r4ds:ggplot2

Hadley Wickham

Table of contents

Welcome

This website us the ggplot visulization sections of Hadley Wickham's book 2nd edition of "R for Data Science".

1 Data visualization

1.1 Introduction

> "The simple graph has brought more information to the data analyst's mind than any other device." — John Tukey

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

This chapter will teach you how to visualize your data using **ggplot2**. We will start by creating a simple scatterplot and use that to introduce aesthetic mappings and geometric objects – the fundamental building blocks of ggplot2. We will then walk you through visualizing distributions of single variables as well as visualizing relationships between two or more variables. We'll finish off with saving your plots and troubleshooting tips.

1.1.1 Prerequisites

This chapter focuses on ggplot2, one of the core packages in the tidyverse. To access the datasets, help pages, and functions used in this chapter, load the tidyverse by running:

```
library(tidyverse)
#> -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
#> v dplyr     1.1.3     v readr     2.1.4
#> v forcats   1.0.0     v stringr   1.5.1
#> v ggplot2   3.4.4     v tibble    3.2.1
#> v lubridate 1.9.3     v tidyr     1.3.0
#> v purrr     1.0.2
#> -- Conflicts ------------------------------------------ tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
#> i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to
```

That one line of code loads the core tidyverse; the packages that you will use in almost every data analysis. It also tells you which functions from the tidyverse conflict with functions in base R (or from other packages you might have loaded)[1].

If you run this code and get the error message `there is no package called 'tidyverse'`, you'll need to first install it, then run `library()` once again.

```
install.packages("tidyverse")
library(tidyverse)
```

You only need to install a package once, but you need to load it every time you start a new session.

In addition to tidyverse, we will also use the **palmerpenguins** package, which includes the `penguins` dataset containing body measurements for penguins on three islands in the Palmer Archipelago, and the ggthemes package, which offers a colorblind safe color palette.

```
library(palmerpenguins)
library(ggthemes)
```

1.2 First steps

Do penguins with longer flippers weigh more or less than penguins with shorter flippers? You probably already have an answer, but try to make your answer precise. What does the relationship between flipper length and body mass look like? Is it positive? Negative? Linear? Nonlinear? Does the relationship vary by the species of the penguin? How about by the island where the penguin lives? Let's create visualizations that we can use to answer these questions.

1.2.1 The `penguins` data frame

You can test your answers to those questions with the `penguins` **data frame** found in palmerpenguins (a.k.a. `palmerpenguins::penguins`). A data frame is a rectangular collection of variables (in the columns) and observations (in the rows). `penguins` contains 344 observations collected and made available by Dr. Kristen Gorman and the Palmer Station, Antarctica LTER[2].

[1]You can eliminate that message and force conflict resolution to happen on demand by using the conflicted package, which becomes more important as you load more packages. You can learn more about conflicted at https://conflicted.r-lib.org.

[2]Horst AM, Hill AP, Gorman KB (2020). palmerpenguins: Palmer Archipelago (Antarctica) penguin data. R package version 0.1.0. https://allisonhorst.github.io/palmerpenguins/. doi: 10.5281/zenodo.3960218.

To make the discussion easier, let's define some terms:

- A **variable** is a quantity, quality, or property that you can measure.
- A **value** is the state of a variable when you measure it. The value of a variable may change from measurement to measurement.
- An **observation** is a set of measurements made under similar conditions (you usually make all of the measurements in an observation at the same time and on the same object). An observation will contain several values, each associated with a different variable. We'll sometimes refer to an observation as a data point.
- **Tabular data** is a set of values, each associated with a variable and an observation. Tabular data is *tidy* if each value is placed in its own "cell", each variable in its own column, and each observation in its own row.

In this context, a variable refers to an attribute of all the penguins, and an observation refers to all the attributes of a single penguin.

Type the name of the data frame in the console and R will print a preview of its contents. Note that it says `tibble` on top of this preview. In the tidyverse, we use special data frames called **tibbles** that you will learn more about soon.

```
penguins
#> # A tibble: 344 x 8
#>    species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
#>    <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
#>  1 Adelie  Torgersen           39.1          18.7               181        3750
#>  2 Adelie  Torgersen           39.5          17.4               186        3800
#>  3 Adelie  Torgersen           40.3          18                 195        3250
#>  4 Adelie  Torgersen           NA            NA                  NA          NA
#>  5 Adelie  Torgersen           36.7          19.3               193        3450
#>  6 Adelie  Torgersen           39.3          20.6               190        3650
#>  7 Adelie  Torgersen           38.9          17.8               181        3625
#>  8 Adelie  Torgersen           39.2          19.6               195        4675
#>  9 Adelie  Torgersen           34.1          18.1               193        3475
#> 10 Adelie  Torgersen           42            20.2               190        4250
#> # i 334 more rows
#> # i 2 more variables: sex <fct>, year <int>
```

This data frame contains 8 columns. For an alternative view, where you can see all variables and the first few observations of each variable, use `glimpse()`. Or, if you're in RStudio, run `View(penguins)` to open an interactive data viewer.

```
glimpse(penguins)
#> Rows: 344
#> Columns: 8
#> $ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel~
#> $ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse~
#> $ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~
#> $ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~
#> $ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~
#> $ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~
#> $ sex               <fct> male, female, female, NA, female, male, female, male~
#> $ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

Among the variables in `penguins` are:

1. `species`: a penguin's species (Adelie, Chinstrap, or Gentoo).
2. `flipper_length_mm`: length of a penguin's flipper, in millimeters.
3. `body_mass_g`: body mass of a penguin, in grams.

To learn more about `penguins`, open its help page by running `?penguins`.

1.2.2 Ultimate goal

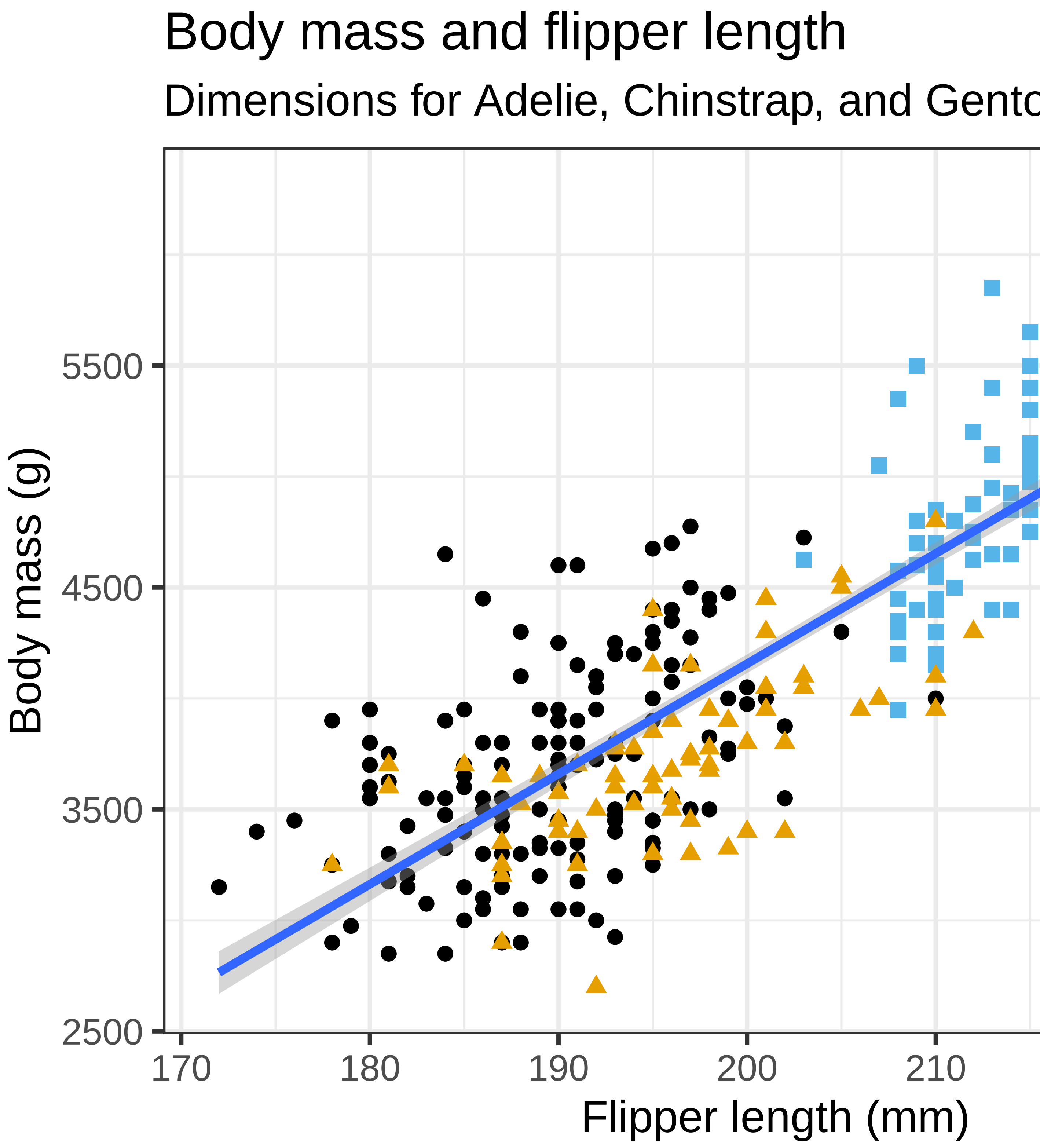
Our ultimate goal in this chapter is to recreate the following visualization displaying the relationship between flipper lengths and body masses of these penguins, taking into consideration the species of the penguin.

1.2.3 Creating a ggplot

Let's recreate this plot step-by-step.

With ggplot2, you begin a plot with the function `ggplot()`, defining a plot object that you then add **layers** to. The first argument of `ggplot()` is the dataset to use in the graph and so `ggplot(data = penguins)` creates an empty graph that is primed to display the `penguins` data, but since we haven't told it how to visualize it yet, for now it's empty. This is not a very exciting plot, but you can think of it like an empty canvas you'll paint the remaining layers of your plot onto.

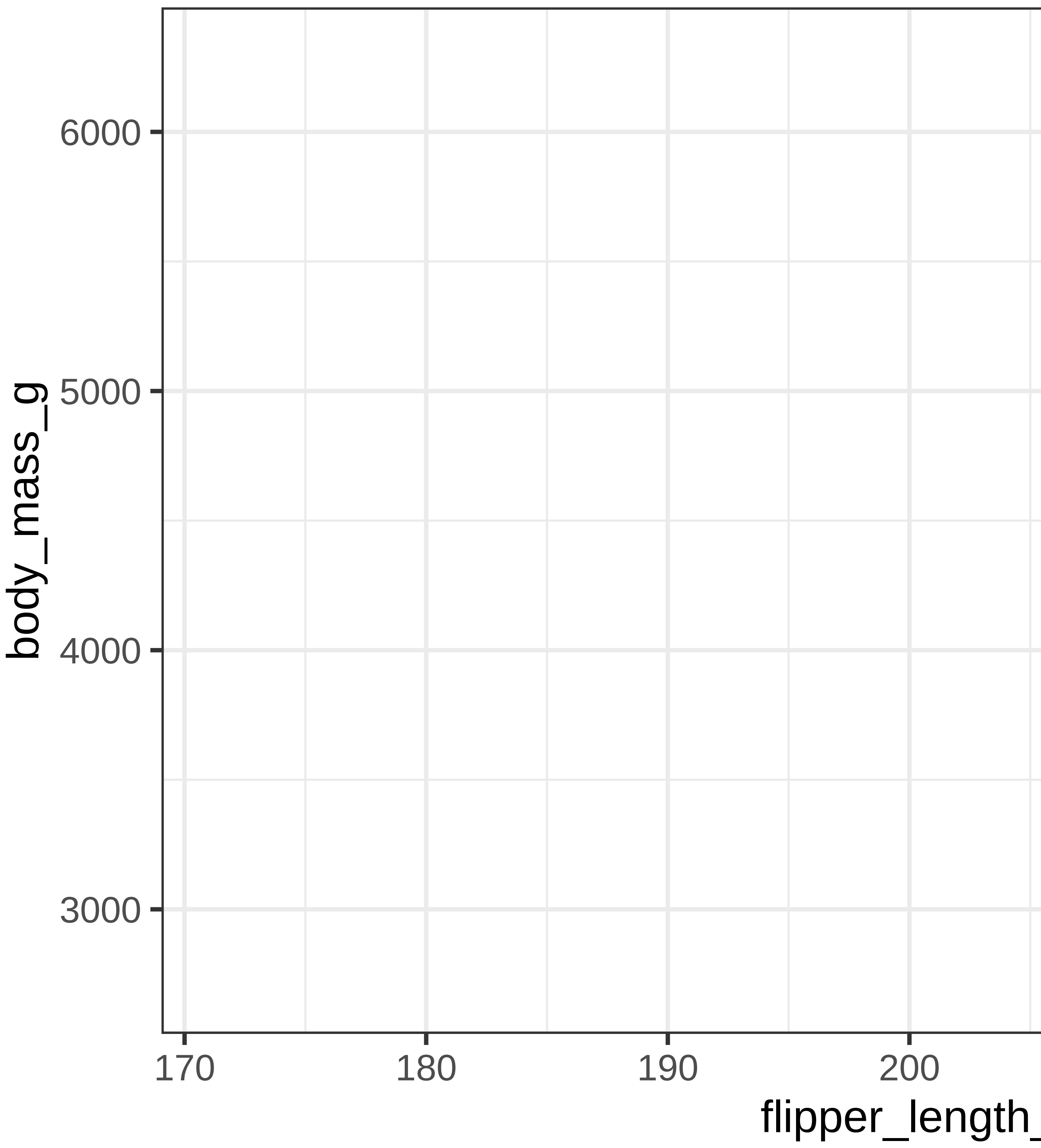
```
ggplot(data = penguins)
```


Body mass and flipper length
Dimensions for Adelie, Chinstrap, and Gento
Body mass (g)
5500
4500
3500
2500
170
180
190
200
210
Flipper length (mm)

Next, we need to tell `ggplot()` how the information from our data will be visually represented. The `mapping` argument of the `ggplot()` function defines how variables in your dataset are mapped to visual properties (**aesthetics**) of your plot. The `mapping` argument is always defined in the `aes()` function, and the `x` and `y` arguments of `aes()` specify which variables to map to the x and y axes. For now, we will only map flipper length to the `x` aesthetic and body mass to the `y` aesthetic. ggplot2 looks for the mapped variables in the `data` argument, in this case, `penguins`.

The following plot shows the result of adding these mappings.

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
)
```

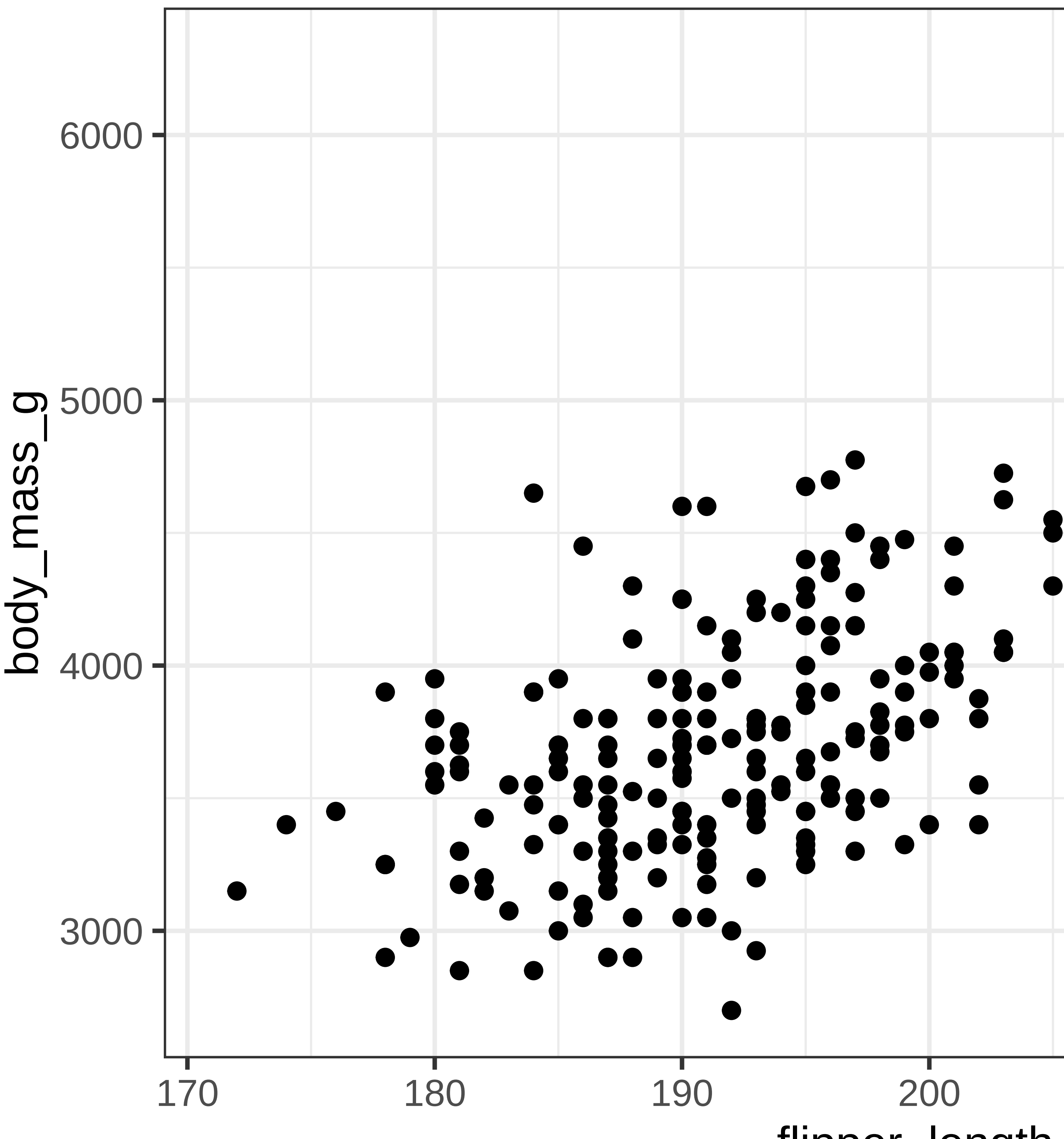

6000
5000
4000
3000
body_mass_g
170
180
190
200
flipper_length_

Our empty canvas now has more structure – it's clear where flipper lengths will be displayed (on the x-axis) and where body masses will be displayed (on the y-axis). But the penguins themselves are not yet on the plot. This is because we have not yet articulated, in our code, how to represent the observations from our data frame on our plot.

To do so, we need to define a **geom**: the geometrical object that a plot uses to represent data. These geometric objects are made available in ggplot2 with functions that start with `geom_`. People often describe plots by the type of geom that the plot uses. For example, bar charts use bar geoms (`geom_bar()`), line charts use line geoms (`geom_line()`), boxplots use boxplot geoms (`geom_boxplot()`), scatterplots use point geoms (`geom_point()`), and so on.

The function `geom_point()` adds a layer of points to your plot, which creates a scatterplot. ggplot2 comes with many geom functions that each adds a different type of layer to a plot. You'll learn a whole bunch of geoms throughout the book, particularly in Chapter 2.

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
) +
  geom_point()
#> Warning: Removed 2 rows containing missing values (`geom_point()`).
```


6000
5000
4000
3000
body_mass_g
170
180
190
200
flipper_length_

Now we have something that looks like what we might think of as a "scatterplot". It doesn't yet match our "ultimate goal" plot, but using this plot we can start answering the question that motivated our exploration: "What does the relationship between flipper length and body mass look like?" The relationship appears to be positive (as flipper length increases, so does body mass), fairly linear (the points are clustered around a line instead of a curve), and moderately strong (there isn't too much scatter around such a line). Penguins with longer flippers are generally larger in terms of their body mass.

Before we add more layers to this plot, let's pause for a moment and review the warning message we got:

> Removed 2 rows containing missing values (`geom_point()`).

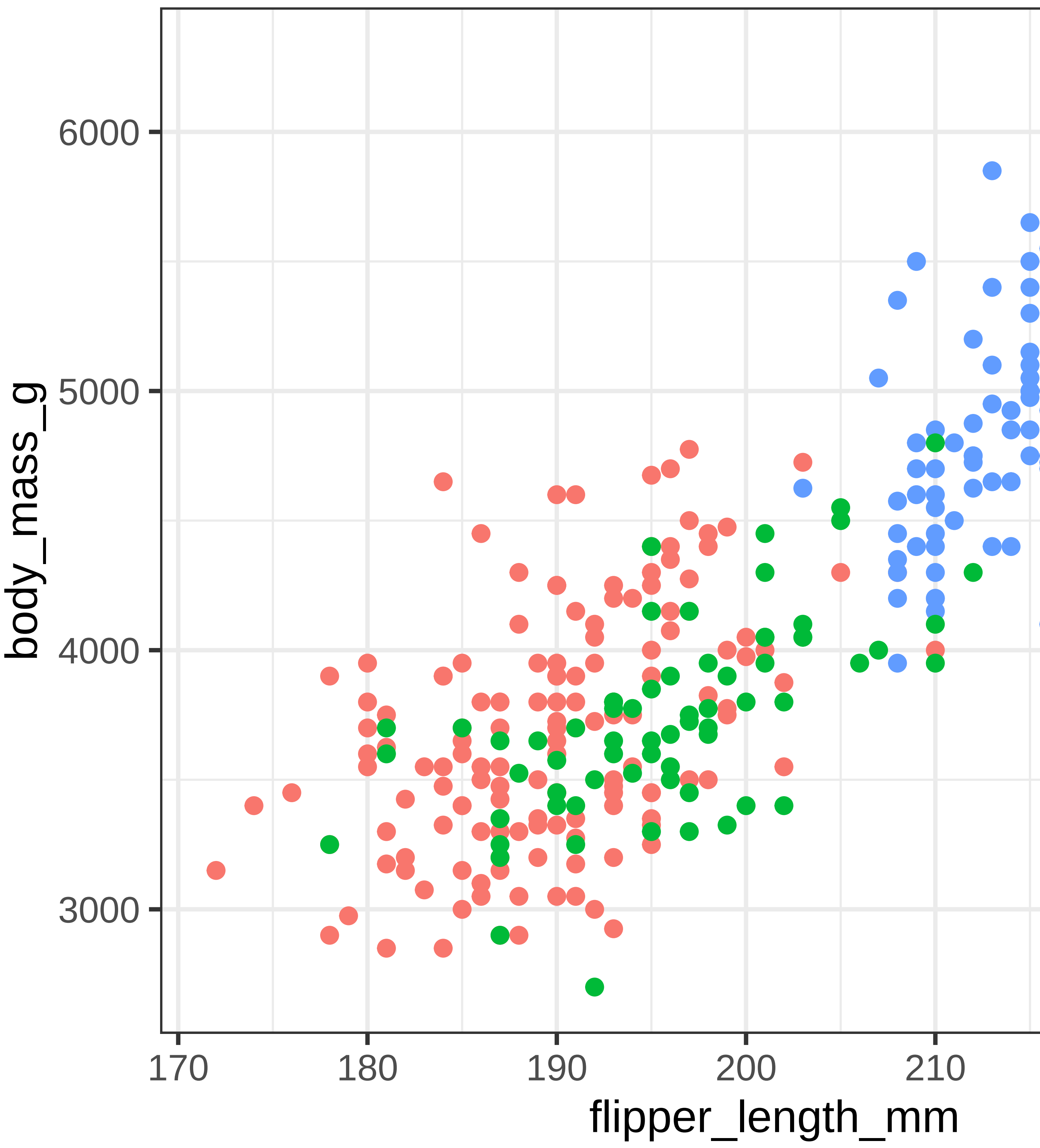
We're seeing this message because there are two penguins in our dataset with missing body mass and/or flipper length values and ggplot2 has no way of representing them on the plot without both of these values. Like R, ggplot2 subscribes to the philosophy that missing values should never silently go missing. This type of warning is probably one of the most common types of warnings you will see when working with real data – missing values are a very common issue and you'll learn more about them throughout the book, particularly in **?@sec-missing-values**. For the remaining plots in this chapter we will suppress this warning so it's not printed alongside every single plot we make.

1.2.4 Adding aesthetics and layers

Scatterplots are useful for displaying the relationship between two numerical variables, but it's always a good idea to be skeptical of any apparent relationship between two variables and ask if there may be other variables that explain or change the nature of this apparent relationship. For example, does the relationship between flipper length and body mass differ by species? Let's incorporate species into our plot and see if this reveals any additional insights into the apparent relationship between these variables. We will do this by representing species with different colored points.

To achieve this, will we need to modify the aesthetic or the geom? If you guessed "in the aesthetic mapping, inside of `aes()`", you're already getting the hang of creating data visualizations with ggplot2! And if not, don't worry. Throughout the book you will make many more ggplots and have many more opportunities to check your intuition as you make them.

```r
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g, color = species)
) +
  geom_point()
```

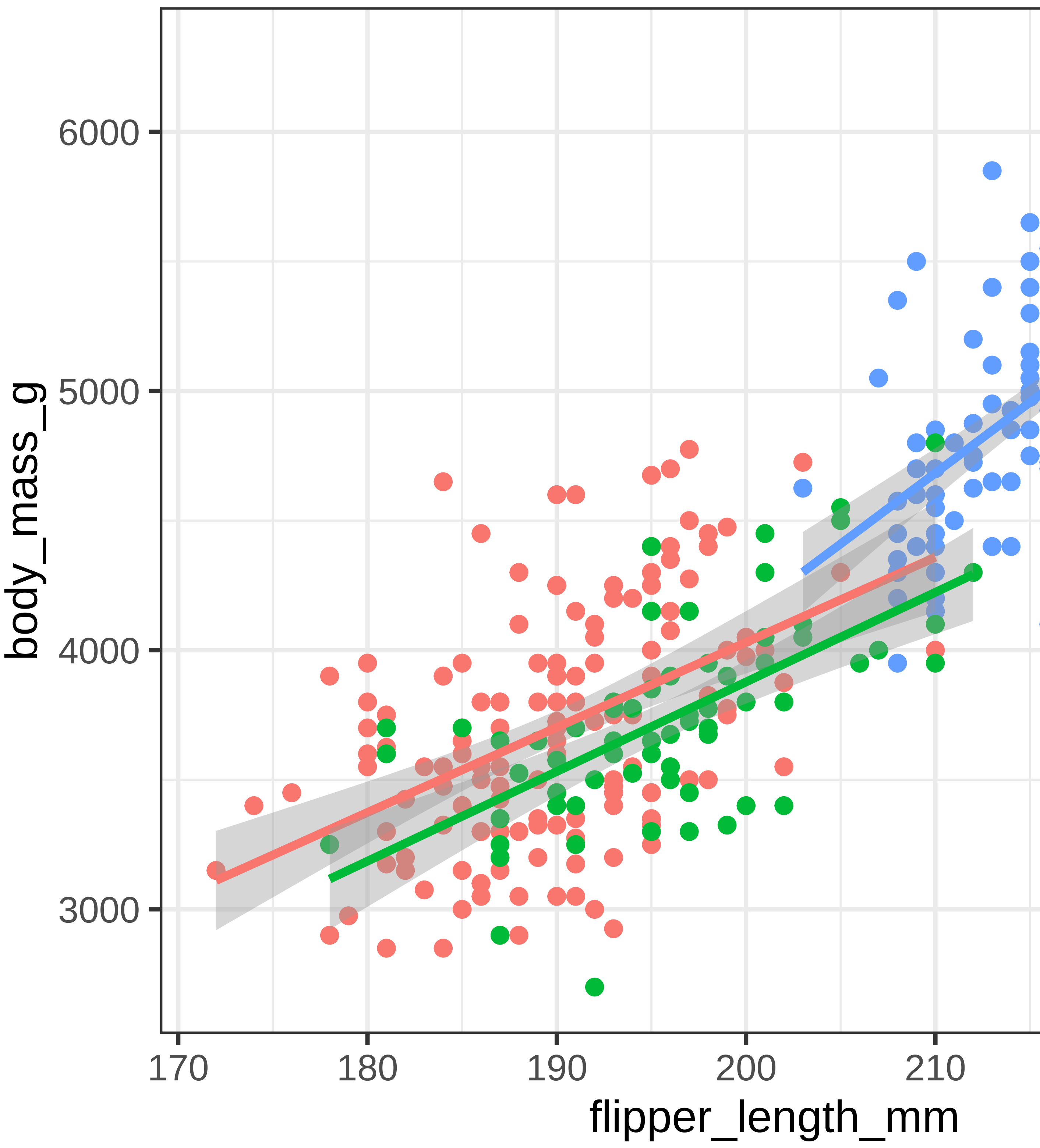

6000
5000
4000
3000
body_mass_g
170
180
190
200
210
flipper_length_mm

When a categorical variable is mapped to an aesthetic, ggplot2 will automatically assign a unique value of the aesthetic (here a unique color) to each unique level of the variable (each of the three species), a process known as **scaling**. ggplot2 will also add a legend that explains which values correspond to which levels.

Now let's add one more layer: a smooth curve displaying the relationship between body mass and flipper length. Before you proceed, refer back to the code above, and think about how we can add this to our existing plot.

Since this is a new geometric object representing our data, we will add a new geom as a layer on top of our point geom: `geom_smooth()`. And we will specify that we want to draw the line of best fit based on a `linear` model with `method = "lm"`.

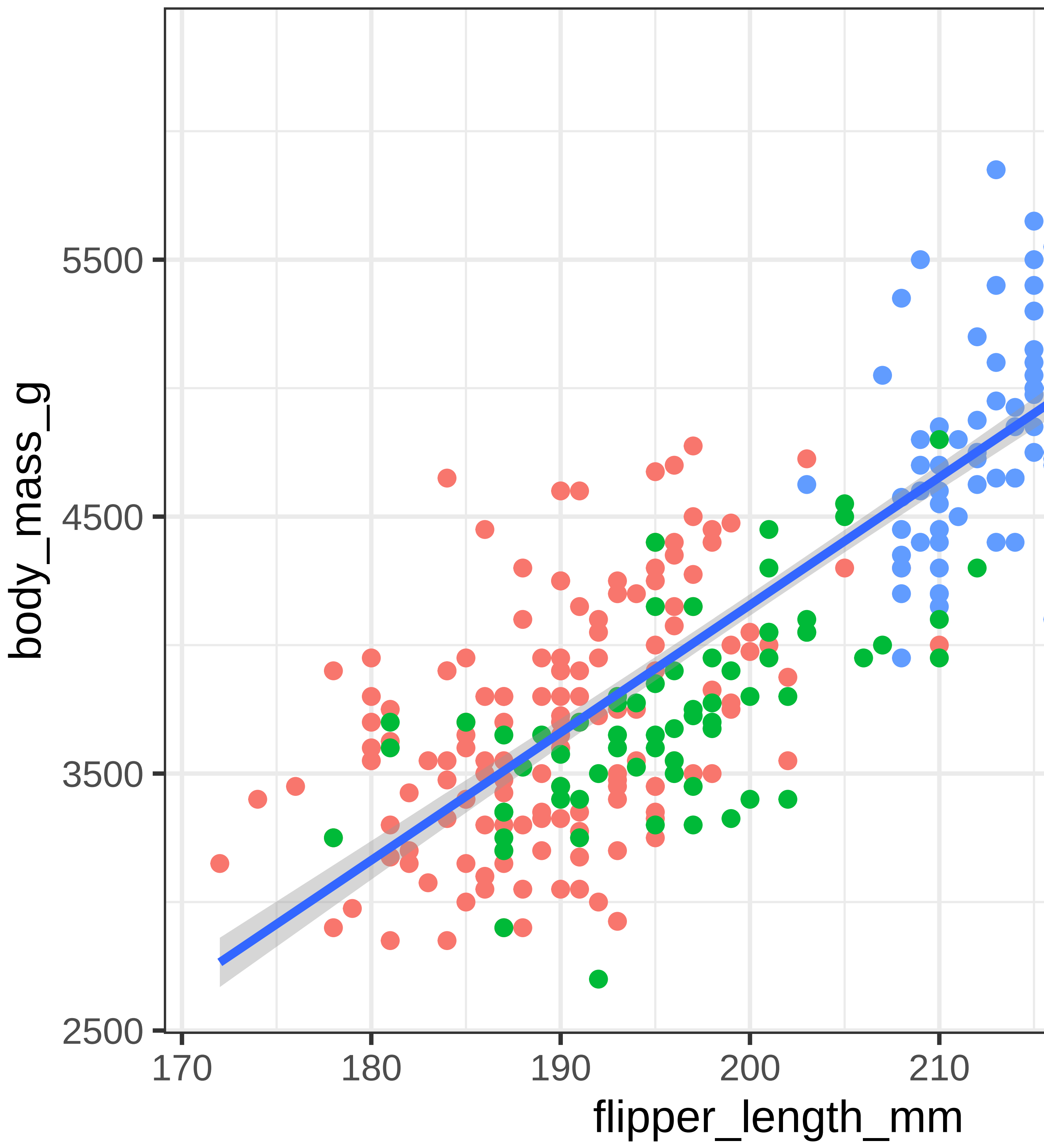
```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g, color = species)
) +
  geom_point() +
  geom_smooth(method = "lm")
```


body_mass_g
6000
5000
4000
3000
170
180
190
200
210
flipper_length_mm

We have successfully added lines, but this plot doesn't look like the plot from Section 1.2.2, which only has one line for the entire dataset as opposed to separate lines for each of the penguin species.

When aesthetic mappings are defined in `ggplot()`, at the *global* level, they're passed down to each of the subsequent geom layers of the plot. However, each geom function in ggplot2 can also take a `mapping` argument, which allows for aesthetic mappings at the *local* level that are added to those inherited from the global level. Since we want points to be colored based on species but don't want the lines to be separated out for them, we should specify `color = species` for `geom_point()` only.

```r
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
) +
  geom_point(mapping = aes(color = species)) +
  geom_smooth(method = "lm")
```

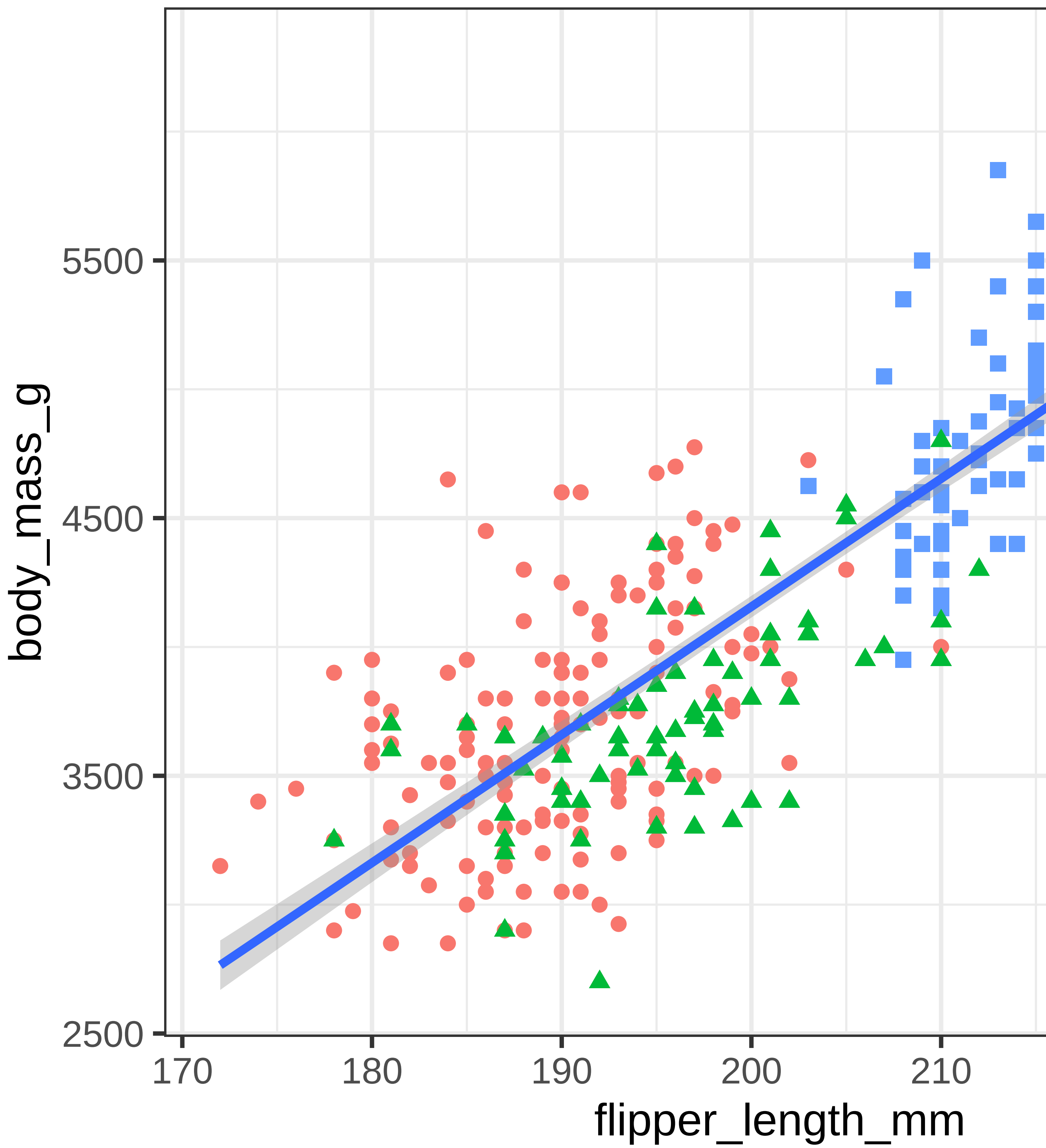


5500
4500
3500
2500
body_mass_g
170
180
190
200
210
flipper_length_mm

Voila! We have something that looks very much like our ultimate goal, though it's not yet perfect. We still need to use different shapes for each species of penguins and improve labels.

It's generally not a good idea to represent information using only colors on a plot, as people perceive colors differently due to color blindness or other color vision differences. Therefore, in addition to color, we can also map `species` to the `shape` aesthetic.

```r
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
) +
  geom_point(mapping = aes(color = species, shape = species)) +
  geom_smooth(method = "lm")
```

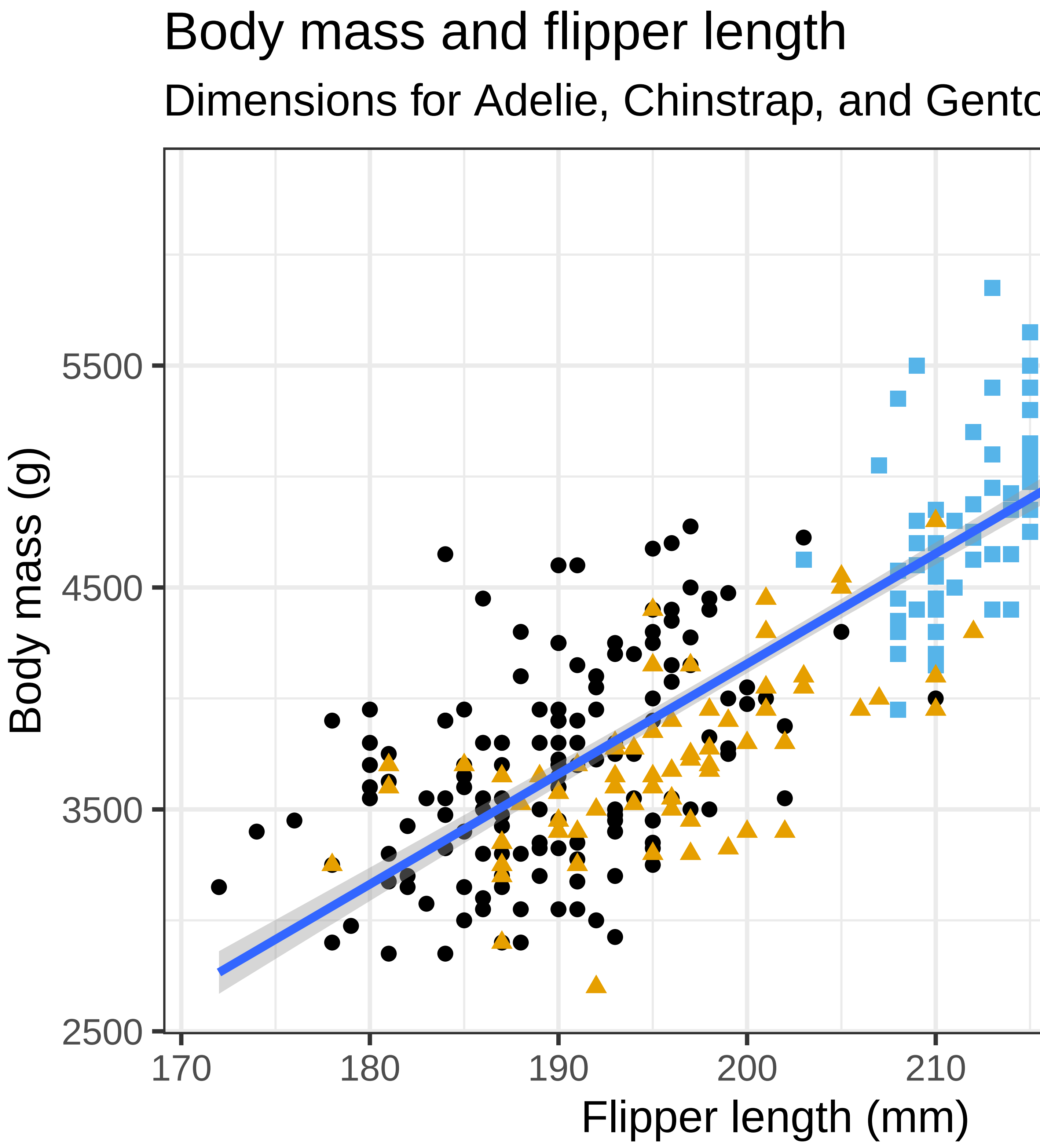


body_mass_g
5500
4500
3500
2500
170
180
190
200
210
flipper_length_mm

Note that the legend is automatically updated to reflect the different shapes of the points as well.

And finally, we can improve the labels of our plot using the `labs()` function in a new layer. Some of the arguments to `labs()` might be self explanatory: `title` adds a title and `subtitle` adds a subtitle to the plot. Other arguments match the aesthetic mappings, `x` is the x-axis label, `y` is the y-axis label, and `color` and `shape` define the label for the legend. In addition, we can improve the color palette to be colorblind safe with the `scale_color_colorblind()` function from the ggthemes package.

```r
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
) +
  geom_point(aes(color = species, shape = species)) +
  geom_smooth(method = "lm") +
  labs(
    title = "Body mass and flipper length",
    subtitle = "Dimensions for Adelie, Chinstrap, and Gentoo Penguins",
    x = "Flipper length (mm)", y = "Body mass (g)",
    color = "Species", shape = "Species"
  ) +
  scale_color_colorblind()
```


Body mass and flipper length
Dimensions for Adelie, Chinstrap, and Gento
Body mass (g)
5500
4500
3500
2500
170
180
190
200
210
Flipper length (mm)

We finally have a plot that perfectly matches our "ultimate goal"!

1.2.5 Exercises

1. How many rows are in `penguins`? How many columns?
2. What does the `bill_depth_mm` variable in the `penguins` data frame describe? Read the help for `?penguins` to find out.
3. Make a scatterplot of `bill_depth_mm` vs. `bill_length_mm`. That is, make a scatterplot with `bill_depth_mm` on the y-axis and `bill_length_mm` on the x-axis. Describe the relationship between these two variables.
4. What happens if you make a scatterplot of `species` vs. `bill_depth_mm`? What might be a better choice of geom?
5. Why does the following give an error and how would you fix it?

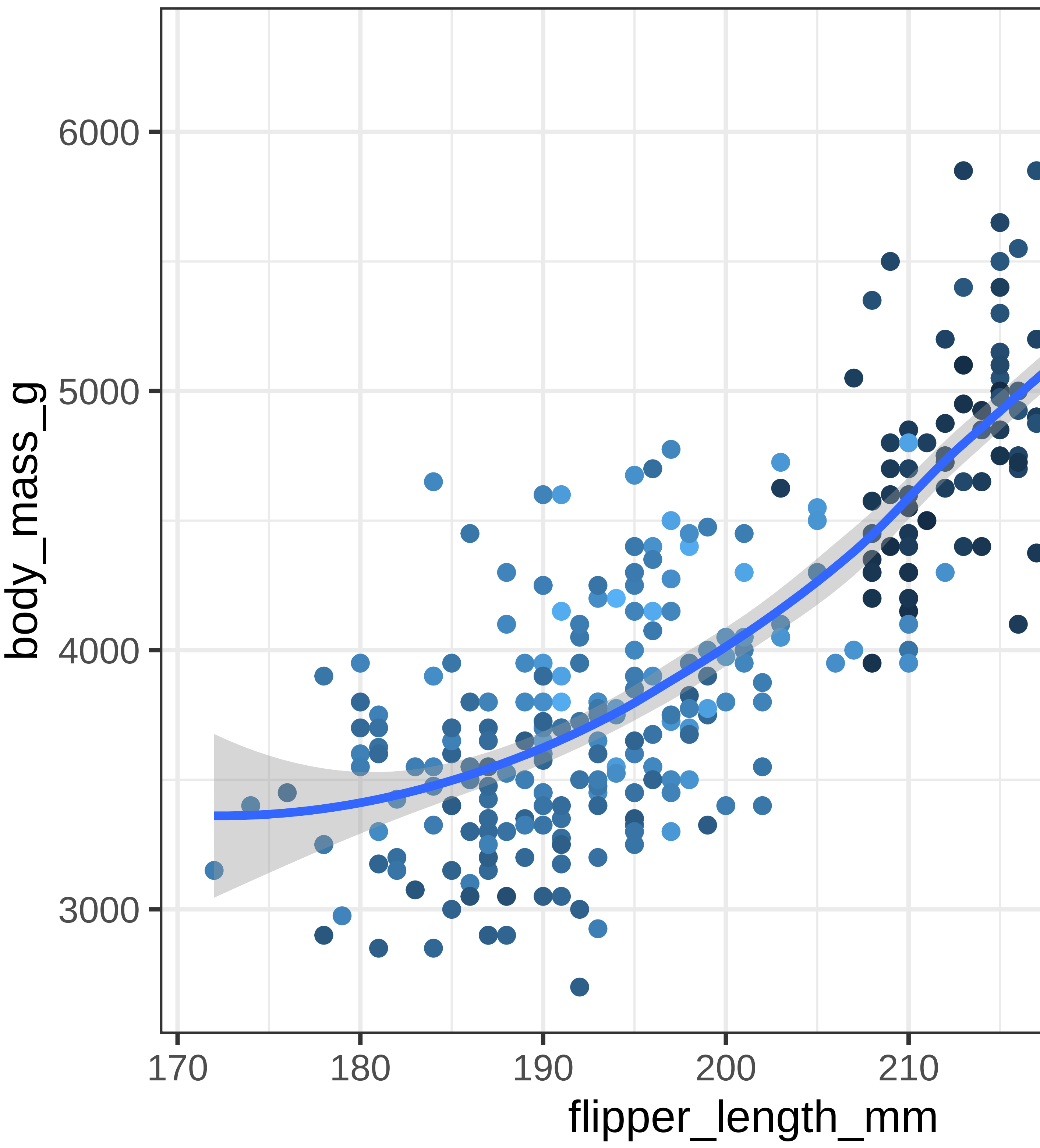
   ```
   ggplot(data = penguins) +
     geom_point()
   ```

6. What does the `na.rm` argument do in `geom_point()`? What is the default value of the argument? Create a scatterplot where you successfully use this argument set to `TRUE`.
7. Add the following caption to the plot you made in the previous exercise: "Data come from the palmerpenguins package." Hint: Take a look at the documentation for `labs()`.
8. Recreate the following visualization. What aesthetic should `bill_depth_mm` be mapped to? And should it be mapped at the global level or at the geom level?
9. 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 = penguins,
     mapping = aes(x = flipper_length_mm, y = body_mass_g, color = island)
   ) +
     geom_point() +
     geom_smooth(se = FALSE)
   ```

10. Will these two graphs look different? Why/why not?

    ```
    ggplot(
      data = penguins,
      mapping = aes(x = flipper_length_mm, y = body_mass_g)
    ```


6000
5000
4000
3000
body_mass_g
170
180
190
200
210
flipper_length_mm

```
) +
  geom_point() +
  geom_smooth()

ggplot() +
  geom_point(
    data = penguins,
    mapping = aes(x = flipper_length_mm, y = body_mass_g)
  ) +
  geom_smooth(
    data = penguins,
    mapping = aes(x = flipper_length_mm, y = body_mass_g)
  )
```

1.3 ggplot2 calls

As we move on from these introductory sections, we'll transition to a more concise expression of ggplot2 code. So far we've been very explicit, which is helpful when you are learning:

```
ggplot(
  data = penguins,
  mapping = aes(x = flipper_length_mm, y = body_mass_g)
) +
  geom_point()
```

Typically, the first one or two arguments to a function are so important that you should know them by heart. The first two arguments to `ggplot()` are `data` and `mapping`, in the remainder of the book, we won't supply those names. That saves typing, and, by reducing the amount of extra text, makes it easier to see what's different between plots. That's a really important programming concern that we'll come back to in **?@sec-functions**.

Rewriting the previous plot more concisely yields:

```
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point()
```

In the future, you'll also learn about the pipe, `|>`, which will allow you to create that plot with:

```r
penguins |>
  ggplot(aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point()
```

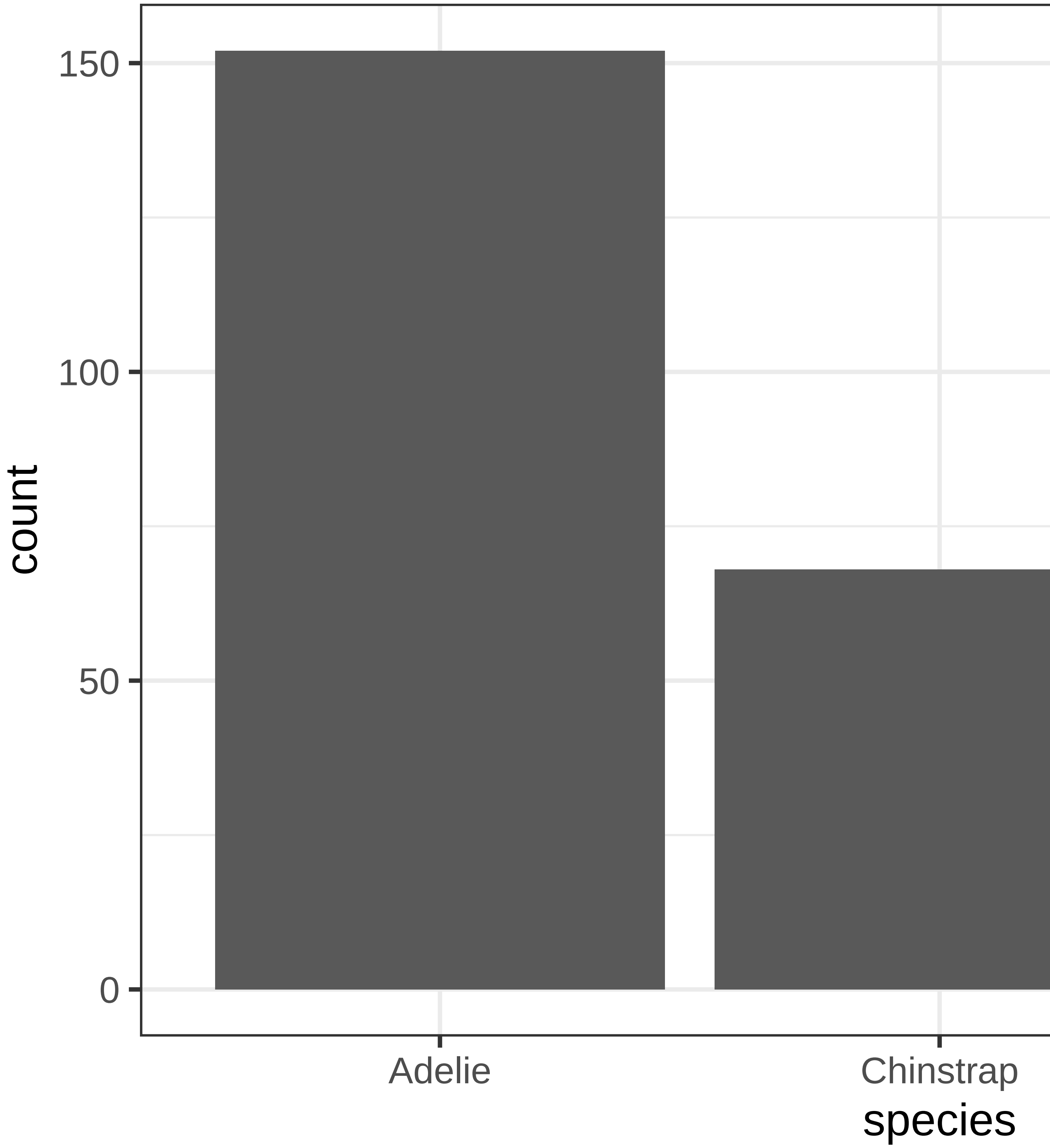
1.4 Visualizing distributions

How you visualize the distribution of a variable depends on the type of variable: categorical or numerical.

1.4.1 A categorical variable

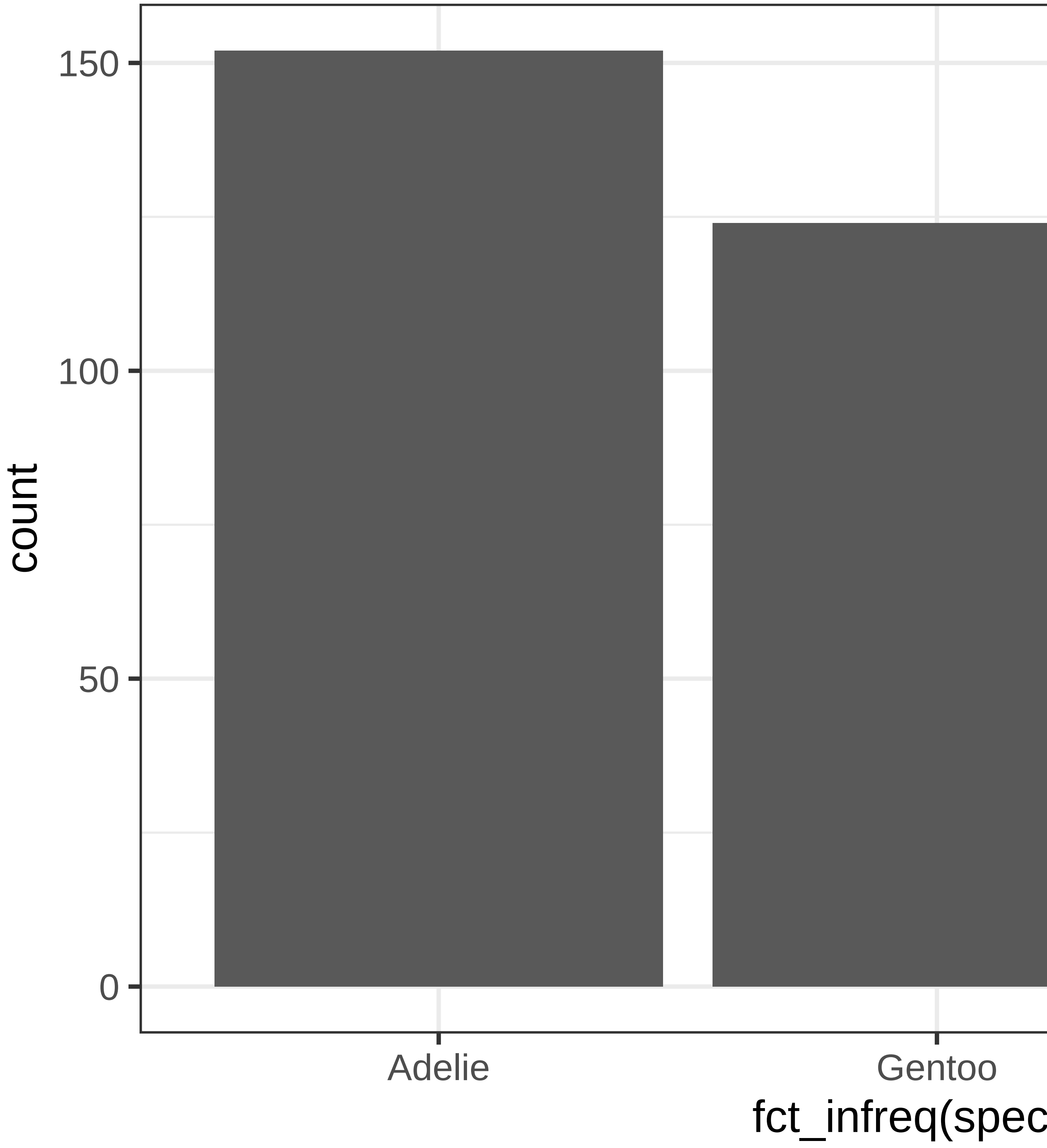
A variable is **categorical** if it can only take one of a small set of values. To examine the distribution of a categorical variable, you can use a bar chart. The height of the bars displays how many observations occurred with each `x` value.

```r
ggplot(penguins, aes(x = species)) +
  geom_bar()
```


150
100
50
0
count
Adelie
Chinstrap
species

In bar plots of categorical variables with non-ordered levels, like the penguin `species` above, it's often preferable to reorder the bars based on their frequencies. Doing so requires transforming the variable to a factor (how R handles categorical data) and then reordering the levels of that factor.

```r
ggplot(penguins, aes(x = fct_infreq(species))) +
  geom_bar()
```


150
100
50
0
count
Adelie
Gentoo
fct_infreq(spec

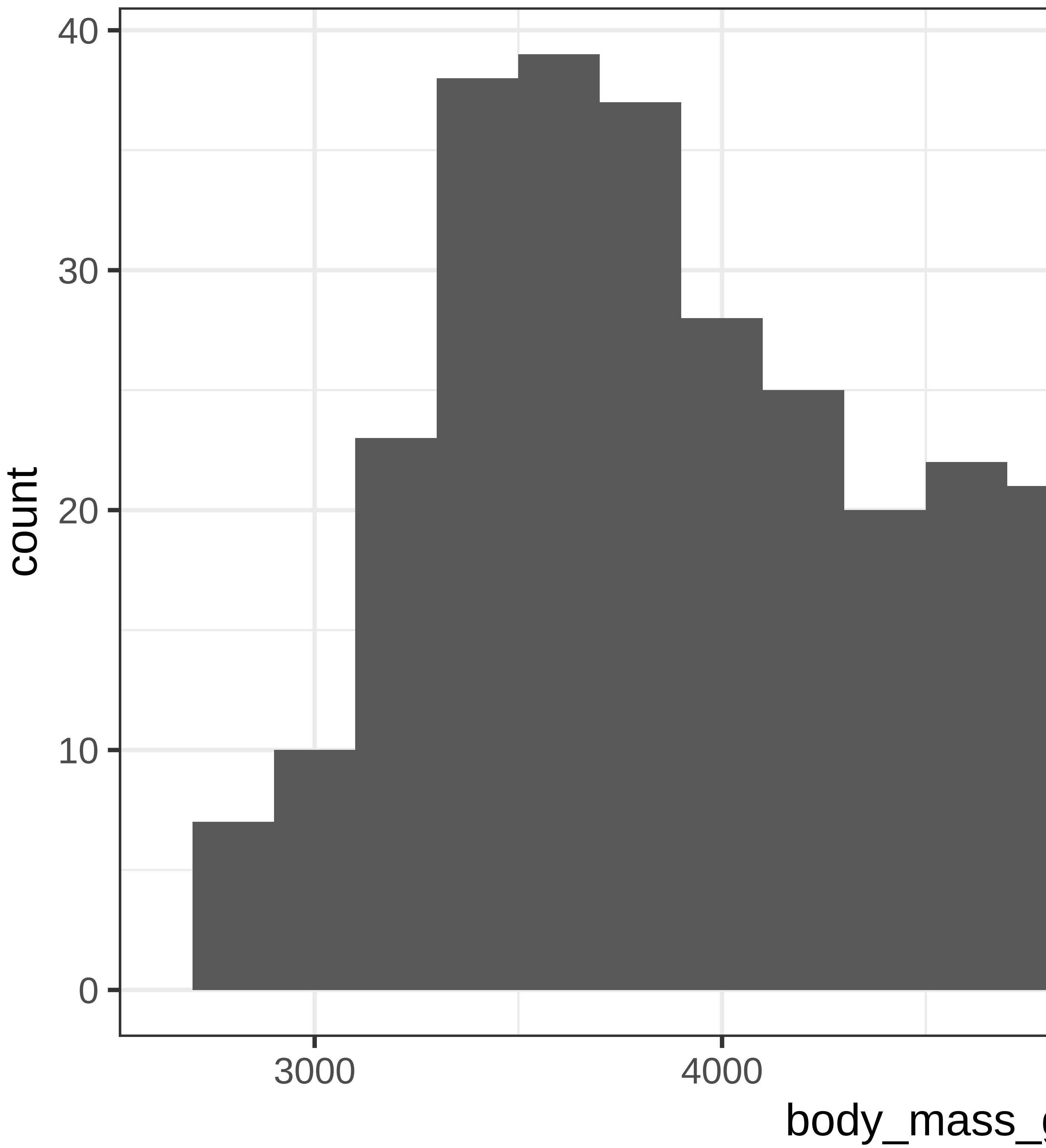
You will learn more about factors and functions for dealing with factors (like `fct_infreq()` shown above) in **?@sec-factors**.

1.4.2 A numerical variable

A variable is **numerical** (or quantitative) if it can take on a wide range of numerical values, and it is sensible to add, subtract, or take averages with those values. Numerical variables can be continuous or discrete.

One commonly used visualization for distributions of continuous variables is a histogram.

```
ggplot(penguins, aes(x = body_mass_g)) +
  geom_histogram(binwidth = 200)
```


40
30
20
10
0
count
3000
4000
body_mass_g

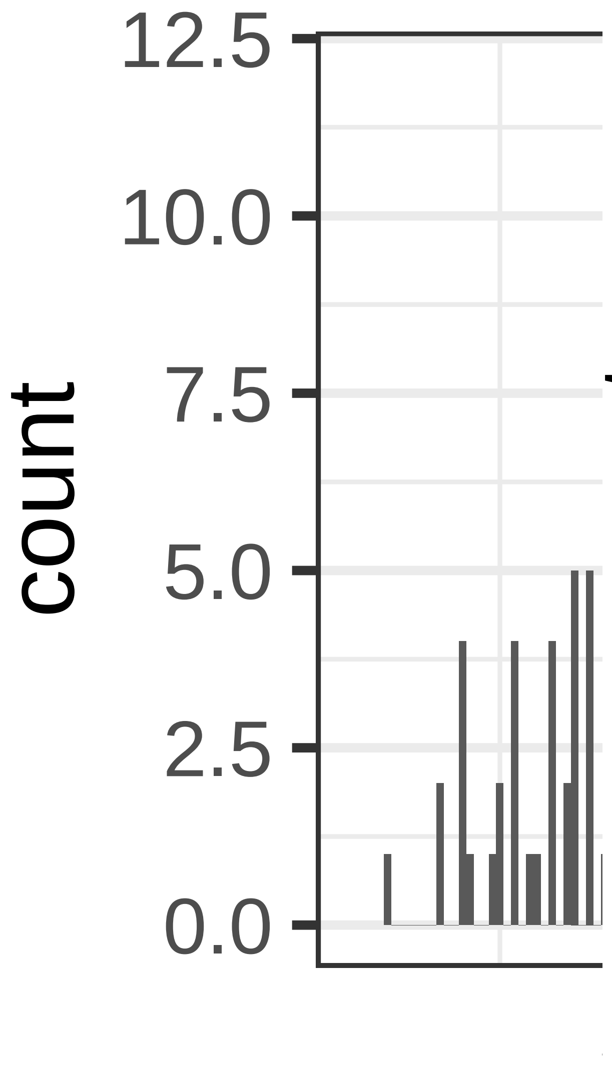
A histogram divides the x-axis into equally spaced bins and then uses the height of a bar to display the number of observations that fall in each bin. In the graph above, the tallest bar shows that 39 observations have a `body_mass_g` value between 3,500 and 3,700 grams, which are the left and right edges of the bar.

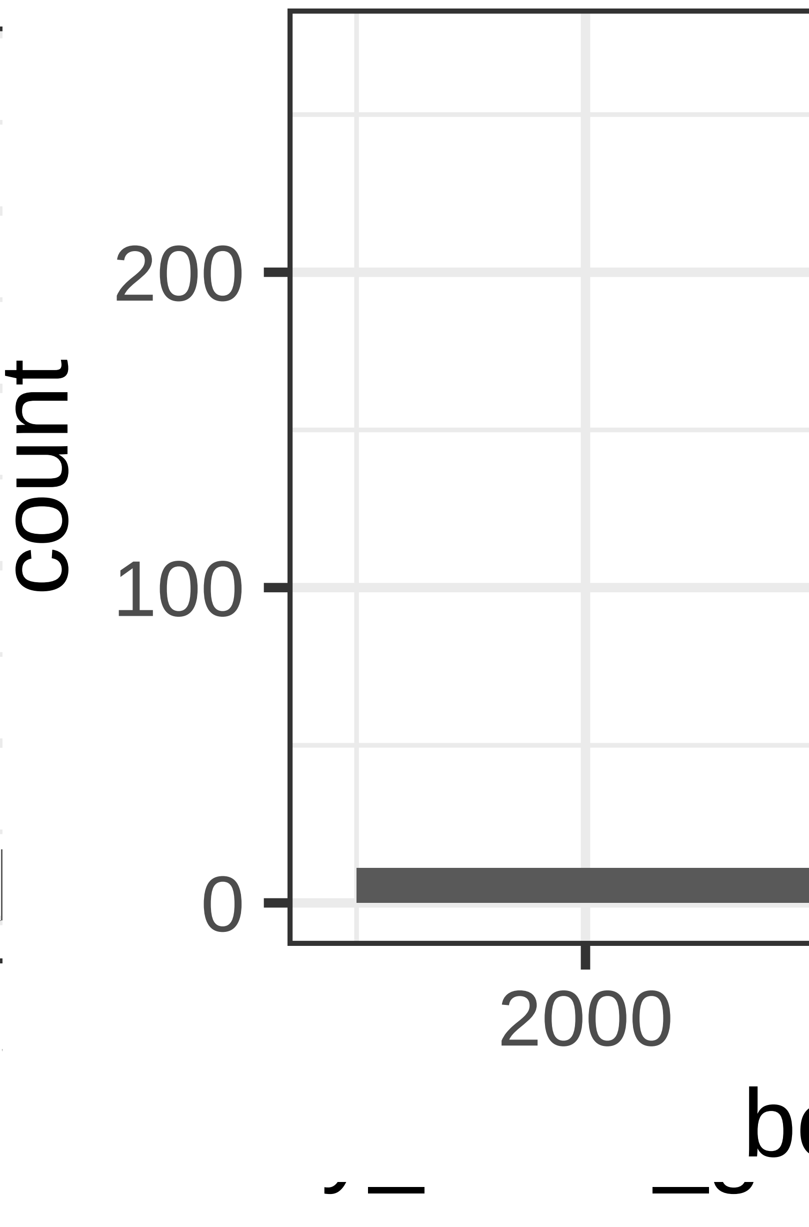
You can set the width of the intervals in a histogram with the binwidth argument, which is measured in the units of the `x` variable. You should always explore a variety of binwidths when working with histograms, as different binwidths can reveal different patterns. In the plots below a binwidth of 20 is too narrow, resulting in too many bars, making it difficult to determine the shape of the distribution. Similarly, a binwidth of 2,000 is too high, resulting in all data being binned into only three bars, and also making it difficult to determine the shape of the distribution. A binwidth of 200 provides a sensible balance.

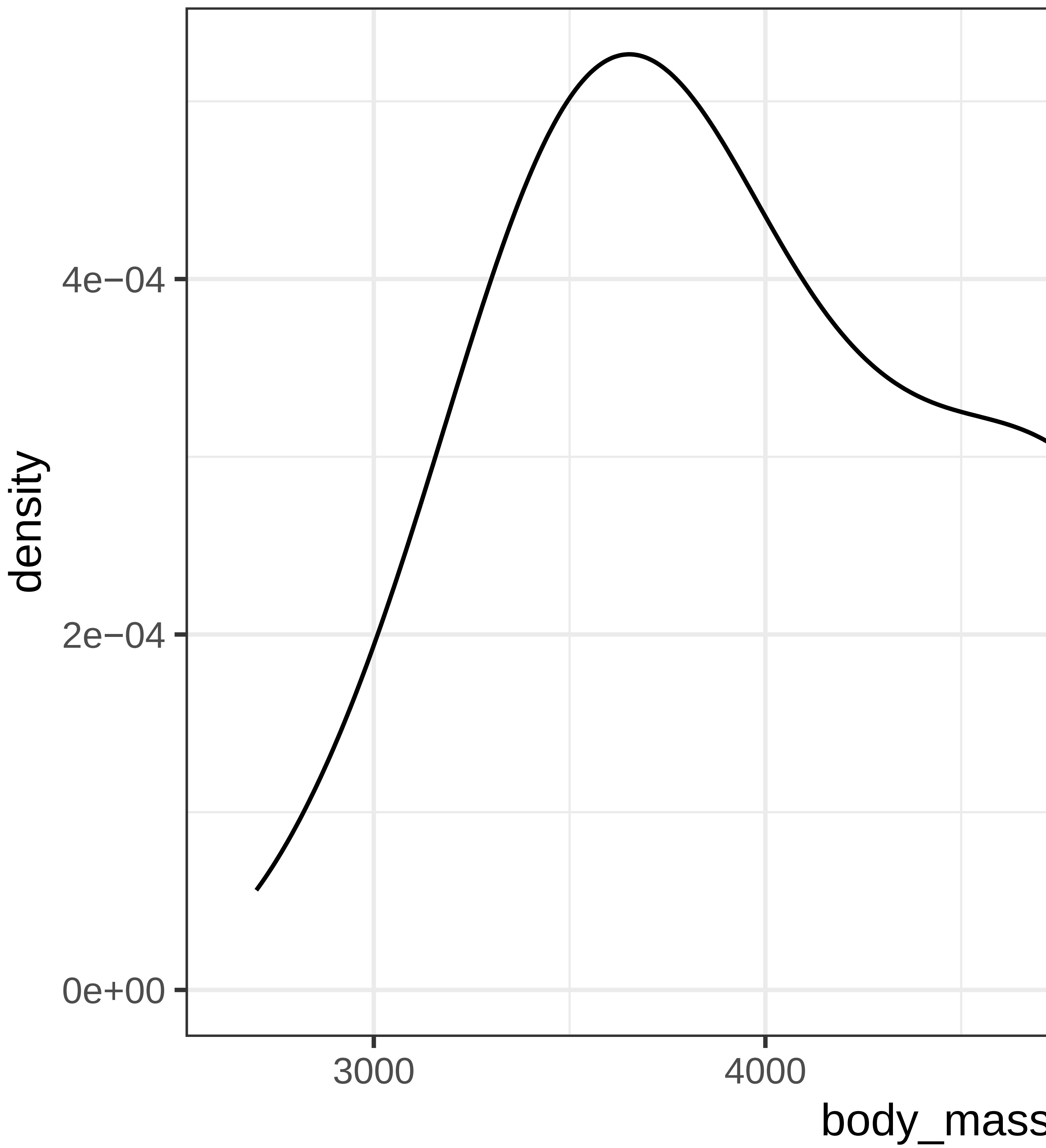
```r
ggplot(penguins, aes(x = body_mass_g)) +
  geom_histogram(binwidth = 20)
ggplot(penguins, aes(x = body_mass_g)) +
  geom_histogram(binwidth = 2000)
```

An alternative visualization for distributions of numerical variables is a density plot. A density plot is a smoothed-out version of a histogram and a practical alternative, particularly for continuous data that comes from an underlying smooth distribution. We won't go into how `geom_density()` estimates the density (you can read more about that in the function documentation), but let's explain how the density curve is drawn with an analogy. Imagine a histogram made out of wooden blocks. Then, imagine that you drop a cooked spaghetti string over it. The shape the spaghetti will take draped over blocks can be thought of as the shape of the density curve. It shows fewer details than a histogram but can make it easier to quickly glean the shape of the distribution, particularly with respect to modes and skewness.

```r
ggplot(penguins, aes(x = body_mass_g)) +
  geom_density()
#> Warning: Removed 2 rows containing non-finite values (`stat_density()`).
```


count
12.5
10.0
7.5
5.0
2.5
0.0


count
200
100
0
2000



4e−04
2e−04
0e+00
density
3000
4000
body_mass

1.4.3 Exercises

1. Make a bar plot of `species` of `penguins`, where you assign `species` to the `y` aesthetic. How is this plot different?

2. How are the following two plots different? Which aesthetic, `color` or `fill`, is more useful for changing the color of bars?

```r
ggplot(penguins, aes(x = species)) +
  geom_bar(color = "red")

ggplot(penguins, aes(x = species)) +
  geom_bar(fill = "red")
```

3. What does the `bins` argument in `geom_histogram()` do?

4. Make a histogram of the `carat` variable in the `diamonds` dataset that is available when you load the tidyverse package. Experiment with different binwidths. What binwidth reveals the most interesting patterns?

1.5 Visualizing relationships

To visualize a relationship we need to have at least two variables mapped to aesthetics of a plot. In the following sections you will learn about commonly used plots for visualizing relationships between two or more variables and the geoms used for creating them.

1.5.1 A numerical and a categorical variable

To visualize the relationship between a numerical and a categorical variable we can use side-by-side box plots. A **boxplot** is a type of visual shorthand for measures of position (percentiles) that describe a distribution. It is also useful for identifying potential outliers. As shown in Figure 1.1, each boxplot consists of:

- A box that indicates the range of the middle half of the data, a distance known as the interquartile range (IQR), stretching from the 25th percentile of the distribution to the 75th percentile. In the middle of the box is a line that displays the median, i.e. 50th percentile, of the distribution. These three lines give you a sense of the spread of the distribution and whether or not the distribution is symmetric about the median or skewed to one side.

- Visual points that display observations that fall more than 1.5 times the IQR from either edge of the box. These outlying points are unusual so are plotted individually.

- A line (or whisker) that extends from each end of the box and goes to the farthest non-outlier point in the distribution.

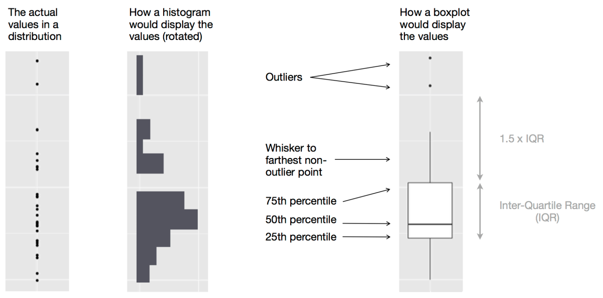
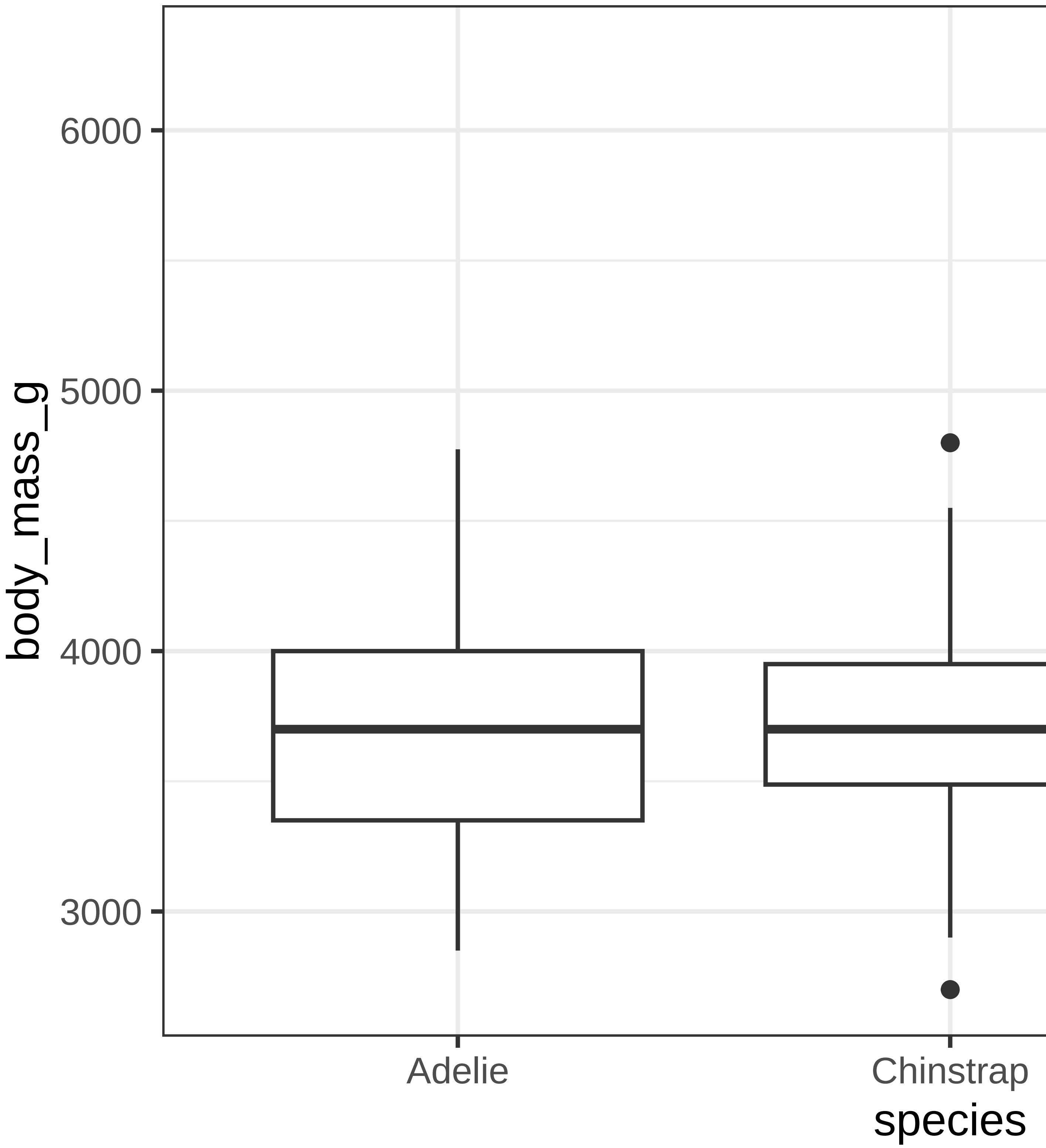

The actual values in a distribution
How a histogram would display the values (rotated)
How a boxplot would display the values
Outliers
Whisker to farthest non-outlier point
75th percentile
50th percentile
25th percentile
1.5 x IQR
Inter-Quartile Range (IQR)

Figure 1.1: Diagram depicting how a boxplot is created.

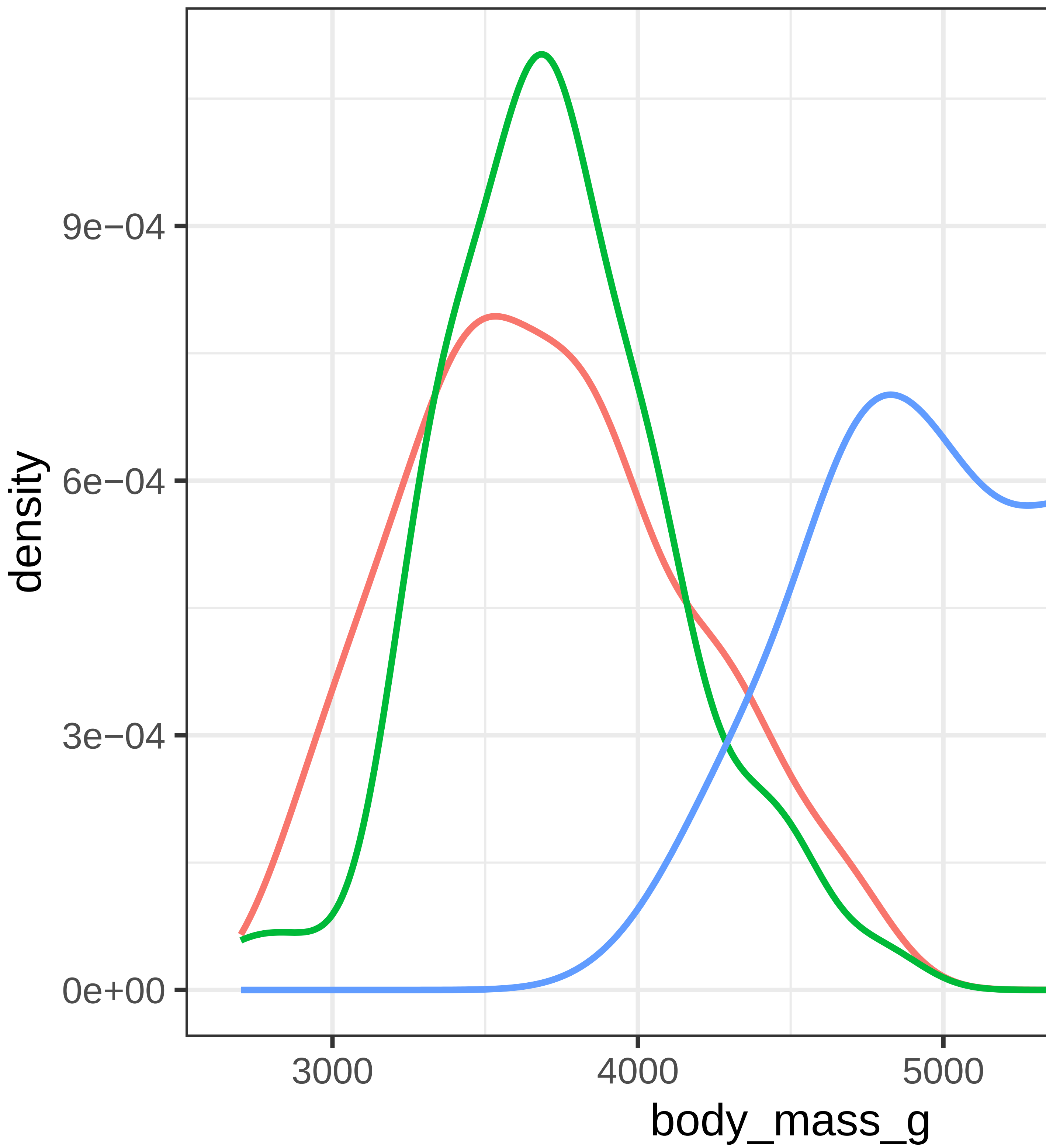
Let's take a look at the distribution of body mass by species using `geom_boxplot()`:

```r
ggplot(penguins, aes(x = species, y = body_mass_g)) +
  geom_boxplot()
```


6000
5000
4000
3000
body_mass_g
Adelie
Chinstrap
species

Alternatively, we can make density plots with `geom_density()`.

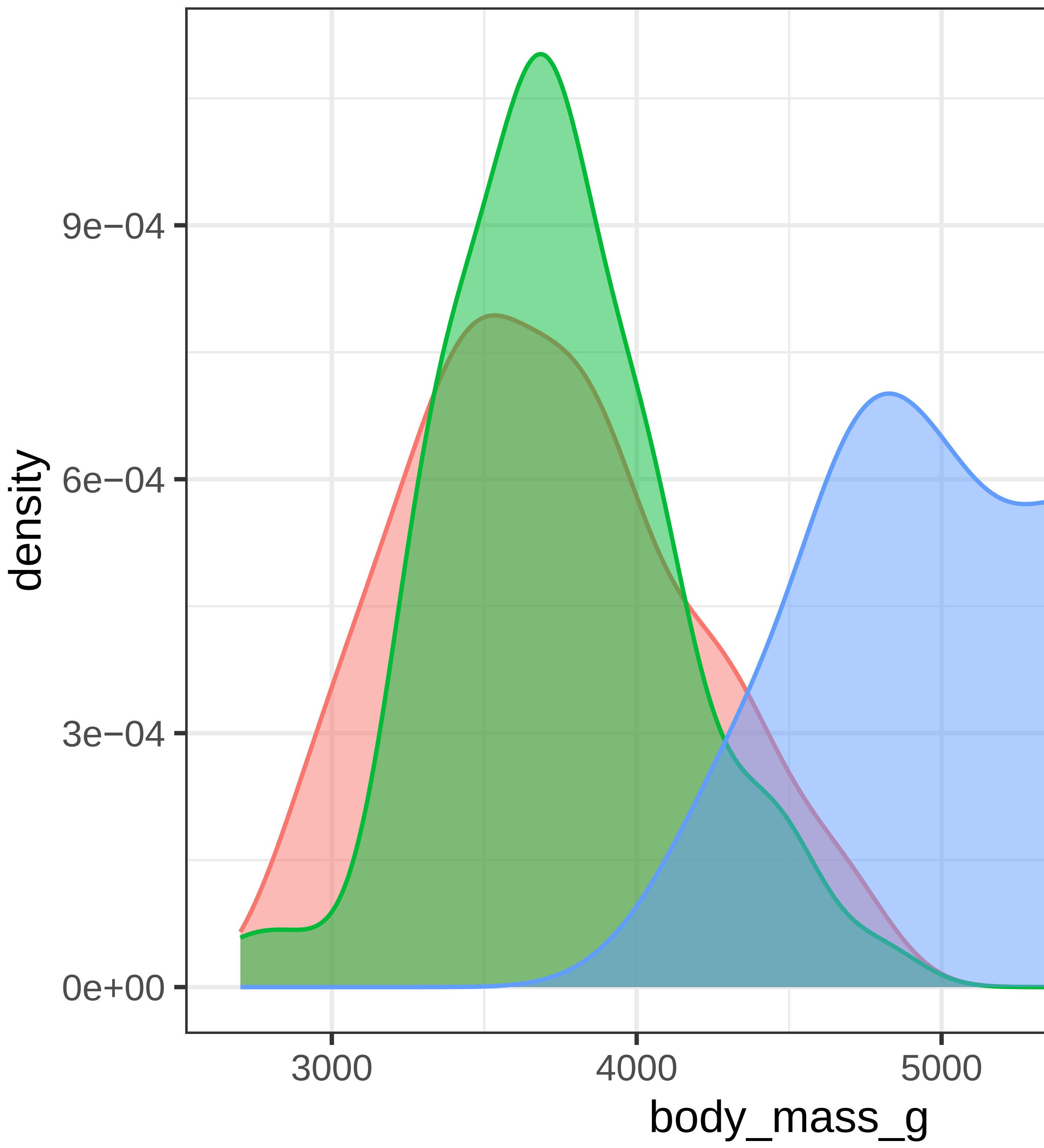
```
ggplot(penguins, aes(x = body_mass_g, color = species)) +
  geom_density(linewidth = 0.75)
```


density
9e−04
6e−04
3e−04
0e+00
3000
4000
5000
body_mass_g

We've also customized the thickness of the lines using the `linewidth` argument in order to make them stand out a bit more against the background.

Additionally, we can map `species` to both `color` and `fill` aesthetics and use the `alpha` aesthetic to add transparency to the filled density curves. This aesthetic takes values between 0 (completely transparent) and 1 (completely opaque). In the following plot it's *set* to 0.5.

```r
ggplot(penguins, aes(x = body_mass_g, color = species, fill = species)) +
  geom_density(alpha = 0.5)
```


9e−04
6e−04
3e−04
0e+00
density
3000
4000
5000
body_mass_g

Note the terminology we have used here:

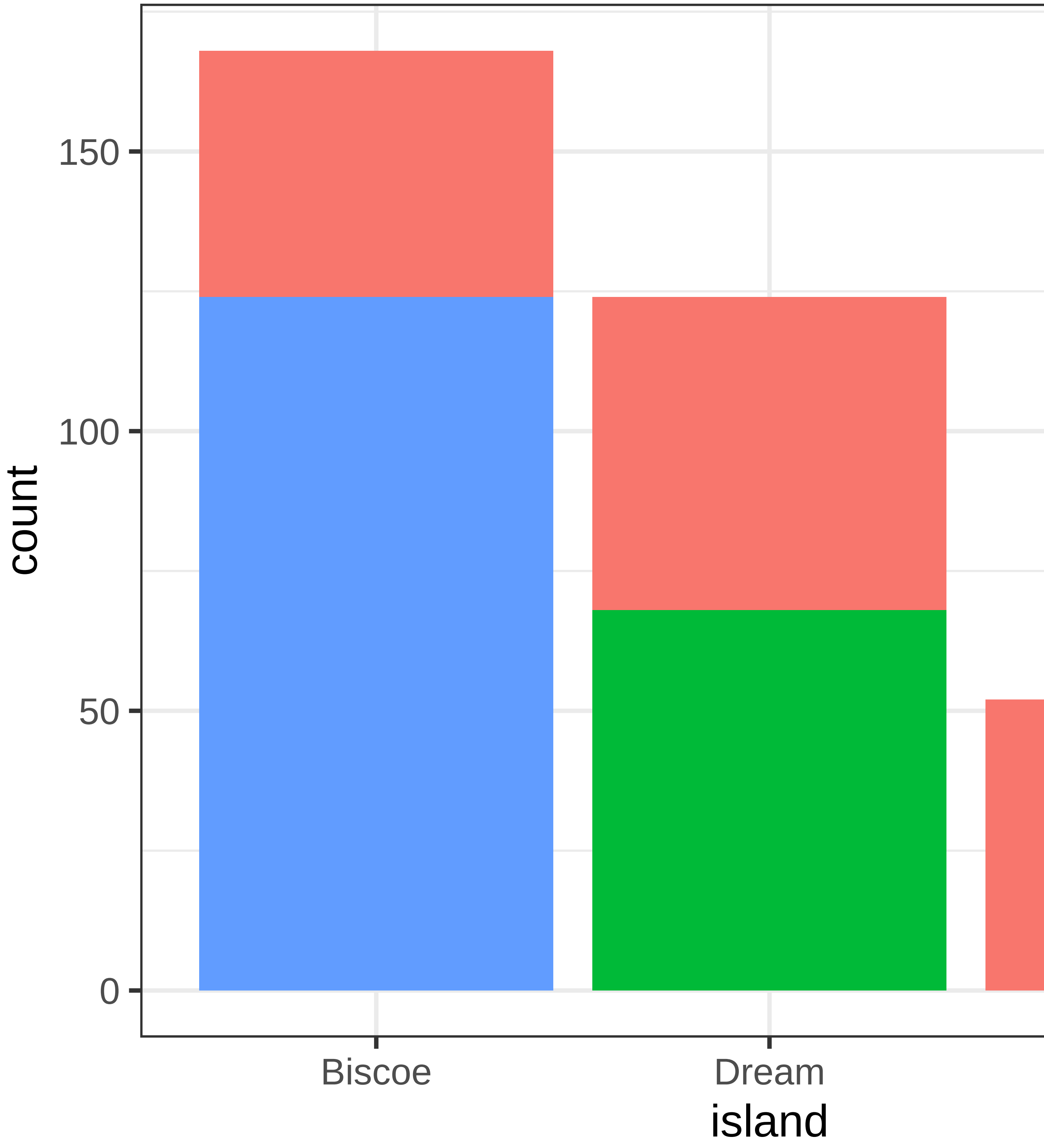
- We *map* variables to aesthetics if we want the visual attribute represented by that aesthetic to vary based on the values of that variable.
- Otherwise, we *set* the value of an aesthetic.

1.5.2 Two categorical variables

We can use stacked bar plots to visualize the relationship between two categorical variables. For example, the following two stacked bar plots both display the relationship between `island` and `species`, or specifically, visualizing the distribution of `species` within each island.

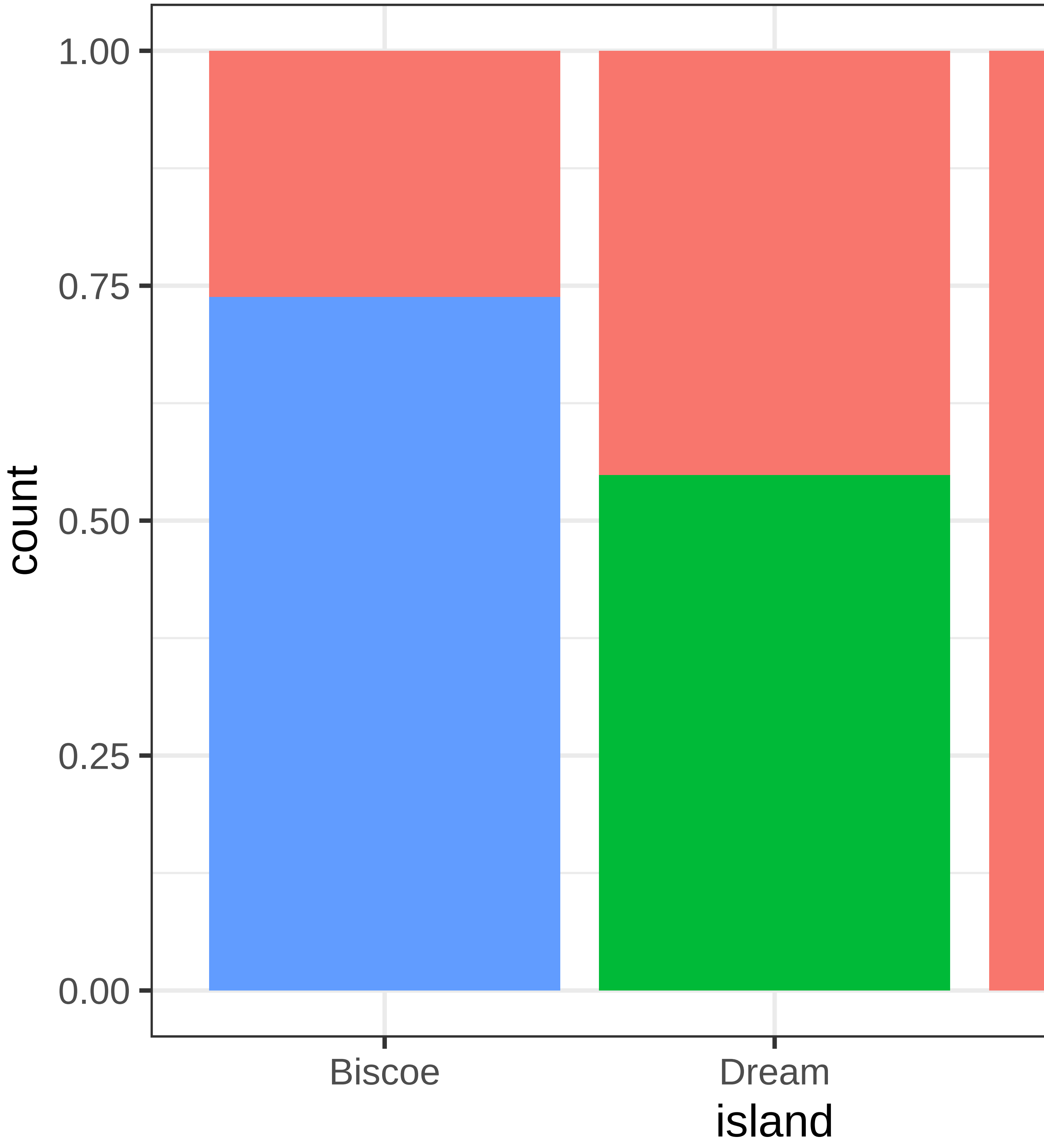
The first plot shows the frequencies of each species of penguins on each island. The plot of frequencies shows that there are equal numbers of Adelies on each island. But we don't have a good sense of the percentage balance within each island.

```r
ggplot(penguins, aes(x = island, fill = species)) +
  geom_bar()
```


count
150
100
50
0
Biscoe
Dream
island

The second plot, a relative frequency plot created by setting `position = "fill"` in the geom, is more useful for comparing species distributions across islands since it's not affected by the unequal numbers of penguins across the islands. Using this plot we can see that Gentoo penguins all live on Biscoe island and make up roughly 75% of the penguins on that island, Chinstrap all live on Dream island and make up roughly 50% of the penguins on that island, and Adelie live on all three islands and make up all of the penguins on Torgersen.

```r
ggplot(penguins, aes(x = island, fill = species)) +
  geom_bar(position = "fill")
```

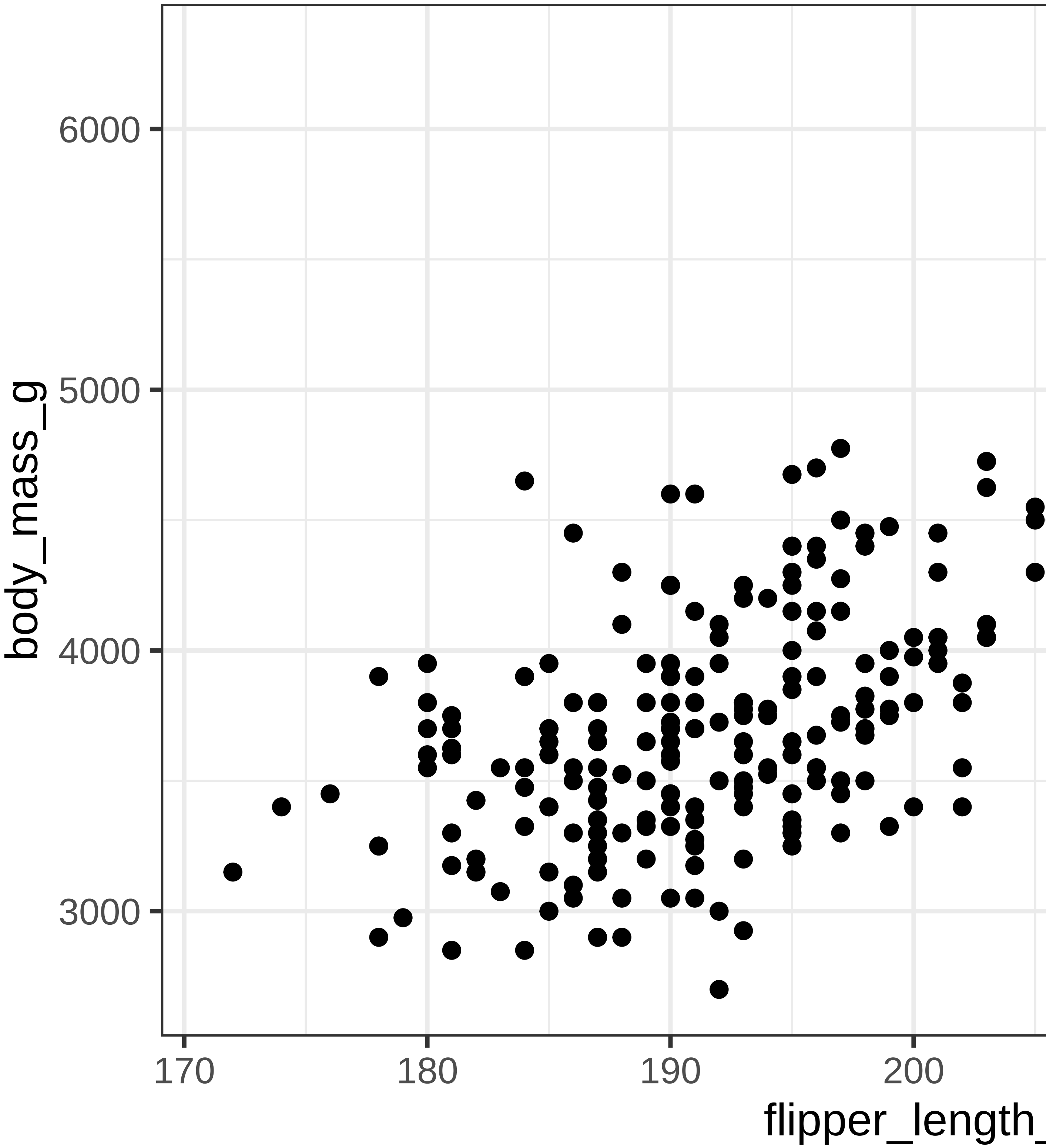

1.00
0.75
0.50
0.25
0.00
count
Biscoe
Dream
island

In creating these bar charts, we map the variable that will be separated into bars to the x aesthetic, and the variable that will change the colors inside the bars to the fill aesthetic.

1.5.3 Two numerical variables

So far you've learned about scatterplots (created with geom_point()) and smooth curves (created with geom_smooth()) for visualizing the relationship between two numerical variables. A scatterplot is probably the most commonly used plot for visualizing the relationship between two numerical variables.

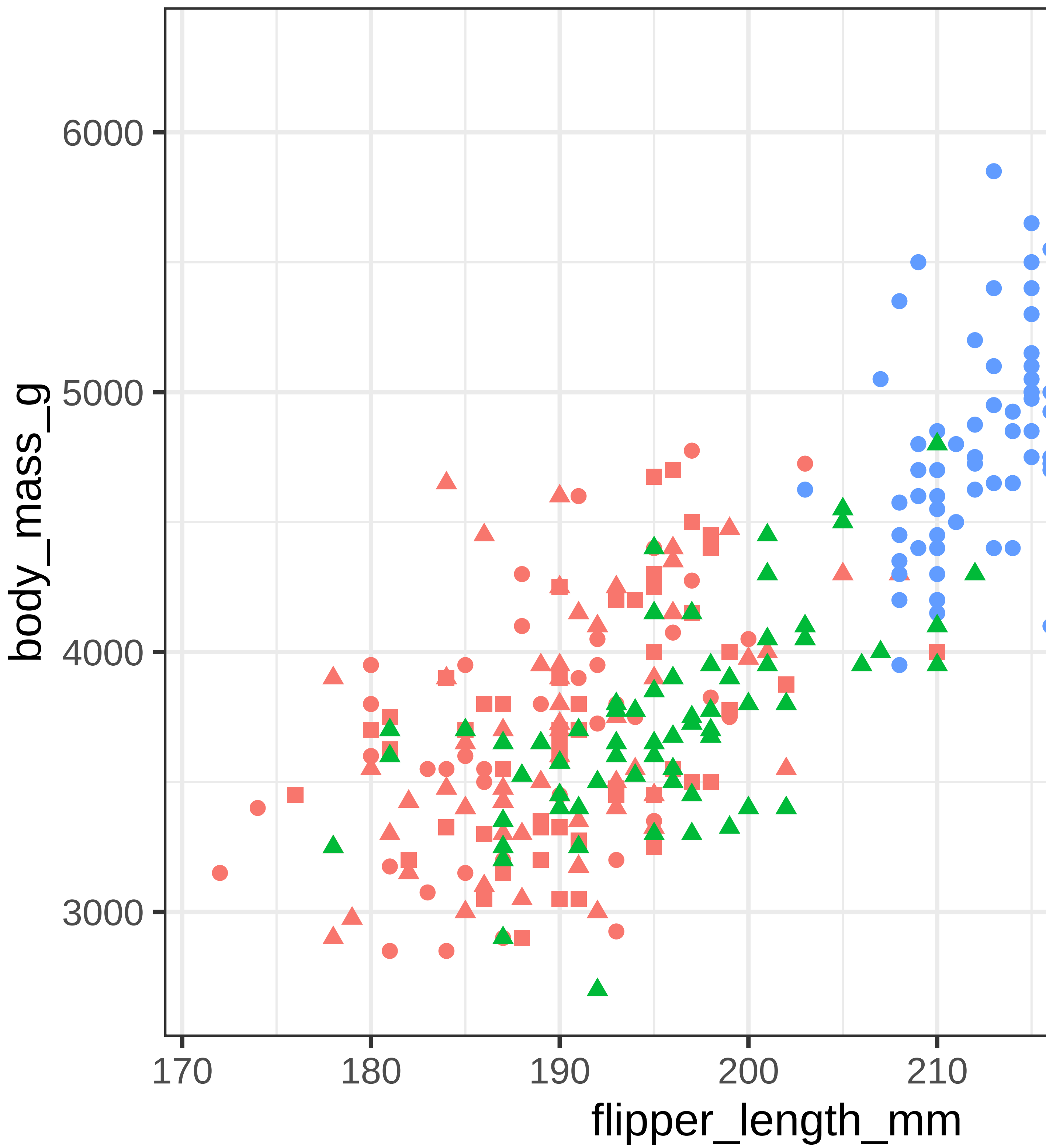
```r
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point()
```


body_mass_g
6000
5000
4000
3000
170
180
190
200
flipper_length_

1.5.4 Three or more variables

As we saw in Section 1.2.4, we can incorporate more variables into a plot by mapping them to additional aesthetics. For example, in the following scatterplot the colors of points represent species and the shapes of points represent islands.

```r
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point(aes(color = species, shape = island))
```

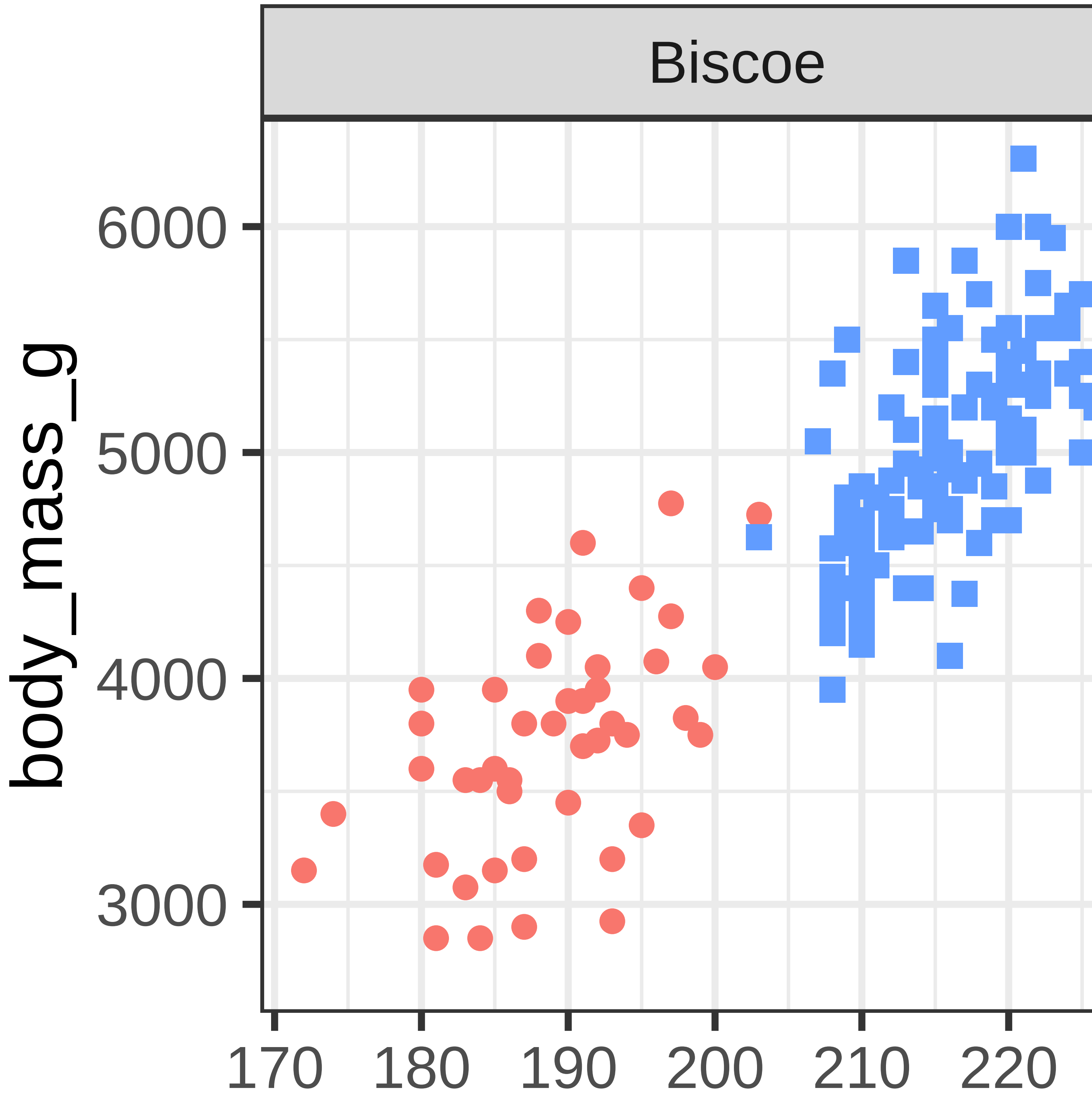

6000
5000
4000
3000
body_mass_g
170
180
190
200
210
flipper_length_mm

However adding too many aesthetic mappings to a plot makes it cluttered and difficult to make sense of. Another way, which is particularly useful for categorical variables, is to split your plot into **facets**, subplots that each display one subset of the data.

To facet your plot by a single variable, use `facet_wrap()`. The first argument of `facet_wrap()` is a formula[3], which you create with `~` followed by a variable name. The variable that you pass to `facet_wrap()` should be categorical.

```r
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point(aes(color = species, shape = species)) +
  facet_wrap(~island)
```

[3]Here "formula" is the name of the thing created by `~`, not a synonym for "equation".


Biscoe
body_mass_g
6000
5000
4000
3000
170
180
190
200
210
220

You will learn about many other geoms for visualizing distributions of variables and relationships between them in Chapter 2.

1.5.5 Exercises

1. The mpg data frame that is bundled with the ggplot2 package contains 234 observations collected by the US Environmental Protection Agency on 38 car models. Which variables in mpg are categorical? Which variables are numerical? (Hint: Type ?mpg to read the documentation for the dataset.) How can you see this information when you run mpg?
2. Make a scatterplot of hwy vs. displ using the mpg data frame. Next, map a third, numerical variable to color, then size, then both color and size, then shape. How do these aesthetics behave differently for categorical vs. numerical variables?
3. In the scatterplot of hwy vs. displ, what happens if you map a third variable to linewidth?
4. What happens if you map the same variable to multiple aesthetics?
5. Make a scatterplot of bill_depth_mm vs. bill_length_mm and color the points by species. What does adding coloring by species reveal about the relationship between these two variables? What about faceting by species?
6. Why does the following yield two separate legends? How would you fix it to combine the two legends?

```
ggplot(
  data = penguins,
  mapping = aes(
    x = bill_length_mm, y = bill_depth_mm,
    color = species, shape = species
  )
) +
  geom_point() +
  labs(color = "Species")
```

7. Create the two following stacked bar plots. Which question can you answer with the first one? Which question can you answer with the second one?

```
ggplot(penguins, aes(x = island, fill = species)) +
  geom_bar(position = "fill")
ggplot(penguins, aes(x = species, fill = island)) +
  geom_bar(position = "fill")
```

1.6 Saving your plots

Once you've made a plot, you might want to get it out of R by saving it as an image that you can use elsewhere. That's the job of `ggsave()`, which will save the plot most recently created to disk:

```r
ggplot(penguins, aes(x = flipper_length_mm, y = body_mass_g)) +
  geom_point()
ggsave(filename = "penguin-plot.png")
```

This will save your plot to your working directory, a concept you'll learn more about in **?@sec-workflow-scripts-projects**.

If you don't specify the `width` and `height` they will be taken from the dimensions of the current plotting device. For reproducible code, you'll want to specify them. You can learn more about `ggsave()` in the documentation.

Generally, however, we recommend that you assemble your final reports using Quarto, a reproducible authoring system that allows you to interleave your code and your prose and automatically include your plots in your write-ups. You will learn more about Quarto in **?@sec-quarto**.

1.6.1 Exercises

1. Run the following lines of code. Which of the two plots is saved as `mpg-plot.png`? Why?

   ```r
   ggplot(mpg, aes(x = class)) +
     geom_bar()
   ggplot(mpg, aes(x = cty, y = hwy)) +
     geom_point()
   ggsave("mpg-plot.png")
   ```

2. What do you need to change in the code above to save the plot as a PDF instead of a PNG? How could you find out what types of image files would work in `ggsave()`?

1.7 Common problems

As you start to run R code, you're likely to run into problems. Don't worry — it happens to everyone. We have all been writing R code for years, but every day we still write code that doesn't work on the first try!

Start by carefully comparing the code that you're running to the code in the book. R is extremely picky, and a misplaced character can make all the difference. Make sure that every `(` is matched with a `)` and every `"` is paired with another `"`. Sometimes you'll run the code and nothing happens. Check the left-hand of your console: if it's a `+`, it means that R doesn't think you've typed a complete expression and it's waiting for you to finish it. In this case, it's usually easy to start from scratch again by pressing ESCAPE to abort processing the current command.

One common problem when creating ggplot2 graphics is to put the `+` in the wrong place: it has to come at the end of the line, not the start. In other words, make sure you haven't accidentally written code like this:

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

If you're still stuck, try the help. You can get help about any R function by running `?function_name` in the console, or highlighting the function name and pressing F1 in RStudio. Don't worry if the help doesn't seem that helpful - instead skip down to the examples and look for code that matches what you're trying to do.

If that doesn't help, carefully read the error message. Sometimes the answer will be buried there! But when you're new to R, even if the answer is in the error message, you might not yet know how to understand it. Another great tool is Google: try googling the error message, as it's likely someone else has had the same problem, and has gotten help online.

1.8 Summary

In this chapter, you've learned the basics of data visualization with ggplot2. We started with the basic idea that underpins ggplot2: a visualization is a mapping from variables in your data to aesthetic properties like position, color, size and shape. You then learned about increasing the complexity and improving the presentation of your plots layer-by-layer. You also learned about commonly used plots for visualizing the distribution of a single variable as well as for visualizing relationships between two or more variables, by leveraging additional aesthetic mappings and/or splitting your plot into small multiples using faceting.

We'll use visualizations again and again throughout this book, introducing new techniques as we need them as well as do a deeper dive into creating visualizations with ggplot2 in Chapter 2 through **?@sec-communication**.

With the basics of visualization under your belt, in the next chapter we're going to switch gears a little and give you some practical workflow advice. We intersperse workflow advice with data science tools throughout this part of the book because it'll help you stay organized as you write increasing amounts of R code.

2 Layers

2.1 Introduction

In Chapter 1, you learned much more than just how to make scatterplots, bar charts, and boxplots. You learned a foundation that you can use to make *any* type of plot with ggplot2.

In this chapter, you'll expand on that foundation as you learn about the layered grammar of graphics. We'll start with a deeper dive into aesthetic mappings, geometric objects, and facets. Then, you will learn about statistical transformations ggplot2 makes under the hood when creating a plot. These transformations are used to calculate new values to plot, such as the heights of bars in a bar plot or medians in a box plot. You will also learn about position adjustments, which modify how geoms are displayed in your plots. Finally, we'll briefly introduce coordinate systems.

We will not cover every single function and option for each of these layers, but we will walk you through the most important and commonly used functionality provided by ggplot2 as well as introduce you to packages that extend ggplot2.

2.1.1 Prerequisites

This chapter focuses on ggplot2. To access the datasets, help pages, and functions used in this chapter, load the tidyverse by running this code:

```
library(tidyverse)
```

2.2 Aesthetic mappings

> "The greatest value of a picture is when it forces us to notice what we never expected to see." — John Tukey

Remember that the `mpg` data frame bundled with the ggplot2 package contains 234 observations on 38 car models.

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

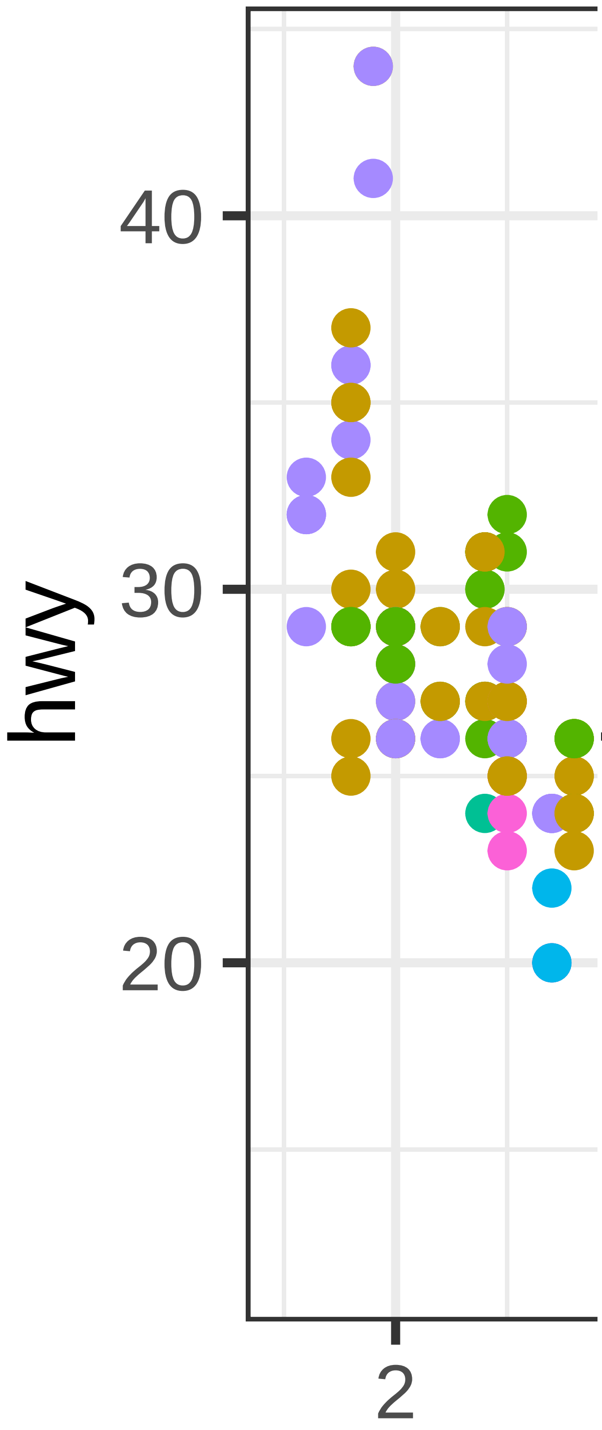
Among the variables in `mpg` are:

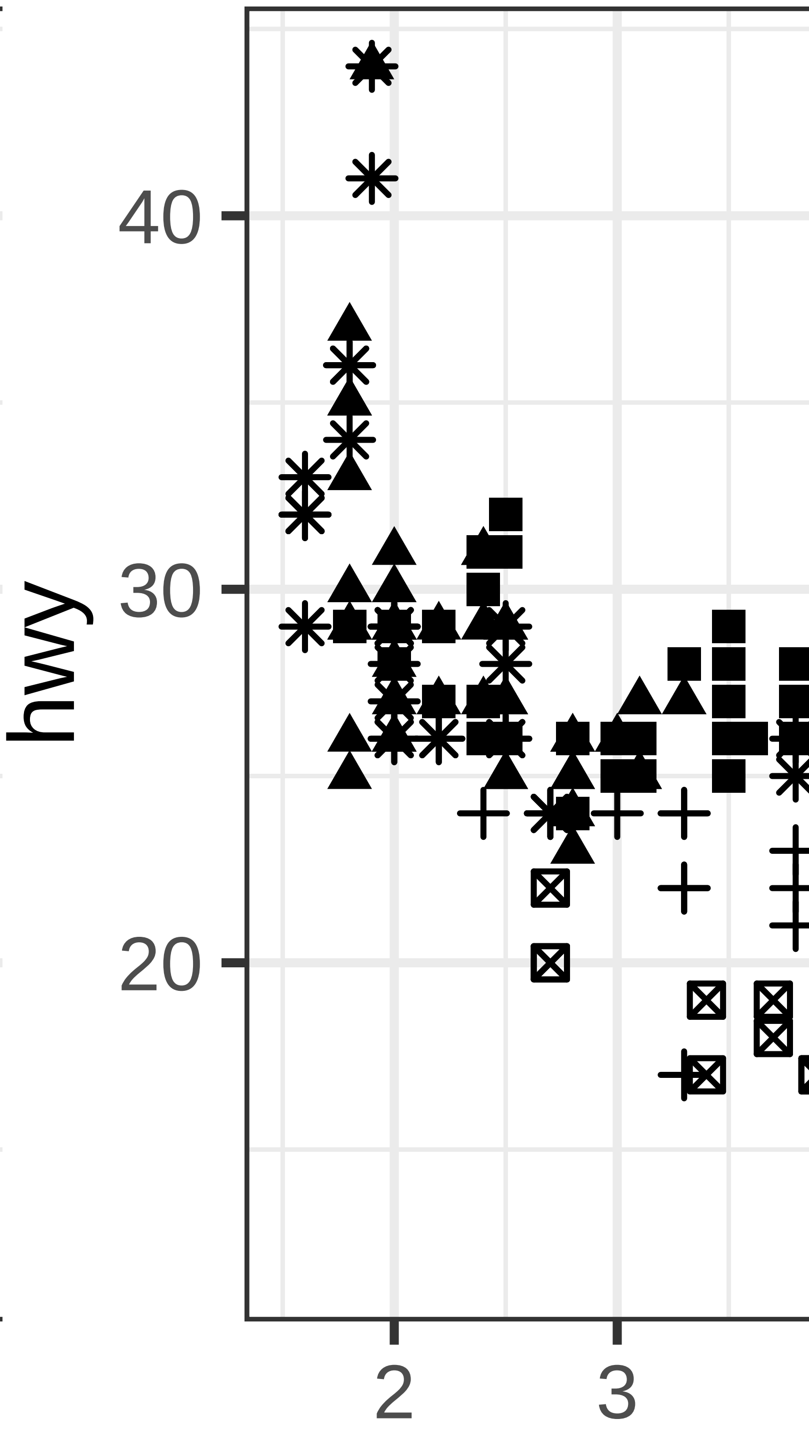
1. `displ`: A car's engine size, in liters. A numerical variable.
2. `hwy`: A car's fuel efficiency on the highway, in miles per gallon (mpg). A car with a low fuel efficiency consumes more fuel than a car with a high fuel efficiency when they travel the same distance. A numerical variable.
3. `class`: Type of car. A categorical variable.

Let's start by visualizing the relationship between `displ` and `hwy` for various `class`es of cars. We can do this with a scatterplot where the numerical variables are mapped to the `x` and `y` aesthetics and the categorical variable is mapped to an aesthetic like `color` or `shape`.

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

# Right
ggplot(mpg, aes(x = displ, y = hwy, shape = class)) +
  geom_point()
#> Warning: The shape palette can deal with a maximum of 6 discrete values because
#> more than 6 becomes difficult to discriminate; you have 7. Consider
#> specifying shapes manually if you must have them.
#> Warning: Removed 62 rows containing missing values (`geom_point()`).
```

When `class` is mapped to `shape`, we get two warnings:


hwy
40
30
20
2


hwy
40
30
20
2
3

> 1: 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.
>
> 2: Removed 62 rows containing missing values (`geom_point()`).

Since ggplot2 will only use six shapes at a time, by default, additional groups will go unplotted when you use the shape aesthetic. The second warning is related – there are 62 SUVs in the dataset and they're not plotted.

Similarly, we can map `class` to `size` or `alpha` aesthetics as well, which control the shape and the transparency of the points, respectively.

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

# Right
ggplot(mpg, aes(x = displ, y = hwy, alpha = class)) +
  geom_point()
#> Warning: Using alpha for a discrete variable is not advised.
```

Both of these produce warnings as well:

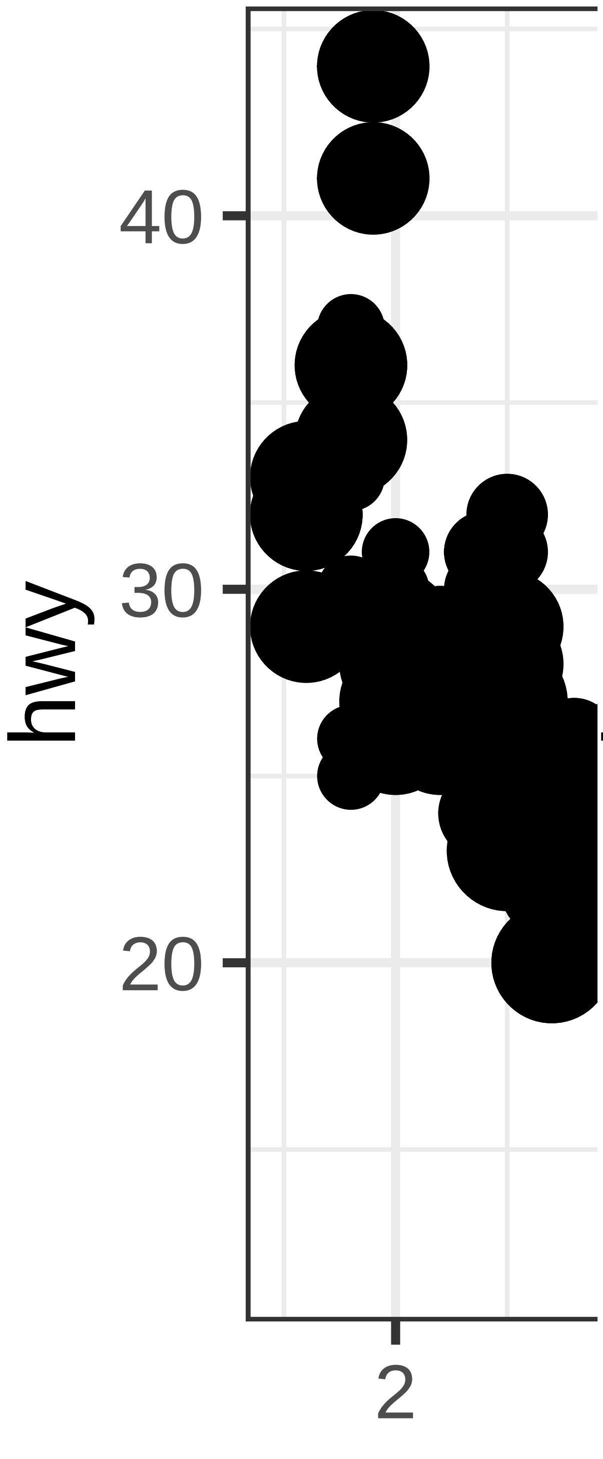
> Using alpha for a discrete variable is not advised.

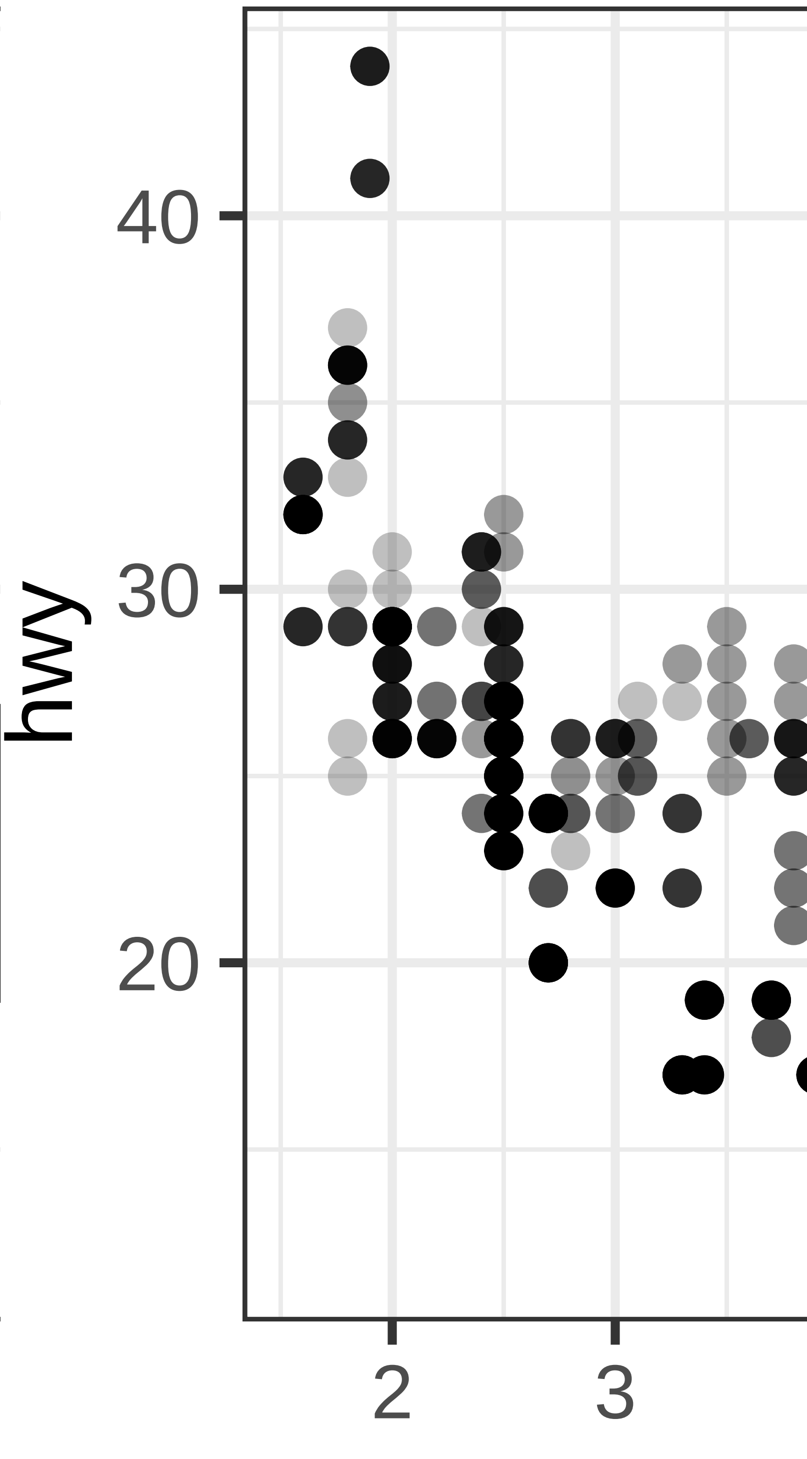
Mapping an unordered discrete (categorical) variable (`class`) to an ordered aesthetic (`size` or `alpha`) is generally not a good idea because it implies a ranking that does not in fact exist.

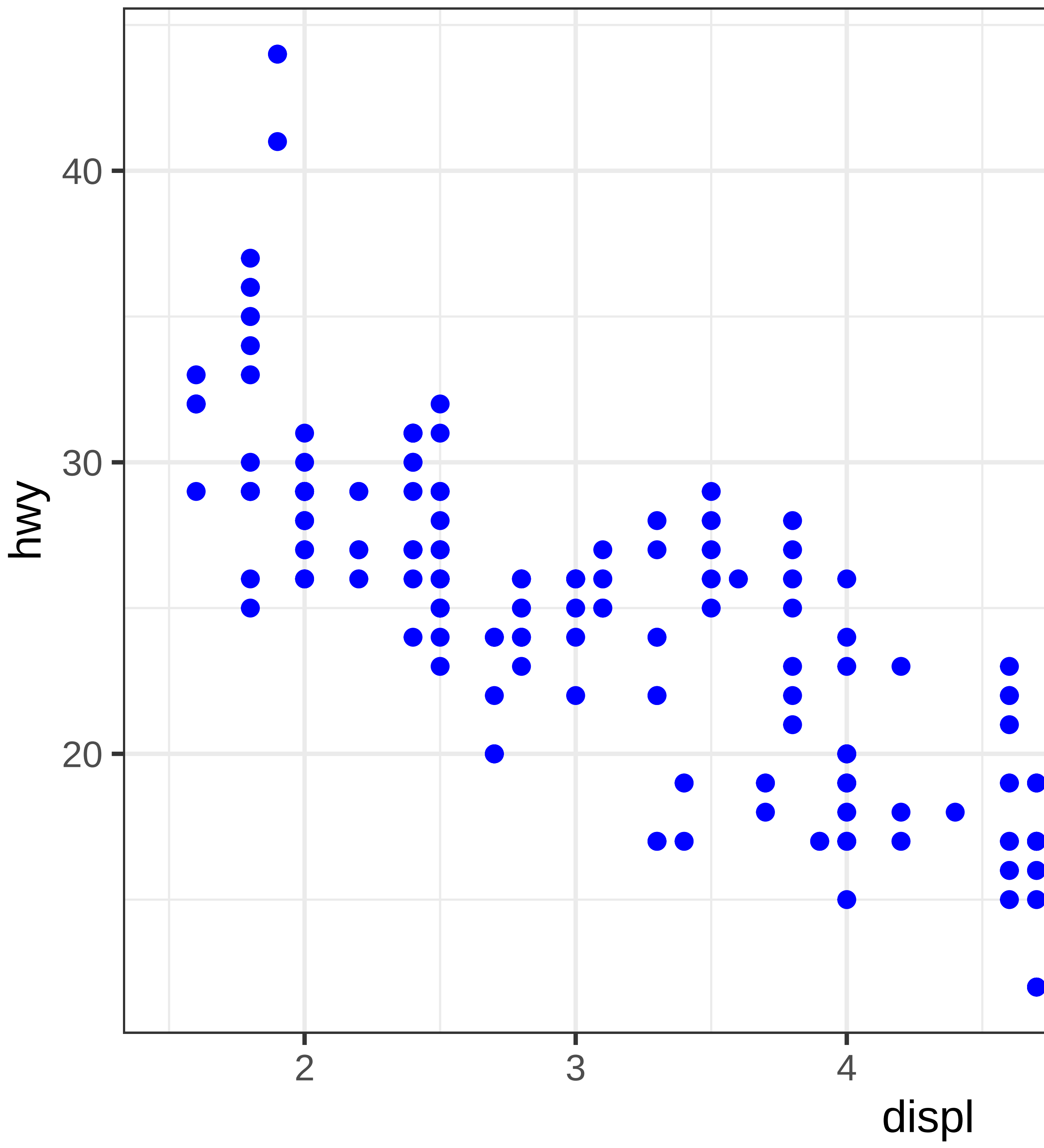
Once you map an aesthetic, ggplot2 takes care of the rest. It selects a reasonable scale to use with the aesthetic, and it constructs a legend that explains the mapping between levels and values. For x and y aesthetics, ggplot2 does not create a legend, but it creates an axis line with tick marks and a label. The axis line provides the same information as a legend; it explains the mapping between locations and values.

You can also set the visual properties of your geom manually as an argument of your geom function (*outside* of `aes()`) instead of relying on a variable mapping to determine the appearance. For example, we can make all of the points in our plot blue:

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


hwy
40
30
20
2


hwy
40
30
20
2
3



40
30
20
hwy
2
3
4
displ

Here, the color doesn't convey information about a variable, but only changes the appearance of the plot. You'll need to pick a value that makes sense for that aesthetic:

- The name of a color as a character string, e.g., `color = "blue"`
- The size of a point in mm, e.g., `size = 1`
- The shape of a point as a number, e.g, `shape = 1`, as shown in Figure 2.1.

So far we have discussed aesthetics that we can map or set in a scatterplot, when using a point geom. You can learn more about all possible aesthetic mappings in the aesthetic specifications vignette at https://ggplot2.tidyverse.org/articles/ggplot2-specs.html.

The specific aesthetics you can use for a plot depend on the geom you use to represent the data. In the next section we dive deeper into geoms.

2.2.1 Exercises

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

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

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

3. What does the `stroke` aesthetic do? What shapes does it work with? (Hint: use `?geom_point`)

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

2.3 Geometric objects

How are these two plots similar?

Both plots contain the same x variable, the same y variable, and both describe the same data. But the plots are not identical. Each plot uses a different geometric object, geom, to represent the data. The plot on the left uses the point geom, and the plot on the right uses the smooth geom, a smooth line fitted to the data.

To change the geom in your plot, change the geom function that you add to `ggplot()`. For instance, to make the plots above, you can use the following code:

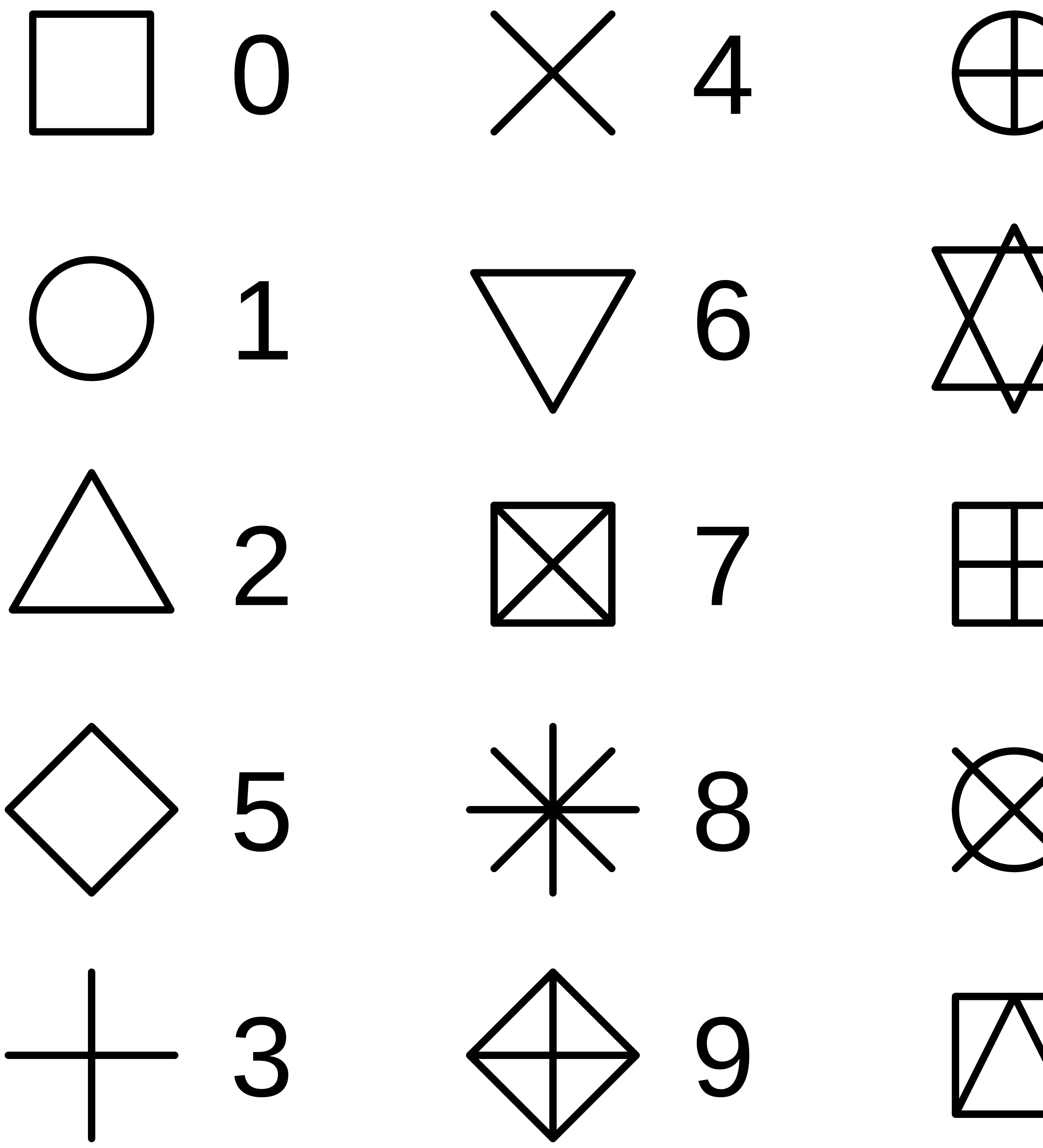
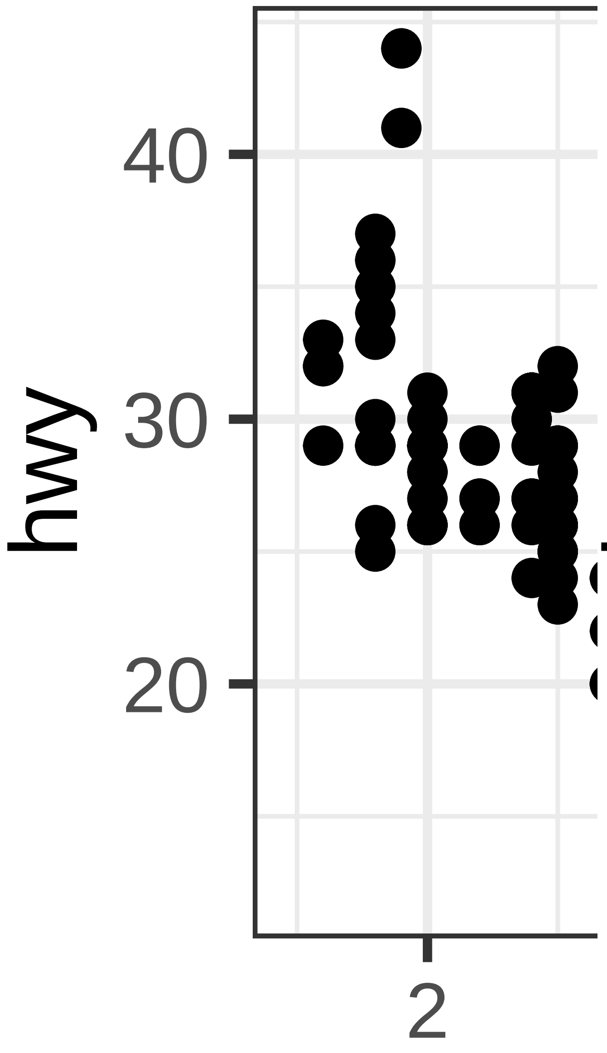
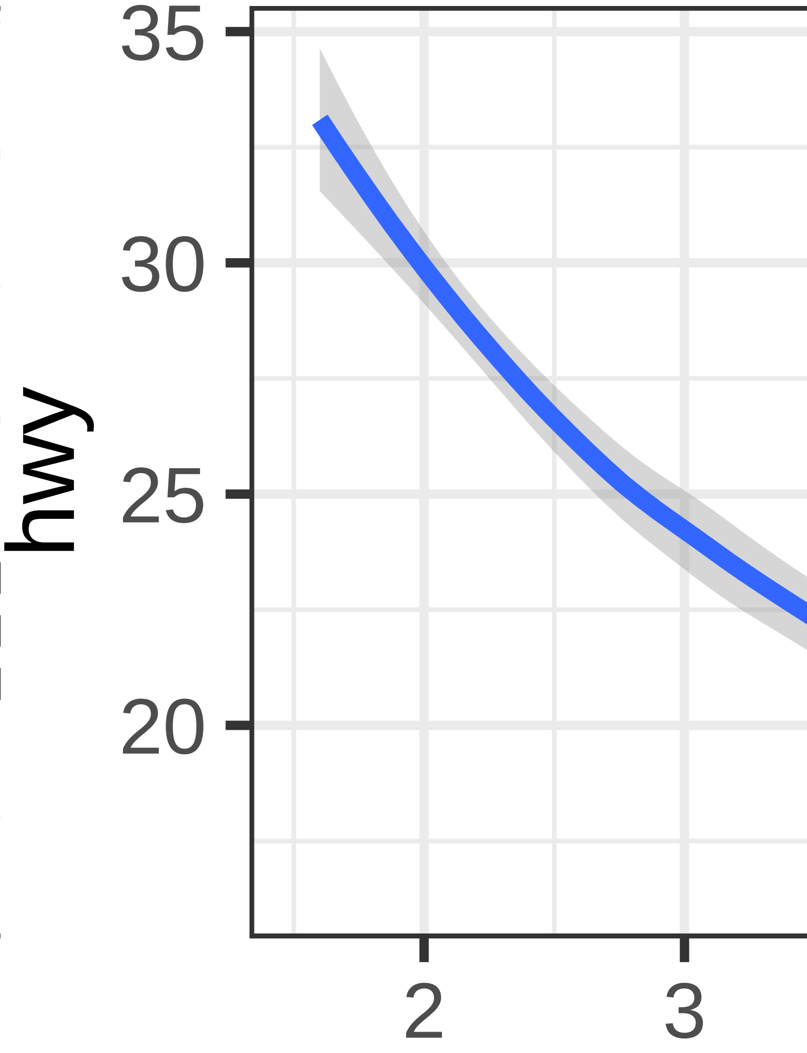


Figure 2.1: R has 25 built-in shapes that are identified by numbers. There are some seeming duplicates: for example, 0, 15, and 22 are all squares. The difference comes from the interaction of the `color` and `fill` aesthetics. The hollow shapes (0–14) have a border determined by `color`; the solid shapes (15–20) are filled with `color`; the filled shapes (21–24) have a border of `color` and are filled with `fill`. Shapes are arranged to keep similar shapes next to each other.


hwy
40
30
20
2


hwy
35
30
25
20
2
3

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

# Right
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth()
#> `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

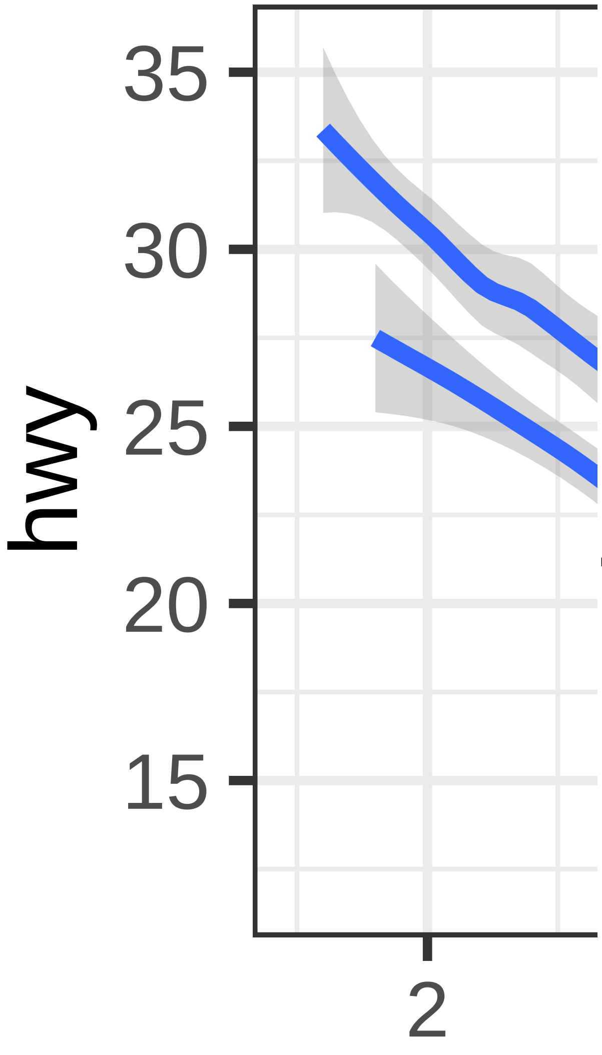
Every geom function in ggplot2 takes a `mapping` argument, either defined locally in the geom layer or globally in the `ggplot()` layer. However, not every aesthetic works with every geom. You could set the shape of a point, but you couldn't set the "shape" of a line. If you try, ggplot2 will silently ignore that aesthetic mapping. 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.

```
# Left
ggplot(mpg, aes(x = displ, y = hwy, shape = drv)) +
  geom_smooth()

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

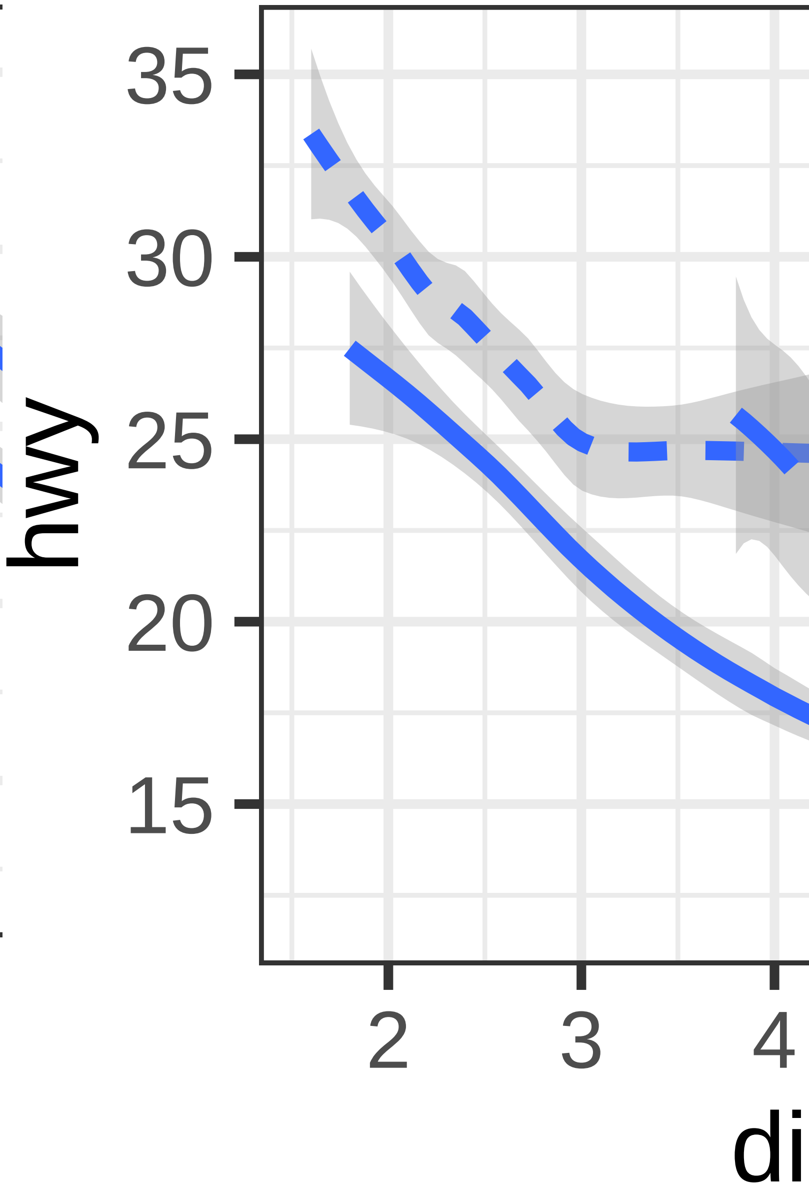
Here, `geom_smooth()` separates the cars into three lines based on their `drv` value, which describes a car's drive train. One line describes all of the points that have a `4` value, one line describes all of the points that have an `f` value, and one line describes all of the points that have an `r` value. Here, `4` stands for four-wheel drive, `f` for front-wheel drive, and `r` for rear-wheel drive.

If this sounds strange, we can make it clearer by overlaying the lines on top of the raw data and then coloring everything according to `drv`.

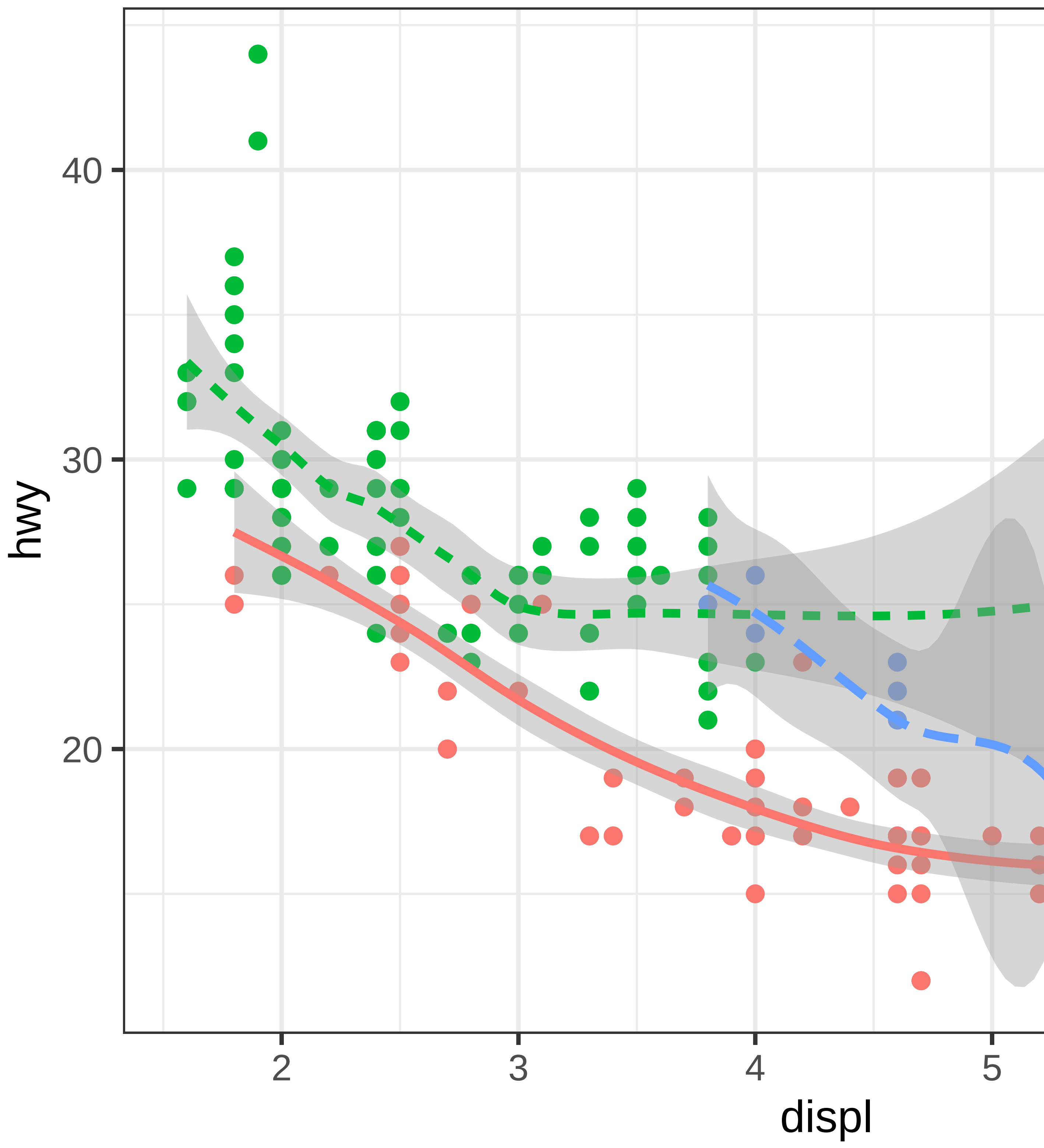
```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(aes(linetype = drv))
```



35
30
25
20
15
hwy
2



35
30
25
20
15
hwy
2
3
4
di



hwy
40
30
20
2
3
4
5
displ

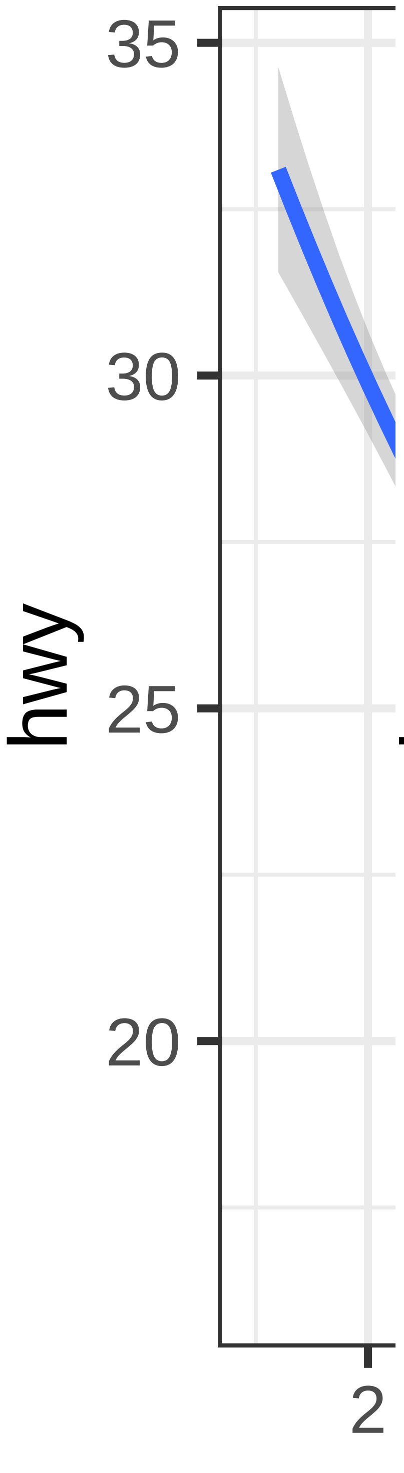
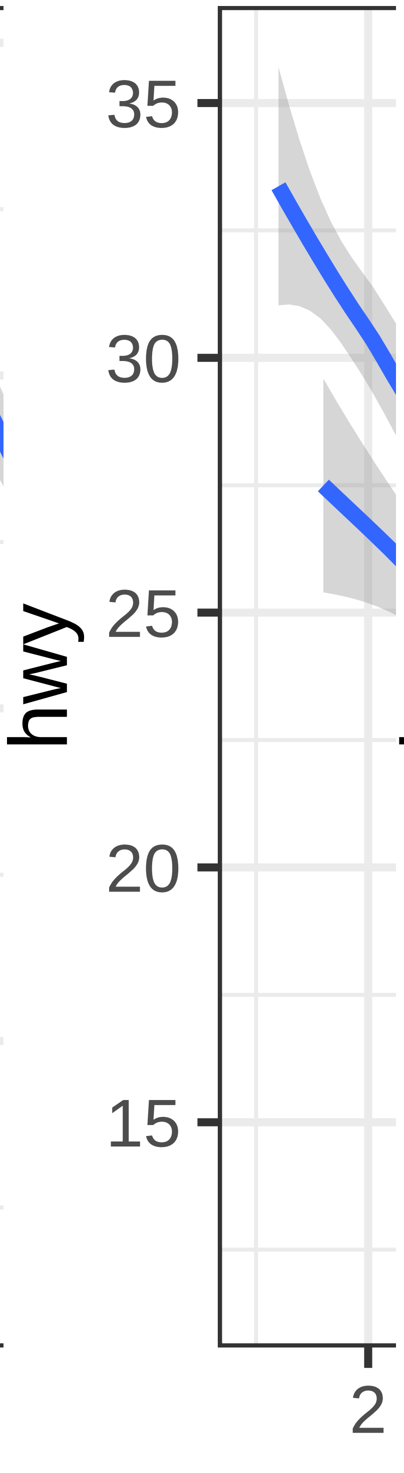
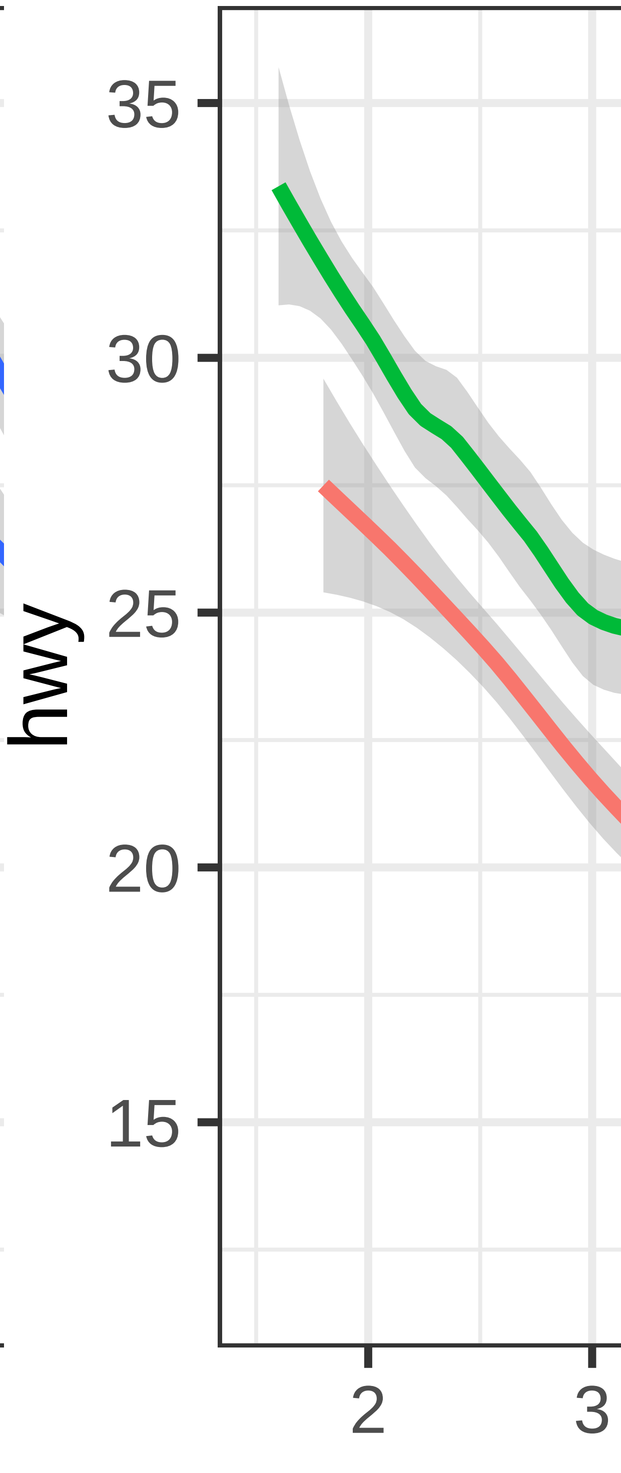
Notice that this plot contains two geoms in the same graph.

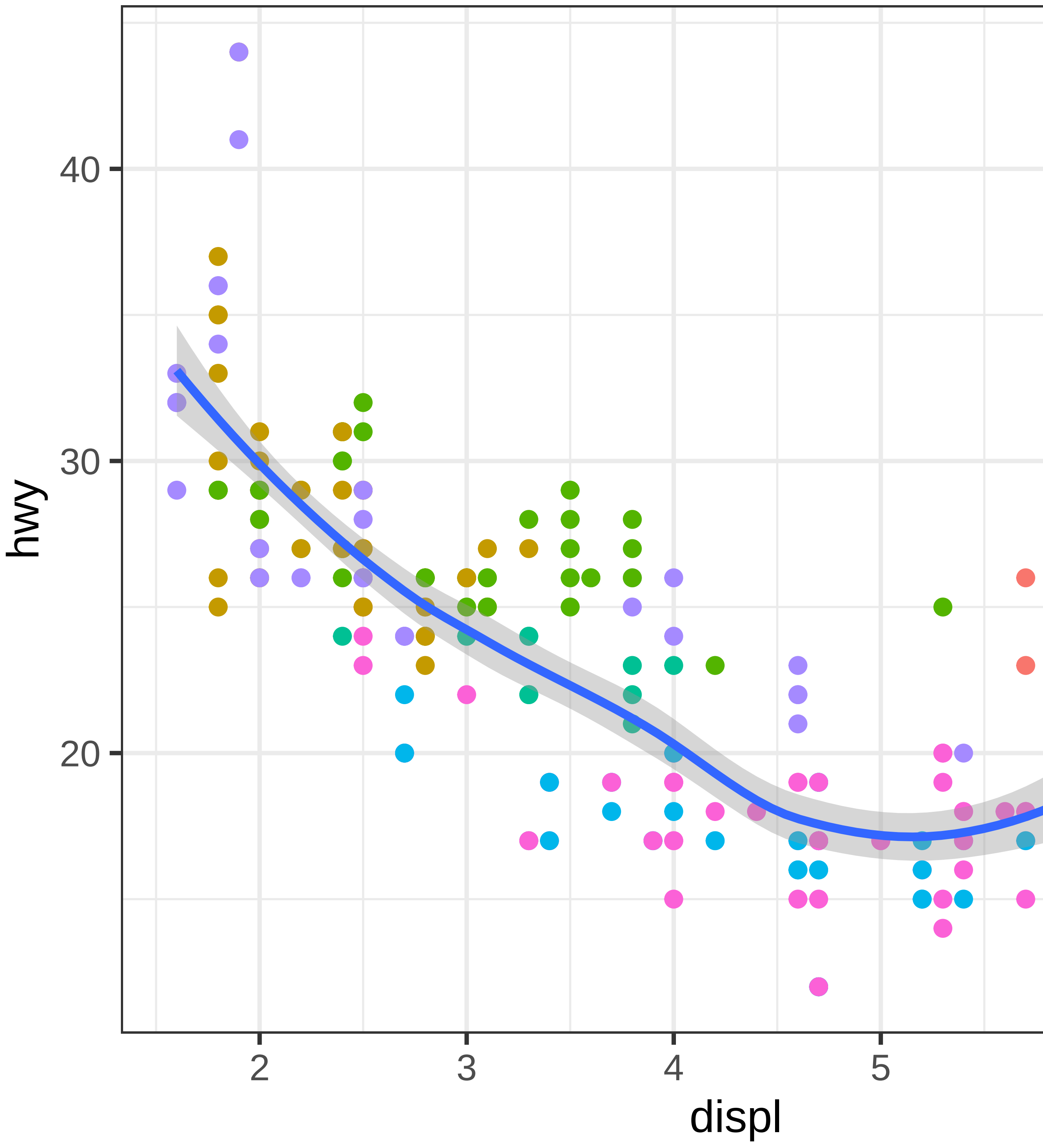
Many geoms, like `geom_smooth()`, use a single geometric object to display multiple rows of data. For these geoms, you can set the `group` aesthetic to a categorical variable to draw multiple objects. ggplot2 will draw a separate object for each unique value of the grouping variable. In practice, ggplot2 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.

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

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

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

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.

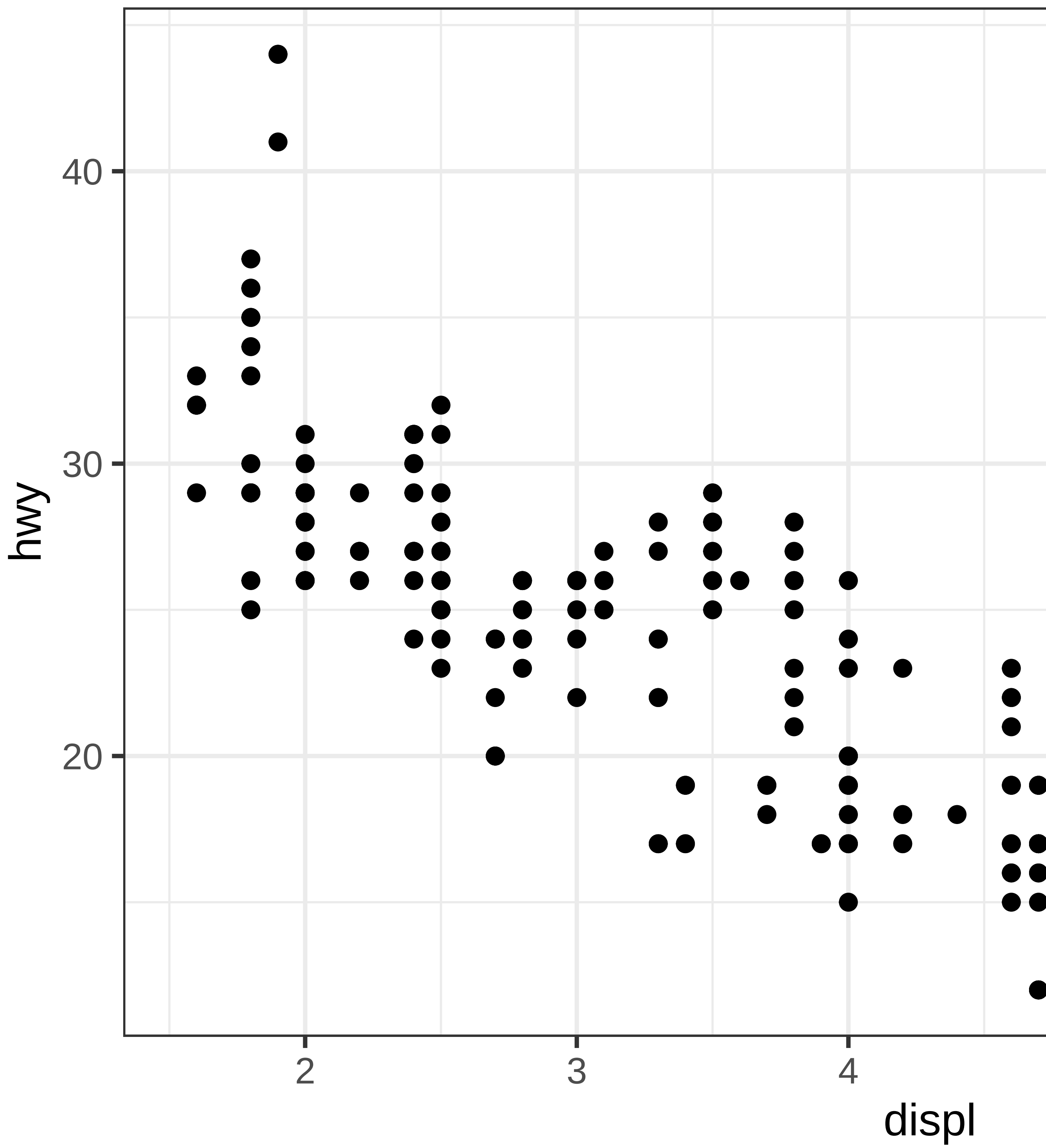
```r
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth()
```


35
30
25
20
hwy
2

35
30
25
20
15
hwy
2

35
30
25
20
15
hwy
2
3


40
30
20
hwy
2
3
4
5
displ

You can use the same idea to specify different `data` for each layer. Here, we use red points as well as open circles to highlight two-seater cars. The local data argument in `geom_point()` overrides the global data argument in `ggplot()` for that layer only.

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



hwy
40
30
20
2
3
4
displ

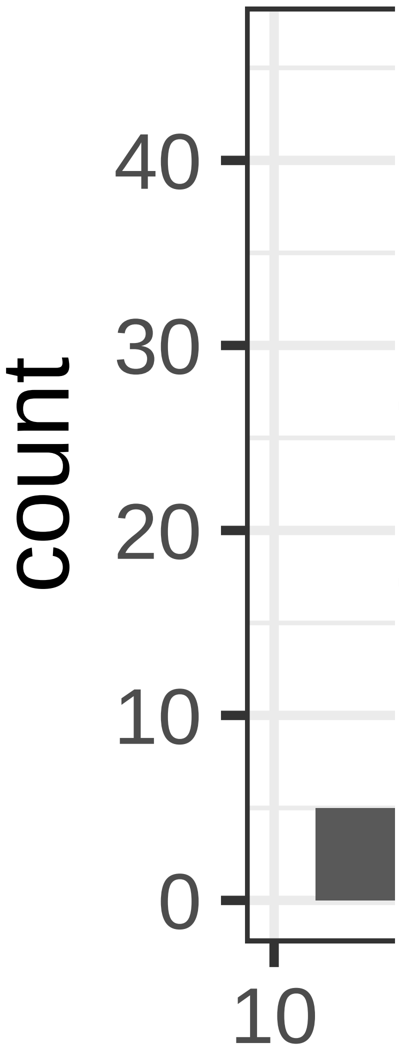
Geoms are the fundamental building blocks of ggplot2. You can completely transform the look of your plot by changing its geom, and different geoms can reveal different features of your data. For example, the histogram and density plot below reveal that the distribution of highway mileage is bimodal and right skewed while the boxplot reveals two potential outliers.

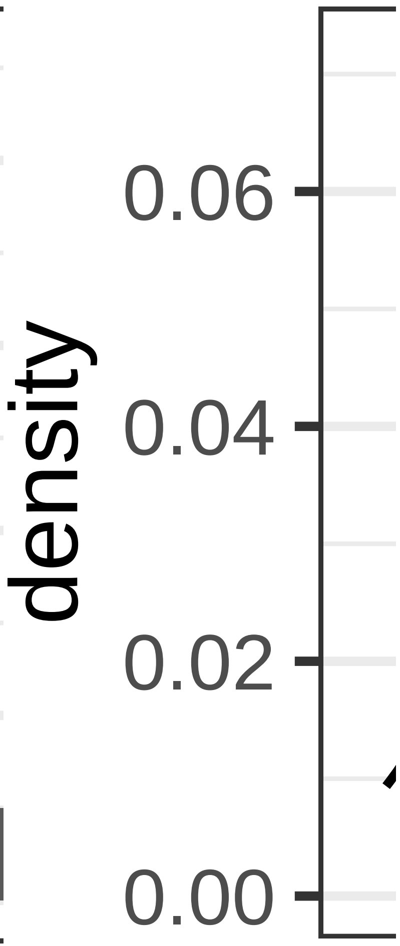
```
# Left
ggplot(mpg, aes(x = hwy)) +
  geom_histogram(binwidth = 2)

# Middle
ggplot(mpg, aes(x = hwy)) +
  geom_density()

# Right
ggplot(mpg, aes(x = hwy)) +
  geom_boxplot()
```

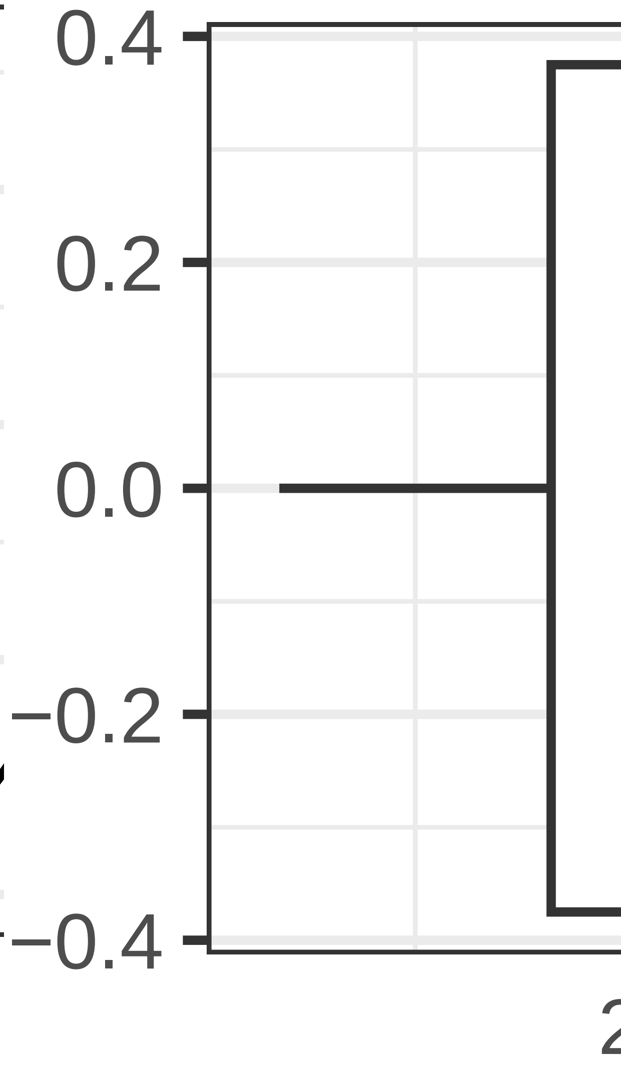
ggplot2 provides more than 40 geoms but these don't cover all possible plots one could make. If you need a different geom, we recommend looking into extension packages first to see if someone else has already implemented it (see https://exts.ggplot2.tidyverse.org/gallery/ for a sampling).

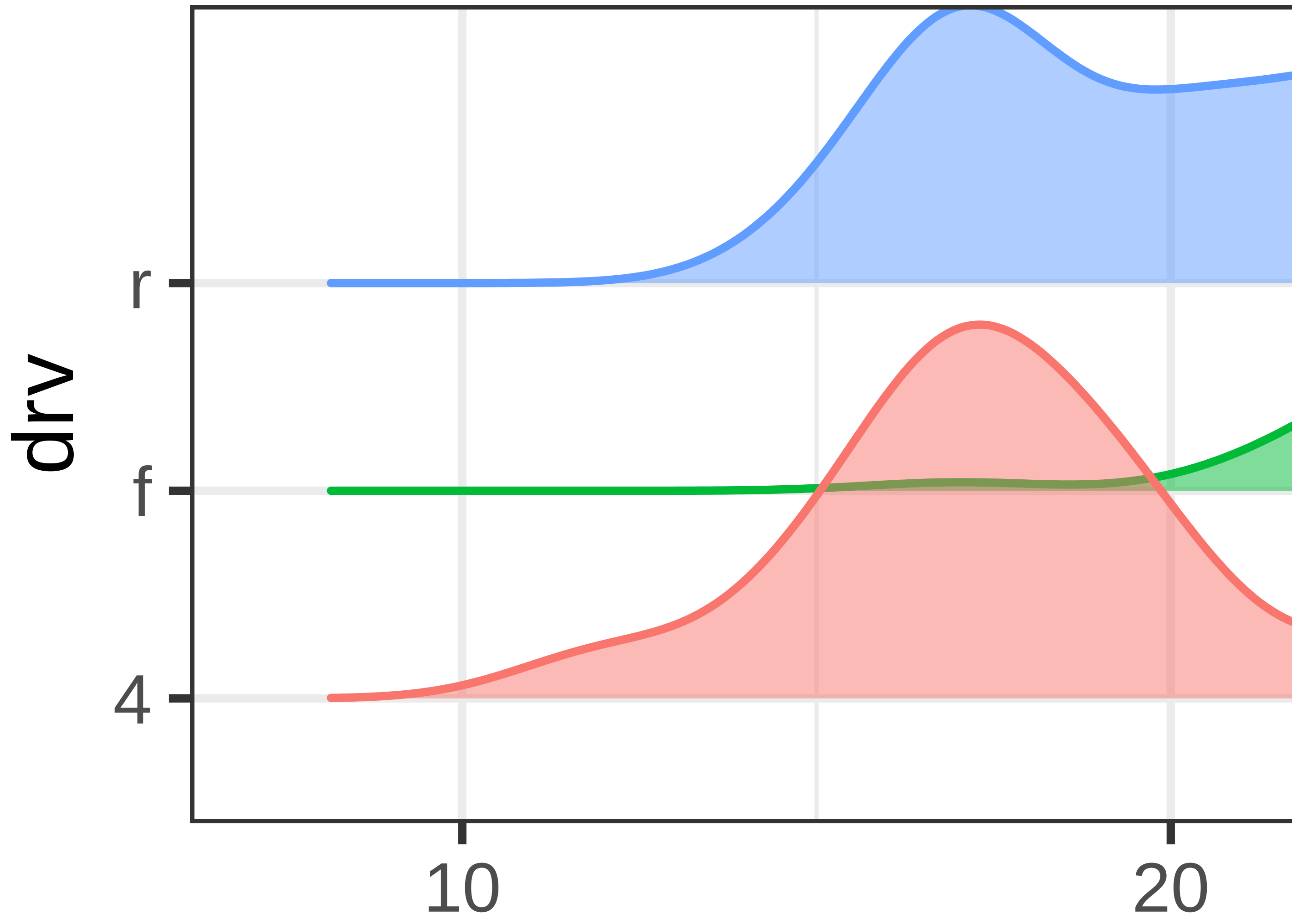
For example, the **ggridges** package (https://wilkelab.org/ggridges) is useful for making ridgeline plots, which can be useful for visualizing the density of a numerical variable for different levels of a categorical variable. In the following plot not only did we use a new geom (`geom_density_ridges()`), but we have also mapped the same variable to multiple aesthetics (`drv` to `y`, `fill`, and `color`) as well as set an aesthetic (`alpha = 0.5`) to make the density curves transparent.

```
library(ggridges)

ggplot(mpg, aes(x = hwy, y = drv, fill = drv, color = drv)) +
  geom_density_ridges(alpha = 0.5, show.legend = FALSE)
#> Picking joint bandwidth of 1.28
```


count
40
30
20
10
0
10


density
0.06
0.04
0.02
0.00


0.4
0.2
0.0
−0.2
−0.4



The best place to get a comprehensive overview of all of the geoms ggplot2 offers, as well as all functions in the package, is the reference page: https://ggplot2.tidyverse.org/reference. To learn more about any single geom, use the help (e.g., `?geom_smooth`).

2.3.1 Exercises

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

2. Earlier in this chapter we used `show.legend` without explaining it:

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

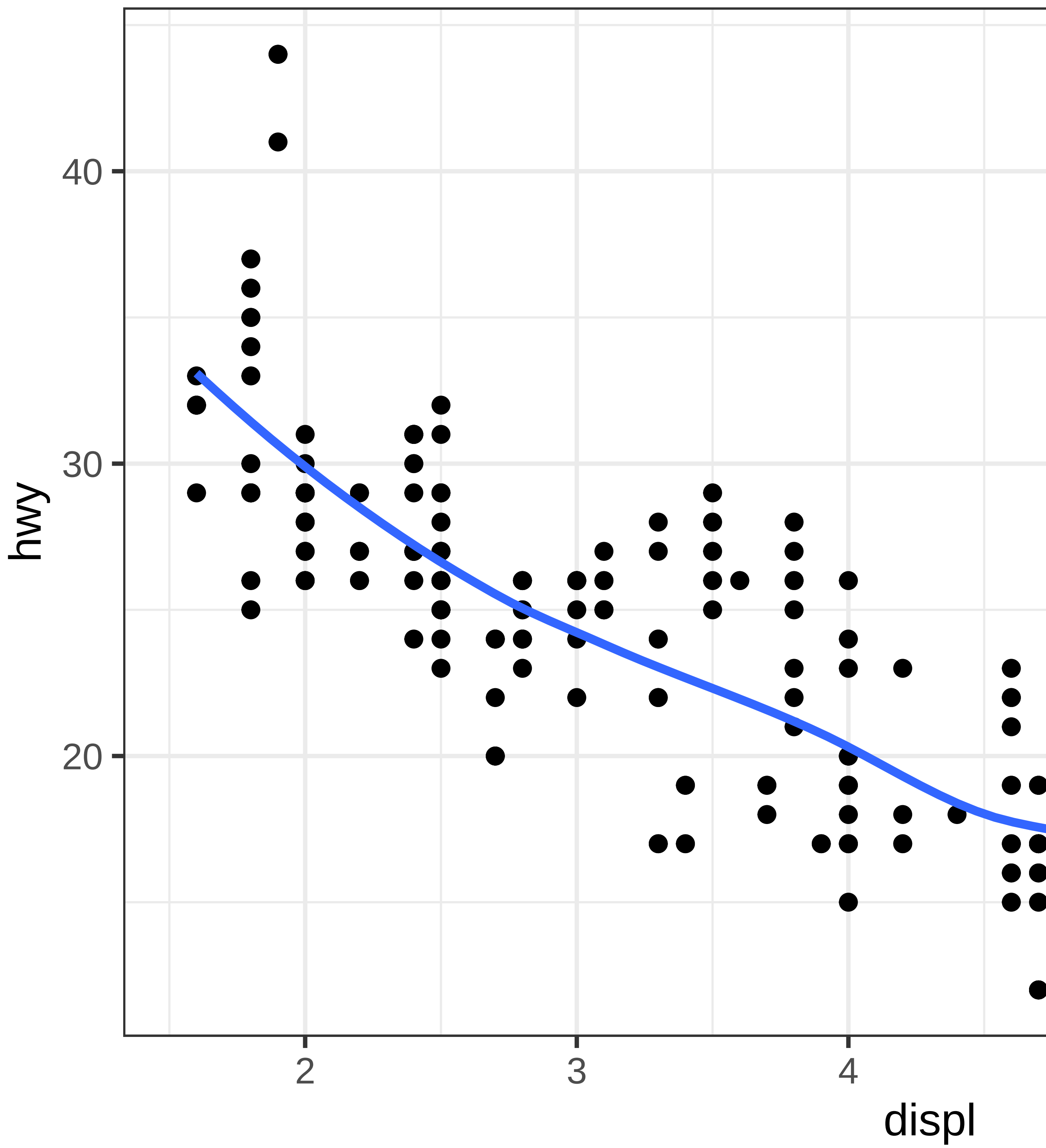
   What does `show.legend = FALSE` do here? What happens if you remove it? Why do you think we used it earlier?

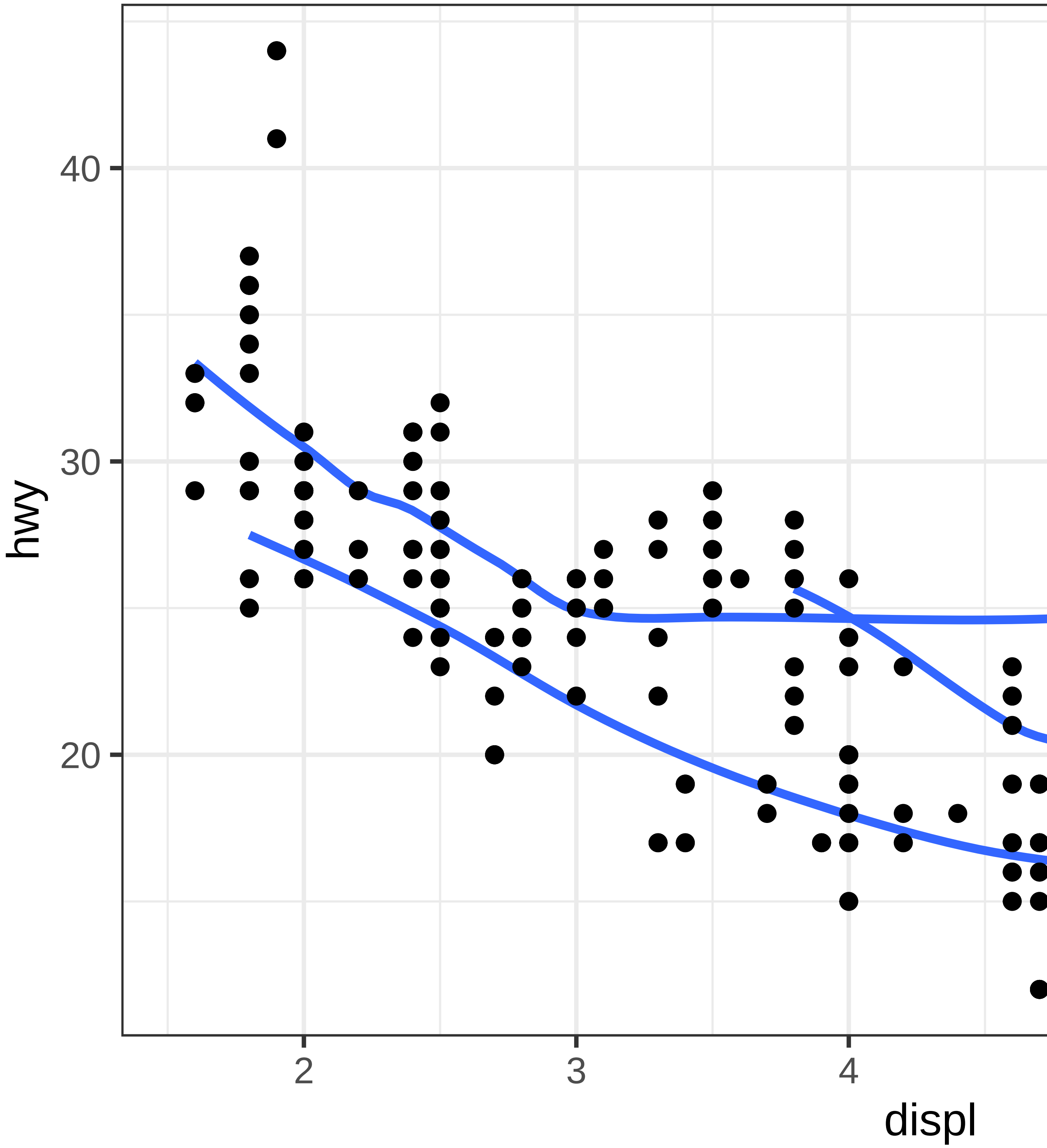
3. What does the `se` argument to `geom_smooth()` do?

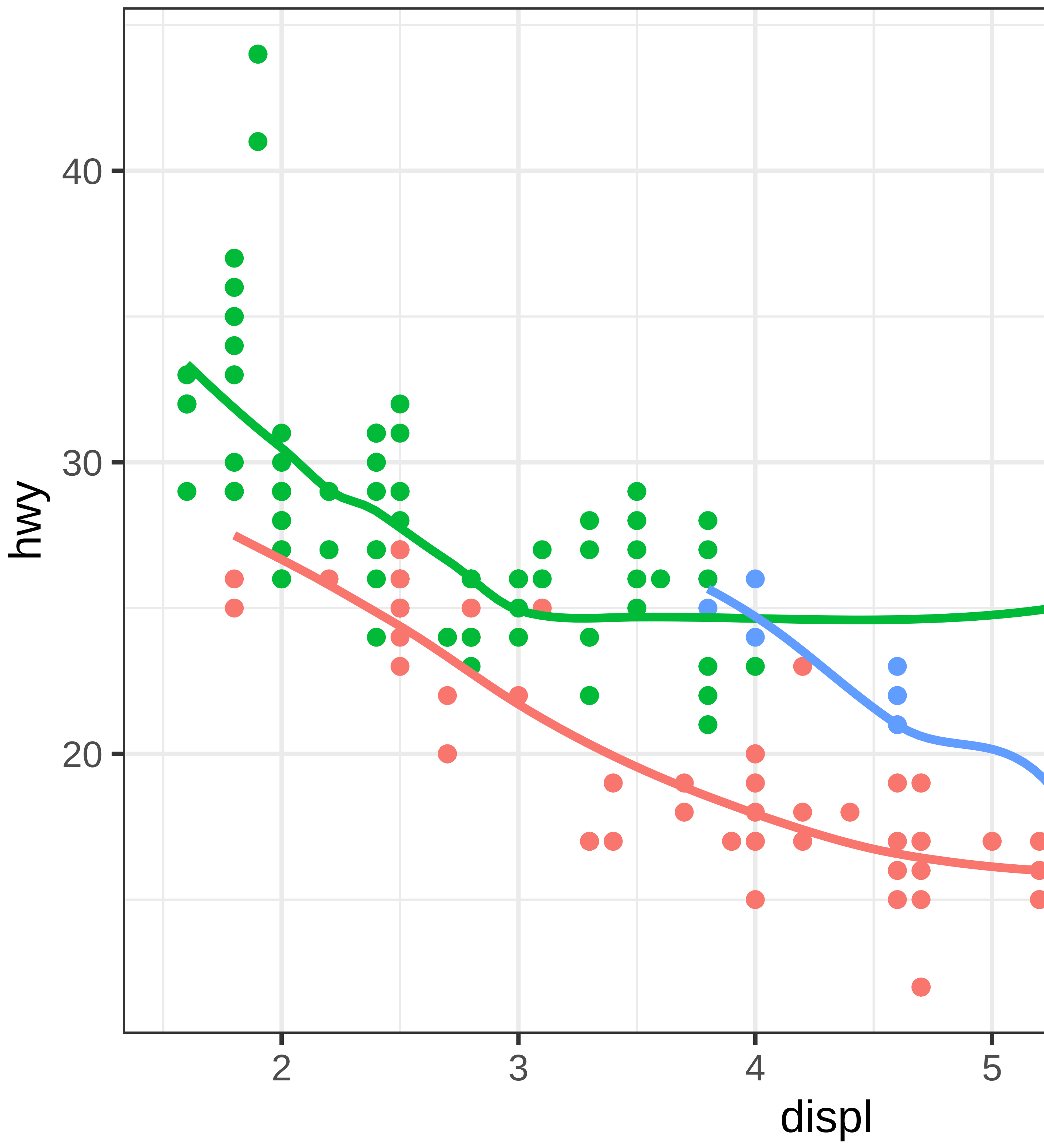
4. Recreate the R code necessary to generate and save the following graphs, and arrange them in a figure. Note that wherever a categorical variable is used in the plot, it's `drv`.

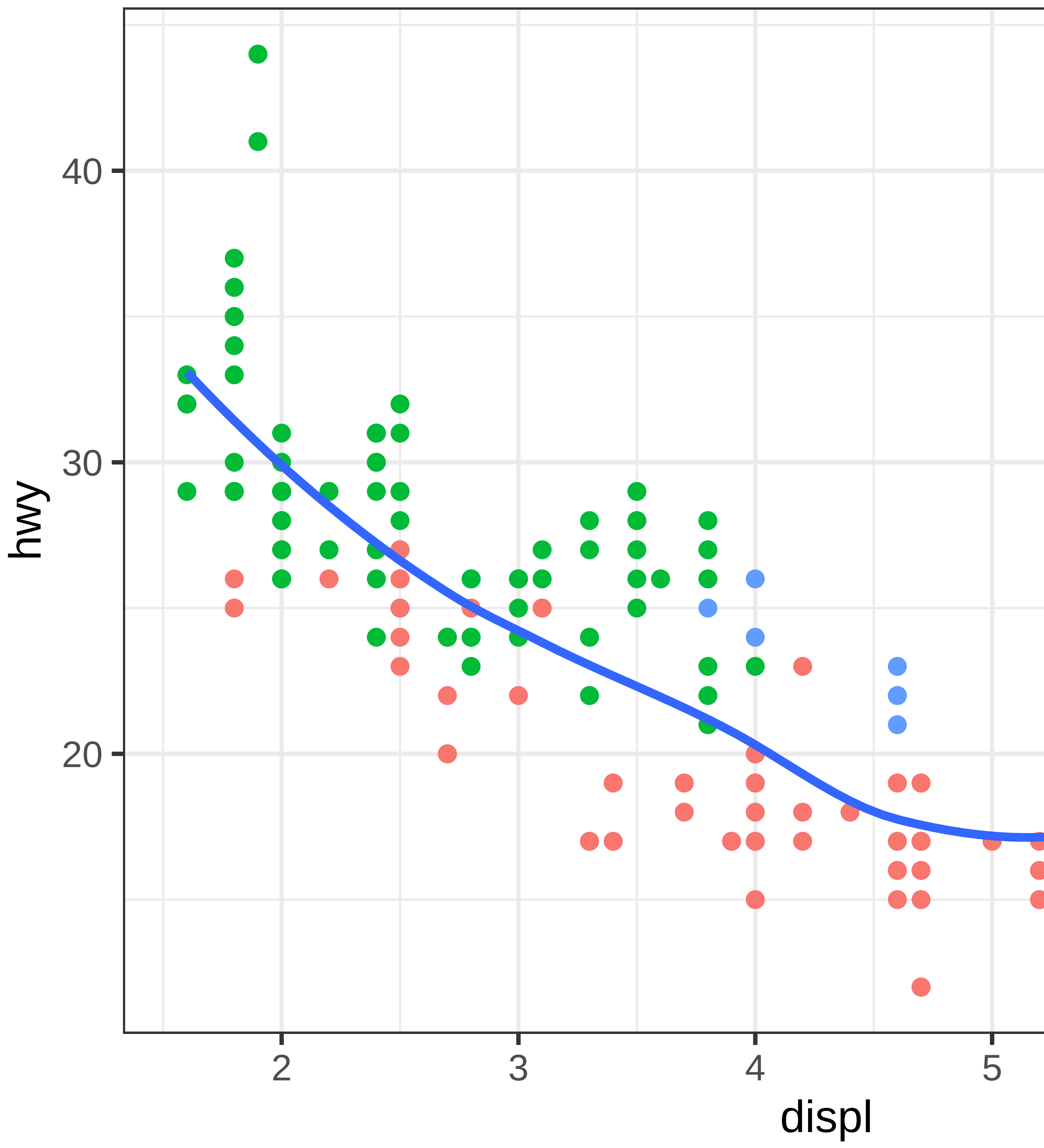
   ```
   #| message: false
   #| layout-ncol: 2
   #| fig-width: 3
   #| fig-alt: |
   #|   There are six scatterplots in this figure, arranged in a 3x2 grid.
   #|   In all plots highway fuel efficiency of cars are on the y-axis and
   #|   engine size is on the x-axis. The first plot shows all points in black
   #|   with a smooth curve overlaid on them. In the second plot points are
   #|   also all black, with separate smooth curves overlaid for each level of
   #|   drive train. On the third plot, points and the smooth curves are
   #|   represented in different colors for each level of drive train. In the
   #|   fourth plot the points are represented in different colors for each
   #|   level of drive train but there is only a single smooth line fitted to
   #|   the whole data. In the fifth plot, points are represented in different
   #|   colors for each level of drive train, and a separate smooth curve with
   #|   different line types are fitted to each level of drive train. And
   #|   finally in the sixth plot points are represented in different colors
   #|   for each level of drive train and they have a thick white border.

   ggplot(mpg, aes(x = displ, y = hwy)) +
     geom_point() +
     geom_smooth(se = FALSE)
   #> `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
   ggplot(mpg, aes(x = displ, y = hwy)) +
     geom_smooth(aes(group = drv), se = FALSE) +
     geom_point()
   #> `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
   ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
     geom_point() +
   ```

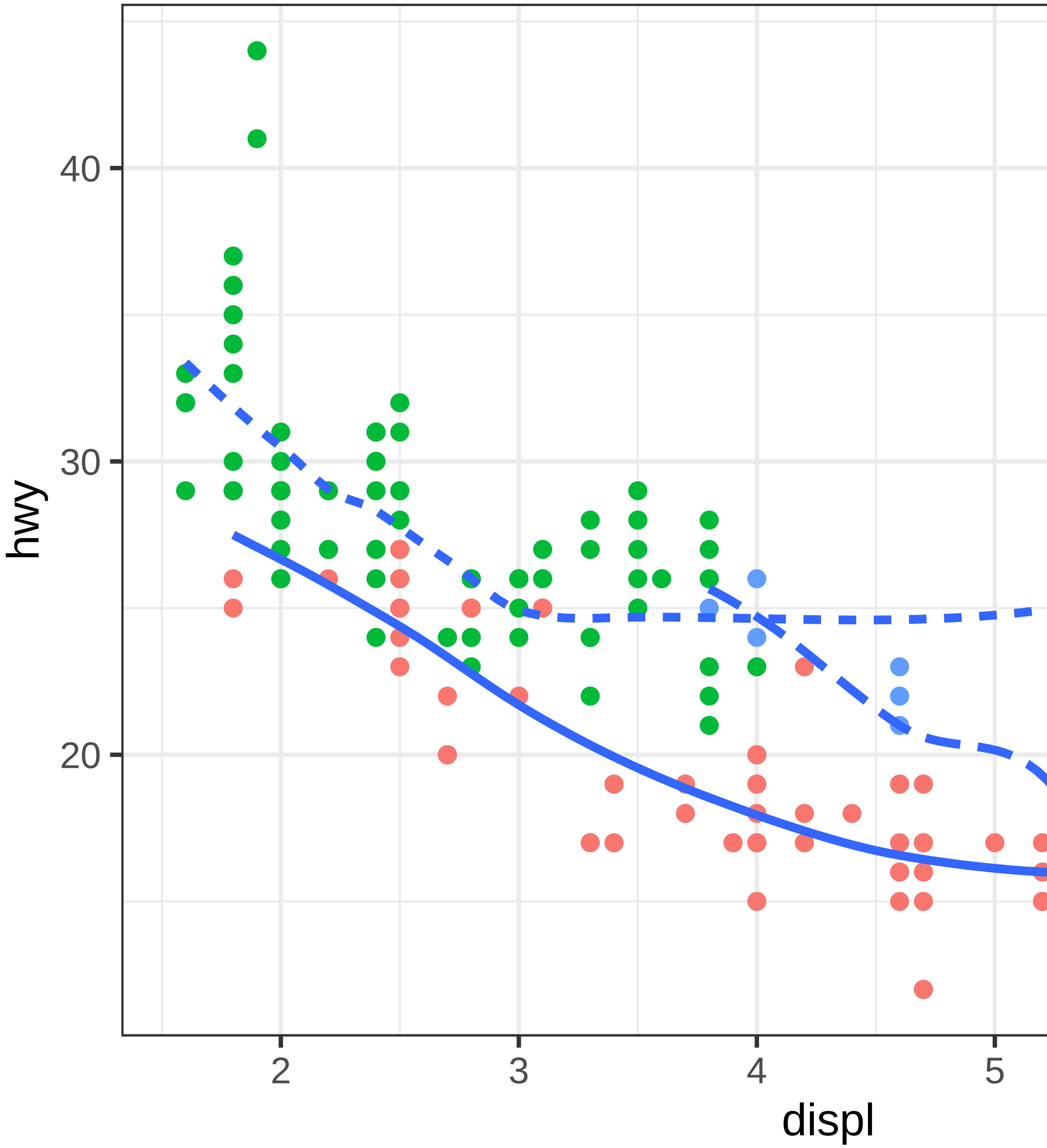
```
  geom_smooth(se = FALSE)
#> `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = drv)) +
  geom_smooth(se = FALSE)
#> `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = drv)) +
  geom_smooth(aes(linetype = drv), se = FALSE)
#> `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(size = 4, color = "white") +
  geom_point(aes(color = drv))
```

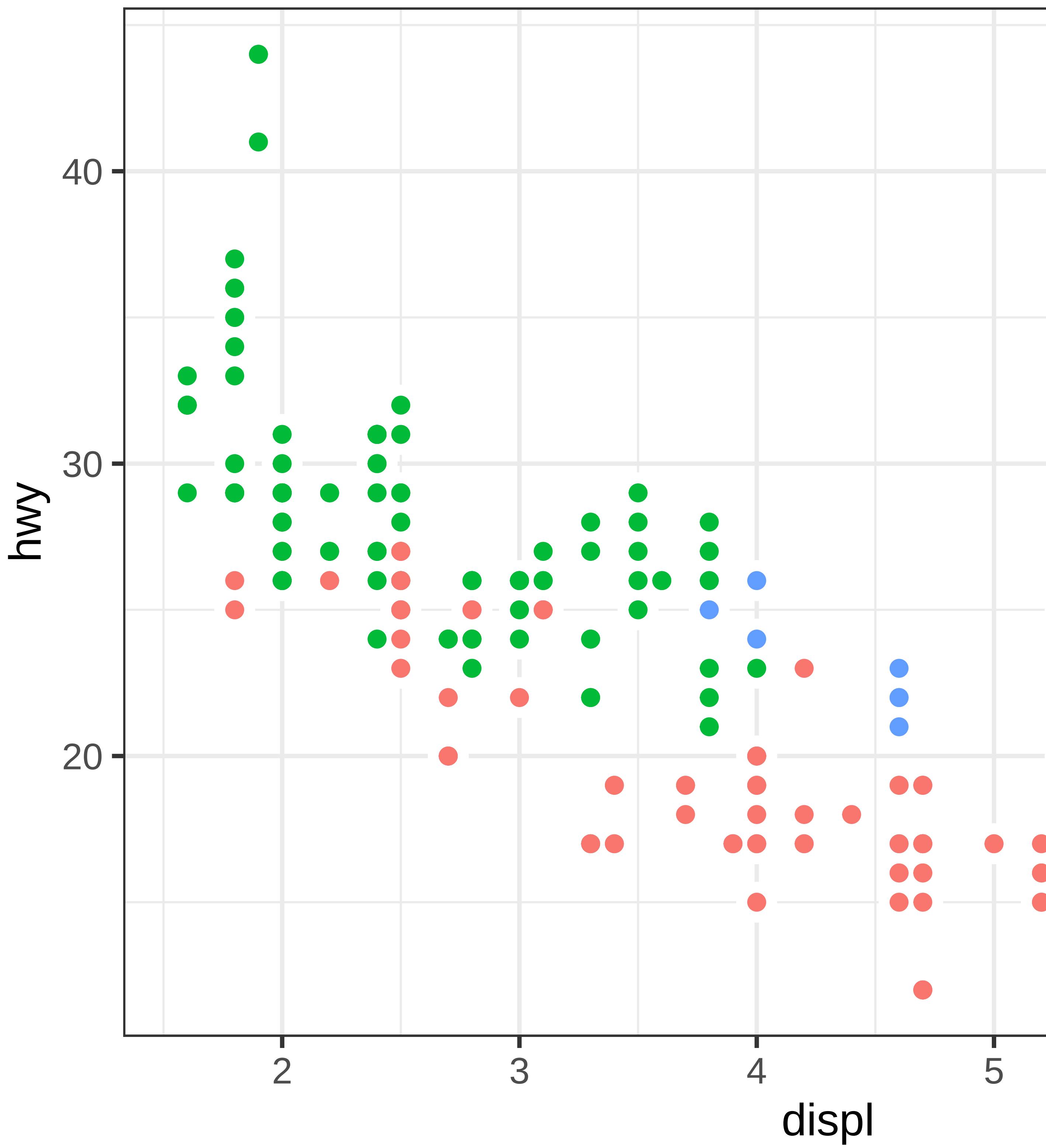

40
30
20
hwy
2
3
4
displ


hwy
40
30
20
2
3
4
displ


40
30
20
hwy
2
3
4
5
displ


hwy
40
30
20
2
3
4
5
displ

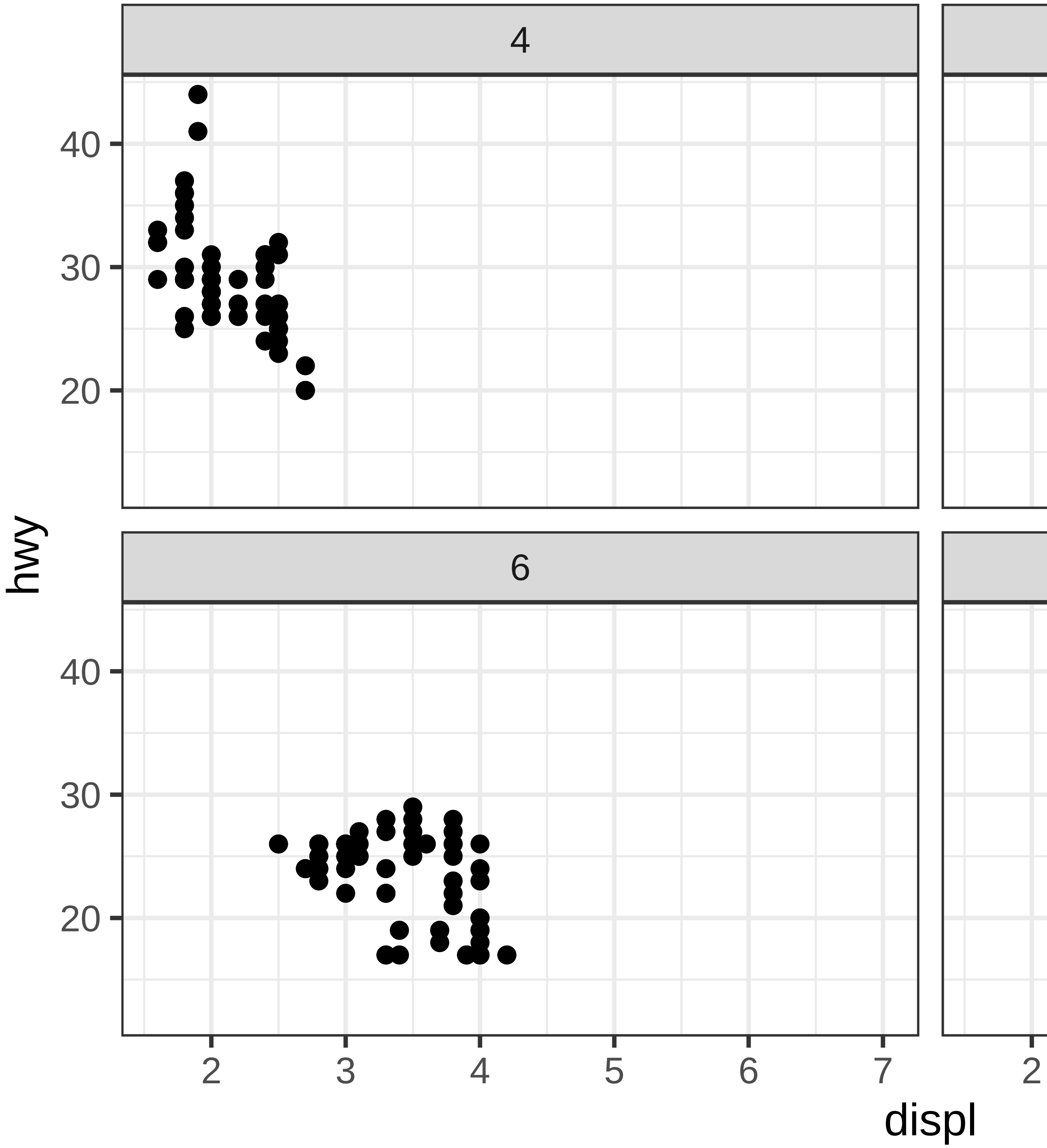

hwy
40
30
20
2
3
4
5
displ


40
30
20
hwy
2
3
4
5
displ

2.4 Facets

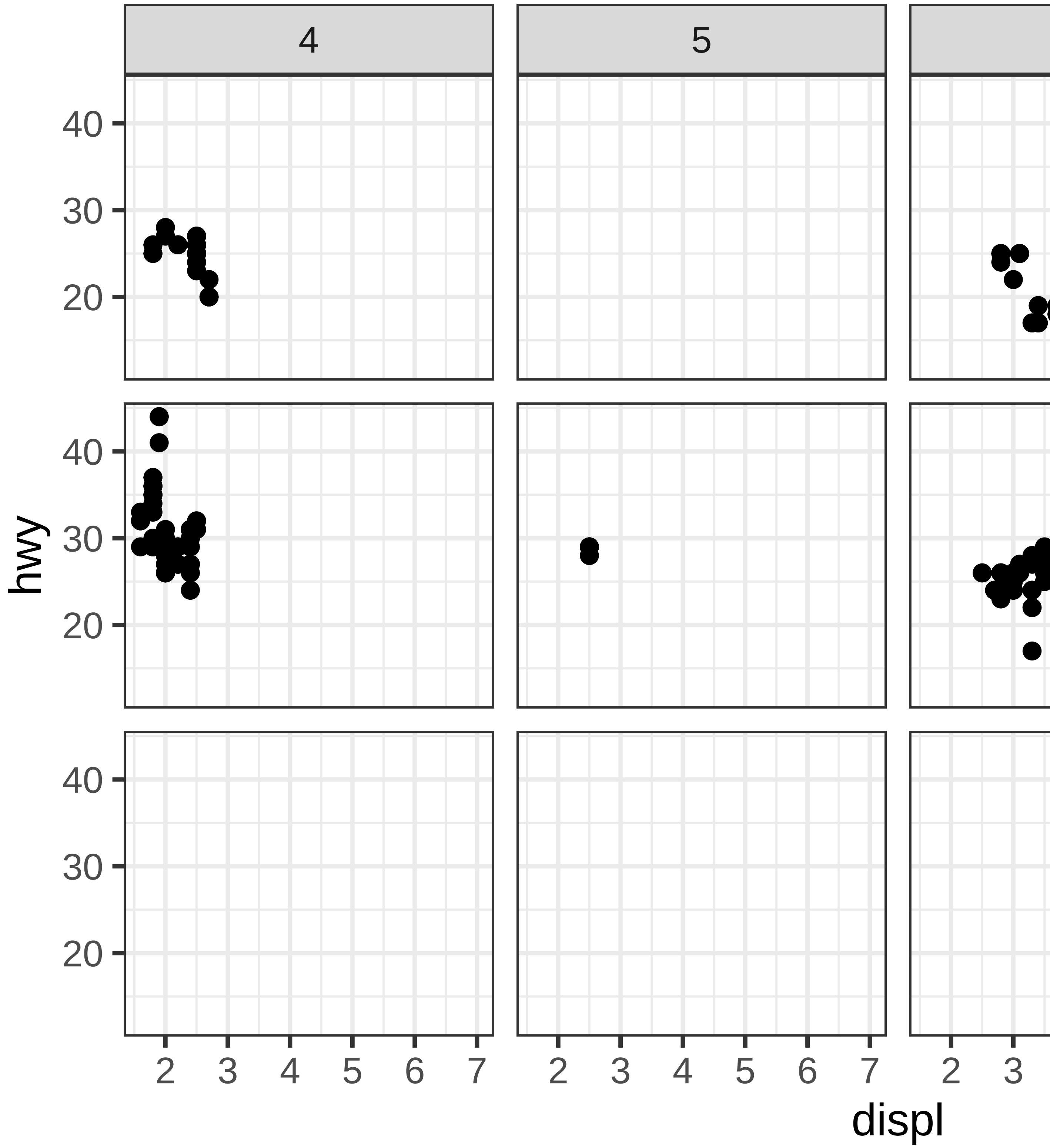
In Chapter 1 you learned about faceting with `facet_wrap()`, which splits a plot into subplots that each display one subset of the data based on a categorical variable.

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


4
6
hwy
displ
40
30
20
2
3
4
5
6
7

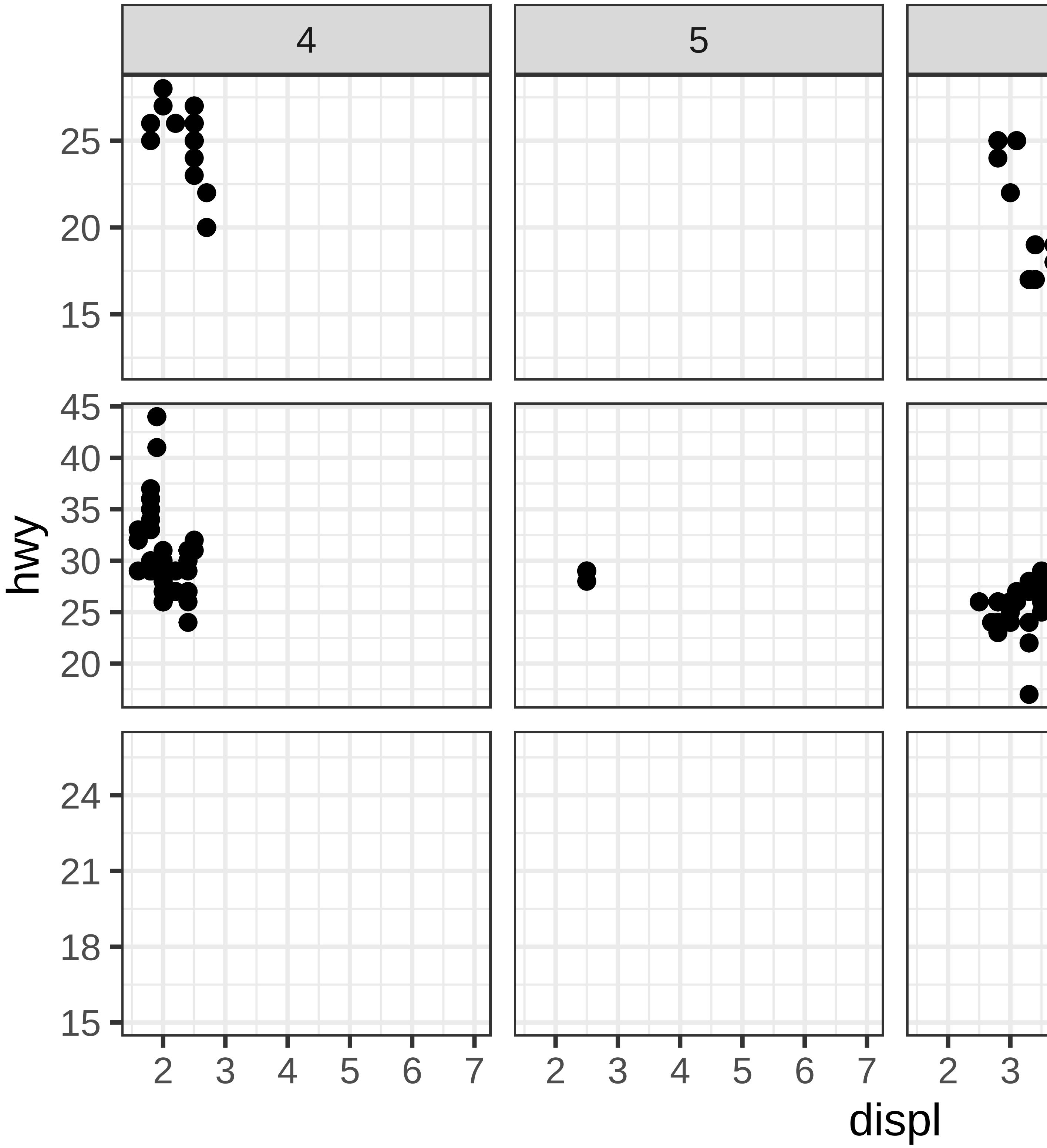
To facet your plot with the combination of two variables, switch from `facet_wrap()` to `facet_grid()`. The first argument of `facet_grid()` is also a formula, but now it's a double sided formula: `rows ~ cols`.

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


4
5
hwy
40
30
20
2
3
4
5
6
7
displ

By default each of the facets share the same scale and range for x and y axes. This is useful when you want to compare data across facets but it can be limiting when you want to visualize the relationship within each facet better. Setting the `scales` argument in a faceting function to `"free"` will allow for different axis scales across both rows and columns, `"free_x"` will allow for different scales across rows, and `"free_y"` will allow for different scales across columns.

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


4
5
hwy
displ
25
20
15
45
40
35
30
25
20
24
21
18
15
2
3
4
5
6
7

2.4.1 Exercises

1. What happens if you facet on a continuous variable?
2. What do the empty cells in the plot above with `facet_grid(drv ~ cyl)` mean? Run the following code. How do they relate to the resulting plot?

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

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

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

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

4. Take the first faceted plot in this section:

   ```r
   ggplot(mpg) +
     geom_point(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?

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

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

   ggplot(mpg, aes(x = displ)) +
     geom_histogram() +
   ```

```
  facet_grid(. ~ drv)
```

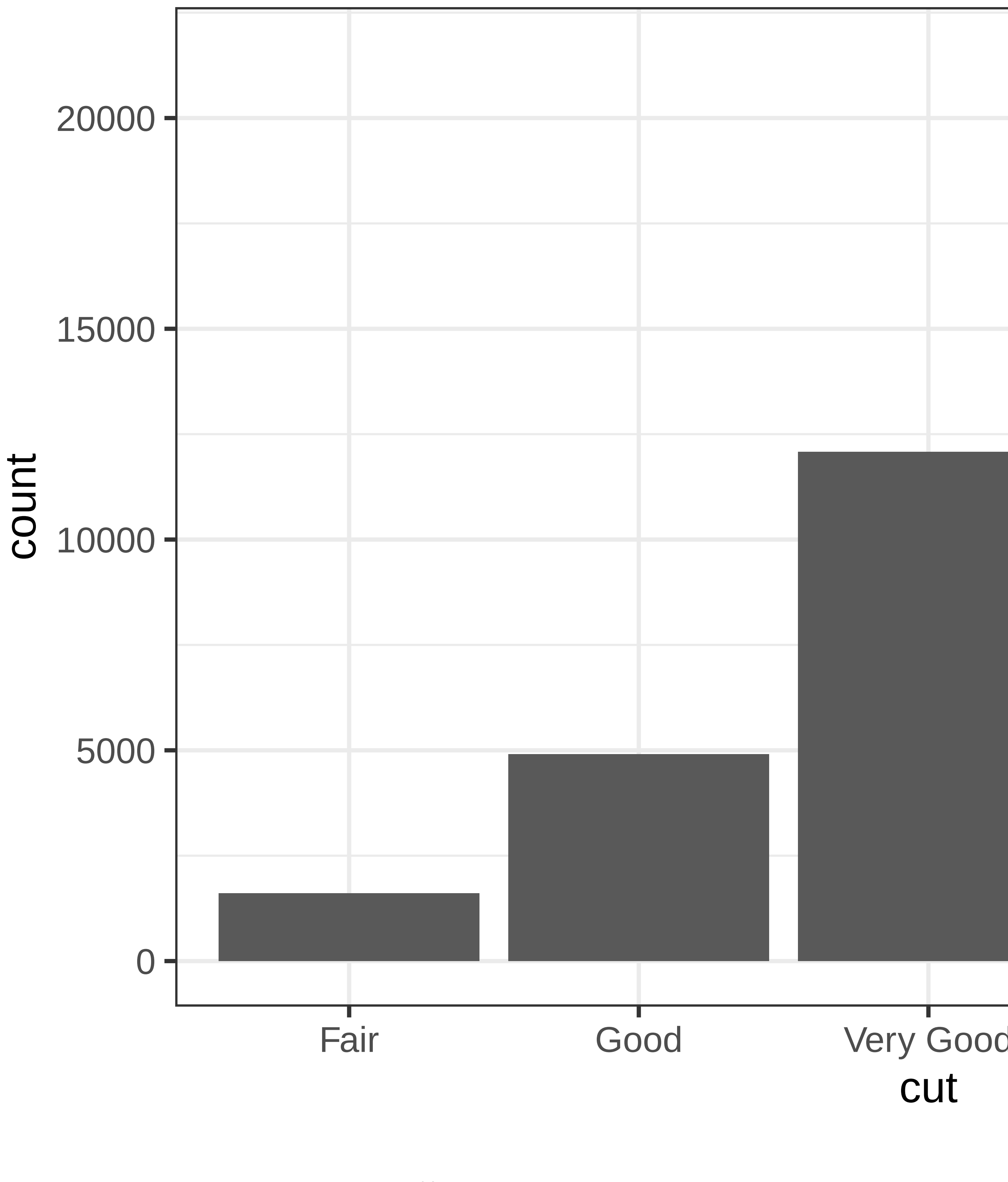
7. Recreate the following plot using `facet_wrap()` instead of `facet_grid()`. How do the positions of the facet labels change?

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

2.5 Statistical transformations

Consider a basic bar chart, drawn with `geom_bar()` or `geom_col()`. The following chart displays the total number of diamonds in the `diamonds` dataset, grouped by `cut`. The `diamonds` dataset is in the ggplot2 package and contains information on ~54,000 diamonds, including the `price`, `carat`, `color`, `clarity`, and `cut` of each diamond. The chart shows that more diamonds are available with high quality cuts than with low quality cuts.

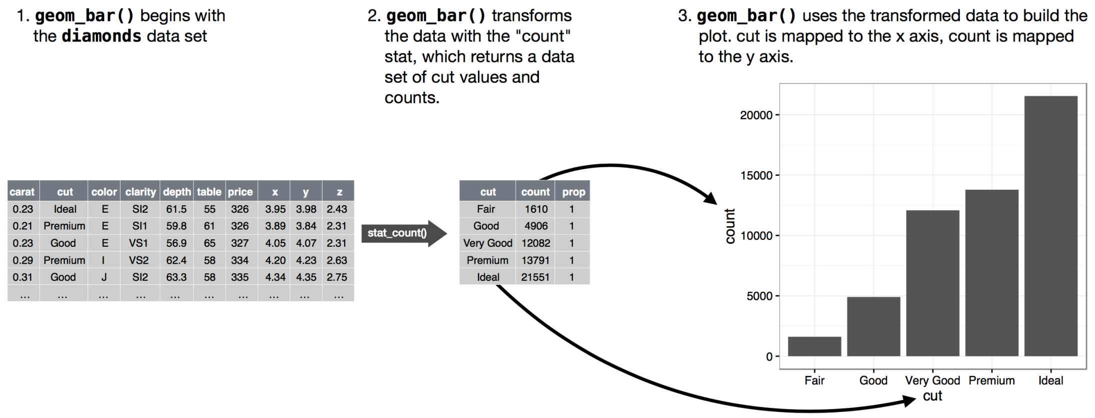
```
ggplot(diamonds, aes(x = cut)) +
  geom_bar()
```


count
20000
15000
10000
5000
0
Fair
Good
Very Good
cut

On the x-axis, the chart displays `cut`, a variable from `diamonds`. On the y-axis, it displays count, but count is not a variable in `diamonds`! Where does count come from? 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 the five-number summary of the distribution and then display that summary as a specially formatted box.

The algorithm used to calculate new values for a graph is called a **stat**, short for statistical transformation. Figure 2.2 shows how this process works with `geom_bar()`.


1. **geom_bar()** begins with the **diamonds** data set
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.
stat_count()
count
20000
15000
10000
5000
0
Fair
Good
Very Good
Premium
Ideal
cut

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
...	...	...	...	...	...	...	...	...	...

cut	count	prop
Fair	1610	1
Good	4906	1
Very Good	12082	1
Premium	13791	1
Ideal	21551	1

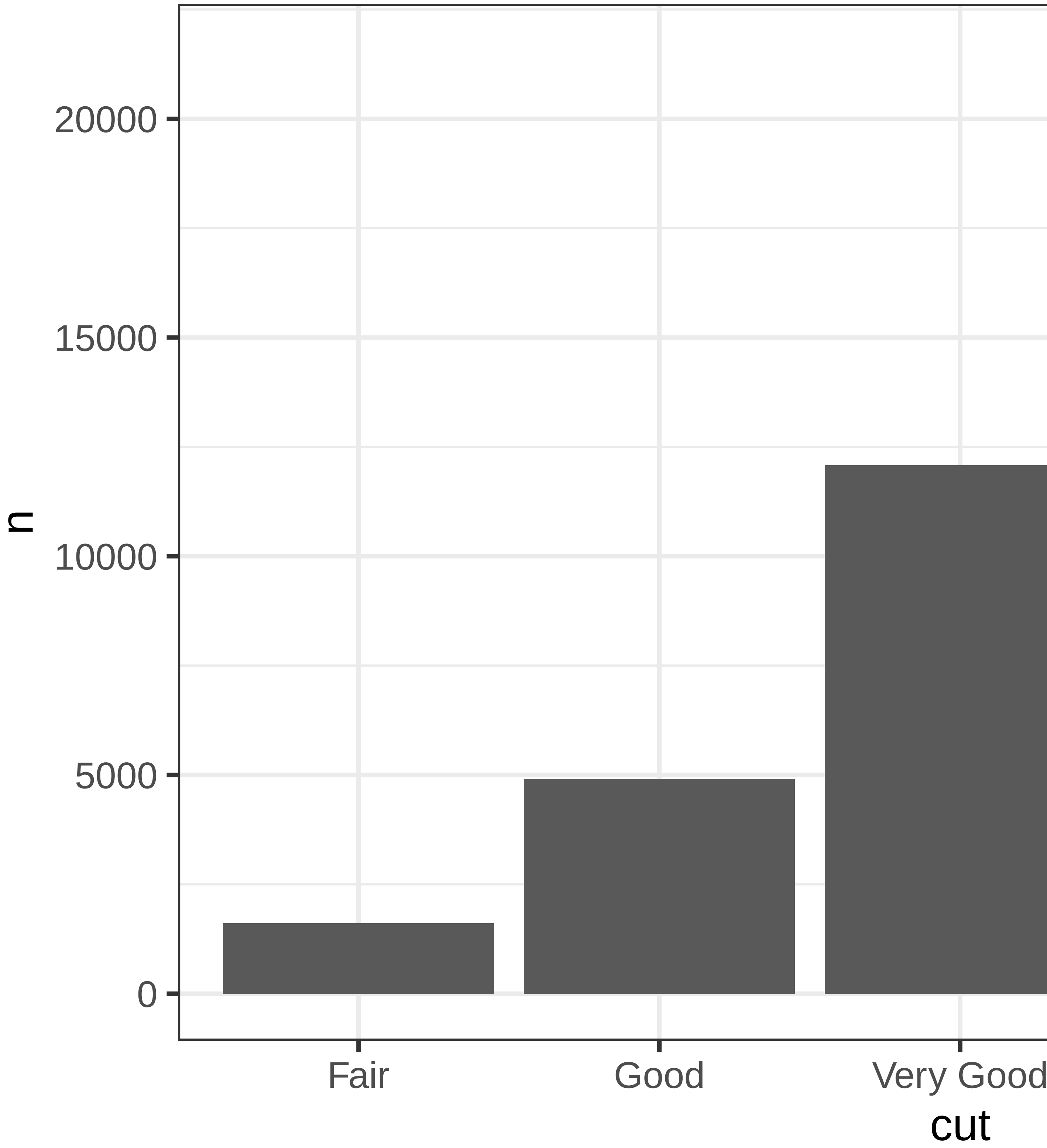
Figure 2.2: When creating a bar chart we first start with the raw data, then aggregate it to count the number of observations in each bar, and finally map those computed variables to plot aesthetics.

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

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. However, there are three reasons why you might need to use a stat explicitly:

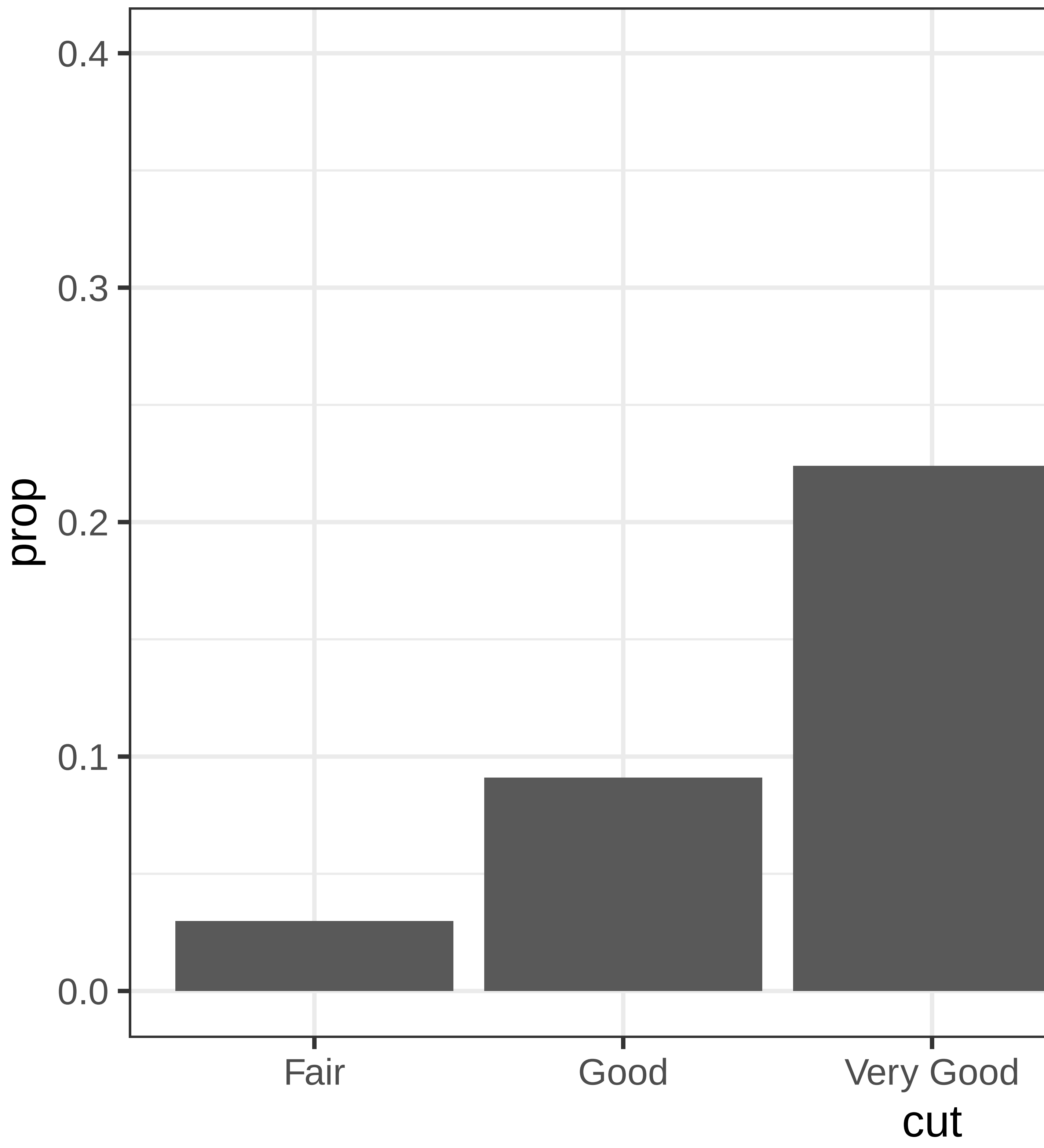
1. You might want to override the default stat. In the code below, we change the stat of `geom_bar()` from count (the default) to identity. This lets us map the height of the bars to the raw values of a y variable.

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


20000
15000
10000
5000
0
n
Fair
Good
Very Good
cut

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 proportions, rather than counts:

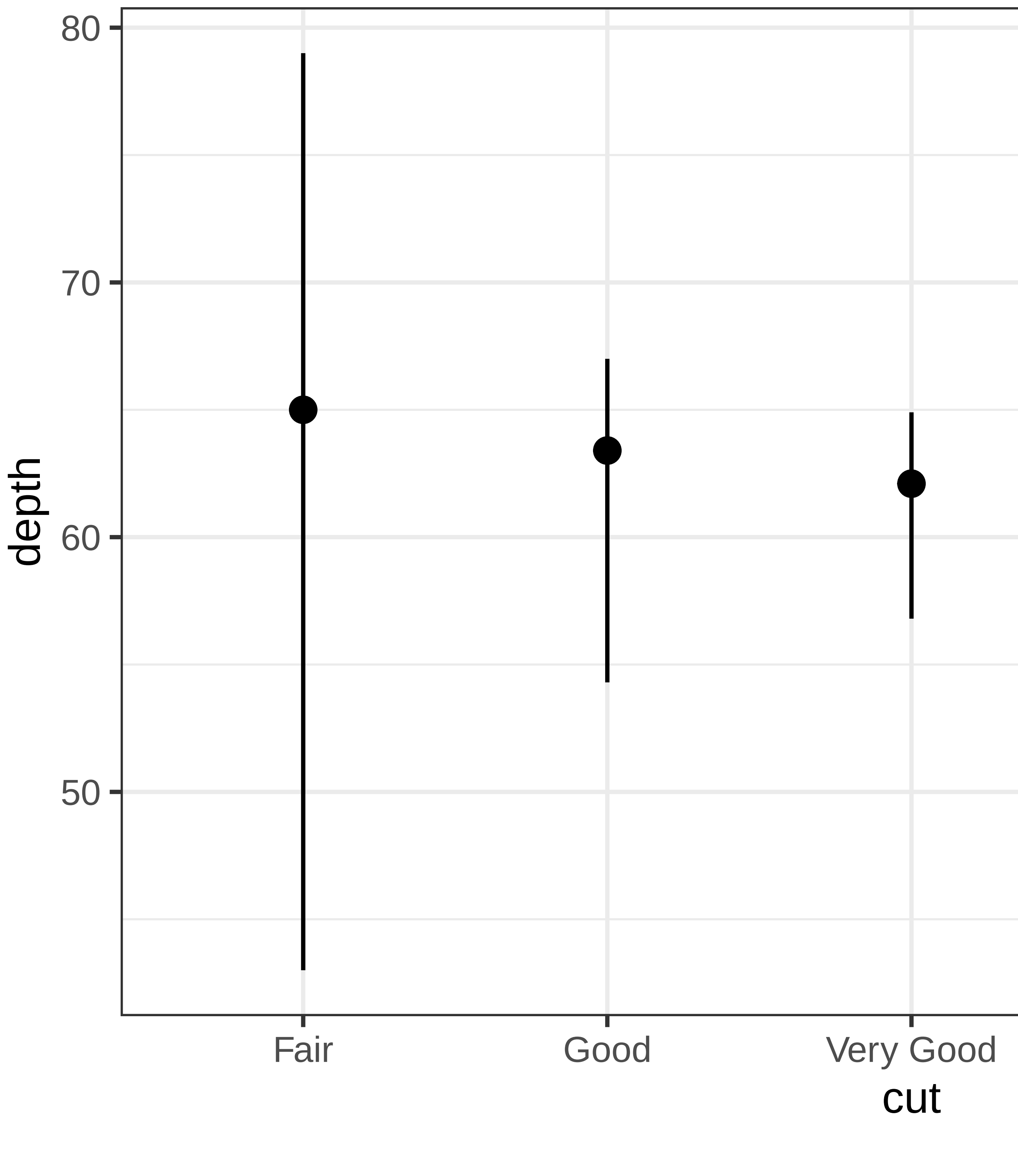
```r
ggplot(diamonds, aes(x = cut, y = after_stat(prop), group = 1)) +
  geom_bar()
```


0.4
0.3
0.2
0.1
0.0
prop
Fair
Good
Very Good
cut

To find the possible variables that can be computed by the stat, look for the section titled "computed variables" in the help for `geom_bar()`.

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

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


80
70
60
50
depth
Fair
Good
Very Good
cut

ggplot2 provides more than 20 stats for you to use. Each stat is a function, so you can get help in the usual way, e.g., `?stat_bin`.

2.5.1 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?
2. What does `geom_col()` do? How is it different from `geom_bar()`?
3. Most geoms and stats come in pairs that are almost always used in concert. Make a list of all the pairs. What do they have in common? (Hint: Read through the documentation.)
4. What variables does `stat_smooth()` compute? What arguments control its behavior?
5. In our proportion bar chart, we needed to set `group = 1`. Why? In other words, what is the problem with these two graphs?

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

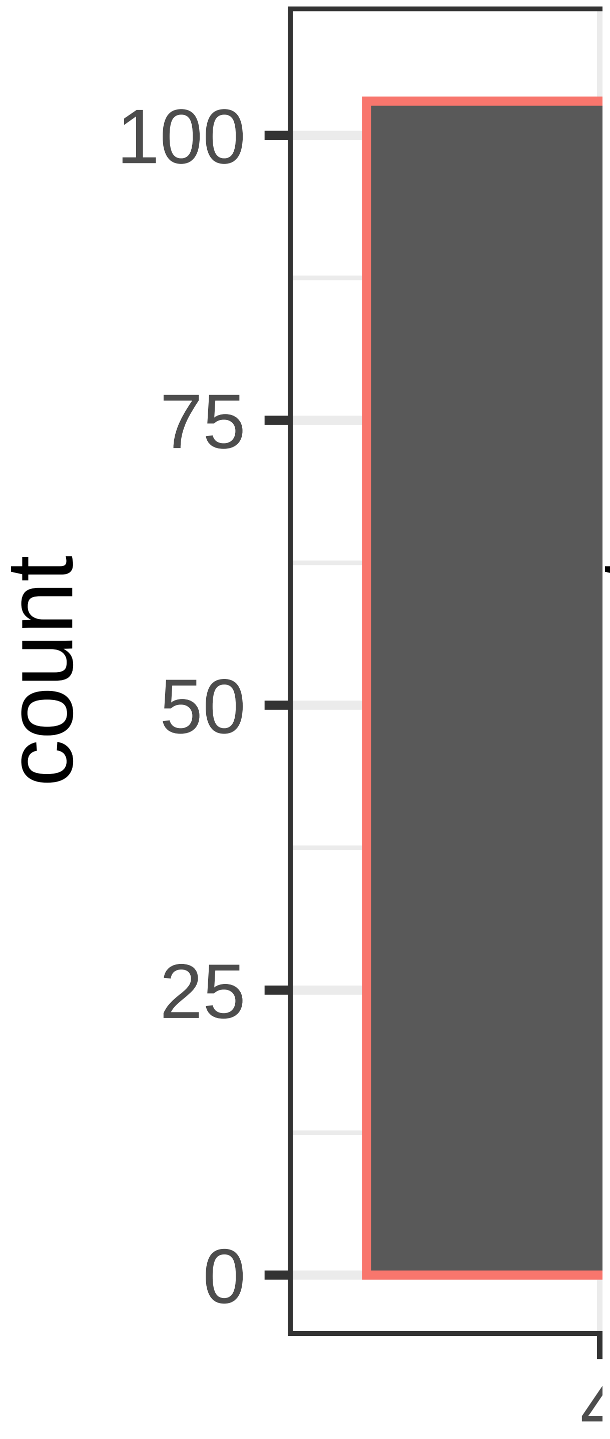
2.6 Position adjustments

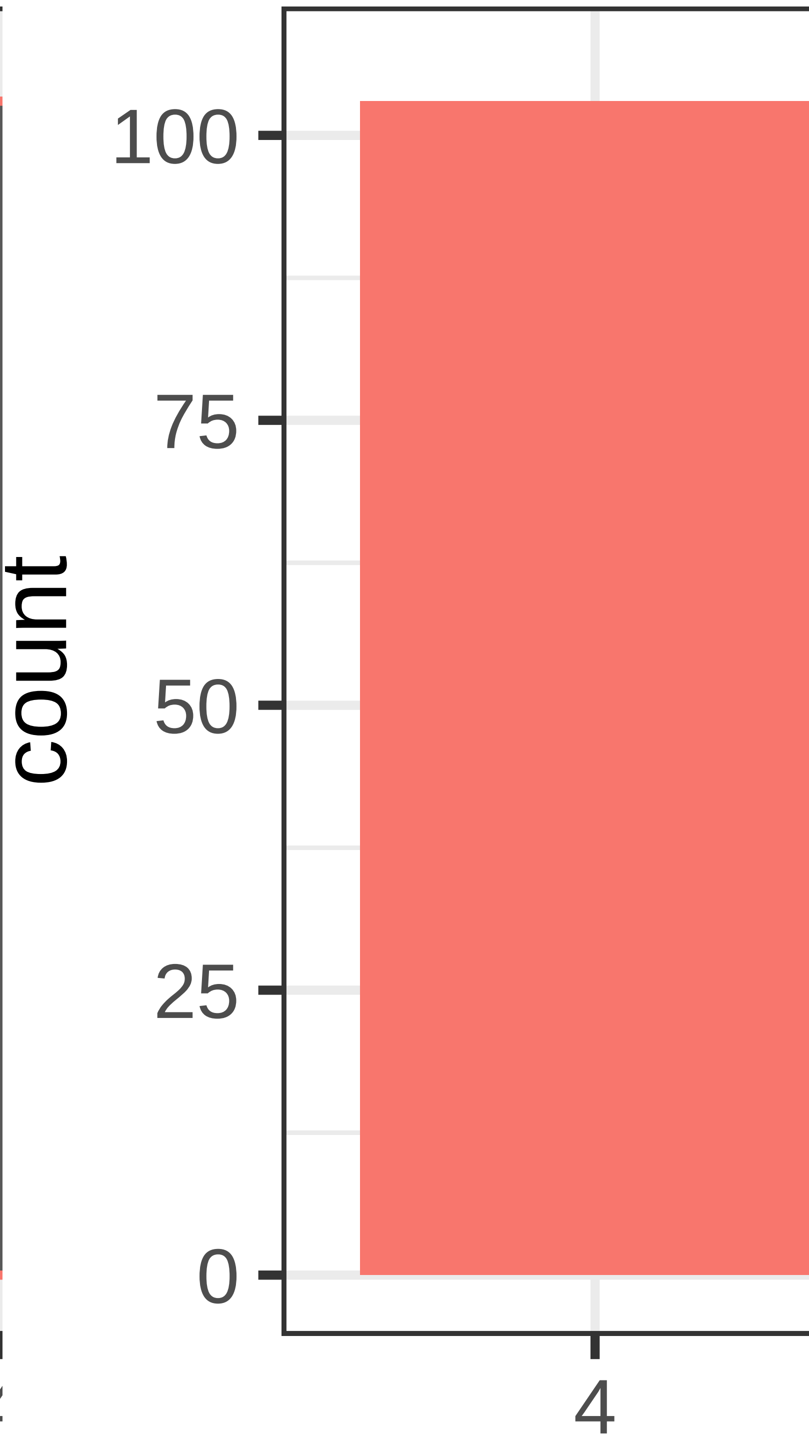
There's one more piece of magic associated with bar charts. You can color a bar chart using either the `color` aesthetic, or, more usefully, the `fill` aesthetic:

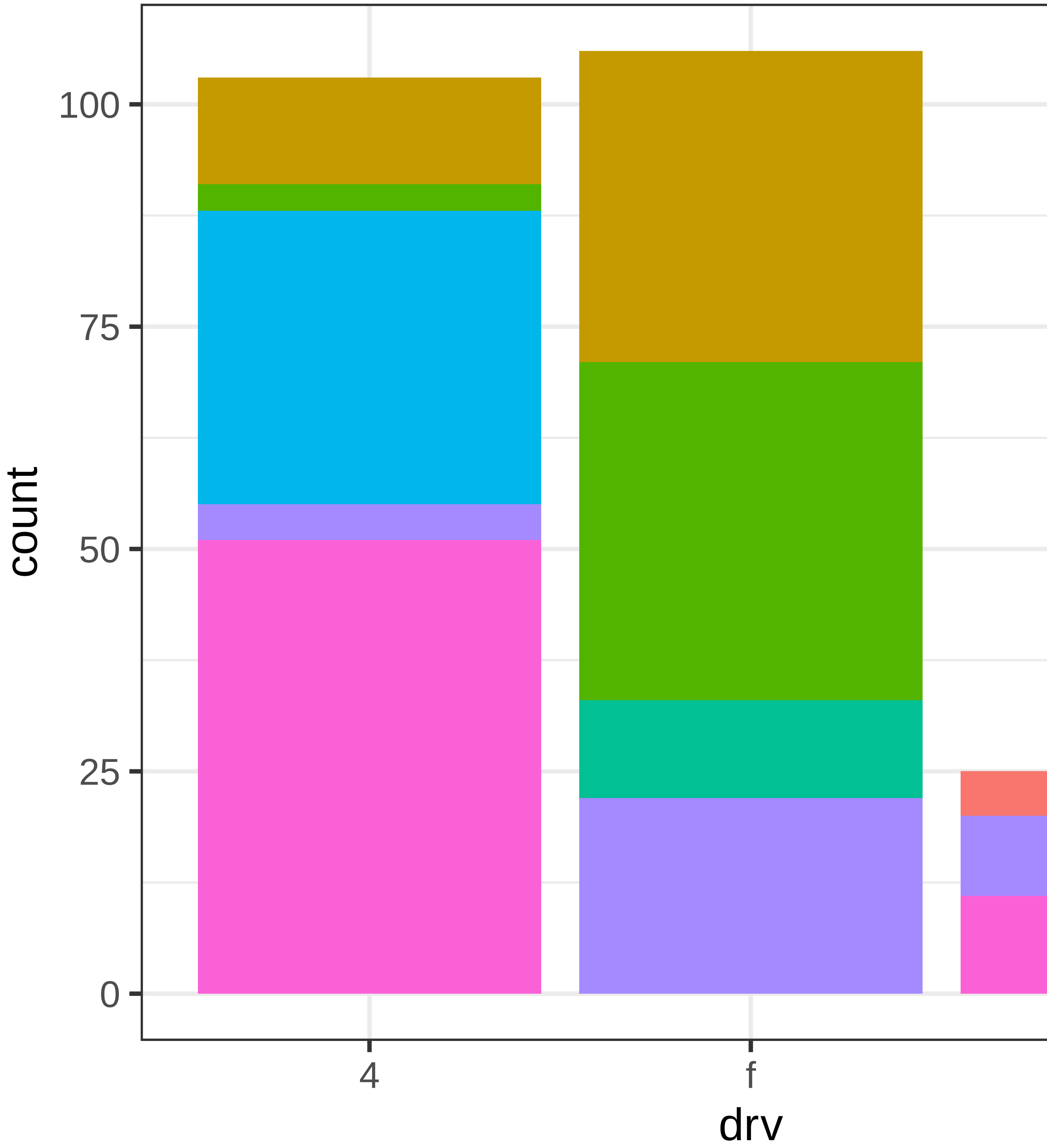
```
# Left
ggplot(mpg, aes(x = drv, color = drv)) +
  geom_bar()

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

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

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


count
100
75
50
25
0
4


count
100
75
50
25
0
4


count
100
75
50
25
0
4
f
drv

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

- `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. 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`.

  ```
  # Left
  ggplot(mpg, aes(x = drv, fill = class)) +
    geom_bar(alpha = 1/5, position = "identity")

  # Right
  ggplot(mpg, aes(x = drv, color = class)) +
    geom_bar(fill = NA, position = "identity")
  ```

  The identity position adjustment is more useful for 2d geoms, like points, where it is the default.

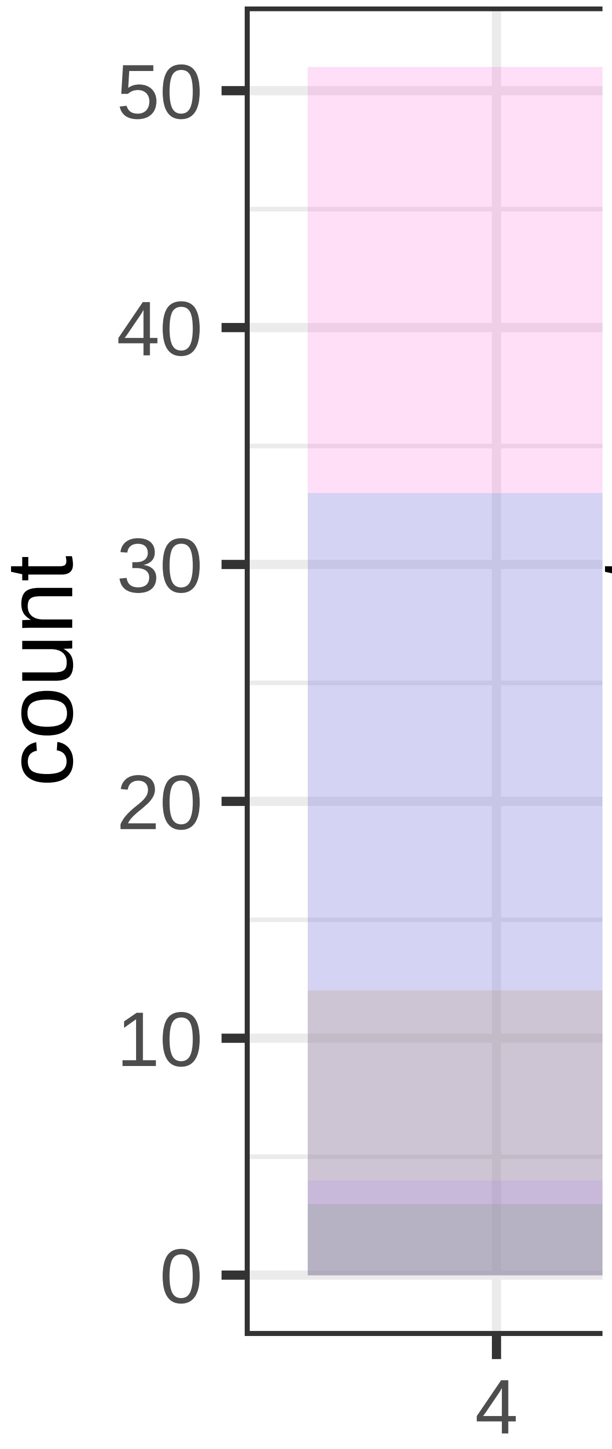
- `position = "fill"` works like stacking, but makes each set of stacked bars the same height. This makes it easier to compare proportions across groups.

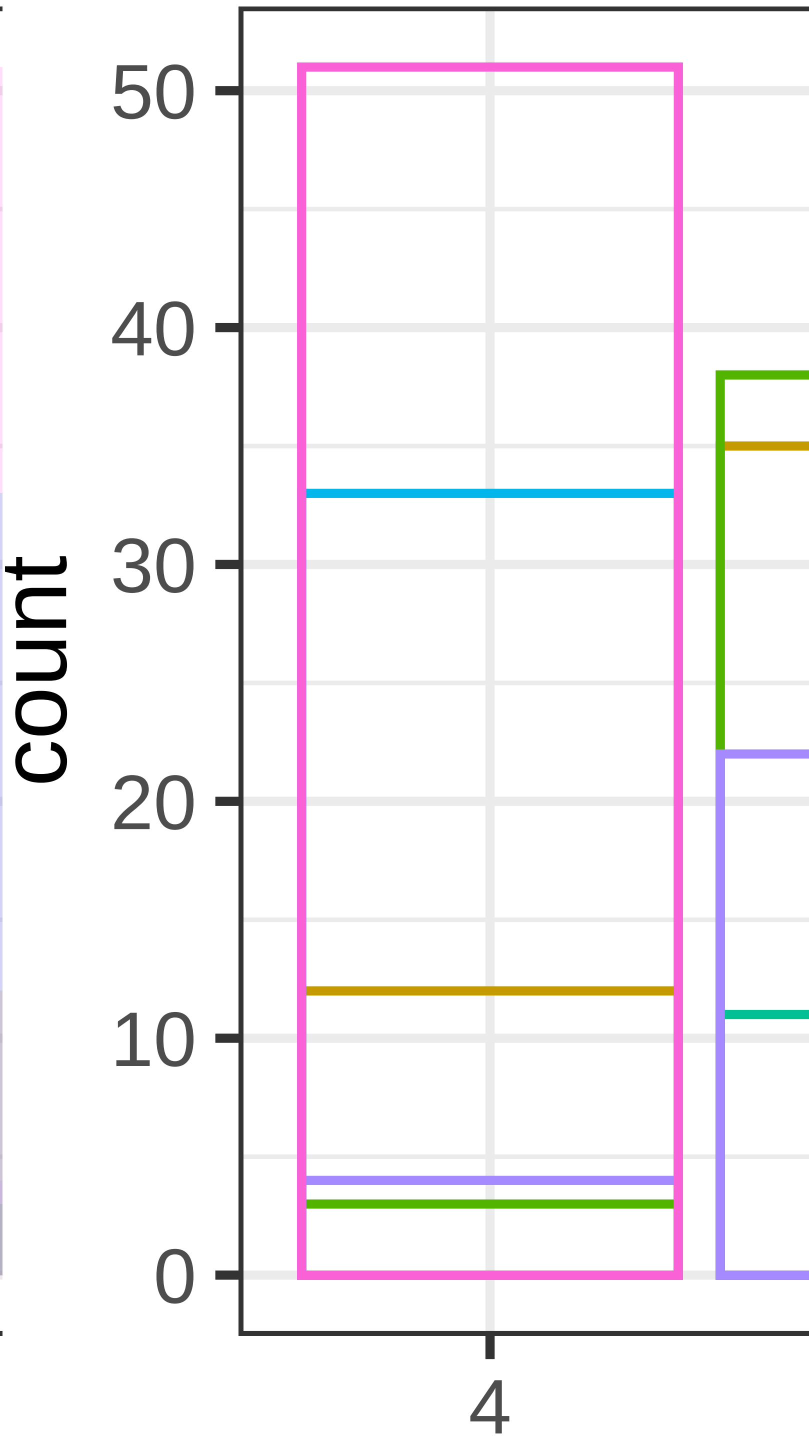
- `position = "dodge"` places overlapping objects directly *beside* one another. This makes it easier to compare individual values.

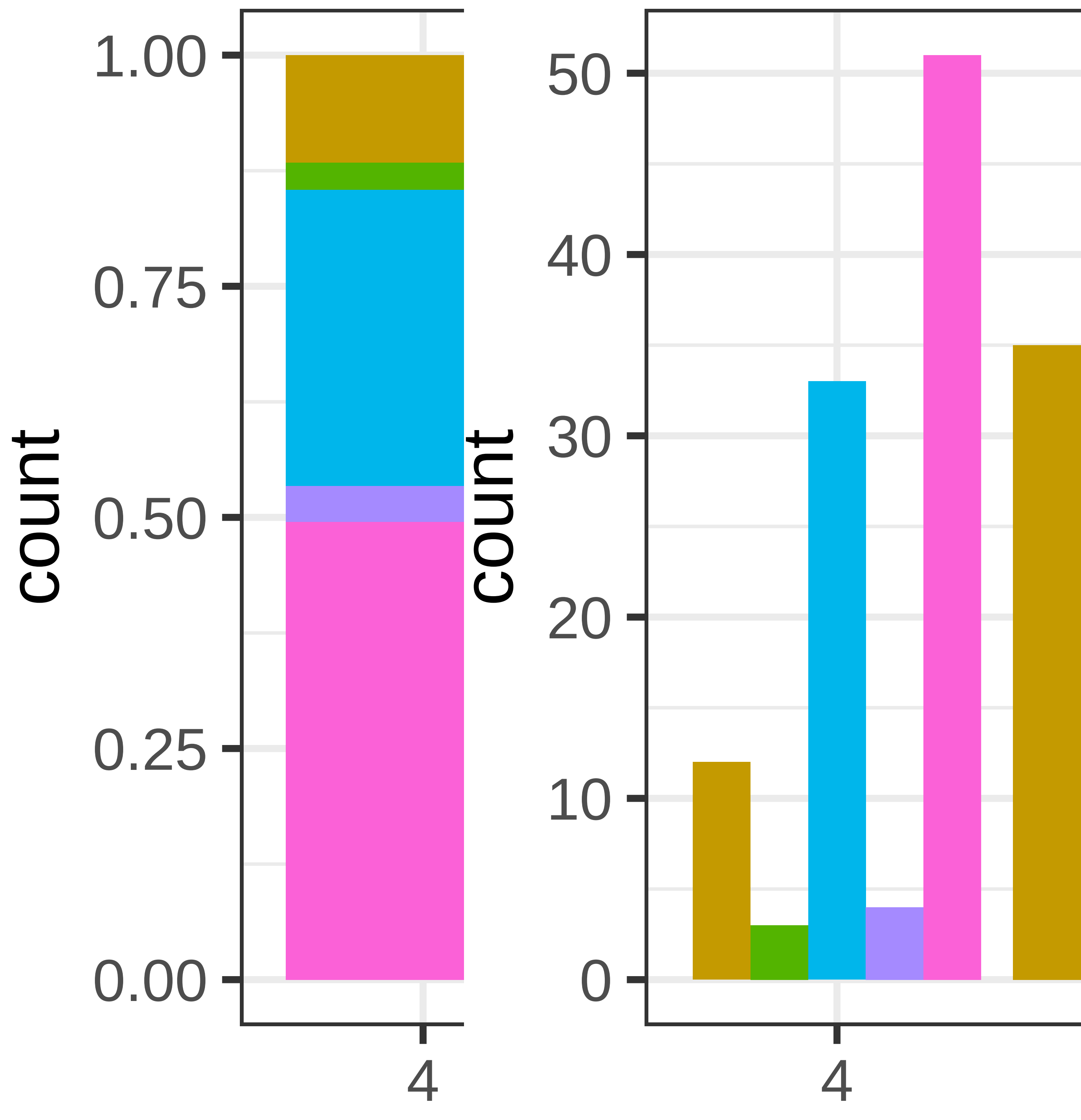
  ```
  # Left
  ggplot(mpg, aes(x = drv, fill = class)) +
    geom_bar(position = "fill")

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

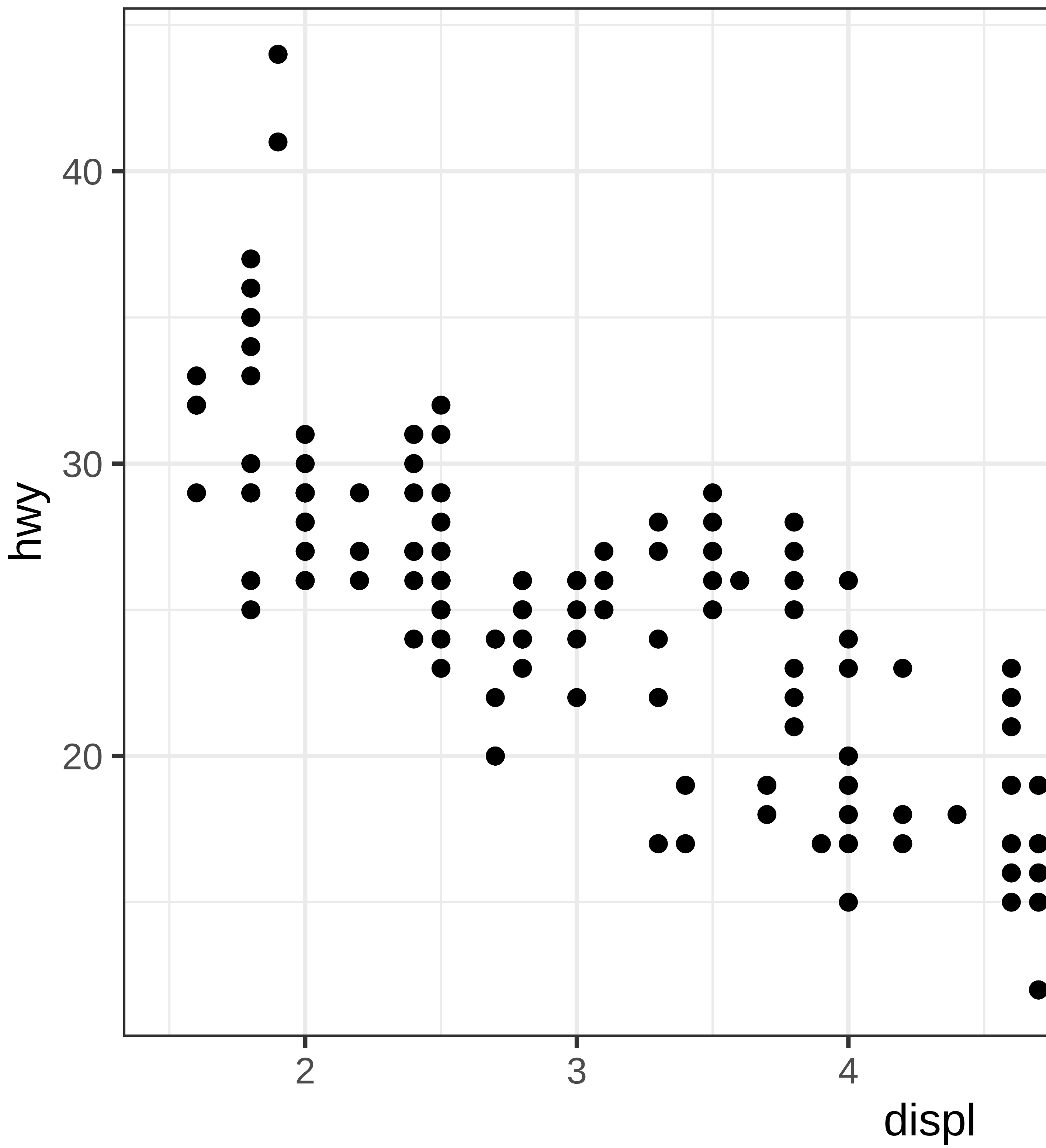
There's one other type of adjustment that's not useful for bar charts, but can be very useful for scatterplots. Recall our first scatterplot. Did you notice that the plot displays only 126 points, even though there are 234 observations in the dataset?

The underlying 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 difficult to see the distribution of the data. Are the data points spread equally throughout the graph, or is there one special combination of `hwy` and `displ` that contains 109 values?


count
50
40
30
20
10
0
4


count
50
40
30
20
10
0
4

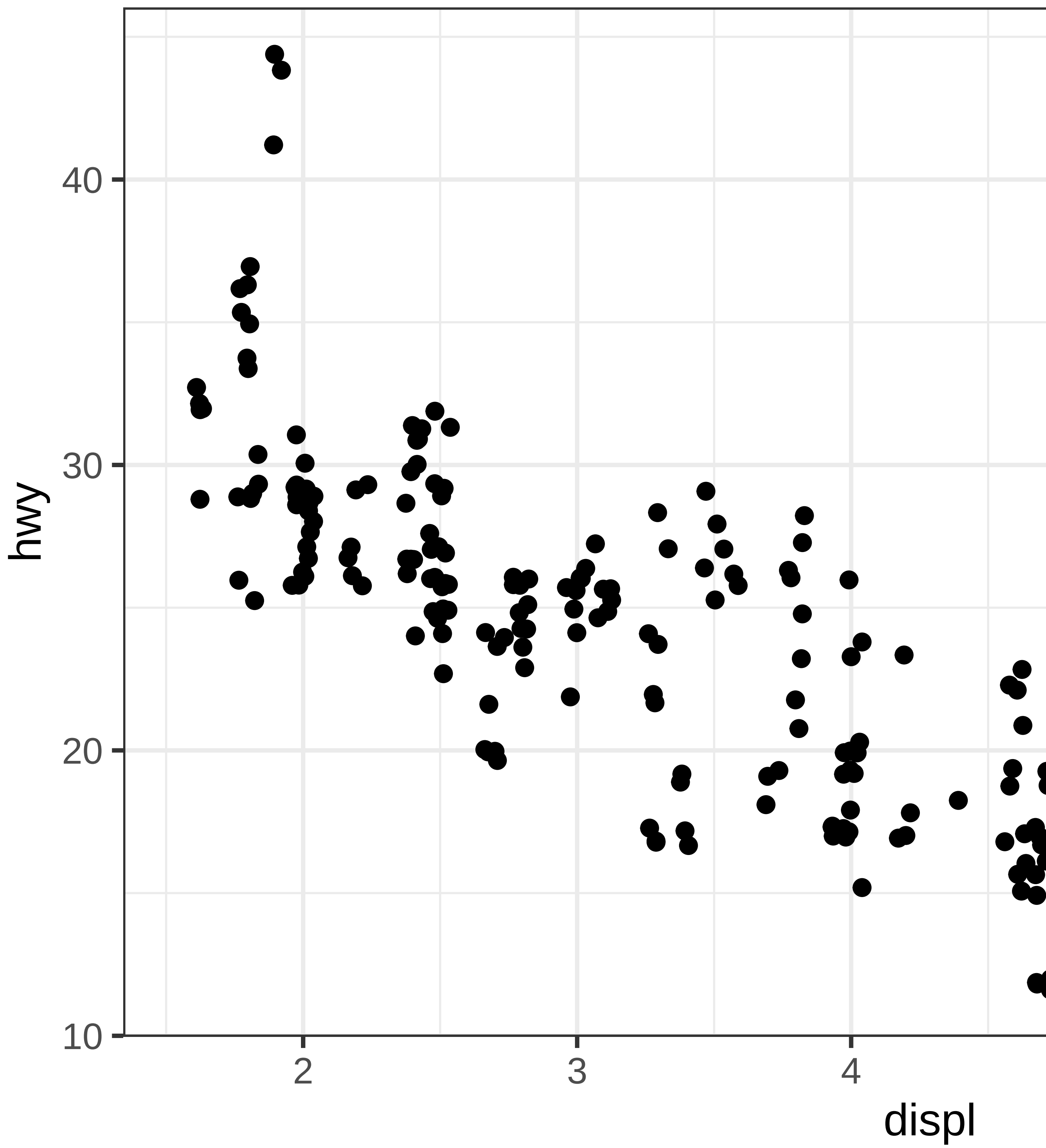

count
1.00
0.75
0.50
0.25
0.00
4
count
50
40
30
20
10
0
4



40
30
20
hwy
2
3
4
displ

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.

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


hwy
40
30
20
10
2
3
4
displ

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()`.

To 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`.

2.6.1 Exercises

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

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

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

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

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

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

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

2.7 The layered grammar of graphics

We can expand on the graphing template you learned in Section 1.3 by adding position adjustments, stats, coordinate systems, and faceting:

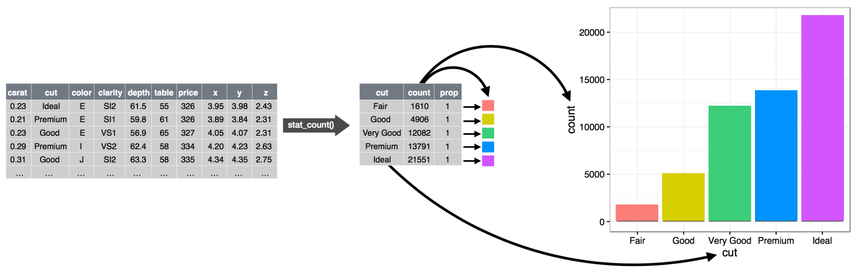
```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
```

```
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION>
```

Our new template takes seven parameters, the bracketed words that appear in the template. In practice, you rarely need to supply all seven parameters to make a graph because ggplot2 will provide useful defaults for everything except the data, the mappings, and the geom function.

The seven parameters in the template compose the grammar of graphics, a formal system for building plots. The grammar of graphics is based on the insight that you can uniquely describe *any* plot as a combination of a dataset, a geom, a set of mappings, a stat, a position adjustment, a coordinate system, a faceting scheme, and a theme.

To see how this works, consider how you could build a basic plot from scratch: you could start with a dataset and then transform it into the information that you want to display (with a stat). Next, you could choose a geometric object to represent each observation in the transformed data. You could then use the aesthetic properties of the geoms to represent variables in the data. You would map the values of each variable to the levels of an aesthetic. These steps are illustrated in Figure 2.3. You'd then select a coordinate system to place the geoms into, using the location of the objects (which is itself an aesthetic property) to display the values of the x and y variables.



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
...	...	...	...	...	...	...	...	...	...

cut	count	prop
Fair	1610	1
Good	4906	1
Very Good	12082	1
Premium	13791	1
Ideal	21551	1

Figure 2.3: Steps for going from raw data to a table of frequencies to a bar plot where the heights of the bar represent the frequencies.

At this point, you would have a complete graph, but you could further adjust the positions of the geoms within the coordinate system (a position adjustment) or split the graph into subplots (faceting). You could also extend the plot by adding one or more additional layers, where each additional layer uses a dataset, a geom, a set of mappings, a stat, and a position adjustment.

You could use this method to build *any* plot that you imagine. In other words, you can use the code template that you've learned in this chapter to build hundreds of thousands of unique plots.

If you'd like to learn more about the theoretical underpinnings of ggplot2, you might enjoy reading "The Layered Grammar of Graphics", the scientific paper that describes the theory of ggplot2 in detail.

2.8 Summary

In this chapter you learned about the layered grammar of graphics starting with aesthetics and geometries to build a simple plot, facets for splitting the plot into subsets, statistics for understanding how geoms are calculated, position adjustments for controlling the fine details of position when geoms might otherwise overlap, and coordinate systems which allow you to fundamentally change what `x` and `y` mean. One layer we have not yet touched on is theme, which we will introduce in **?@sec-themes**.

Two very useful resources for getting an overview of the complete ggplot2 functionality are the ggplot2 cheatsheet (which you can find at https://posit.co/resources/cheatsheets) and the ggplot2 package website (https://ggplot2.tidyverse.org).

An important lesson you should take from this chapter is that when you feel the need for a geom that is not provided by ggplot2, it's always a good idea to look into whether someone else has already solved your problem by creating a ggplot2 extension package that offers that geom.