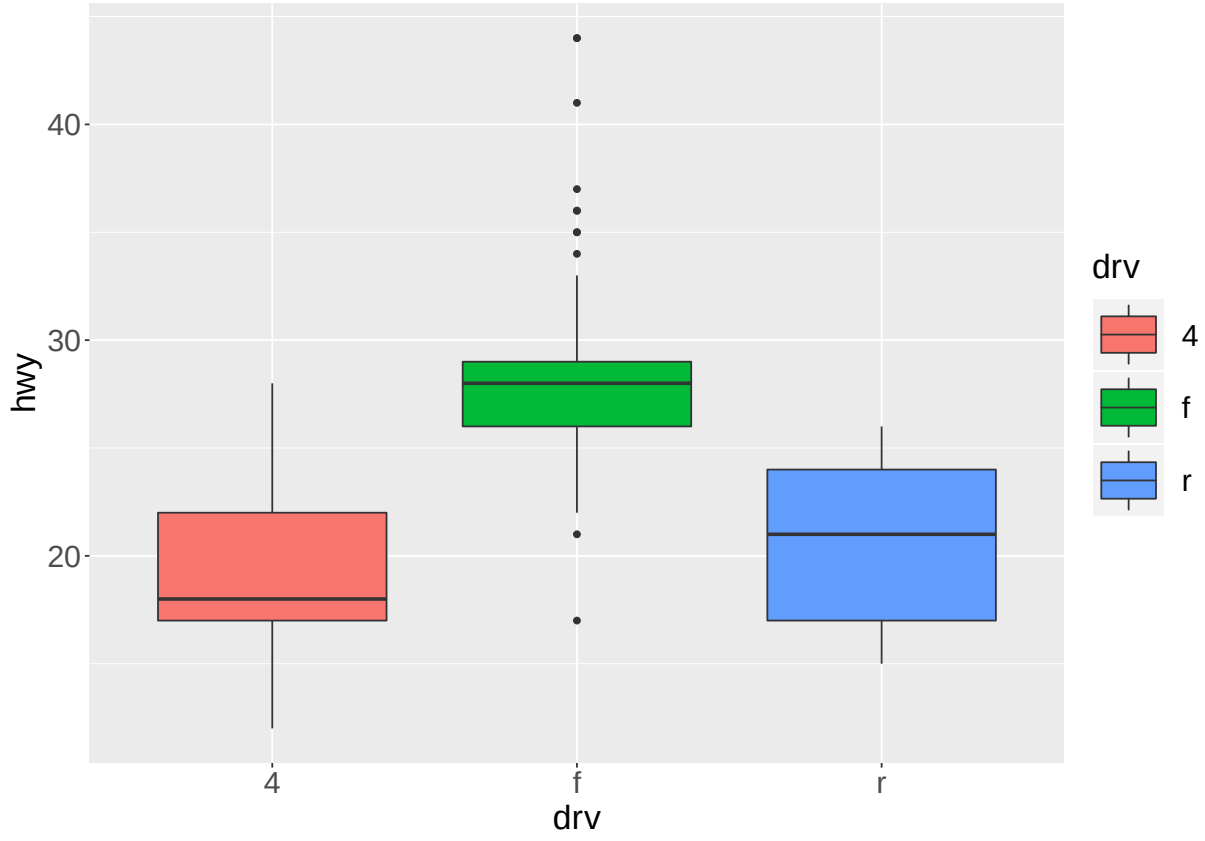
Boxplots

```
ggplot(data = mpg,
       aes(x = drv, y = hwy, color = drv)) +
  geom_boxplot()
```



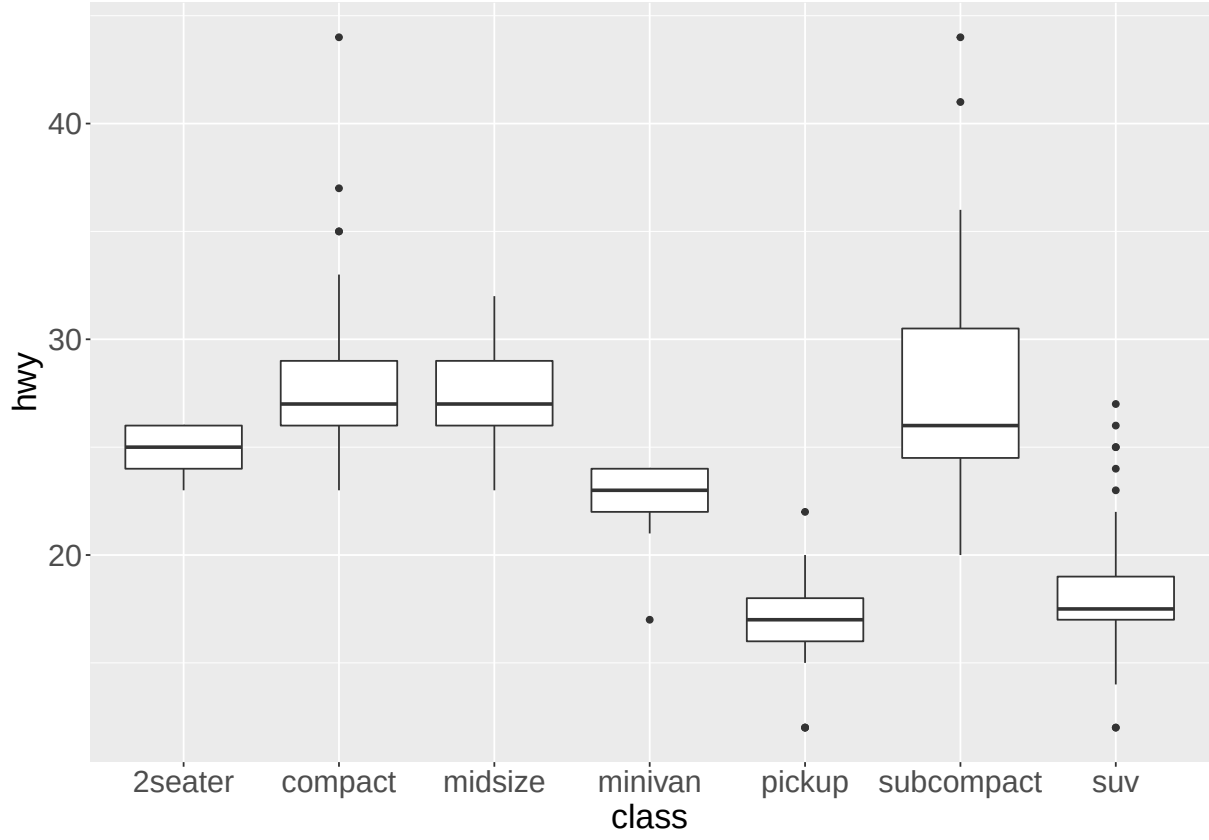
Boxplots

```
ggplot(data = mpg,
       aes(x = drv, y = hwy, fill = drv)) +
  geom_boxplot()
```



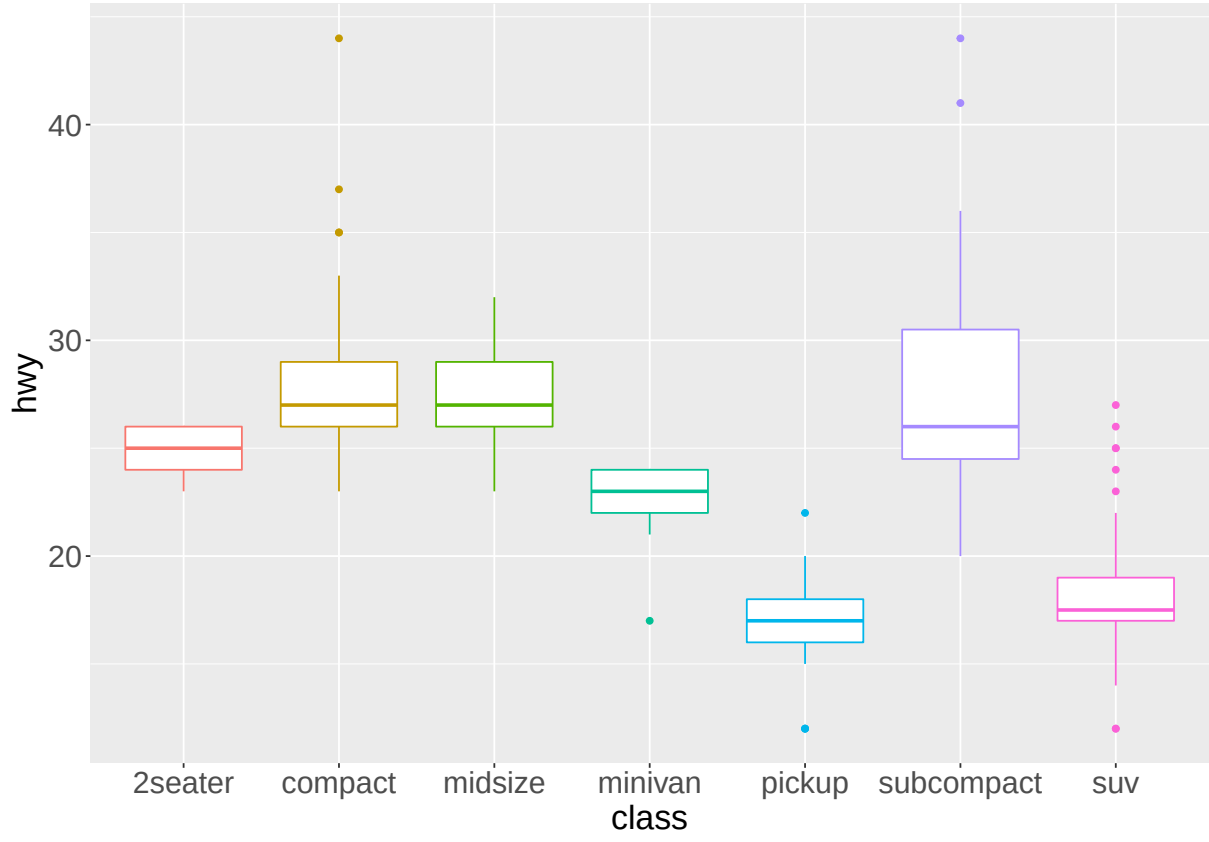
Boxplots

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



Boxplots

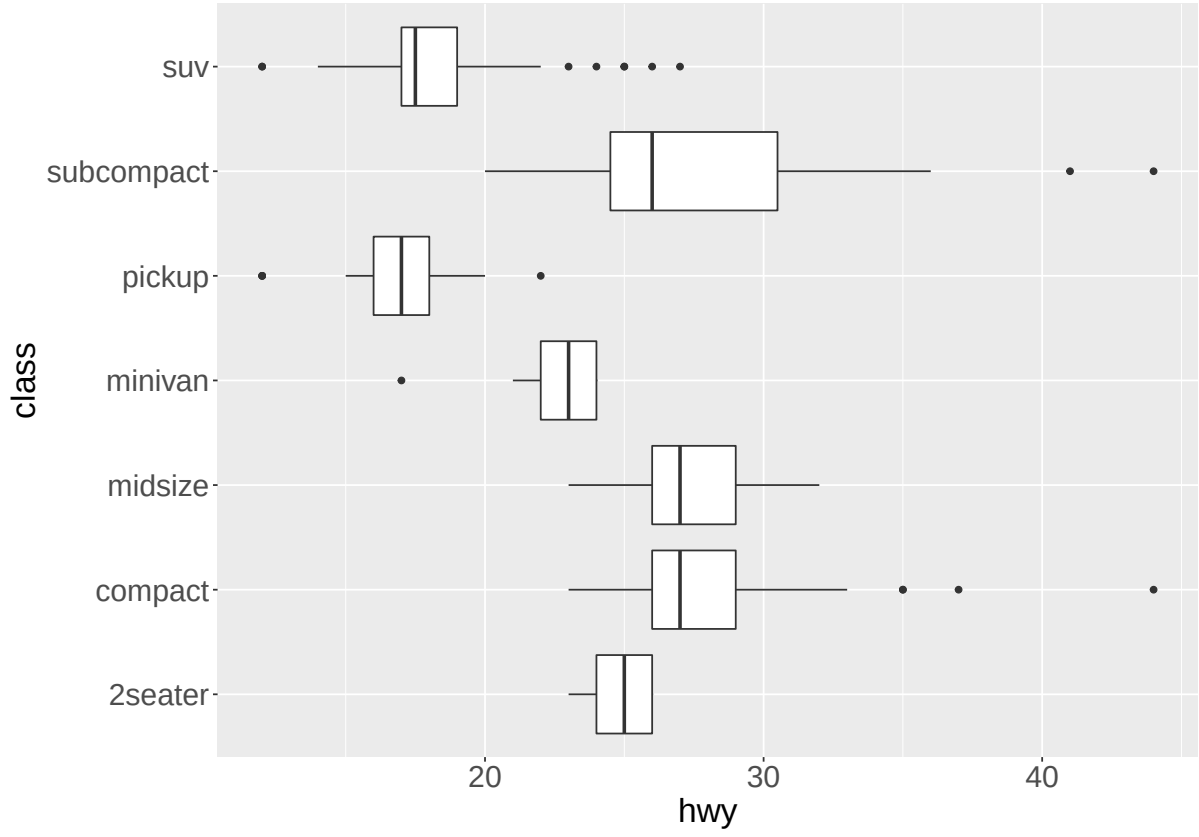
```
ggplot(data = mpg,
       aes(x = class, y = hwy, color = class)) +
  geom_boxplot(show.legend = FALSE)
```



Coordinate systems

`coord_flip()` flips the coordinate system:

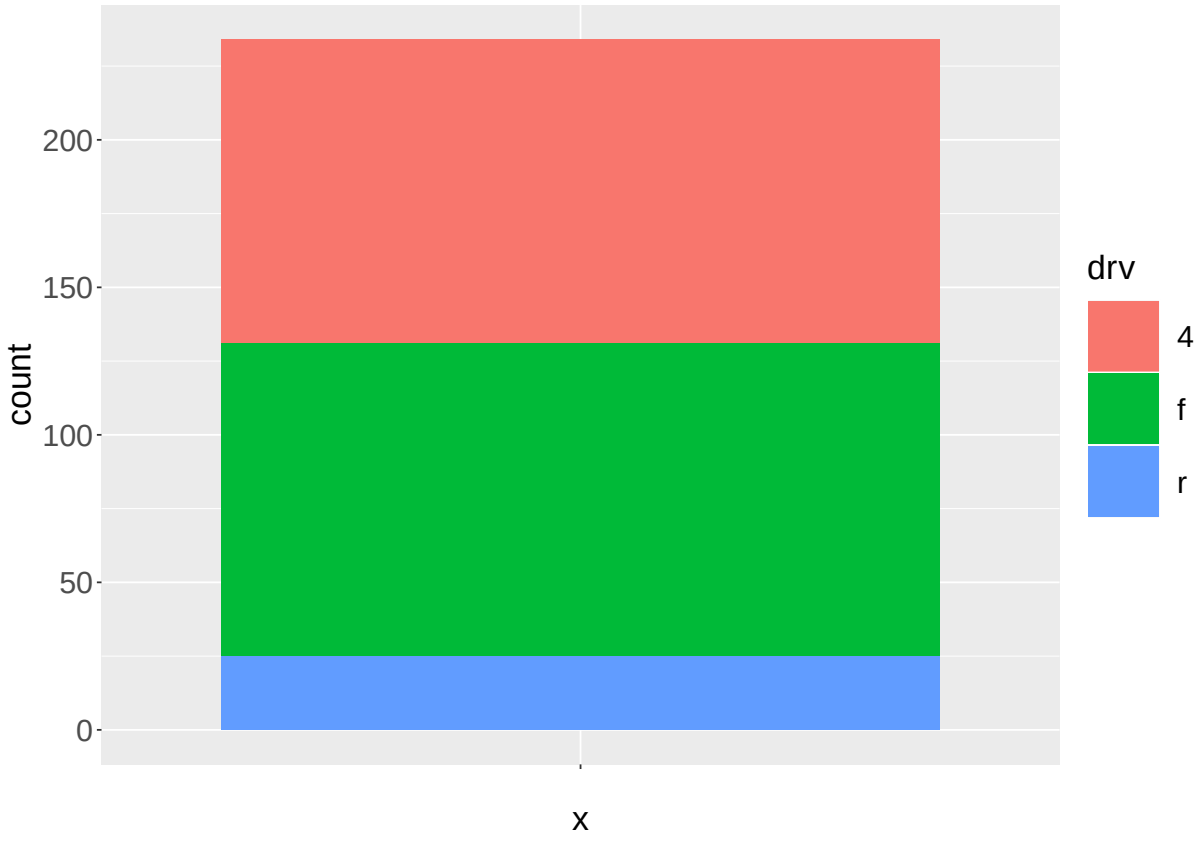
```
ggplot(data = mpg, aes(x = class, y = hwy)) +
  geom_boxplot() +
  coord_flip()
```



Coordinate systems

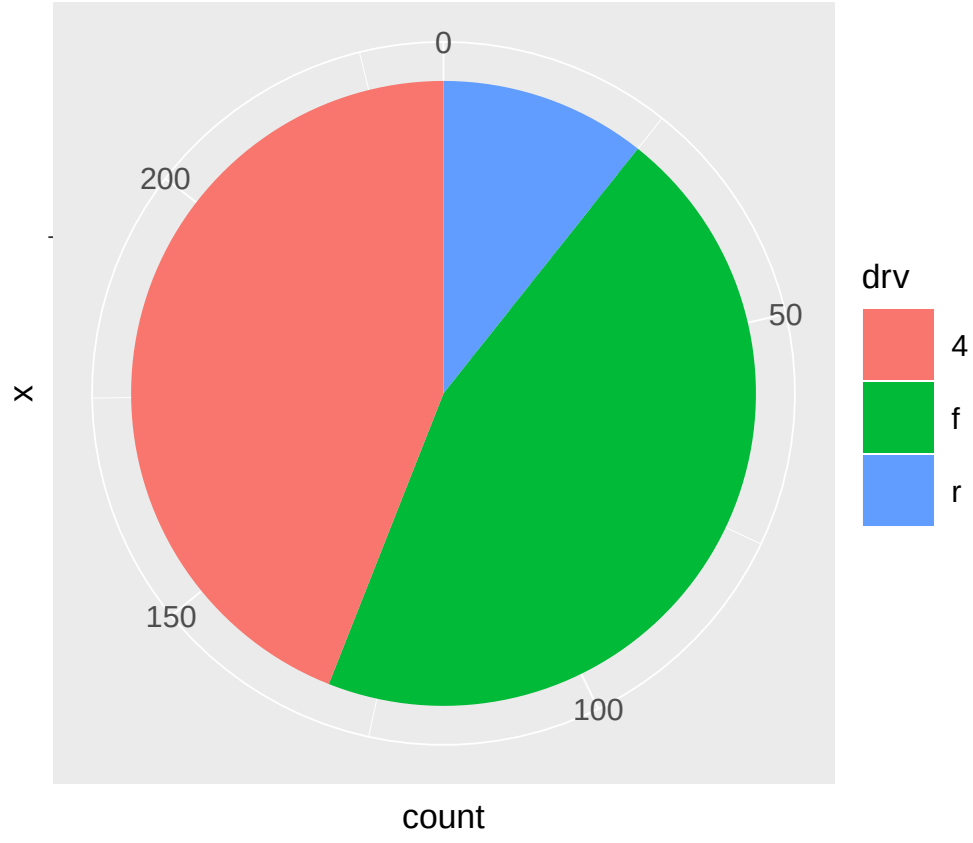
A pie chart is a stacked bar chart with polar coordinates:

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



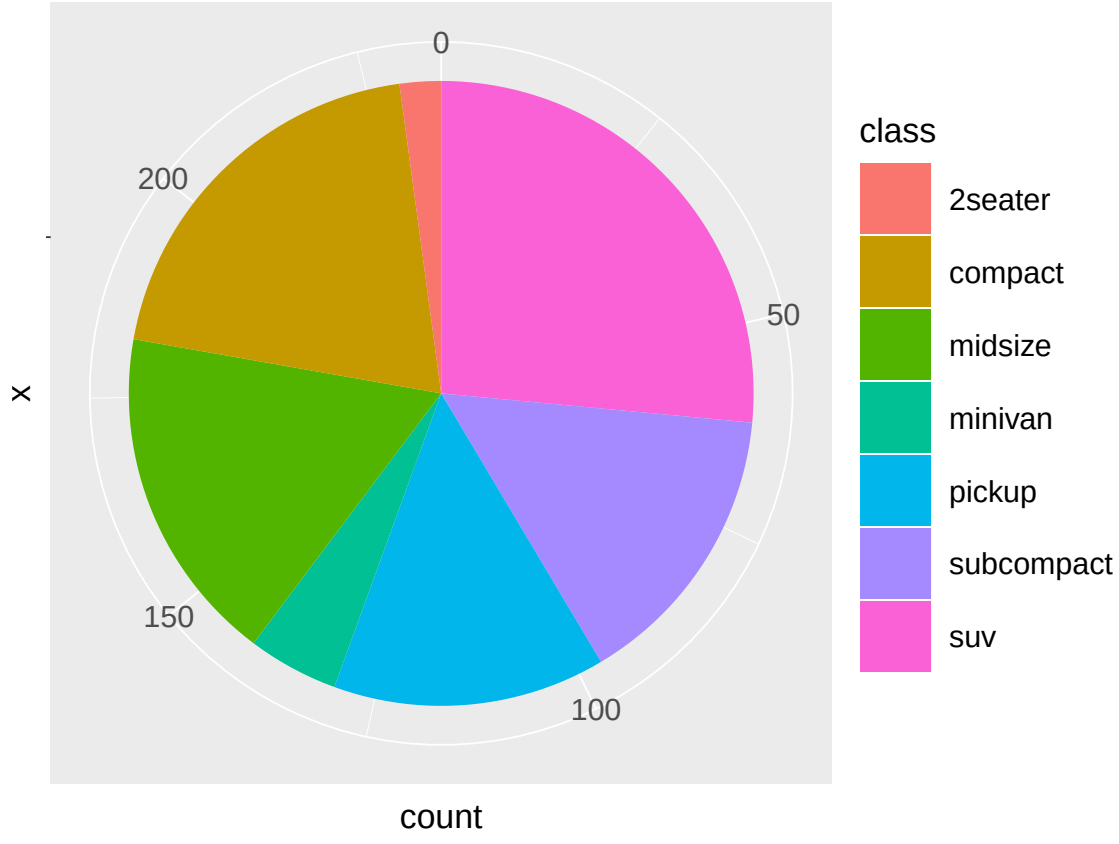
Coordinate systems

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



Coordinate systems

```
ggplot(mpg, aes(x = "", fill = class)) +
  geom_bar() +
  coord_polar("y")
```



The ggplot template

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

Saving plots

Let's build a scatterplot and save it on the working directory:

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point()

ggsave("my-plot.pdf")
```

Saving plots

We can save graphs in other locations by providing the path to `ggsave()`:

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point()

ggsave("outputs/my-plot.pdf")
```

Saving plots

We can use the `plot` argument of `ggsave()` to save any graph that is assigned to an objects:

```
my_plot <- ggplot(mpg, aes(displ, hwy)) +
  geom_point()

ggsave("outputs/my-plot-v2.pdf", plot = my_plot)
```

The default value of the `plot` argument is `last_plot()`.

Bibliography

Wichkam, Hadley, and Garrett Grolemund. 2016. "R for Data Science." *O'Reilly*.

Wickham, Hadley. 2010. "A Layered Grammar of Graphics." *Journal of Computational and Graphical Statistics* 19 (1). Taylor & Francis: 3–28.