Introduction to ggplot2

Dr. Austin Brown

Kennesaw State University

Introduction

- You may have heard the old adage, "A picture is worth a thousand words."
- For a statistician/data analyst, this still holds true!
- Graphical representations of data (or "data visualizations") can lead to insights not obvious from a scan of a dataset.
- Additionally, graphs are useful in conveying the results of statistical models/tests to people without quantitative training.

Introduction

- Within R, there are two primary methods for creating graphs: the Base R method and the `ggplot2` method. We're going to restrict our focus mostly to the latter method.

- The "gg" in `ggplot2` is shorthand for "grammar of graphics."

- It offers an incredible array of options for plotting all sorts of data types (including maps).

- We're going to learn how to plot some of the more common graphs as well as get an understanding of some options available to us. Learning to use `ggplot2` could honestly be a class (or two) by itself, so what we're learning is going to be an introduction, but please know it can be used in many, many ways.

Creating a ggplot

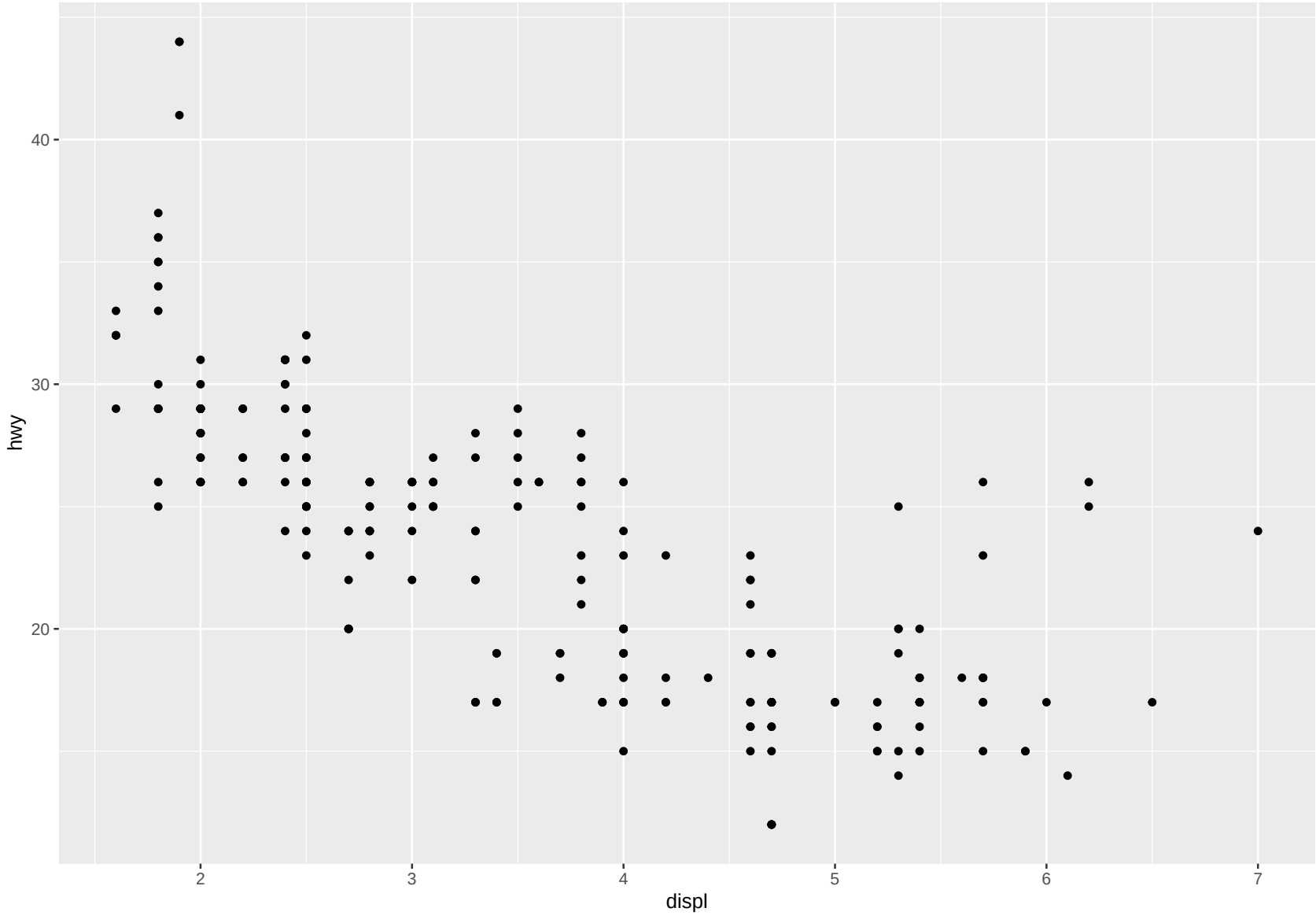
- Let's check out a basic example. Using the `mpg` dataframe, suppose we want to explore the relationship between a car's engine displacement (measured in liters) and its highway miles per gallon (mpg).

- Since both of these are quantitative variables, the most appropriate plot would be a scatterplot.

Creating a ggplot

```
library(tidyverse)
mpg <- ggplot2::mpg
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ,y = hwy))
```

Creating a ggplot



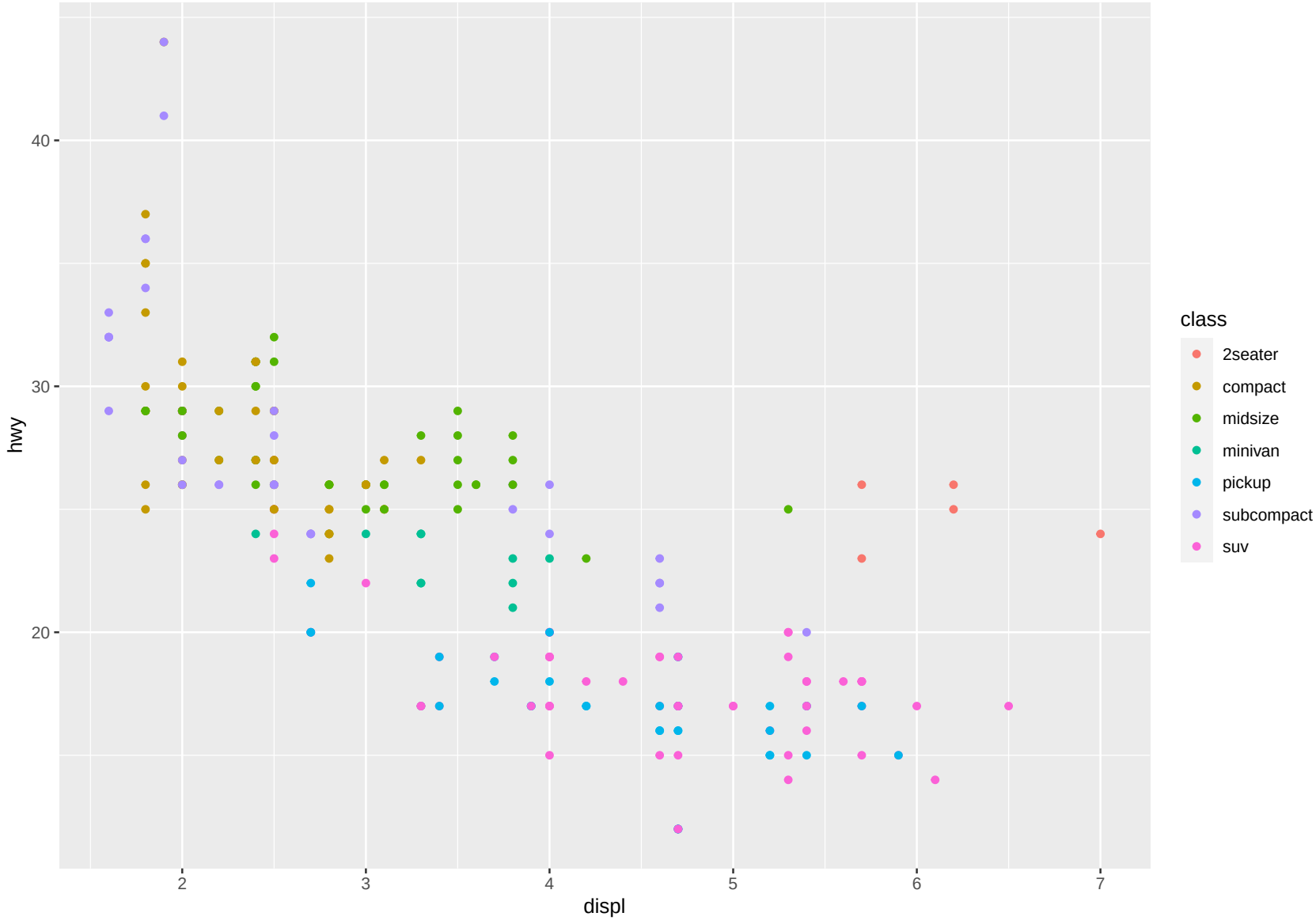
Creating a ggplot

- The visual characteristics of the objects in our plots are referred to as "aesthetics" (hence the `aes()` function).
- `aes()` controls lots of different aspects of our plots, including the size, shape, and color of the objects (bars, points, lines, etc) in the plots.
- To illustrate, let's say we now want to see the relationship between engine displacement and highway mpg controlling for the type of vehicle (e.g., minivan, compact, etc.)

Creating a ggplot

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

Creating a ggplot



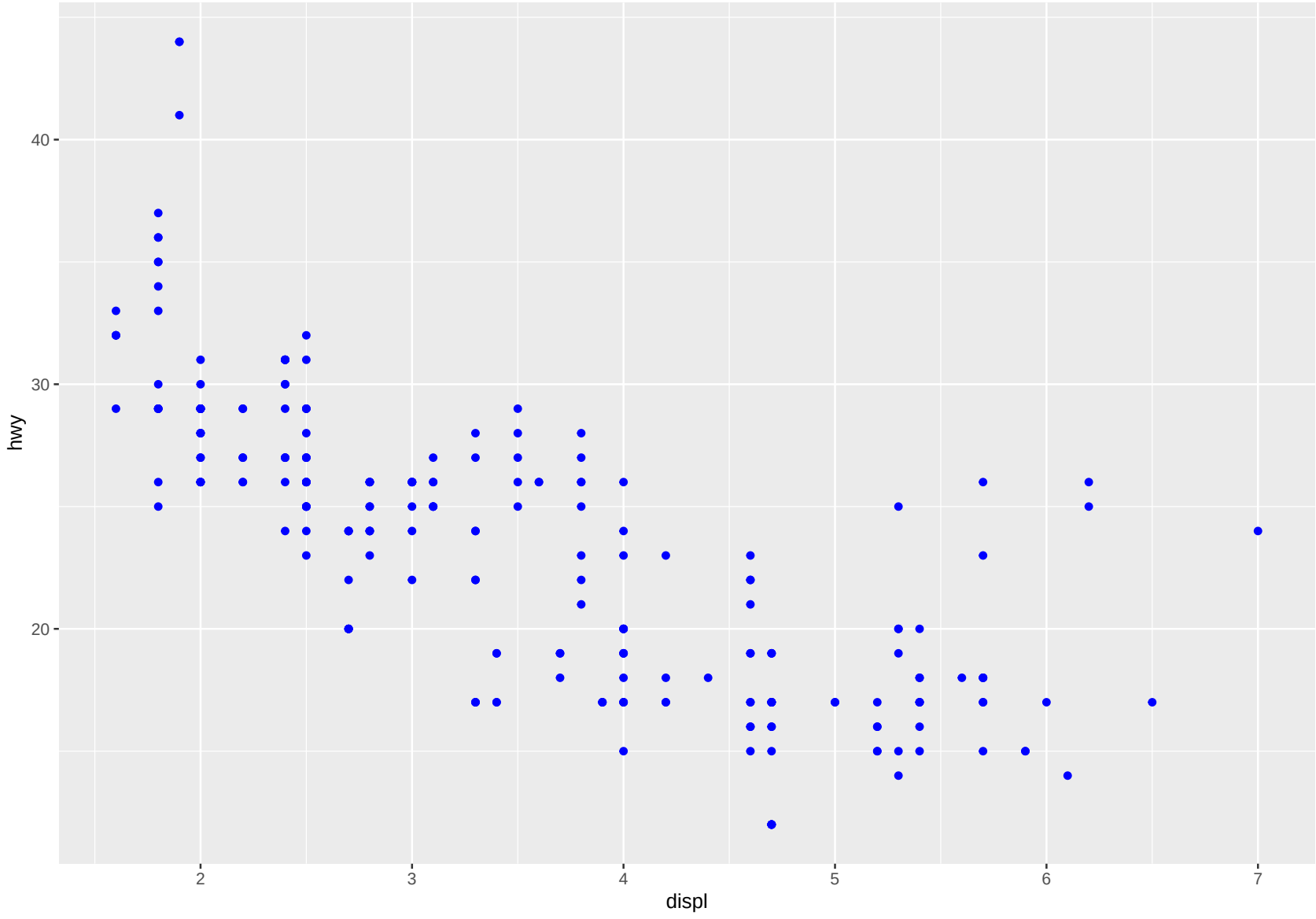
Creating a ggplot

- Now, we also have options that we can specify outside of the `aes()` function,including those we can specify inside of the `aes()` function.

- Let's say, for example, that we want the color of our points to be uniformly blue.

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

Creating a ggplot

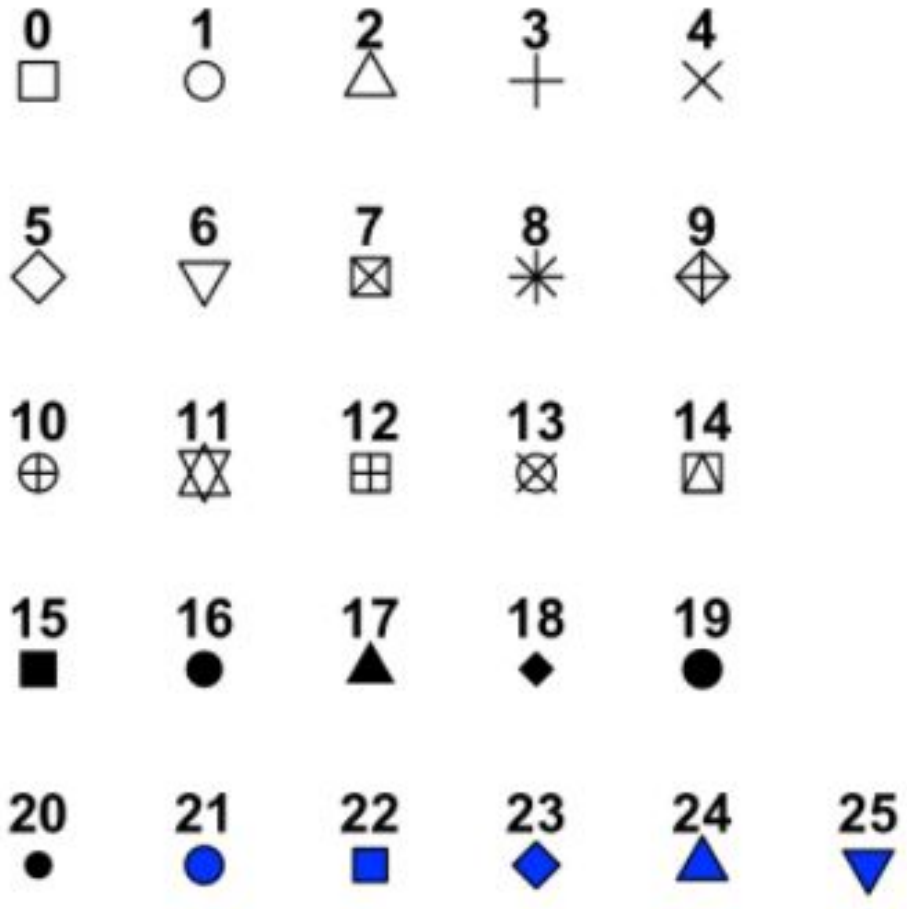


Creating a ggplot

- If we want to have a uniform aesthetic, we have to manually specify it outside of the `aes()` function with a value that is accepted by the aesthetic.

Aesthetic	Input
`color`	Name of color in quotes
`size`	Size of the points in millimeters
`shape`	A number (1-15) that indicates a shape
`alpha`	A number between 0 and 1 that indicates the opacity

Creating a ggplot



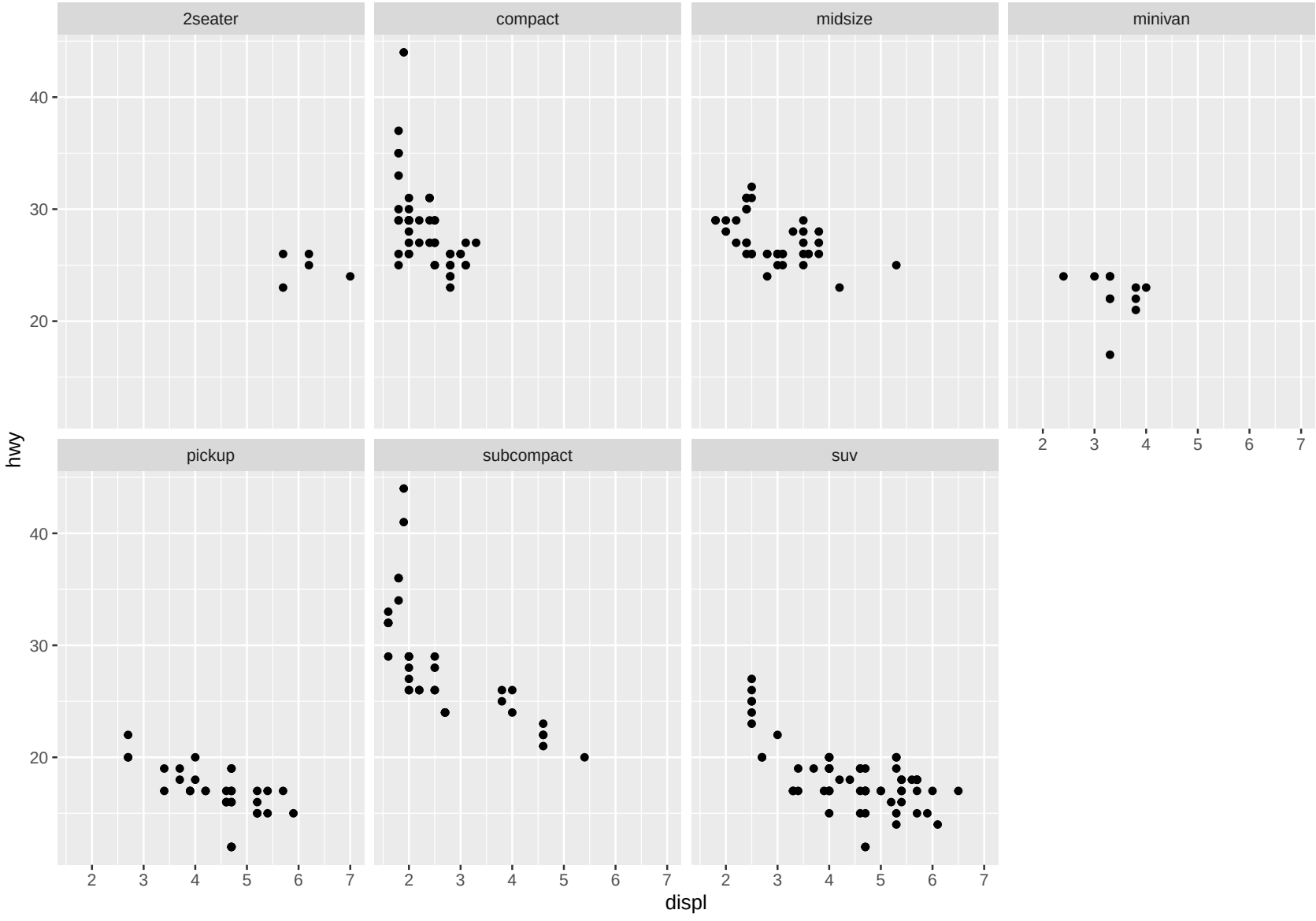
Facets

- In the last example, we were able to glean new information about our dataset by using the `aes()` function.
- We can also create separate graphs for each level of a categorical variable.
- Instead of creating a different dataframe for each level and creating separate graphs, we can take advantage of the `facet_wrap` function to have `ggplot` do the heavy lifting for us.

Facets

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

Facets

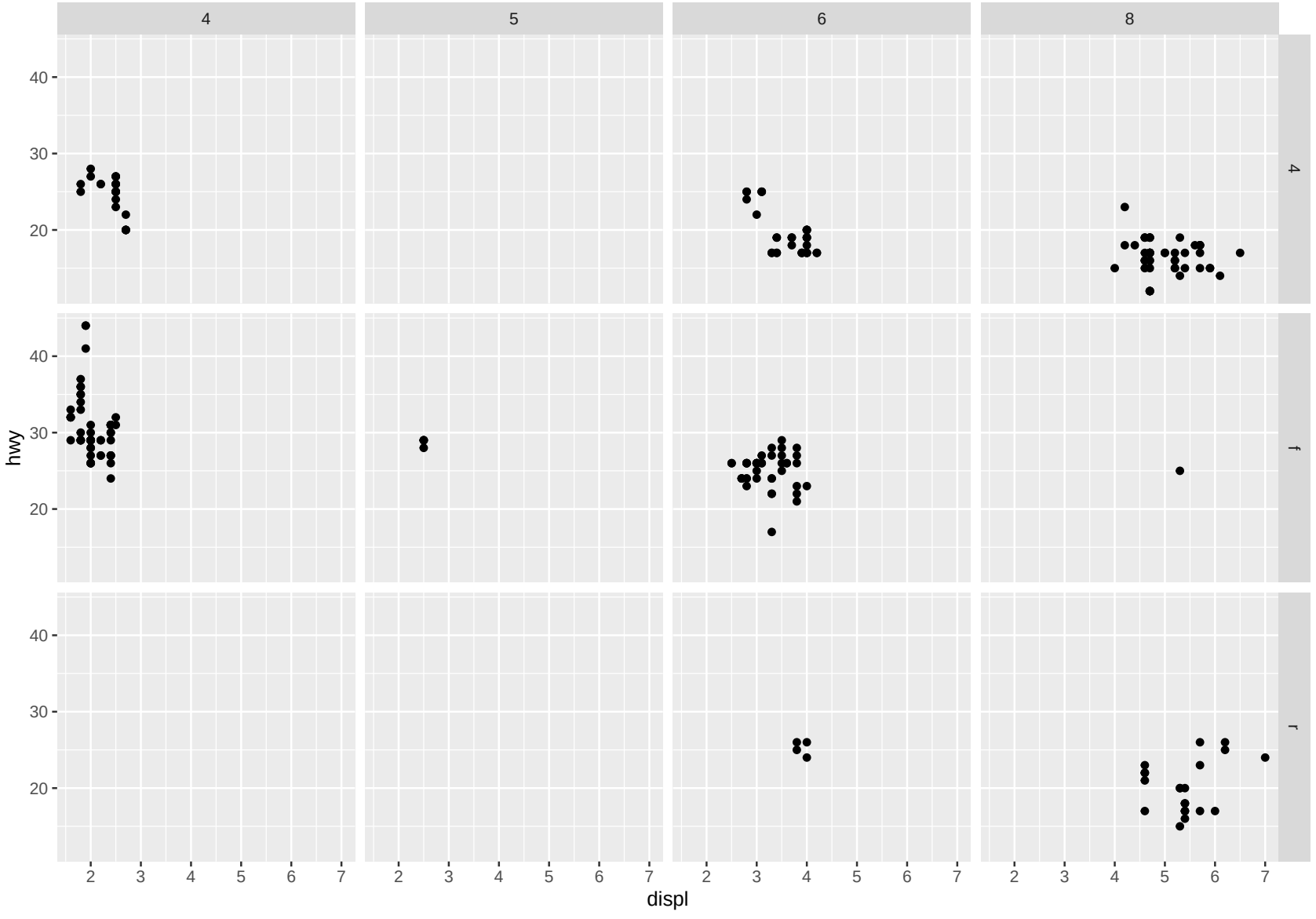

2seater
compact
midsize
minivan
pickup
subcompact
suv
hwy
displ
40
30
20
2
3
4
5
6
7

Facets

- Alternative method of faceting using `facet_grid`:

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

Facets



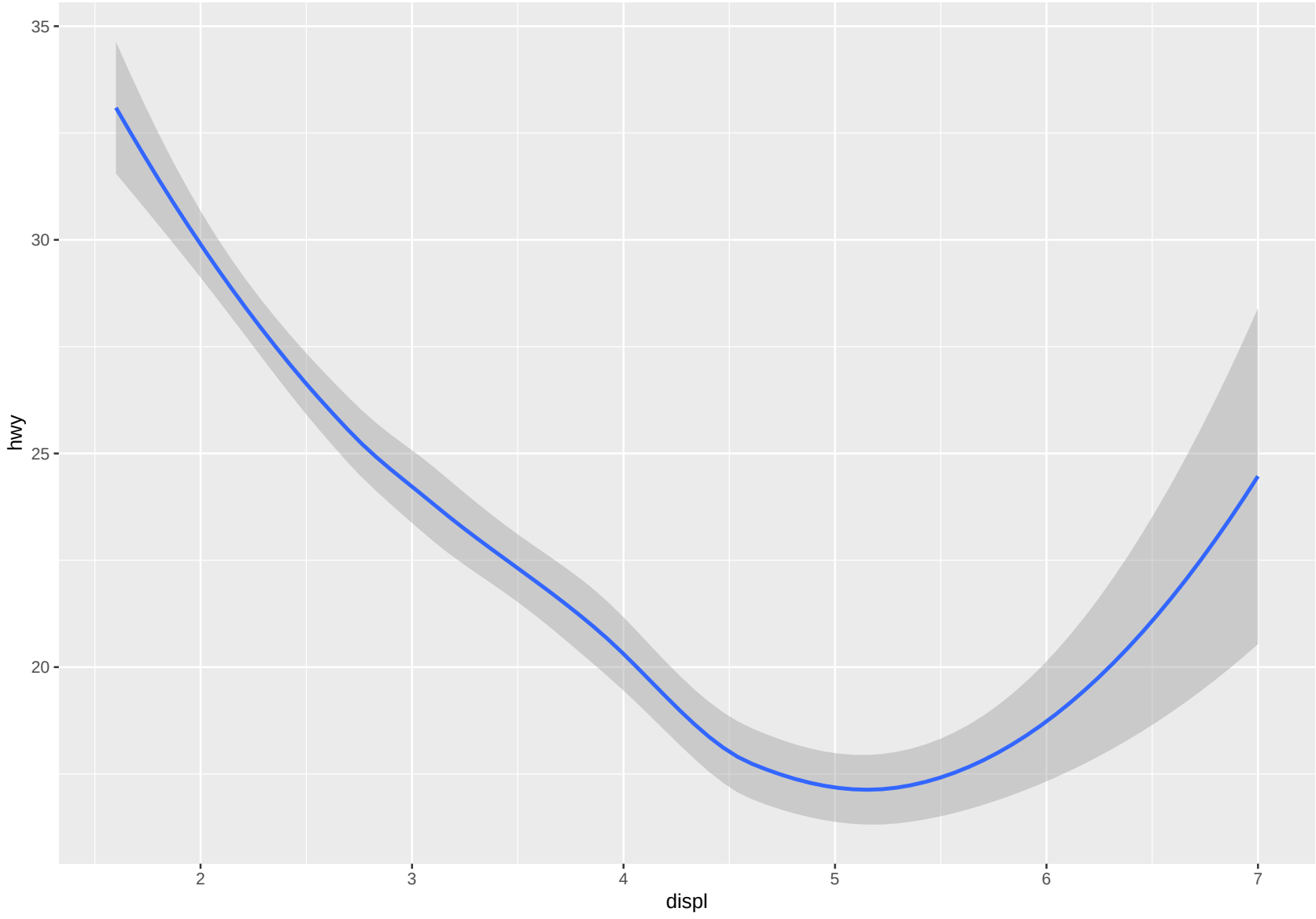
Geometric Objects

- In `ggplot`, the way that we generate different graphics is by using different *geoms*.
- For example, if we wanted to build a histogram, we'd use the histogram geom. If we wanted to build a bar graph, we'd use the bar geom, and so on.
- We can use different geoms on the same data and variables to generate different graphs.

Geometric Objects

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

Geometric Objects



Geometric Objects

- One important thing that should be pointed out is that not every aesthetic works with every geom.
- For example, this code won't run because the `shape` aesthetic won't work with the smooth geom.

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

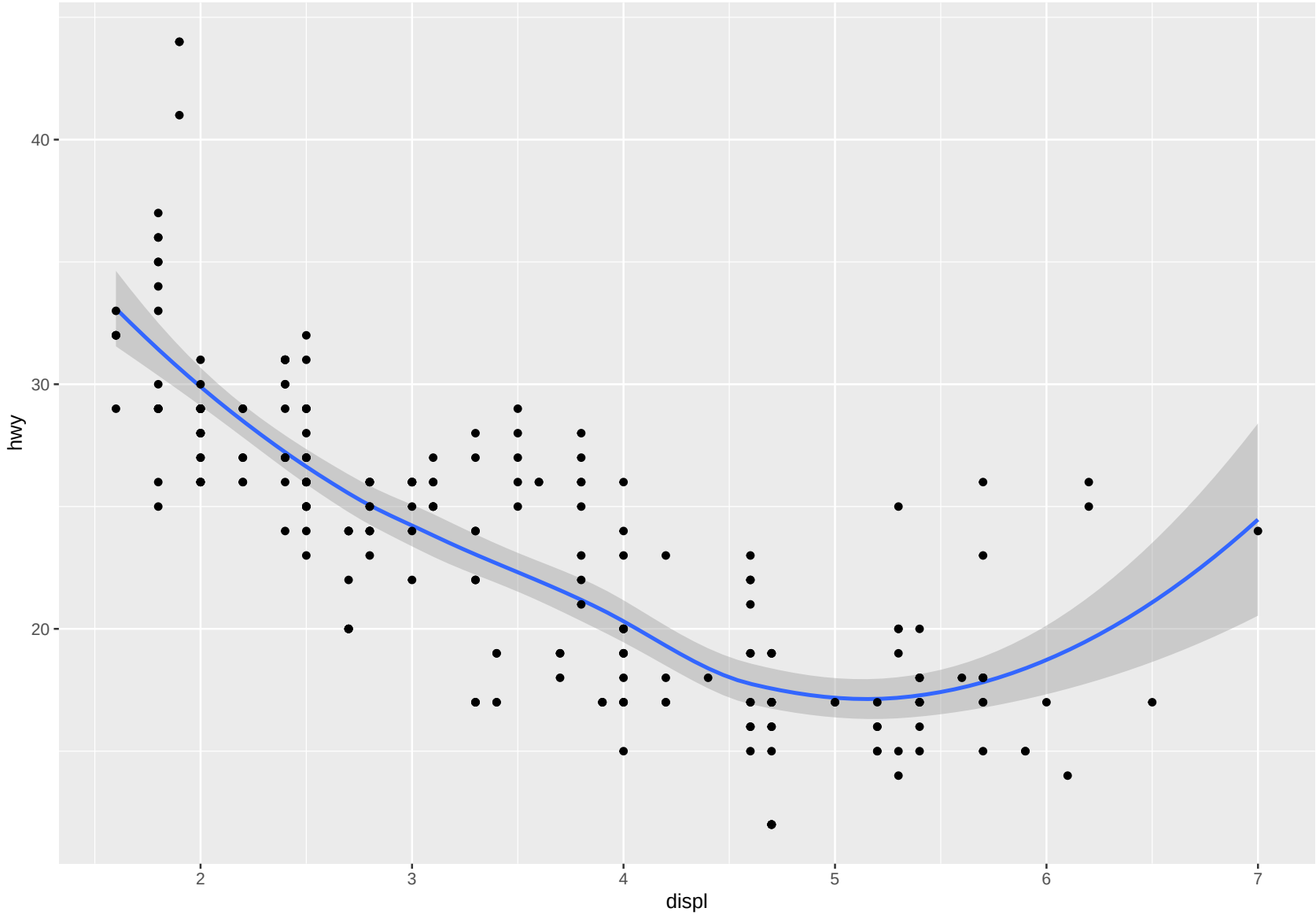
Geometric Objects

- One of the coolest aspects of `ggplot2` is that we can use multiple geoms together in order to build really nice plots.

- For example, we can use the smooth and points geoms together to make a nice, informative plot.

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

Geometric Objects

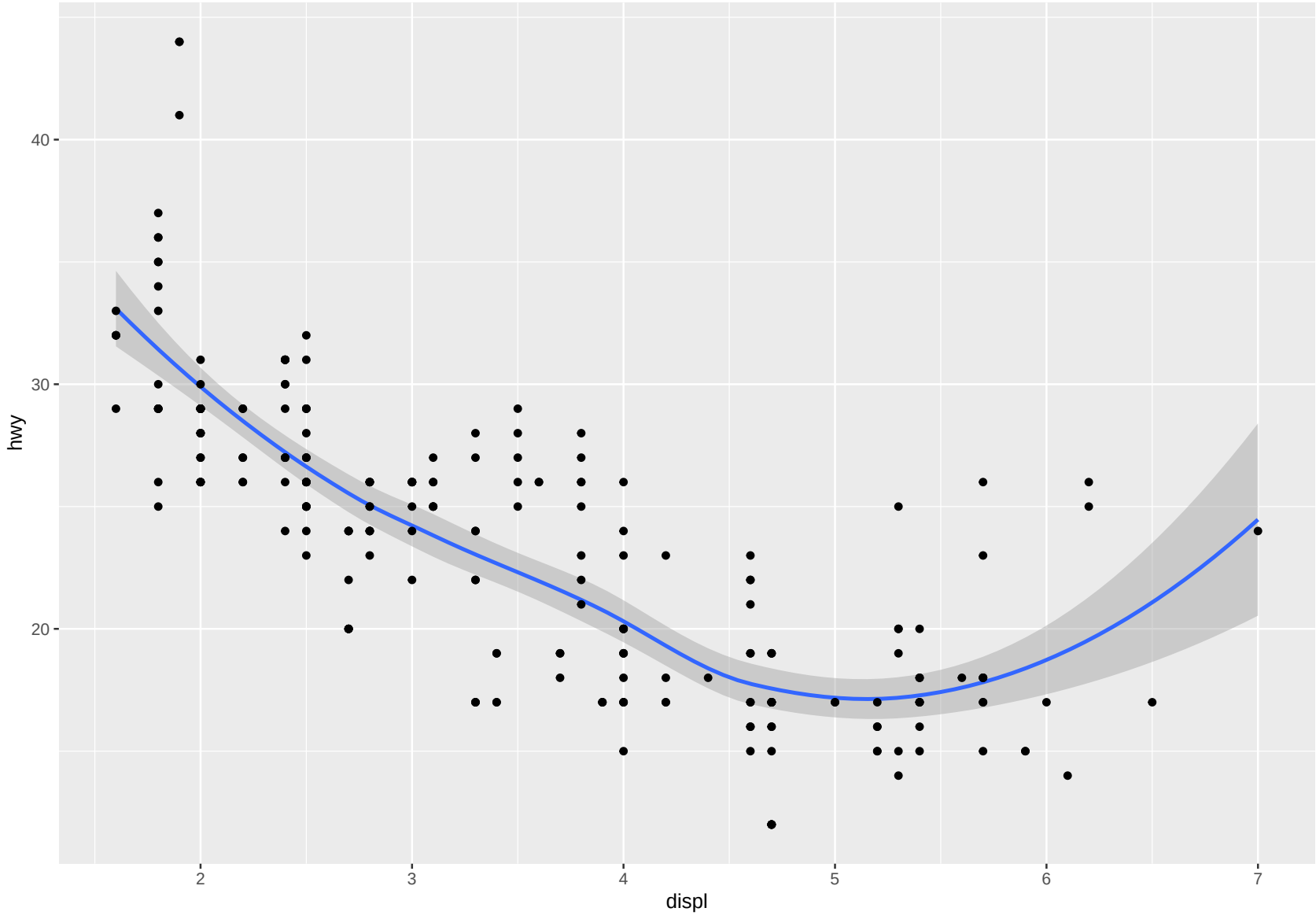


Geometric Objects

- Now, another cool feature of `ggplot` is the ability to specify global aesthetics in order to cut down on keystrokes:

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

Geometric Objects



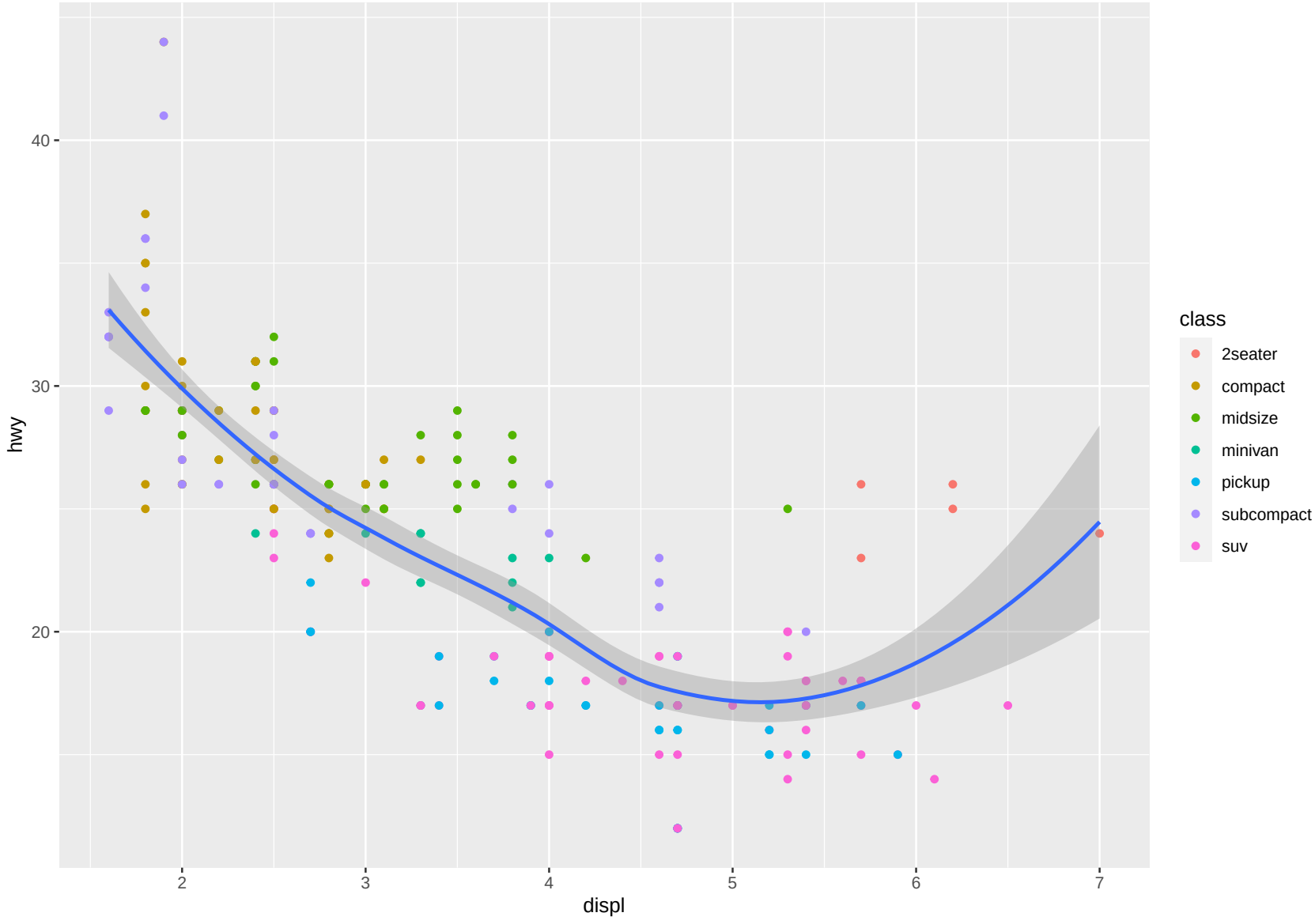
Geometric Objects

- What's happening behind the scenes is that aesthetics specified in the `ggplot` function are mapped to each of the geoms used in a given plot.

- We can specify different options within the various geoms that will only map to those geoms. For example:

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

Geometric Objects



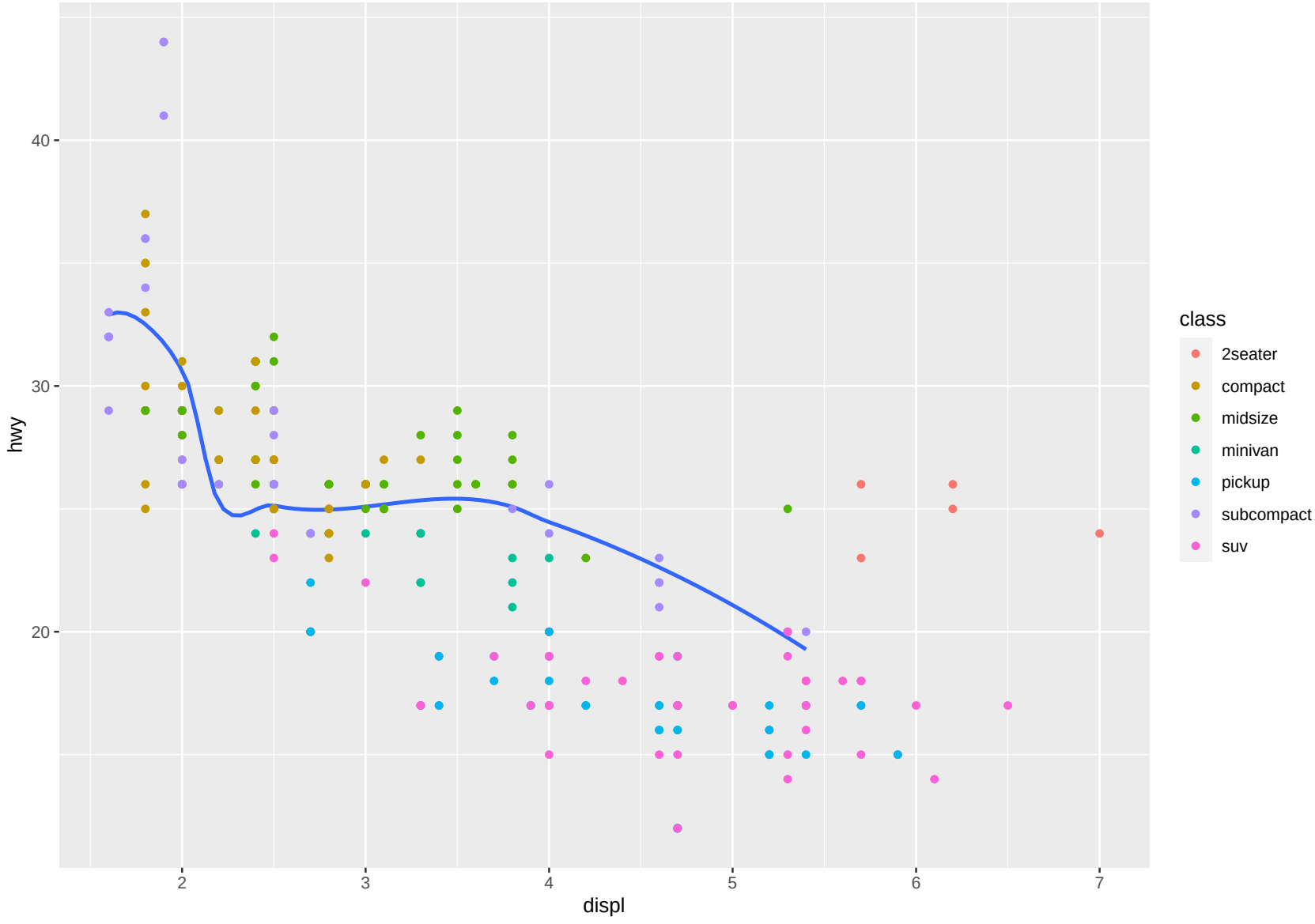
Geometric Objects

- We can also use `dplyr` functions to help us both at the global level as well as at the geom level.

- For example, let's say for our smooth geom, we only want to do that for one of the subcompact vehicle class.

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

Geometric Objects



Geometric Objects

- Notice, when we're using `dplyr` locally, we put it inside of the specific geom.

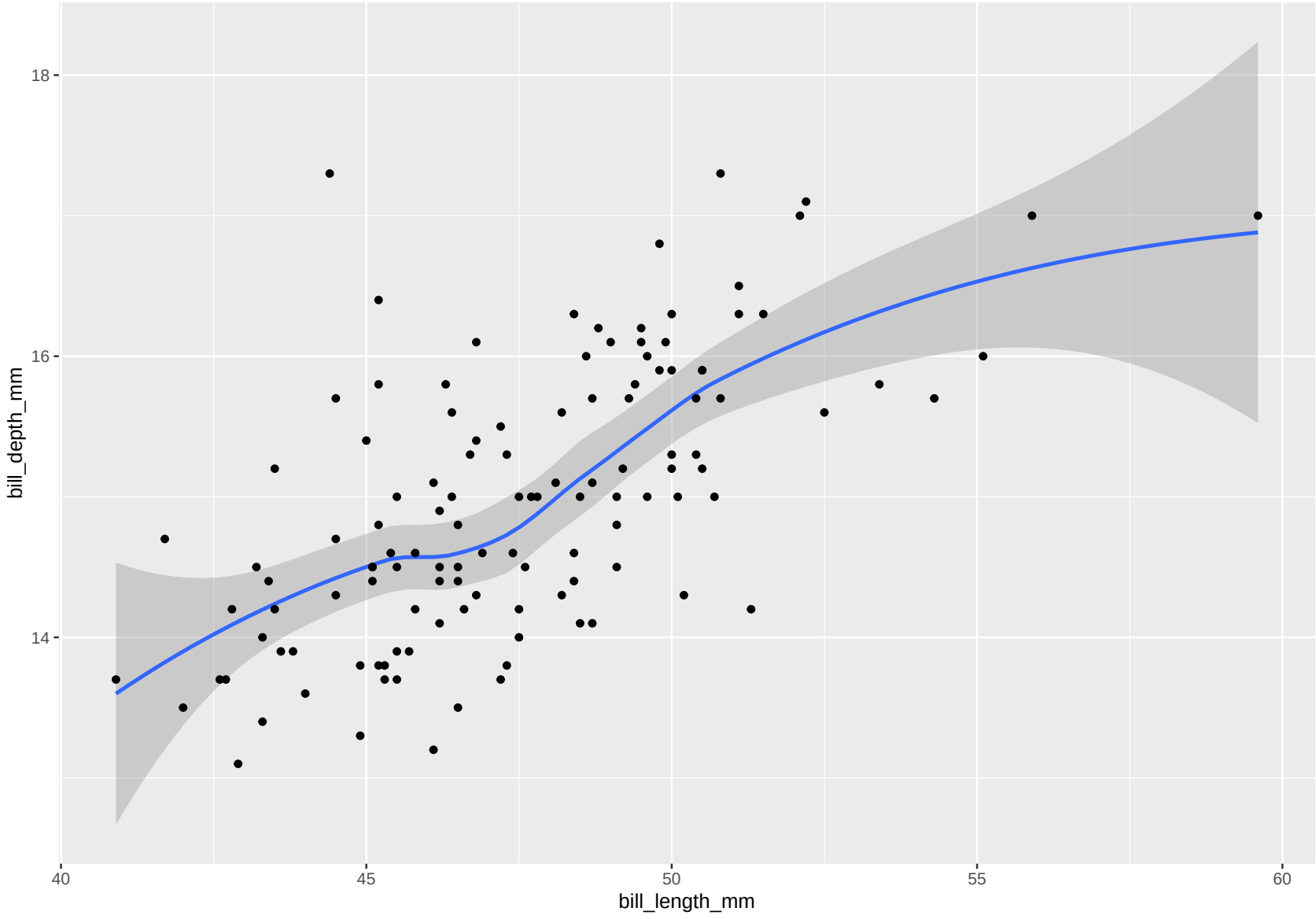
- If we want to use `dplyr` globally, we put it either inside of the `ggplot` function, or combining the `dplyr` function with `ggplot` using piping.

Geometric Objects

- Inside of the ggplot function:

```
penguins <- palmerpenguins::penguins
ggplot(data =
         penguins |>
         filter(island == "Biscoe" &
                  species == "Gentoo"),
           aes(x = bill_length_mm,y=bill_depth_mm)) +
  geom_smooth() +
  geom_point()
```

Geometric Objects

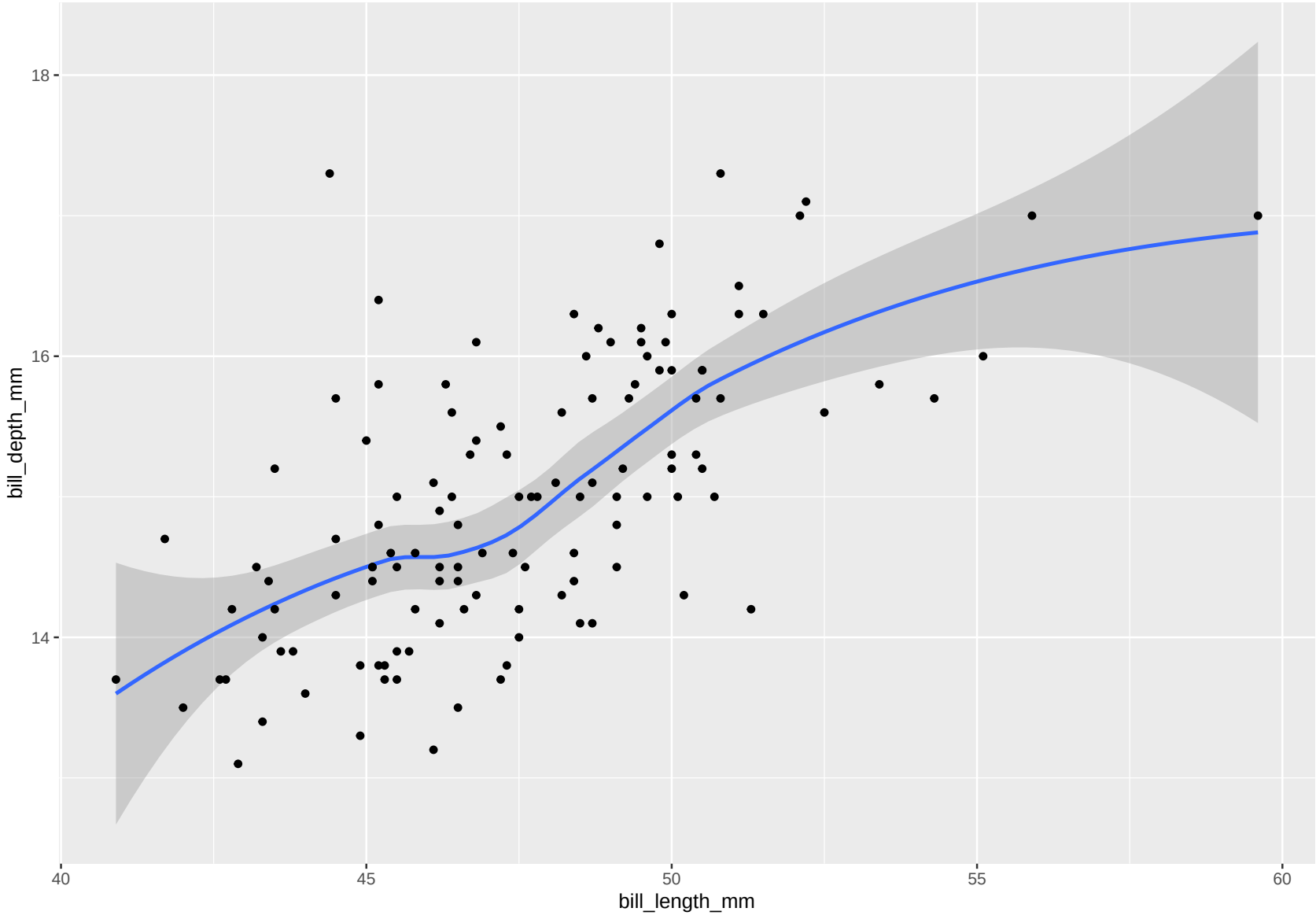


Geometric Objects

- Outside of ggplot2 altoghether:

```
penguins |>
  filter(island == "Biscoe" &
           species == "Gentoo") |>
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm)) +
    geom_smooth() +
    geom_point()
```

Geometric Objects



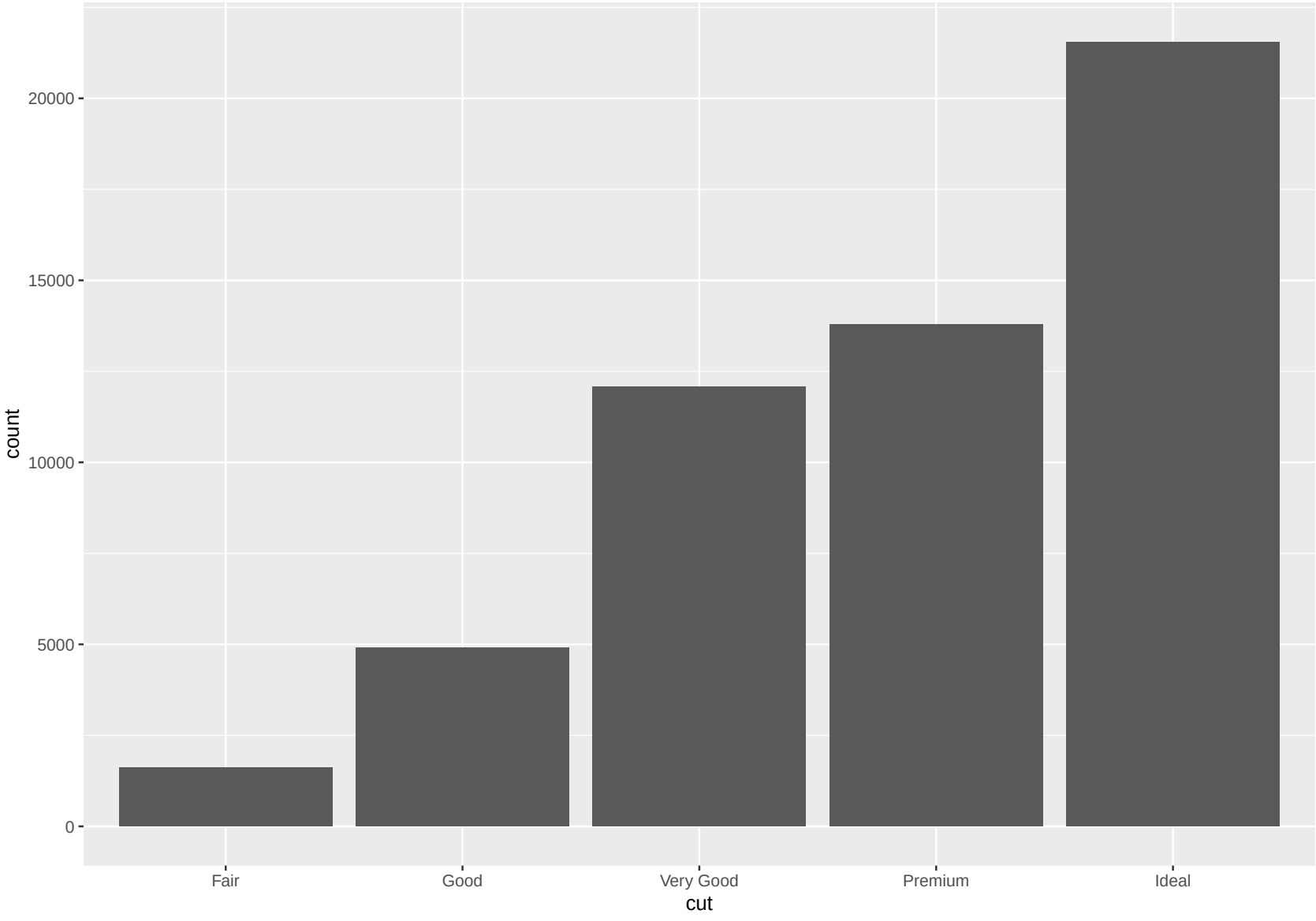
Statistical Transformations

- Sometimes when we're wanting to make a graph, we want to plot a statistical summary of a variable instead of its raw value.
- This is particularly evident for categorical data. Say, for example, we wanted to plot the frequencies of the levels of a categorical variable in a bar chart.
- Using `ggplot2::diamonds` dataframe, let's plot a bar chart.

Statistical Transformations

```
diamonds <- ggplot2::diamonds
ggplot(diamonds) +
  geom_bar(aes(x=cut))
```

Statistical Transformations



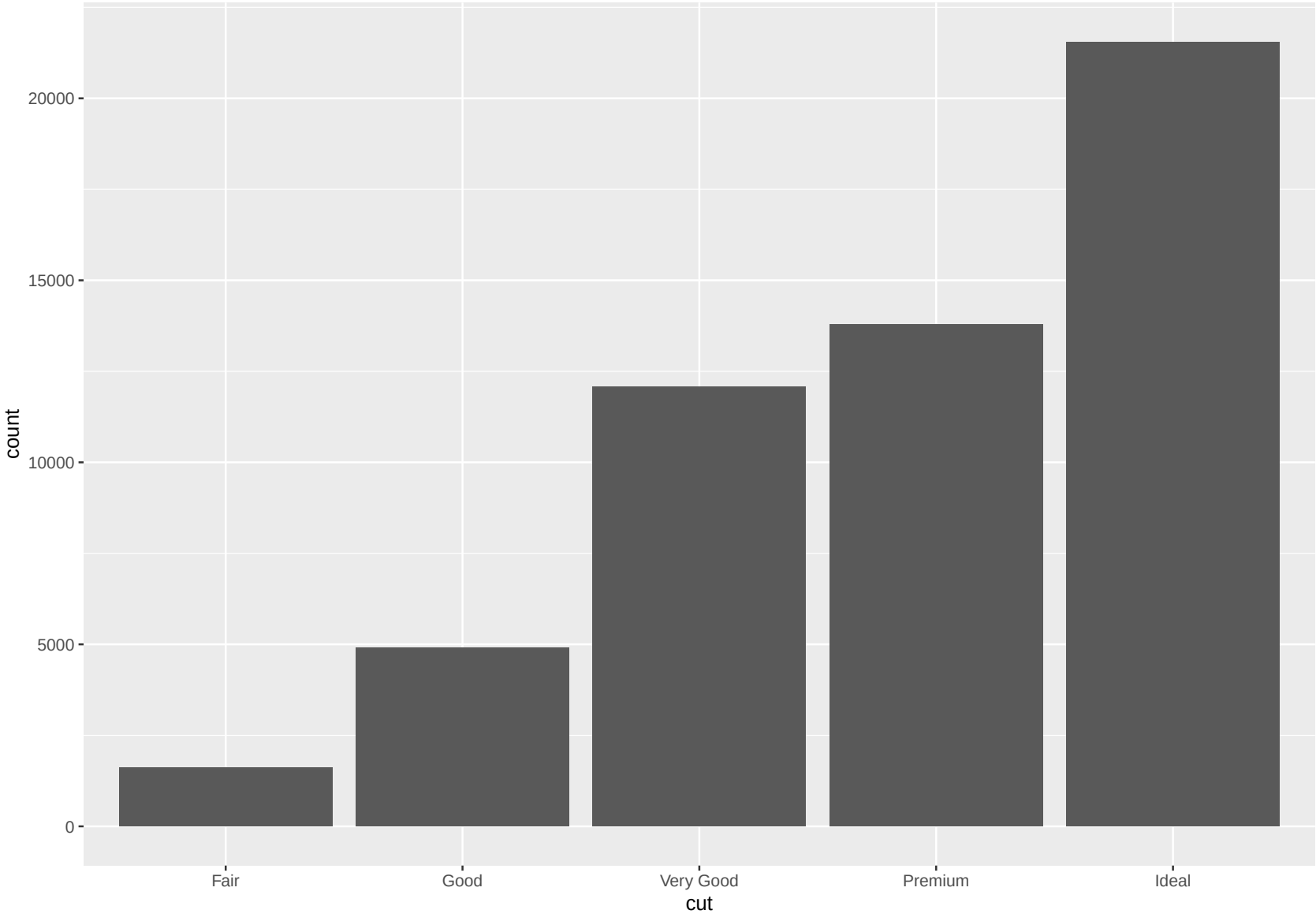
Statistical Transformations

- What you'll notice is that on the y-axis, we have a variable called "count" despite that variable not being in the `diamonds` dataframe.
- This is because behind the scenes, `ggplot` implicitly understands that we're wanting to plot frequencies.
- This implicit understanding stems from stat functions working in tandem with a particular geom. For the bar geom, it's the `stat_count` function.
- We can interchange this function for the bar geom and it yields the same result.

Statistical Transformations

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

Statistical Transformations



Statistical Transformations

- Many times, we don't have to worry about using the stat function because the geom function is clearer as far as code readability goes.
- However, there may be instances where we need to override the defaults.
- One instance, in the case of bar charts, might be because we already have the data in contingency table form with all the frequencies already tabulated.

Statistical Transformations

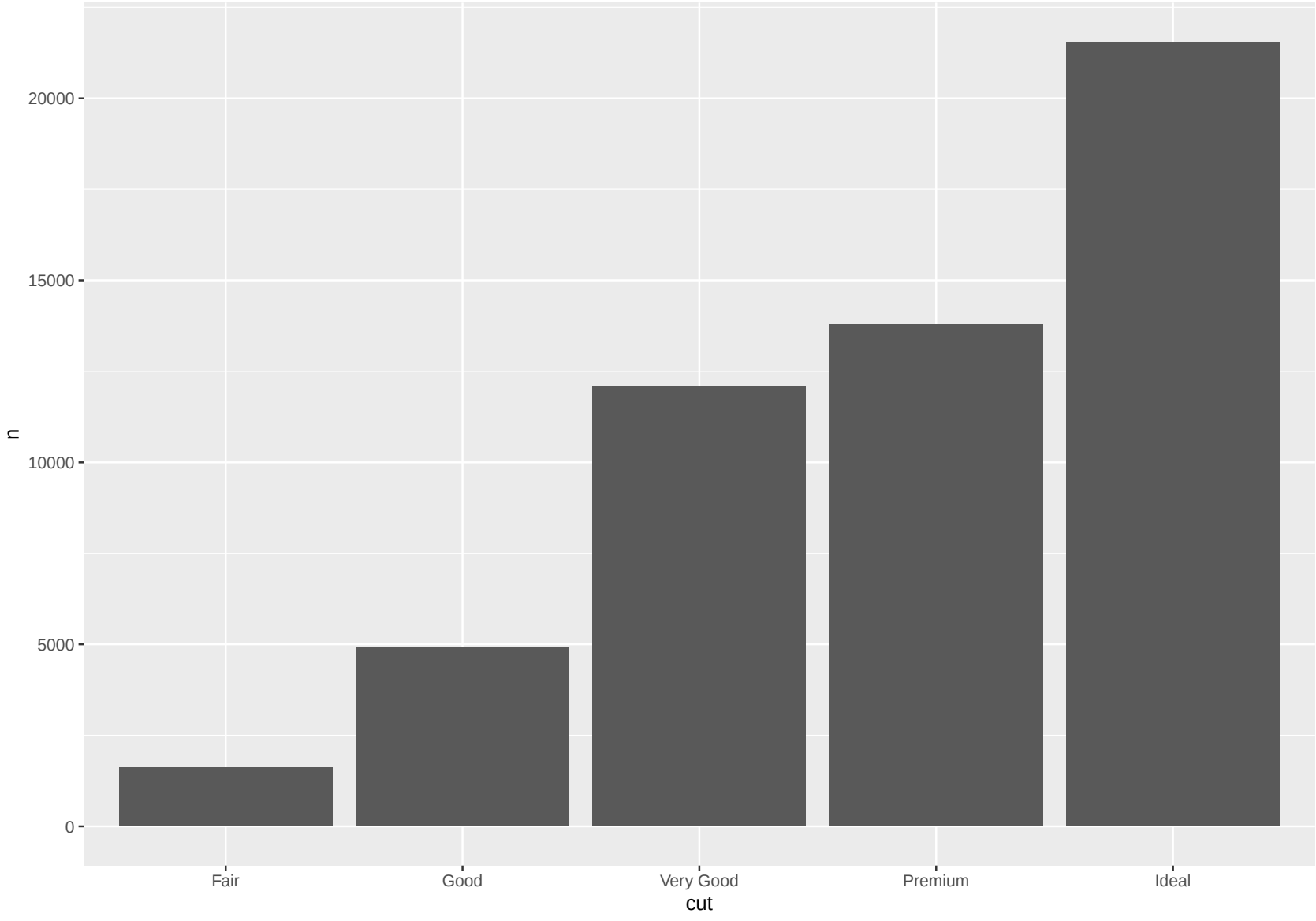
```
## Count Up Number of Cuts in Diamonds Dataframe ##
cuts_df <- diamonds |>
  group_by(cut) |>
  count()
cuts_df
```

```
## # A tibble: 5 x 2
## # Groups:   cut [5]
##   cut           n
##   <ord>     <int>
## 1 Fair       1610
## 2 Good       4906
## 3 Very Good 12082
## 4 Premium   13791
## 5 Ideal     21551
```

Statistical Transformations

```
ggplot(cuts_df) +
  geom_bar(aes(x=cut,y=n),stat="identity")
```

Statistical Transformations



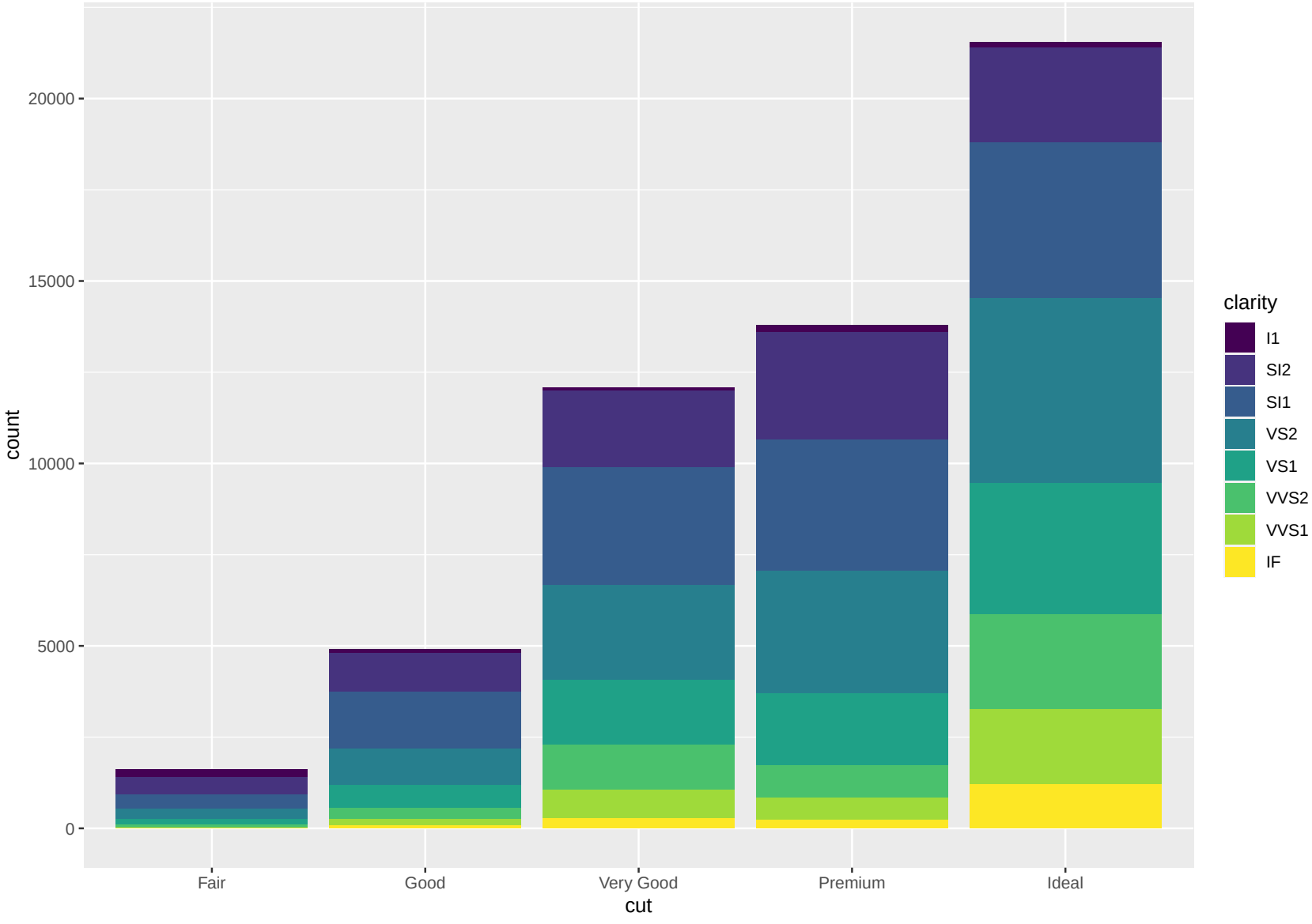
Position Adjustments

- Another cool bit of functionality available in `ggplot` is the ability to adjust the position of a data point in the context of the graph.

- Let's look at a couple of examples using the `diamonds` dataframe to really see this in play. Let's say we want to create a stacked bar chart where we are comparing the proportion of clarity categories against the levels of diamond cuts:

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

Position Adjustments



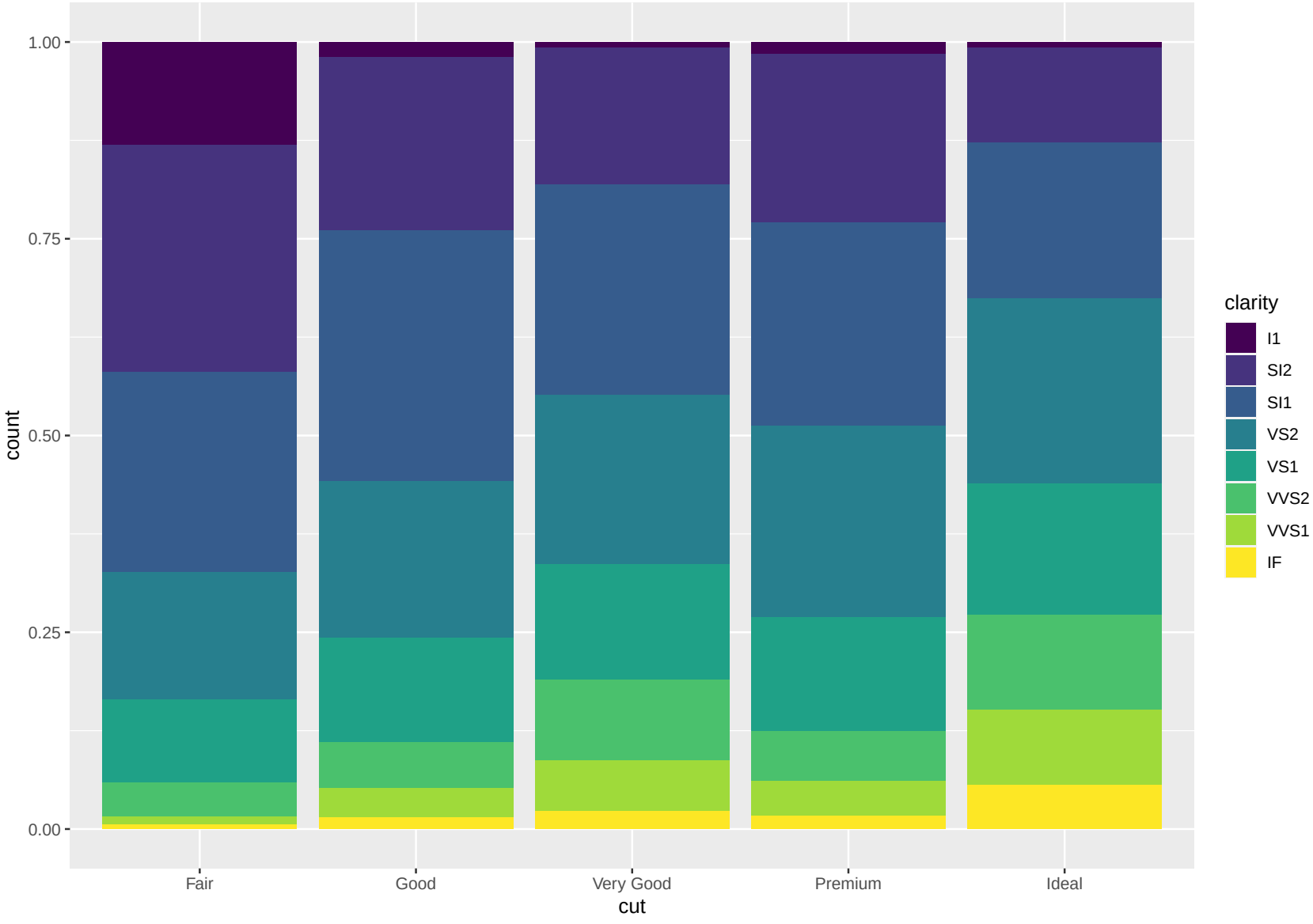
Position Adjustments

- What happened? By default, the position argument is set to stack. So if that's what we want, there's nothing else we have to do.
- However, there may be instances when we want to change the position argument in order to obtain a different plot.
- For example, typically, a raw stacked bar chart like this where the sample sizes across the levels of our x-axis variable are not equal or pretty imbalanced, a 100% stacked bar chart might be preferable for comparison purposes.

Position Adjustments

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

Position Adjustments



Position Adjustments

- Another commonly used type of position adjustment is really useful for generating scatterplots with lots of data points.

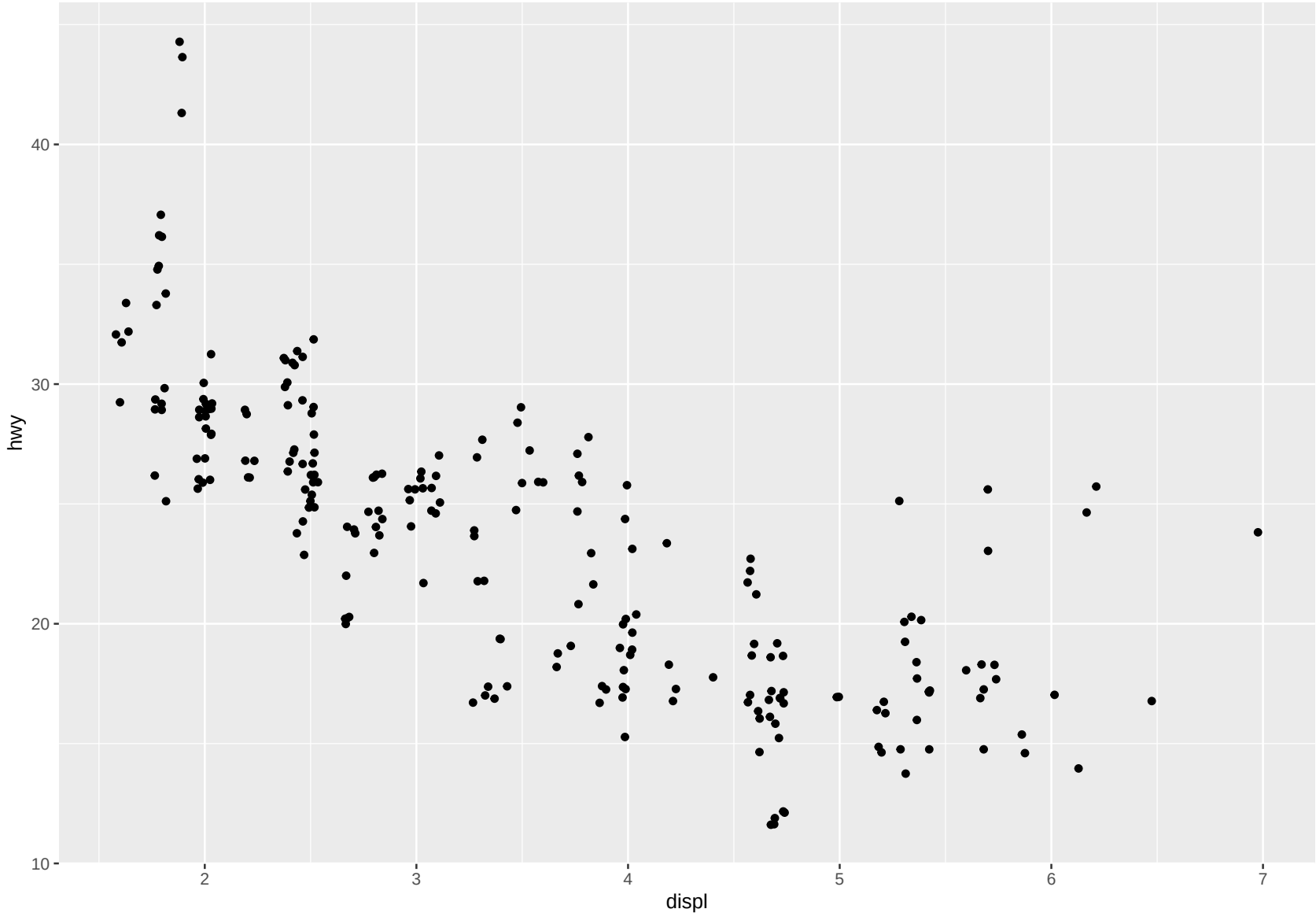
- When we have a lot of data points that are tightly grouped, they tend to overlap to where not all of the points display. This could potentially lead to incorrect conclusions, especially with respect to observation density.

- For the point geom, the default position is "identity." If we change it to "jitter," `ggplot` adds some random noise to each point so they stop overlapping, but the relationship between the two variables isn't significantly modified.

Position Adjustments

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

Position Adjustments



Adding Titles and Changing Themes

- One of the characteristics of an effective graph is clear titles. Most graphs ought to stand alone without you being there to explain what everything means.

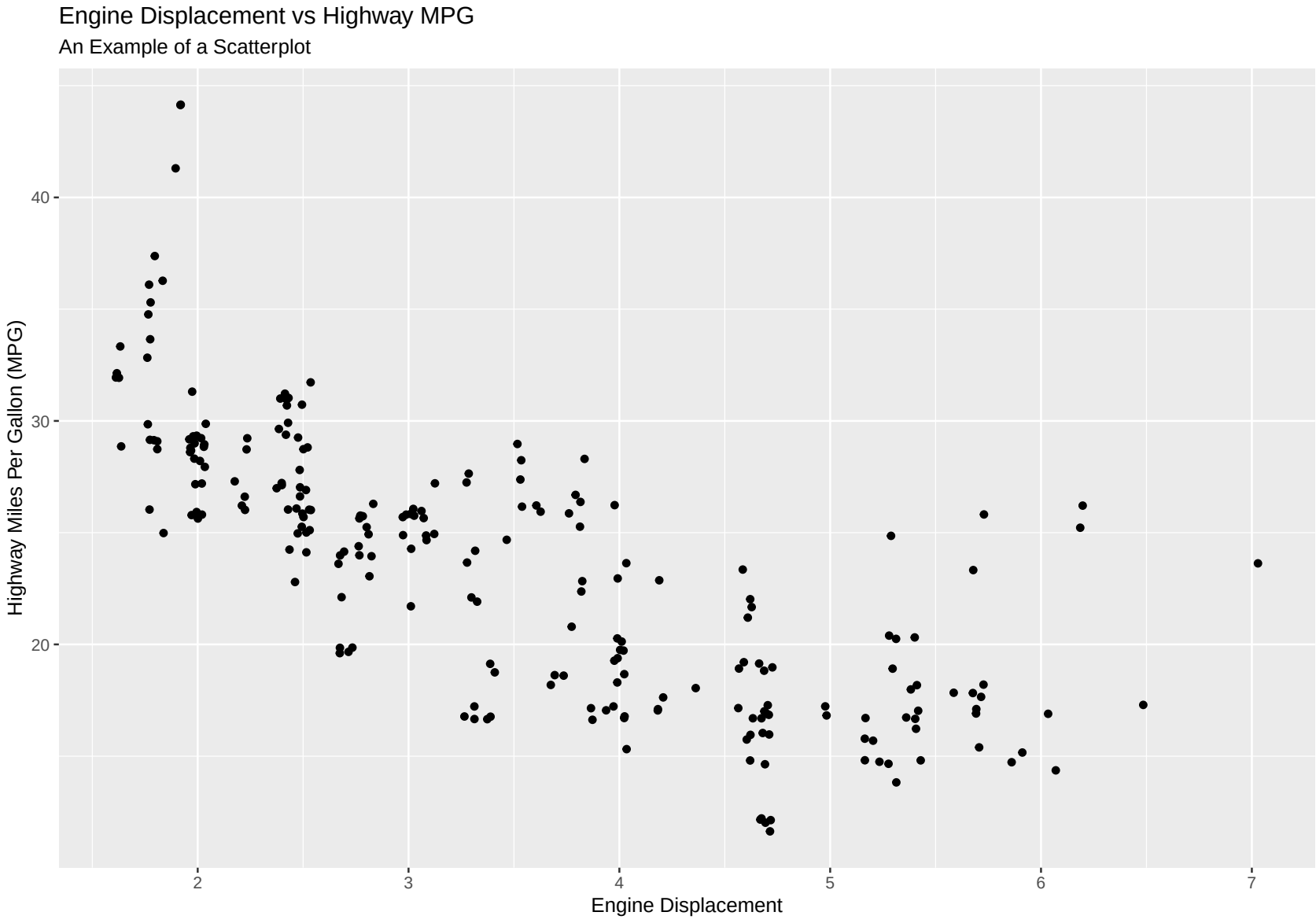
- So far, we haven't made any modifications to our graph or axes titles. However, we can do this quite easily with the `labs` function.

- For the plot we just generated, we can see that there is no main title and the axes titles are lower cased. Let's change that.

Adding Titles and Changing Themes

```
mpg |>
  ggplot(aes(x=displ,y=hwy)) +
  geom_point(position="jitter") +
  labs(x = "Engine Displacement",
       y = "Highway Miles Per Gallon (MPG)",
       title = "Engine Displacement vs Highway MPG",
       subtitle = "An Example of a Scatterplot")
```

Adding Titles and Changing Themes



Adding Titles and Changing Themes

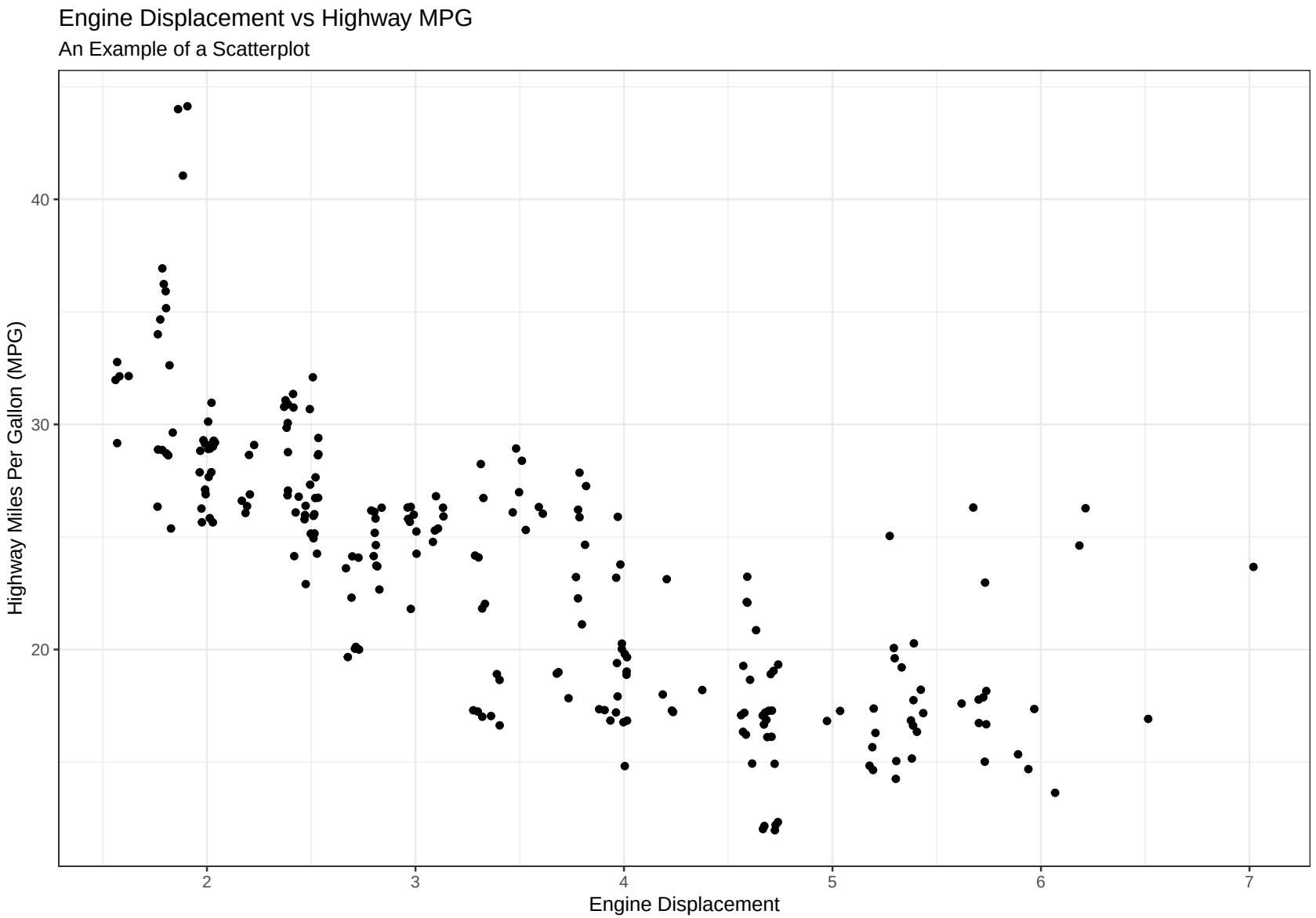
- We also have lots of options for changing the themes of our plots.
- Two big reasons why I change themes are: (1) I don't like the gray gridded background and (2) For a lot of academic journals, if you want to print figures in color, they make you pay to include that (it's nuts, I know).
- Changing themes is easy! We just add whatever theme we want to our main plot.

Adding Titles and Changing Themes

```r
## Adding Titles and Changing Themes
p <- mpg |>
  ggplot(aes(x=displ,y=hwy)) +
  geom_point(position="jitter") +
  labs(x = "Engine Displacement",
       y = "Highway Miles Per Gallon (MPG)",
       title = "Engine Displacement vs Highway MPG",
       subtitle = "An Example of a Scatterplot")
```

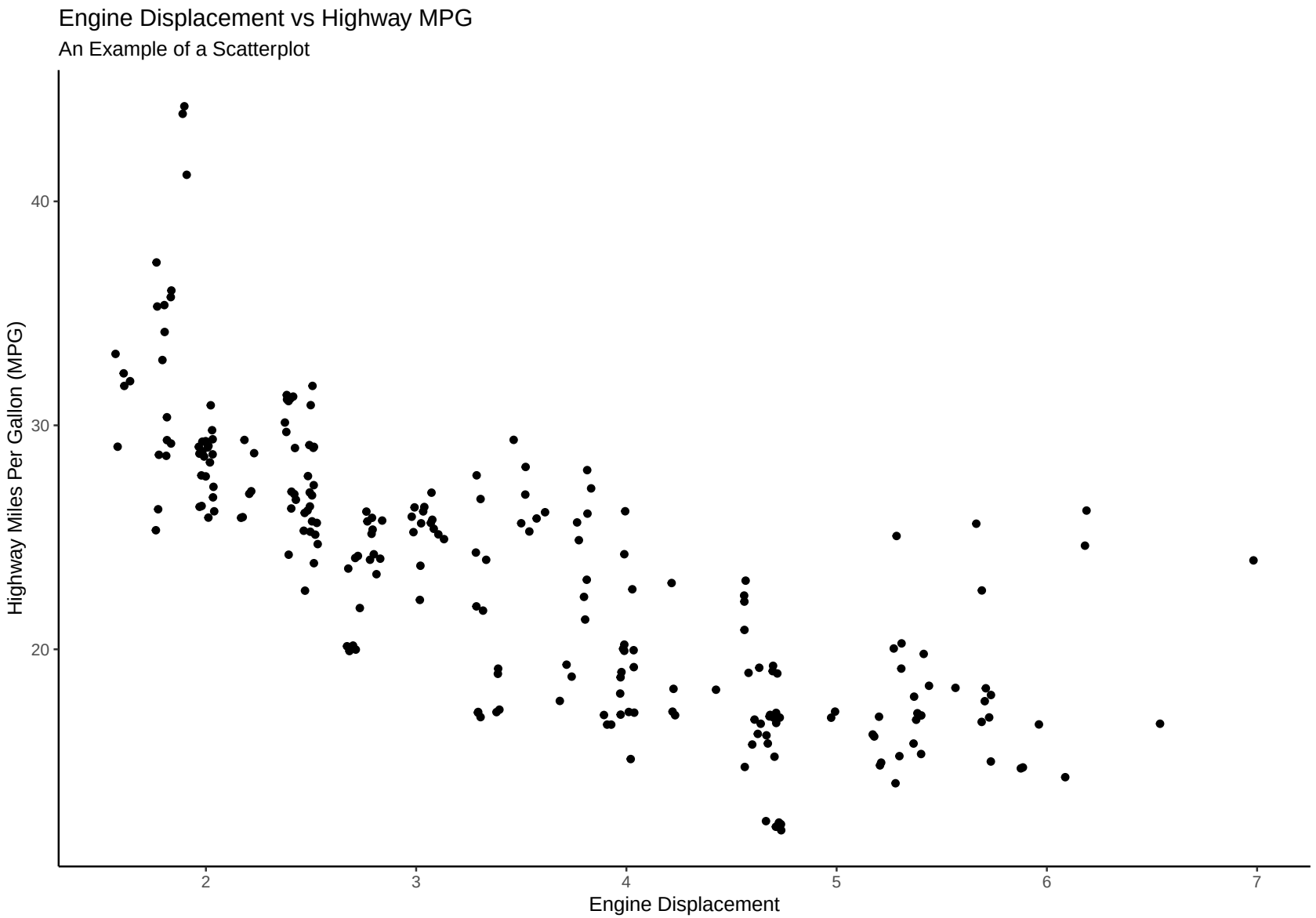
Adding Titles and Changing Themes

```
p + theme_bw()
```



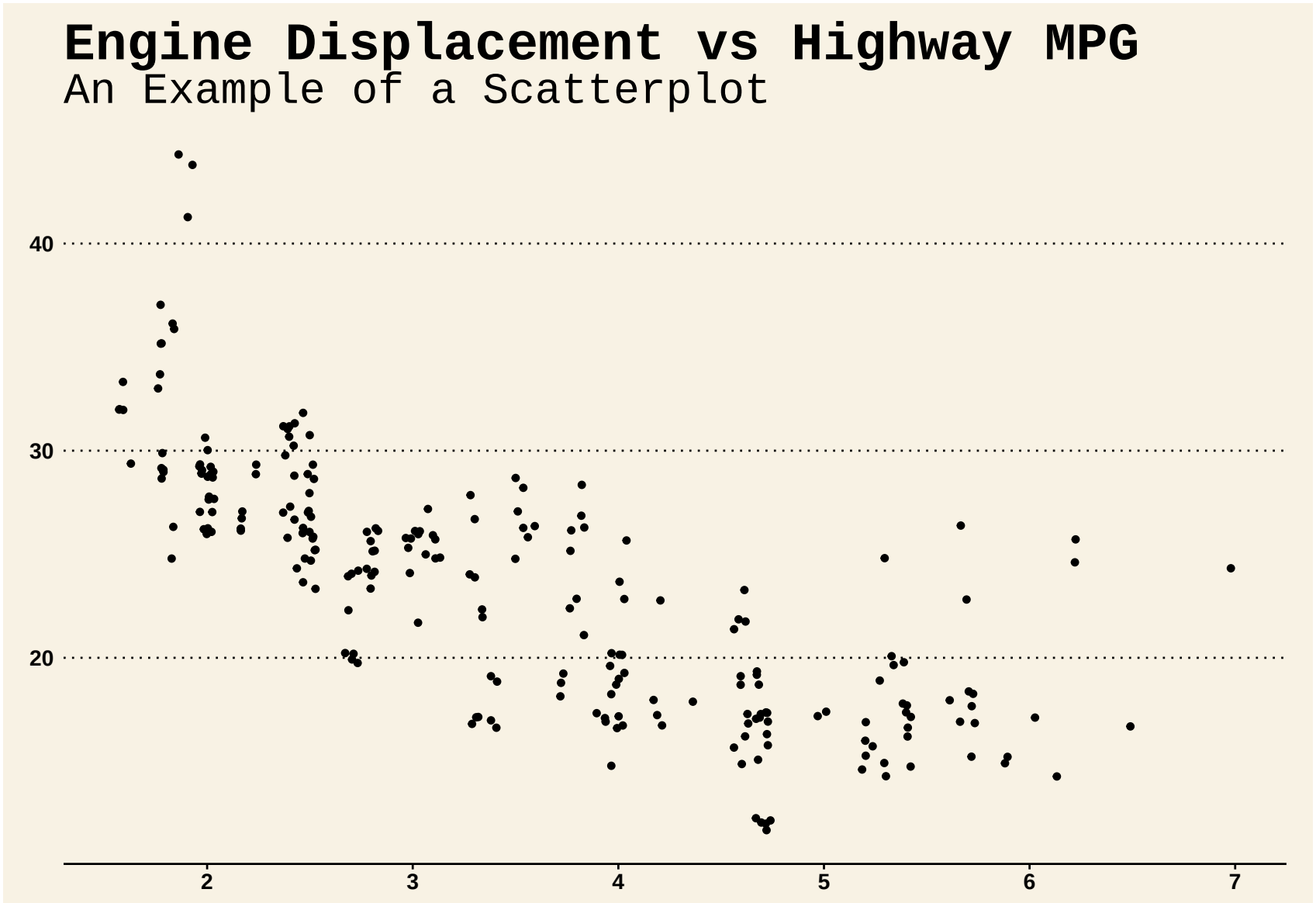
Adding Titles and Changing Themes

```
p + theme_classic()
```



Adding Titles and Changing Themes

```
library(ggthemes)
p + theme_wsj()
```



Practice Problem

- Using the `Lahman::Batting` dataset, let's plot a bar chart of Austin Riley's hits by category for the 2022 regular season.
- Once we generate this plot, what are a couple of things we could do to improve it?