week5_2

Hakan Mehmetcik

Layers, Geoms, and Facets: The art of Visual Communications

Layers

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.

```
glimpse(mpg)
```

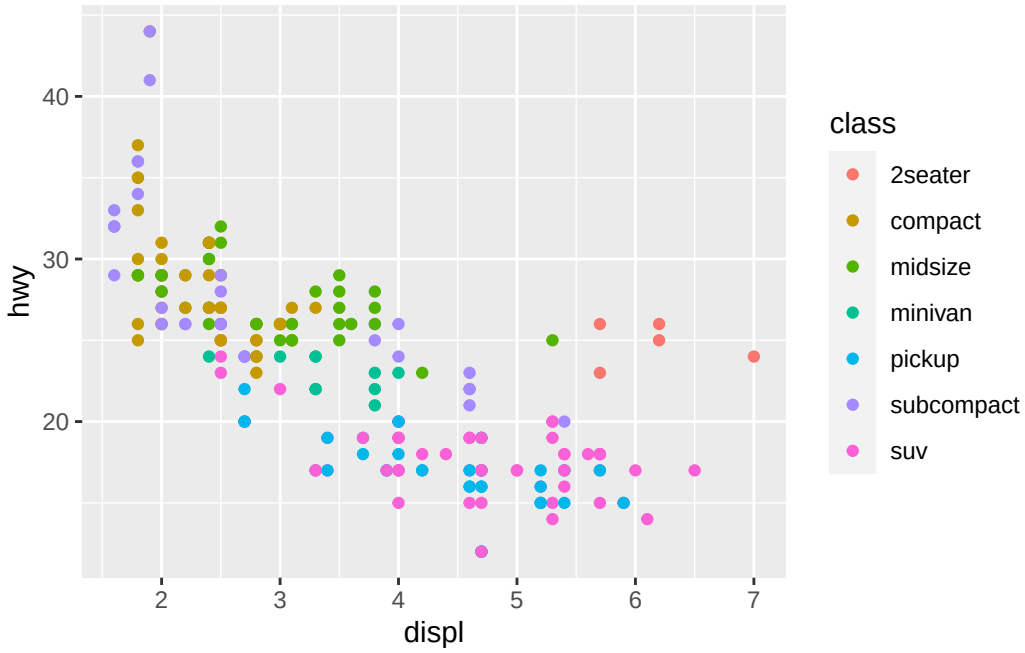
Rows: 234 Columns: 11 $ manufacturer “audi”, “audi”, “audi”, “audi”, “audi”, “audi”, “audi”, “~ $ model ”a4”, “a4”, “a4”, “a4”, “a4”, “a4”, “a4”, “a4 quattro”, “~ $ displ 1.8, 1.8, 2.0, 2.0, 2.8, 2.8, 3.1, 1.8, 1.8, 2.0, 2.0, 2.~ $ year 1999, 1999, 2008, 2008, 1999, 1999, 2008, 1999, 1999, 200~ $ cyl 4, 4, 4, 4, 6, 6, 6, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 8, 8, ~ $ trans ”auto(l5)“,”manual(m5)“,”manual(m6)“,”auto(av)“,”auto~ $ drv “f”, “f”, “f”, “f”, “f”, “f”, “f”, “4”, “4”, “4”, “4”, “4~ $ cty 18, 21, 20, 21, 16, 18, 18, 18, 16, 20, 19, 15, 17, 17, 1~ $ hwy 29, 29, 31, 30, 26, 26, 27, 26, 25, 28, 27, 25, 25, 25, 2~ $ fl ”p”, “p”, “p”, “p”, “p”, “p”, “p”, “p”, “p”, “p”, “p”, “p~ $ class ”compact”, “compact”, “compact”, “compact”, “compact”, “c~

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`.

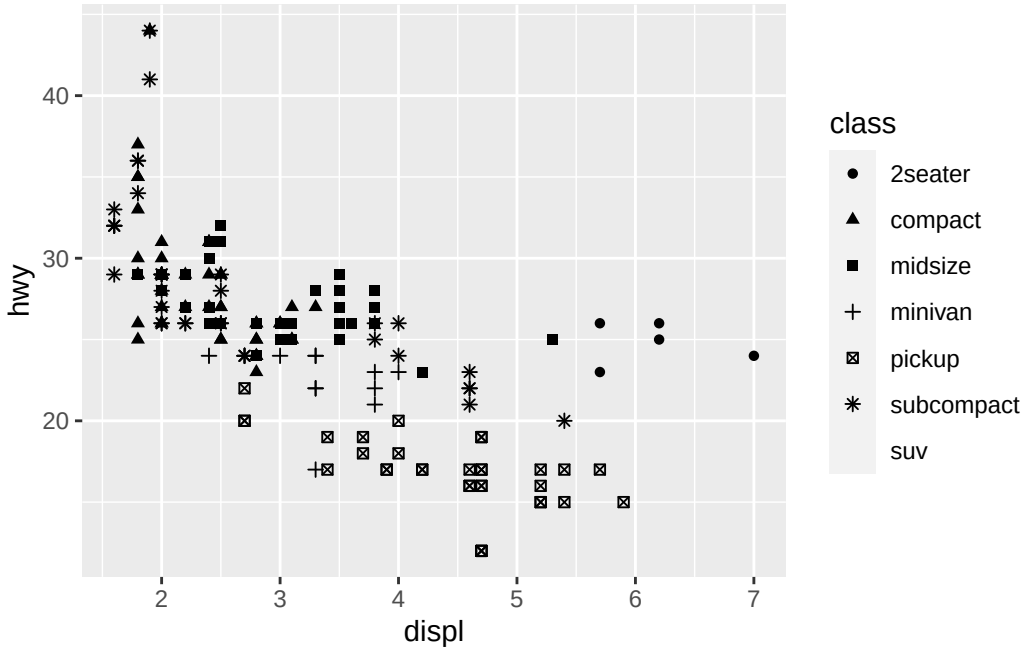
```
# Left
ggplot(mpg, aes(x = displ, y = hwy, color = class)) +
  geom_point()
```




```
# 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()`).
```

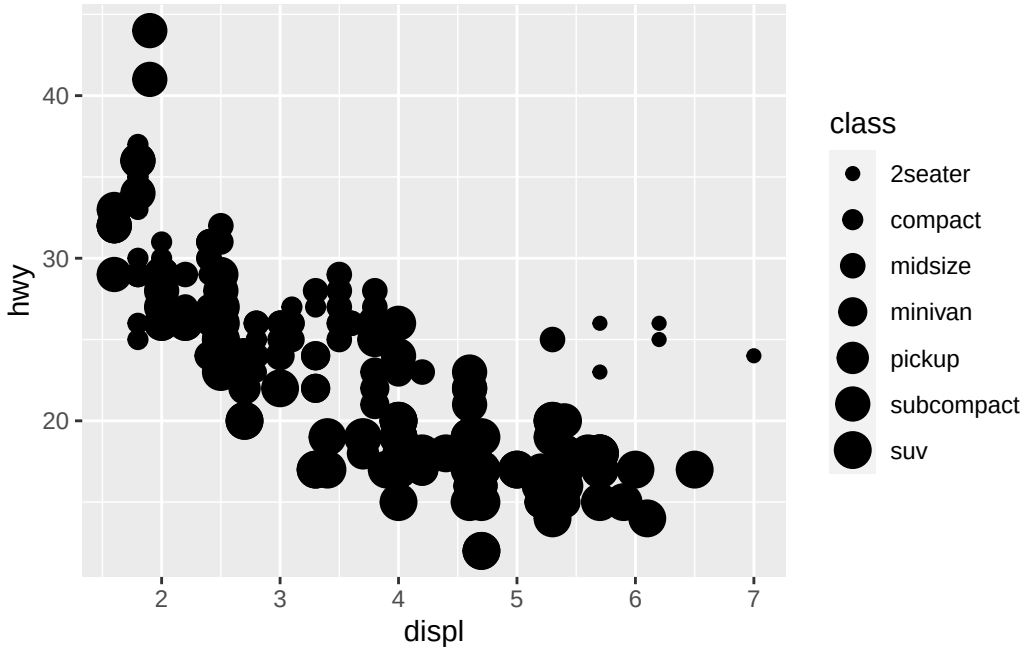



```
#> Warning: The shape palette can deal with a maximum of 6 discrete values because more
#> than 6 becomes difficult to discriminate
#>   you have requested 7 values. Consider specifying shapes manually if you
#>    need that many have them.
#> Warning: Removed 62 rows containing missing values or values outside the scale range
#> (`geom_point()`).
```

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

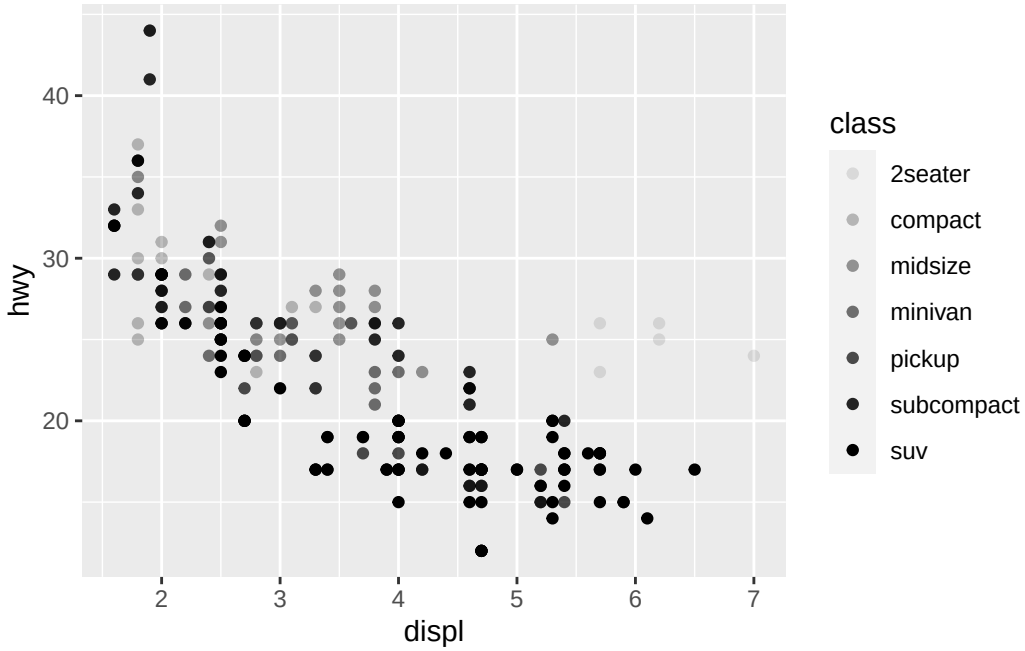
```
# Left
ggplot(mpg, aes(x = displ, y = hwy, size = class)) +
  geom_point()
```

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



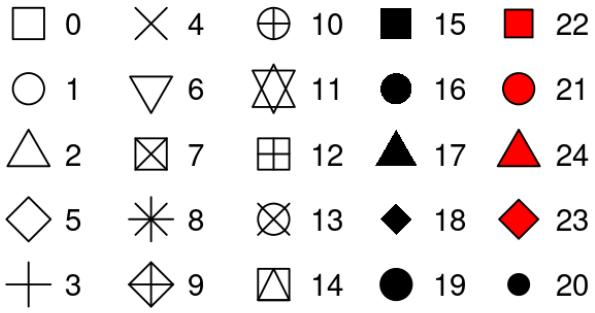

```
#> Warning: Using size for a discrete variable is not advised.

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

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




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

> **Note**
>
> 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.



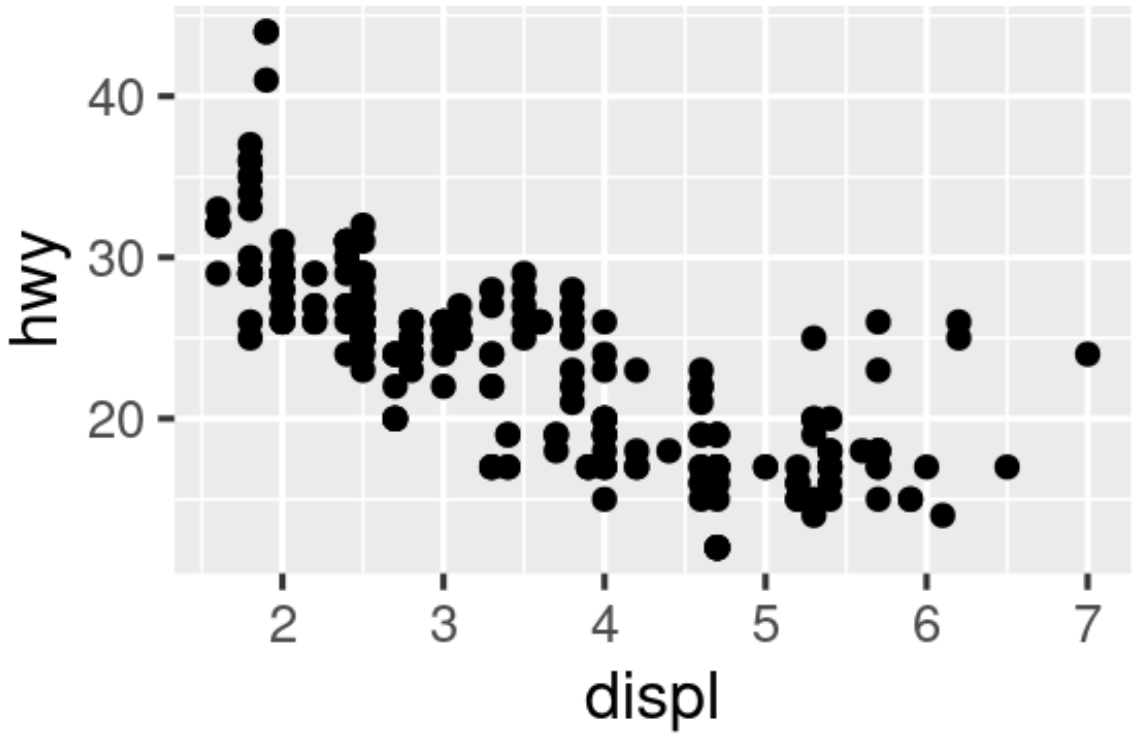
Here, the color doesn't convey information about a variable, but only changes the appearance

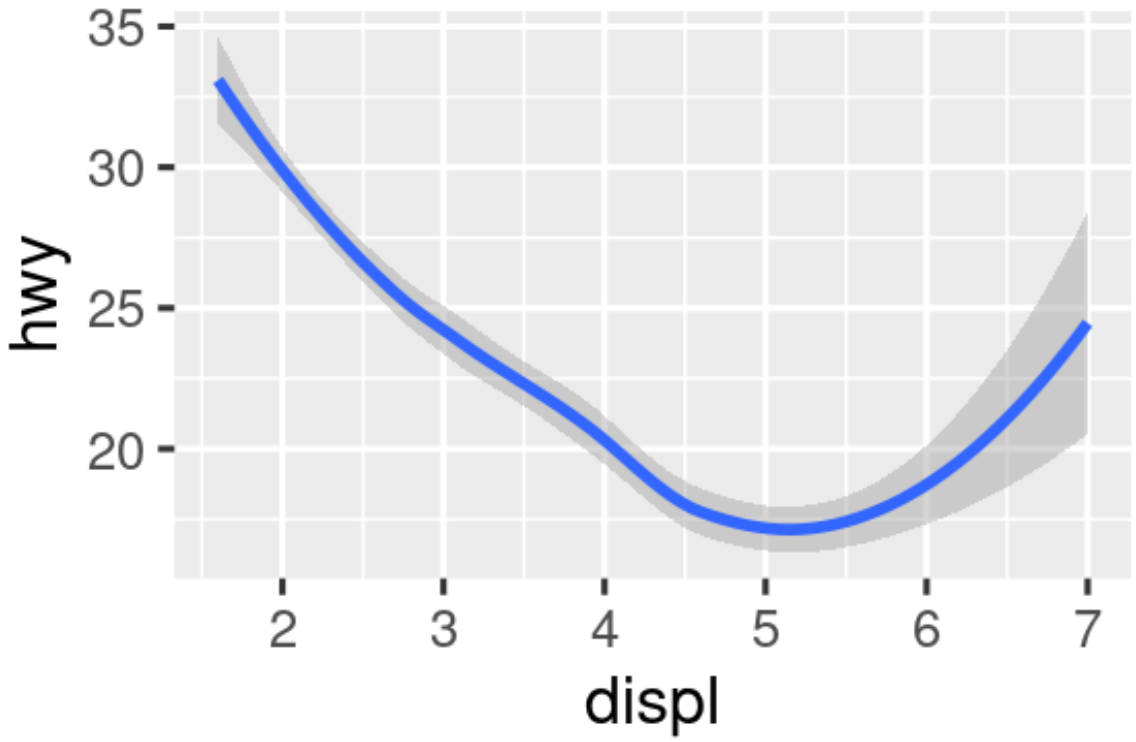
of the plot. You'll need to pick a value that makes sense for that aesthetic:

- 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 the figure above.

> **Note**
>
> You can learn more about all possible aesthetic mappings in the aesthetic specifications vignette at https://ggplot2.tidyverse.org/articles/ggplot2-specs.html.

Geometric Objects

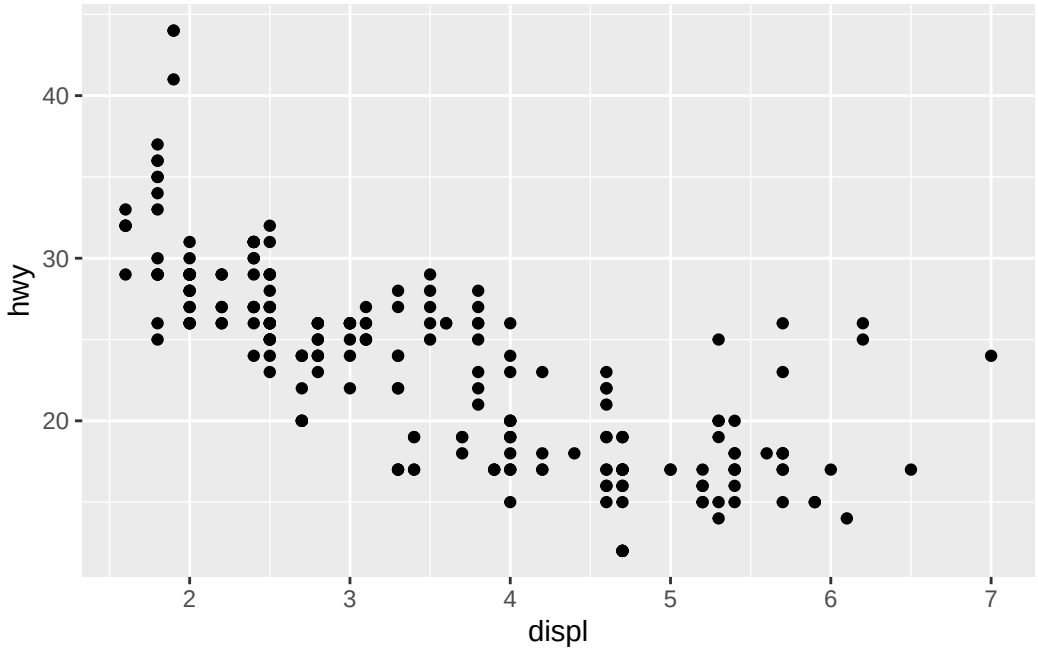




Both plots contain the same x variable, the same y variable, and both describe the same data. But the plots are not identical. Each plot uses a different geometric object, geom, to represent the data. The plot on the left uses the point geom, and the plot on the right uses the smooth geom, a smooth line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

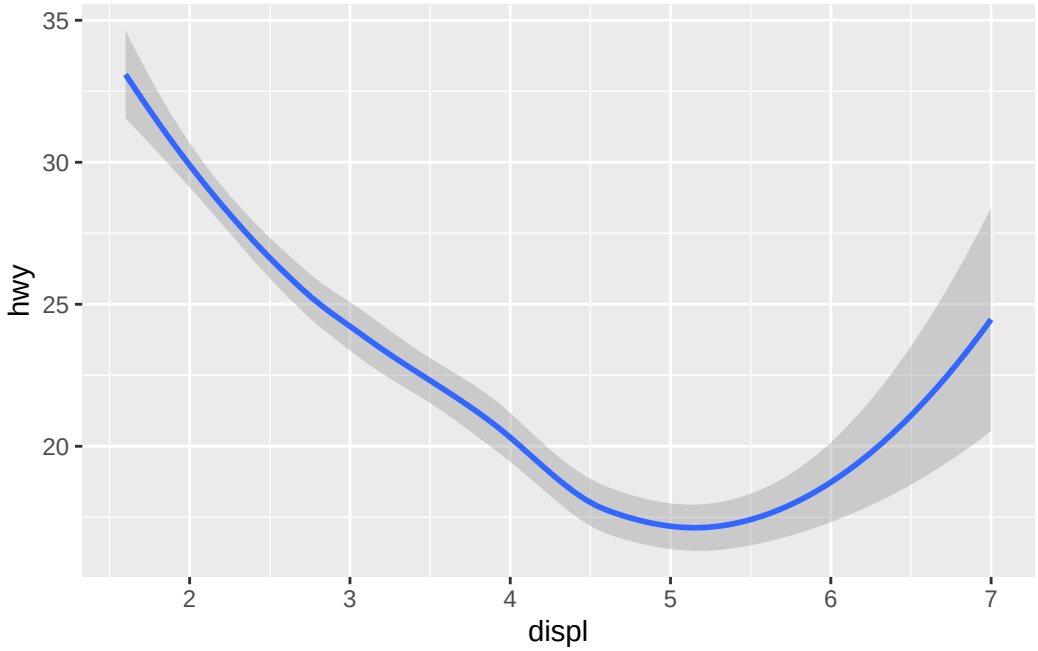
To change the geom in your plot, change the geom function that you add to `ggplot()`. For instance, to make the plots above, you can use the following code:

```
# 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'
```

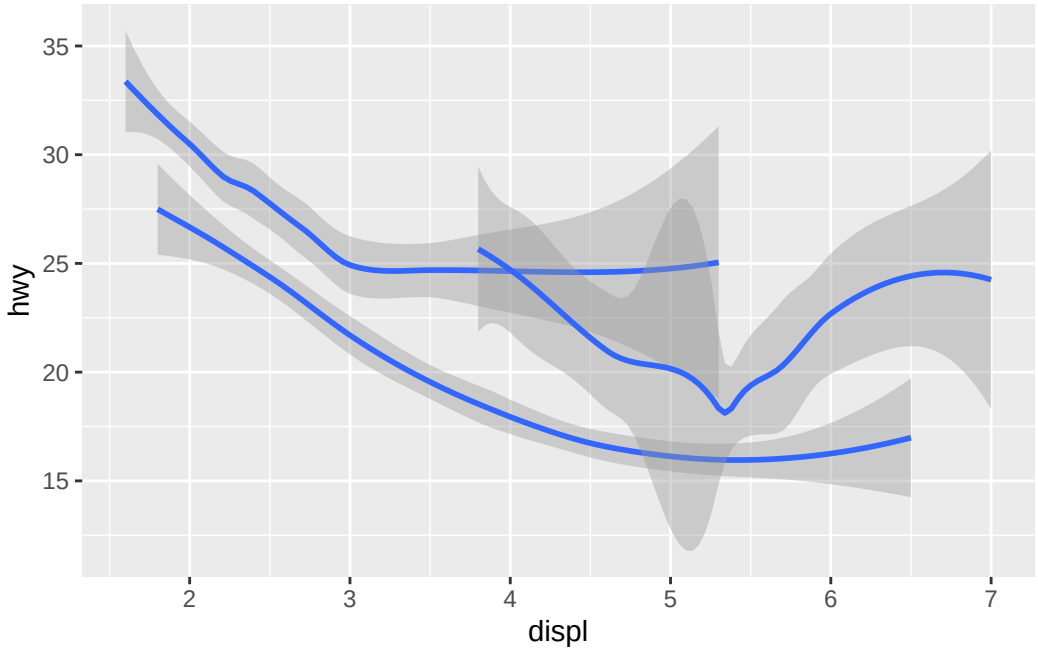



```
#> `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

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.

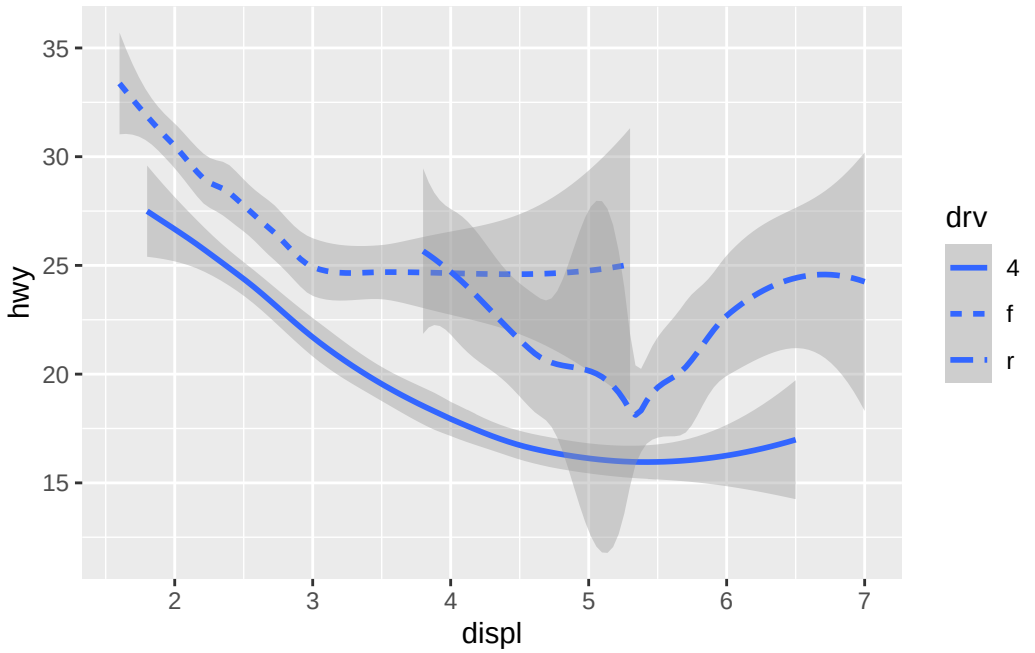
```
# Left
ggplot(mpg, aes(x = displ, y = hwy, shape = drv)) +
  geom_smooth()
```

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




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

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

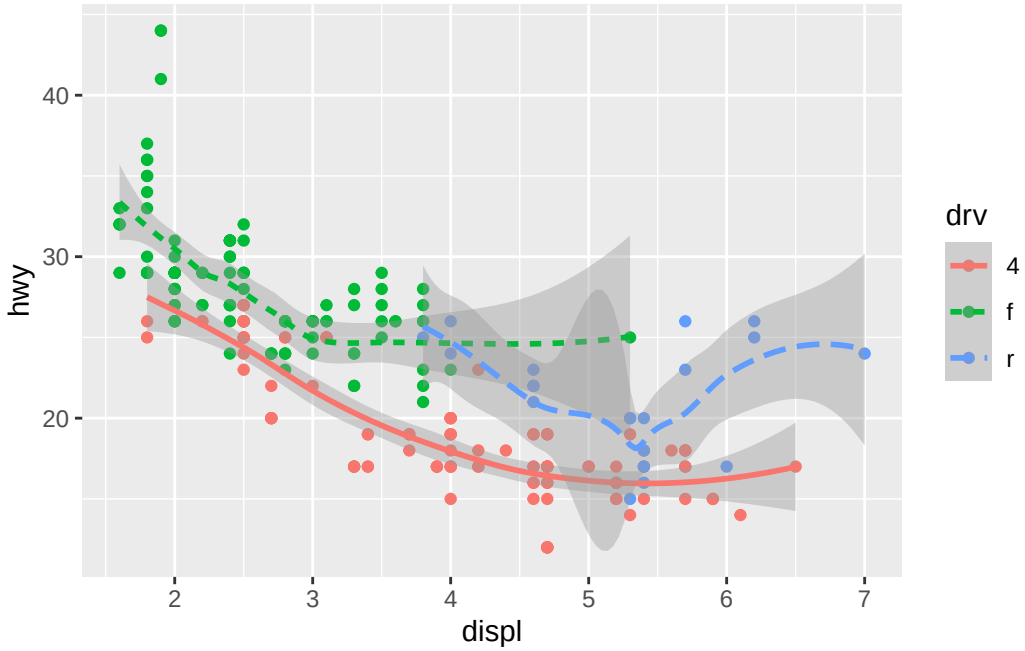


Here, `geom_smooth()` separates the cars into three lines based on their `drv` value, which describes a car's drive train. One line describes all of the points that have a `4` value, one line describes all of the points that have an `f` value, and one line describes all of the points that have an `r`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

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

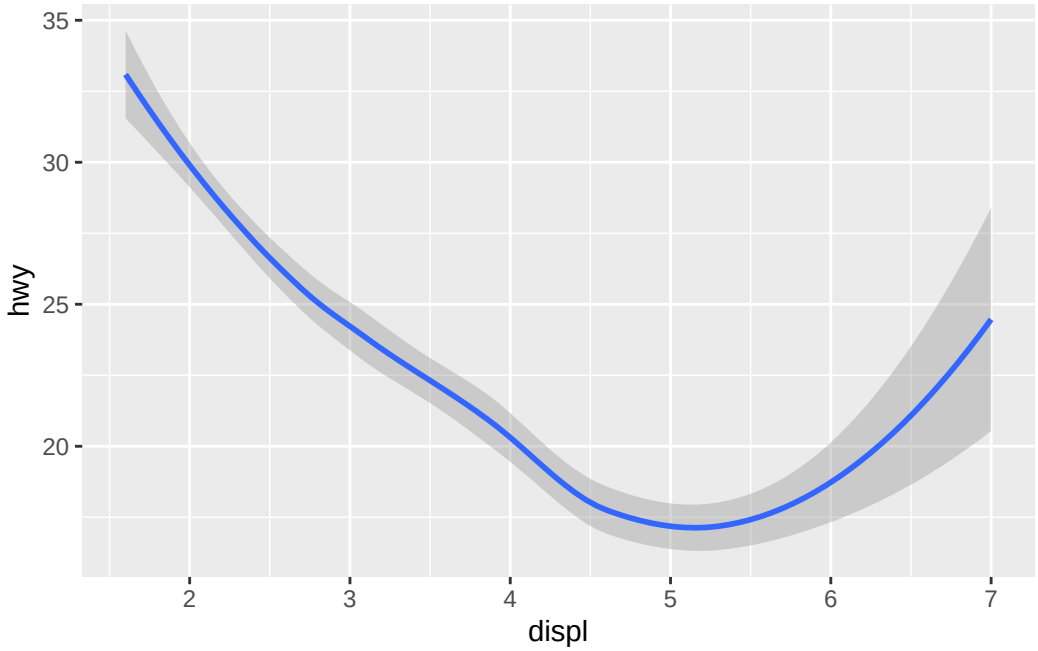


Notice that this plot contains two geoms in the same graph.

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.

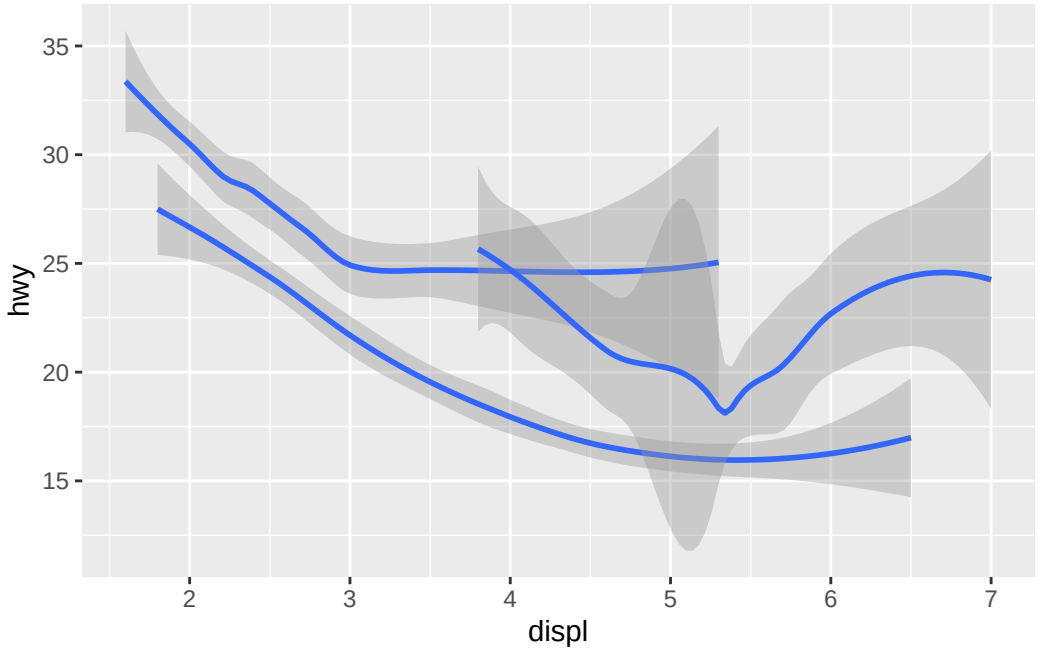
```
# First
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth()
```

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



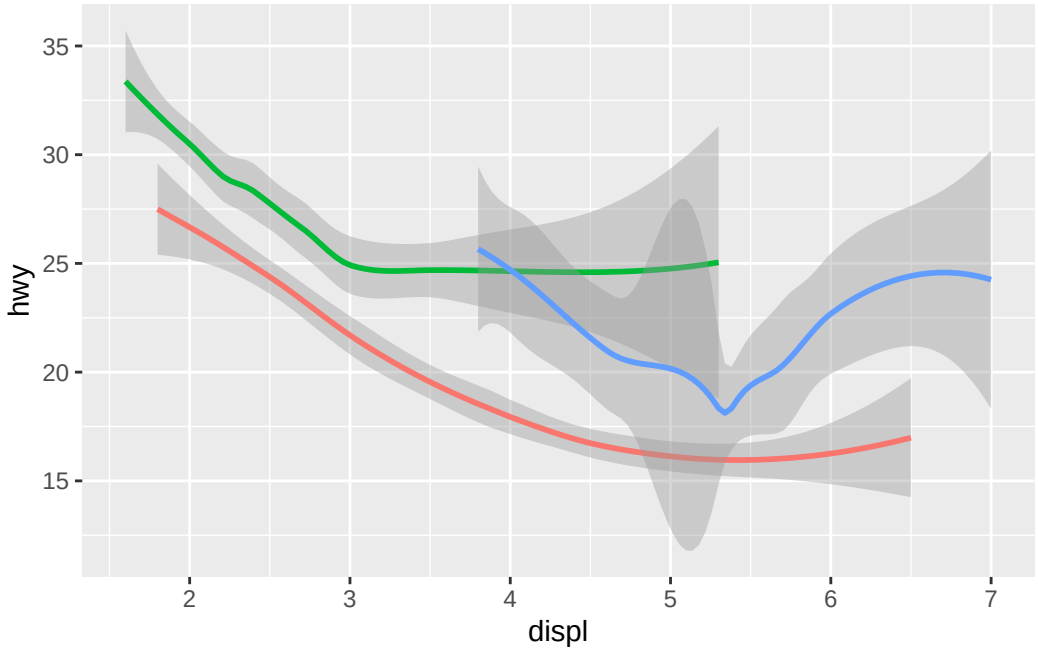

```
# Second
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth(aes(group = drv))
```

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




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

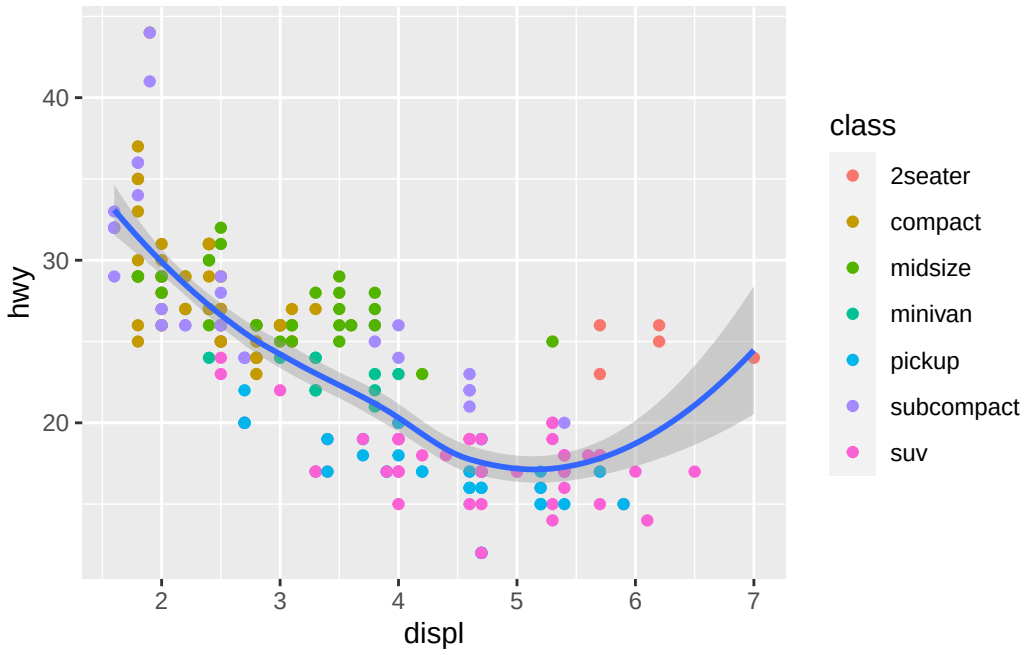
```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



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

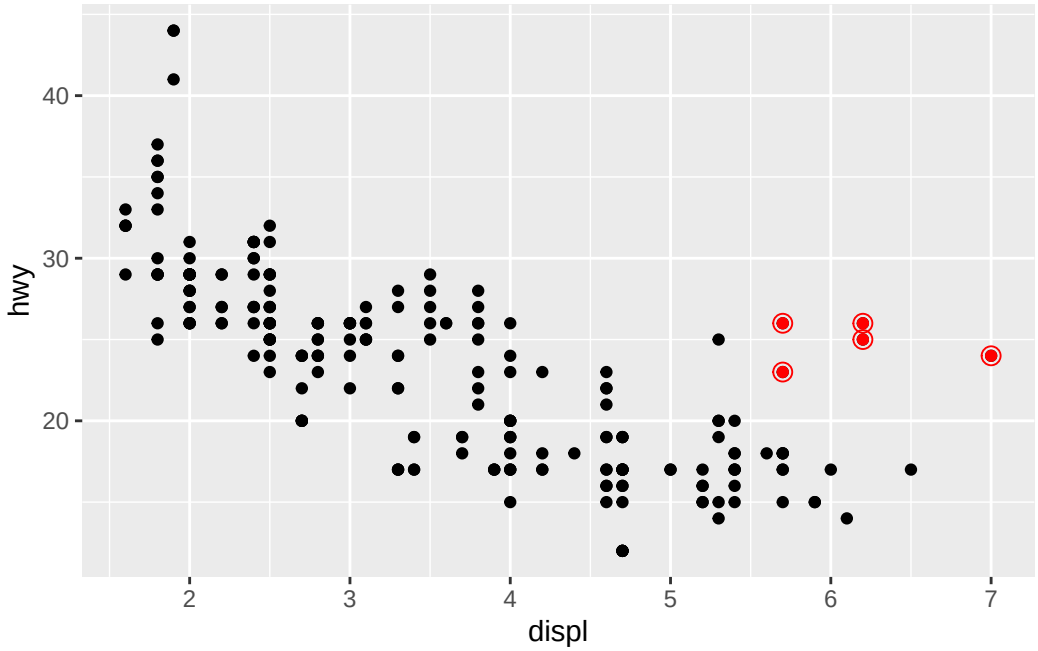
```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth()
```

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



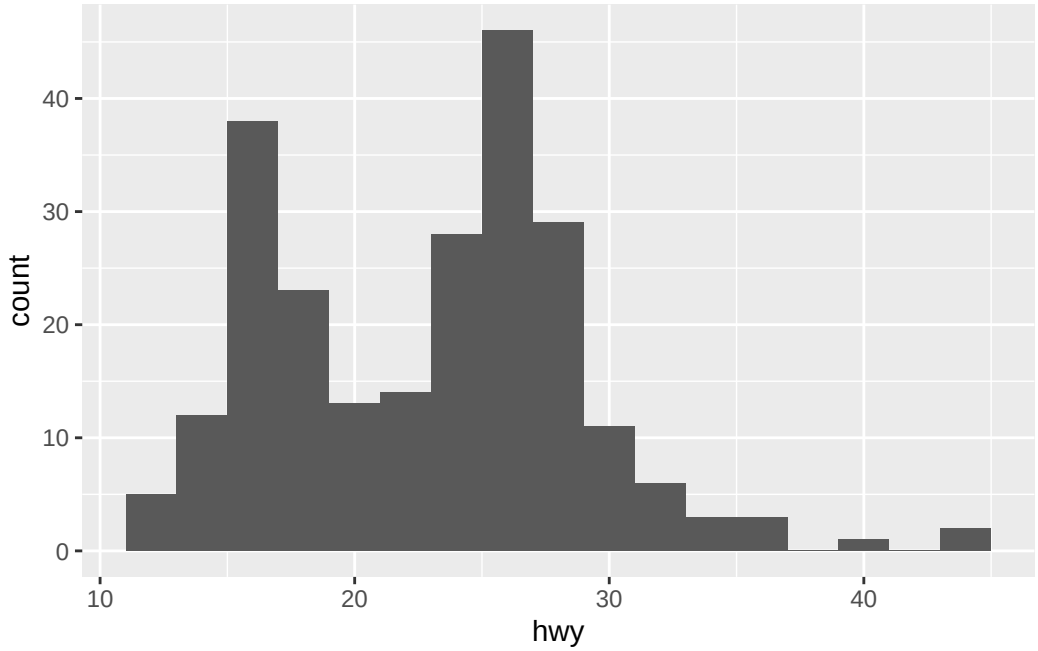
You can use the same idea to specify different `data` for each layer. Here, we use red points as well as open circles to highlight two-seater cars. The local data argument in `geom_point()` overrides the global data argument in `ggplot()` for that layer only.

```
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"
  )
```

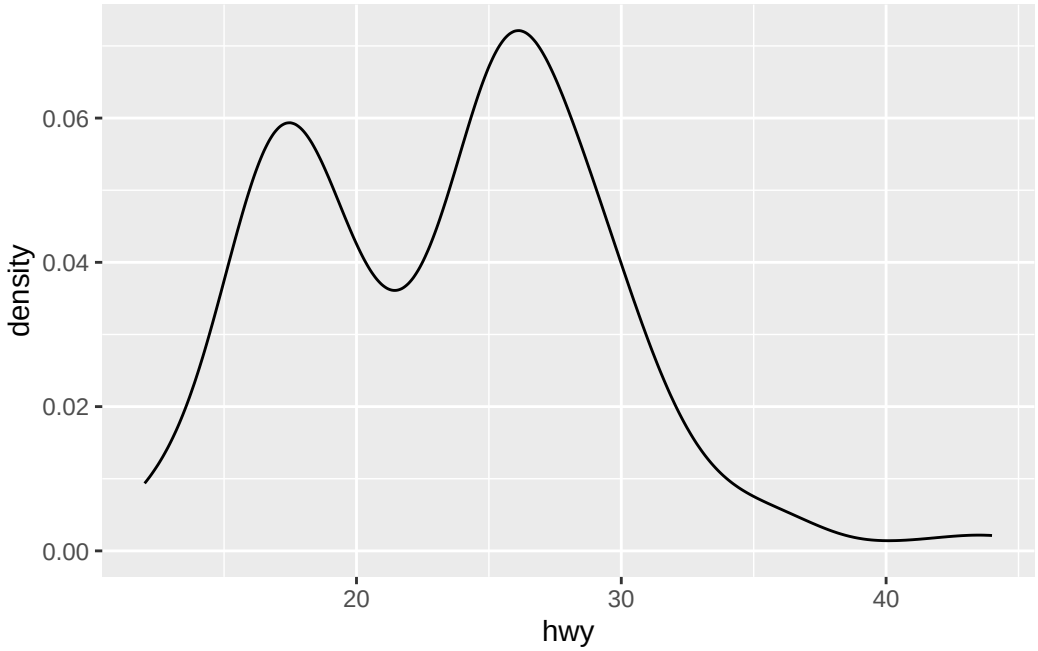


Geoms are the fundamental building blocks of ggplot2. You can completely transform the look of your plot by changing its geom, and different geoms can reveal different features of your data. For example, the histogram and density plot below reveal that the distribution of highway mileage is bimodal and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

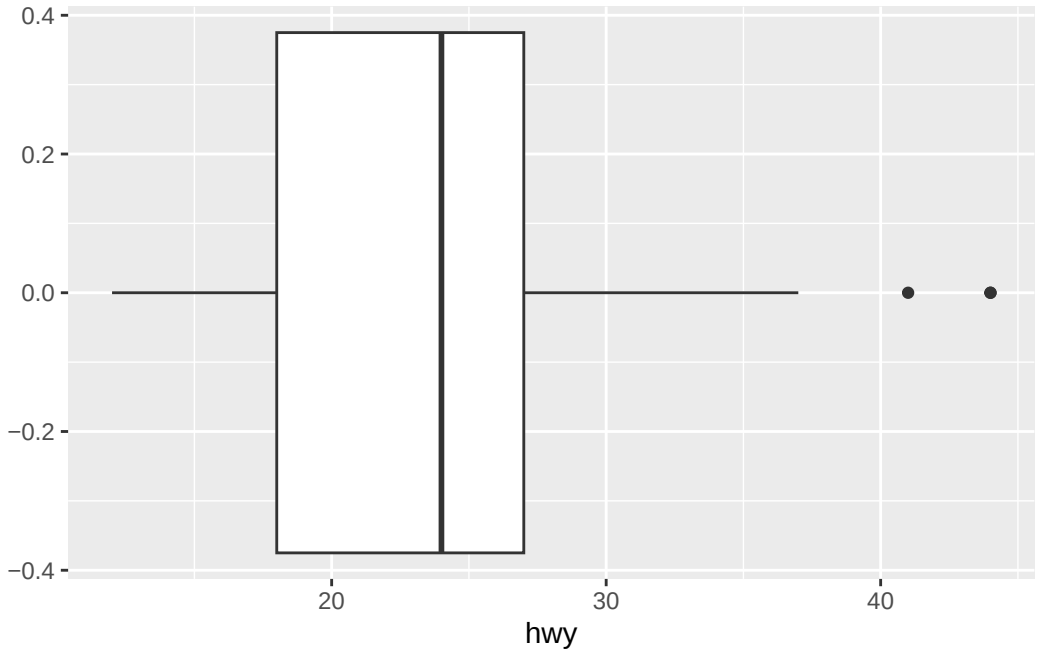
```
# First
ggplot(mpg, aes(x = hwy)) +
  geom_histogram(binwidth = 2)
```




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



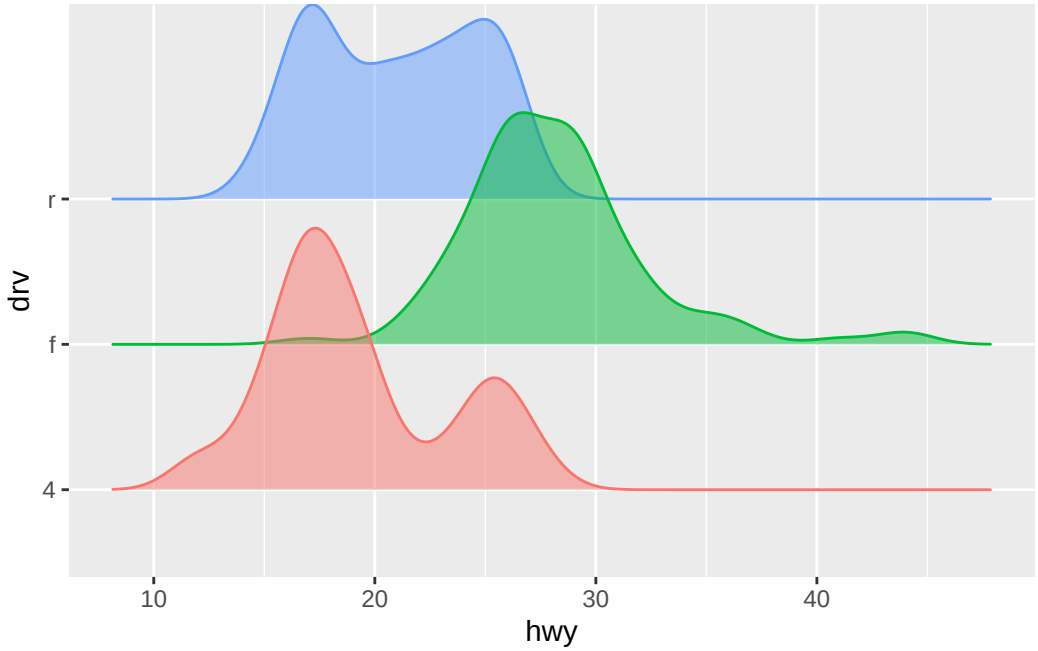

```
# Third
ggplot(mpg, aes(x = hwy)) +
  geom_boxplot()
```



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
```



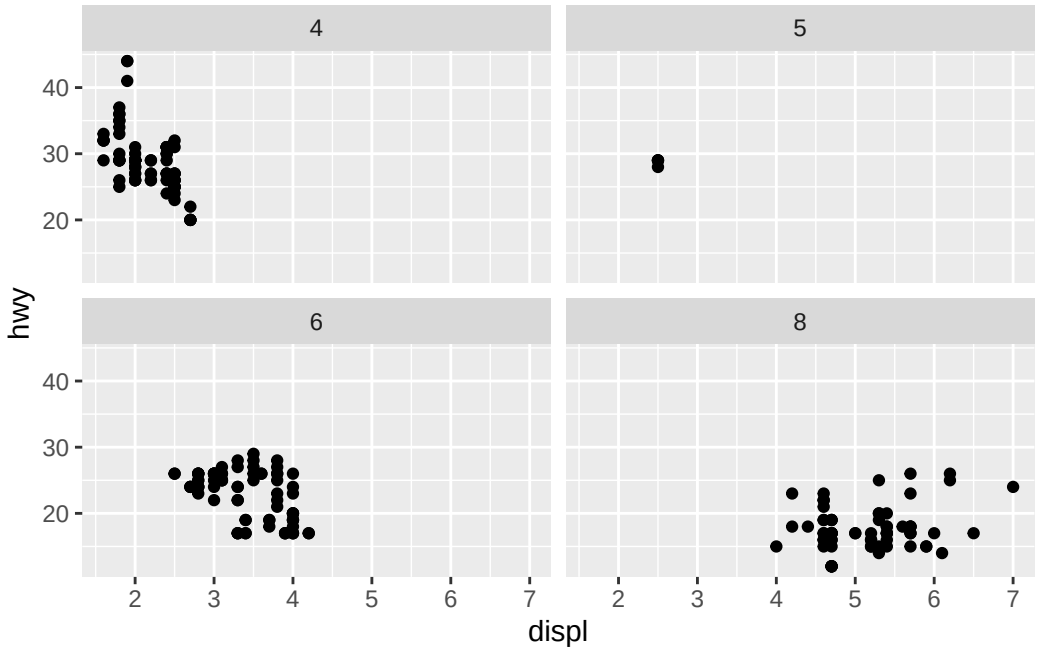

```
#> Picking joint bandwidth of 1.28
```

> **Note**
>
> 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`).

Facets

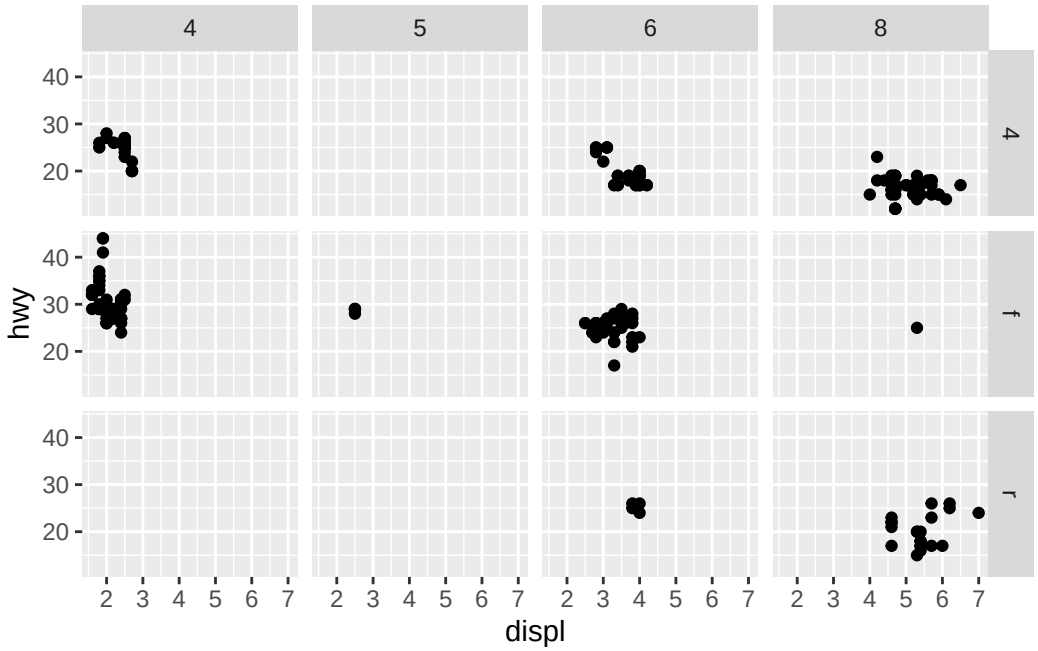
You have already learned about faceting with `facet_wrap()`, which splits a plot into subplots that each display one subset of the data based on a categorical variable.

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



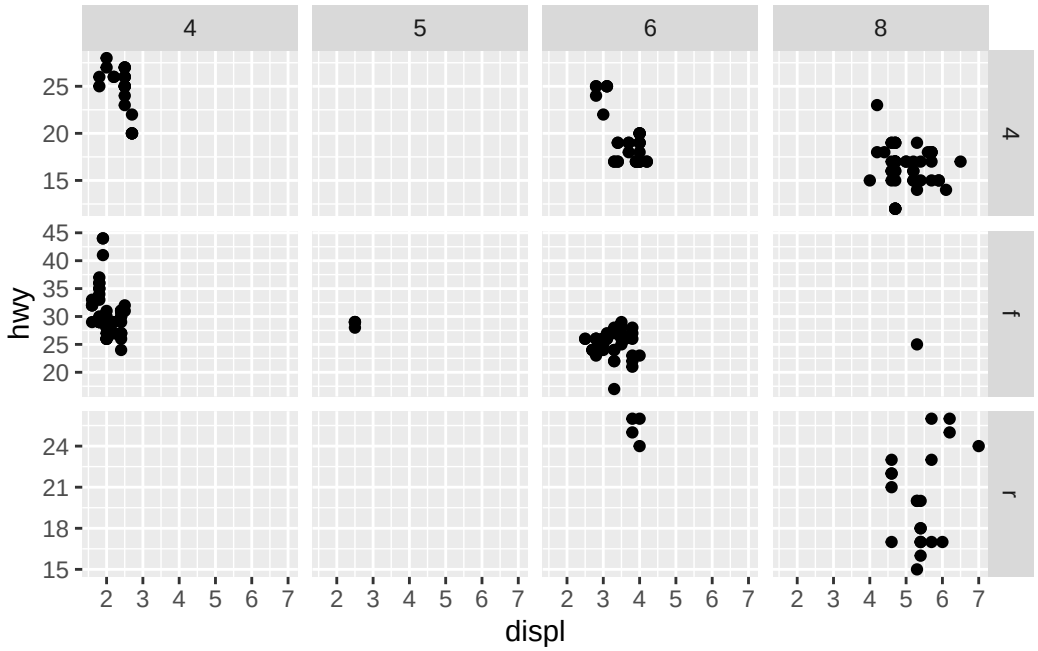
To facet your plot with the combination of two variables, switch from `facet_wrap()` to `facet_grid()`. The first argument of `facet_grid()` is also a formula, but now it's a double sided formula: `rows ~ cols`.

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



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.

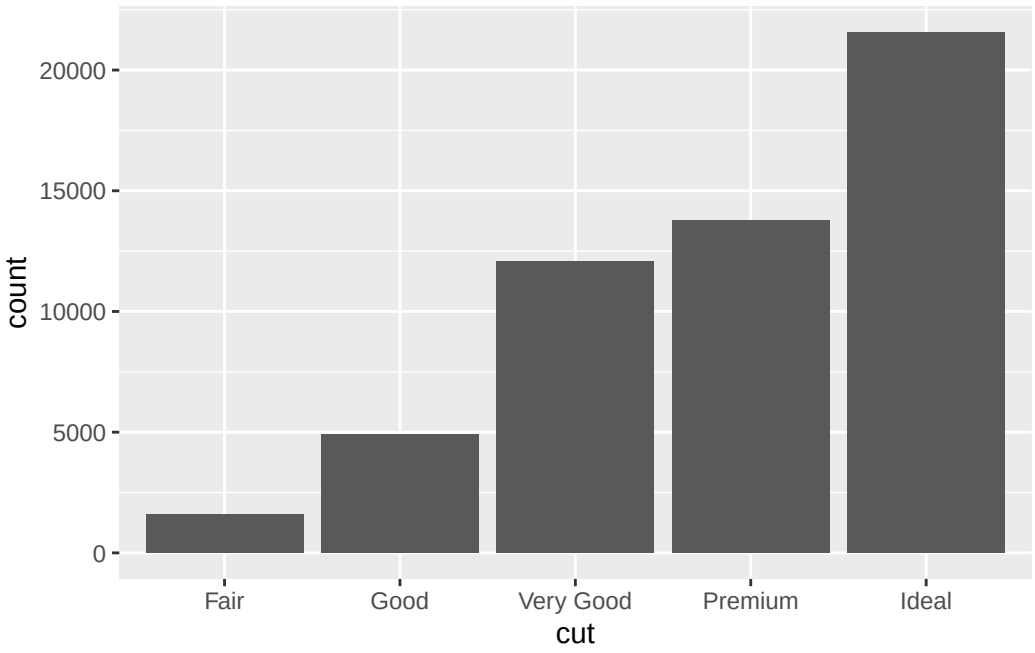
```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(drv ~ cyl, scales = "free_y")
```



Statistical Transformations

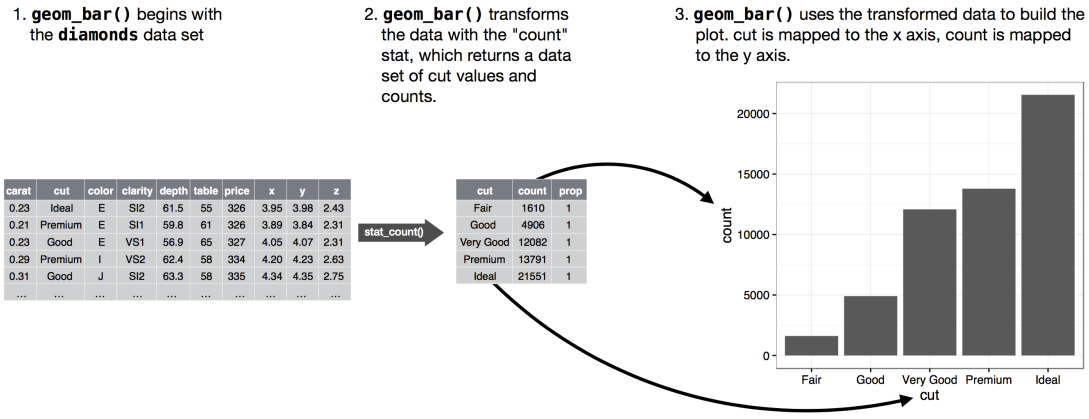
Consider a basic bar chart, drawn with `geom_bar()` or `geom_col()`. The following chart displays the total number of diamonds in the `diamonds` dataset, grouped by `cut`. The `diamonds` dataset is in the ggplot2 package and contains information on ~54,000 diamonds, including the `price`, `carat`, `color`, `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()
```



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:

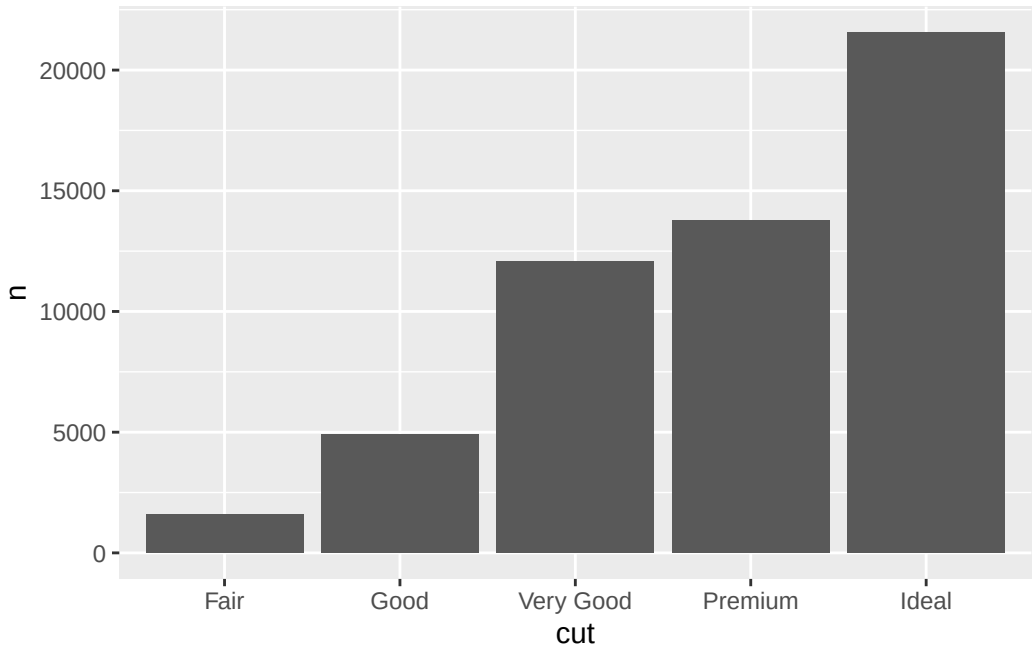
- Bar charts, histograms, and frequency polygons bin your data and then plot bin counts, the number of points that fall in each bin.
- Smoothers fit a model to your data and then plot predictions from the model.
- Boxplots compute the five-number summary of the distribution and then display that summary as a specially formatted box.
- The algorithm used to calculate new values for a graph is called a stat, short for statistical transformation. Figure below shows how this process works with geom_bar().



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:

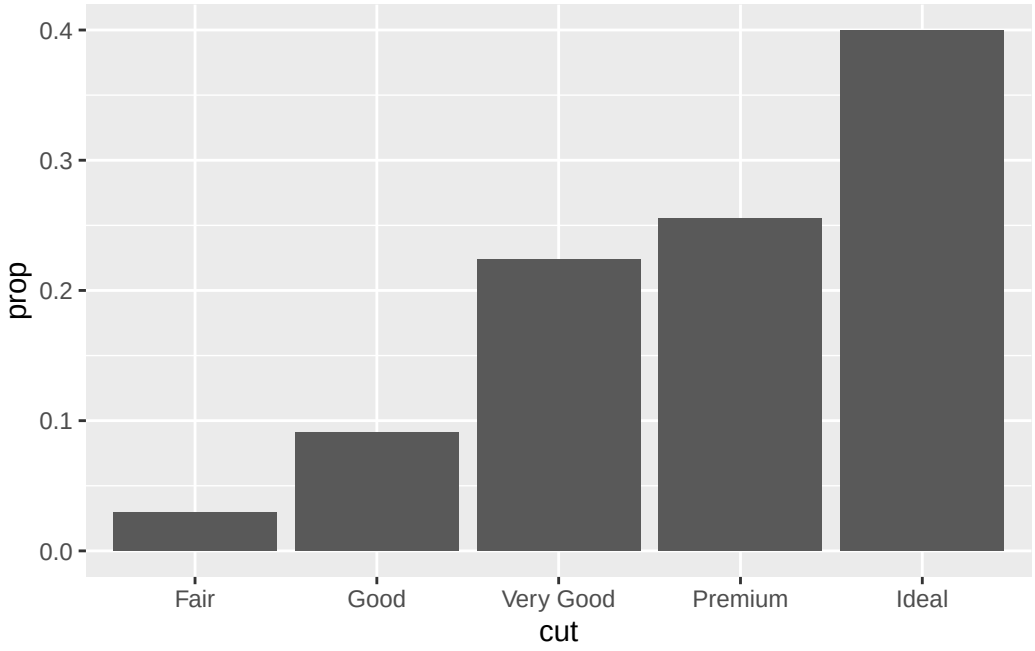
1. You might want to override the default stat. In the code below, we change the stat of `geom_bar()` from count (the default) to identity. This lets us map the height of the bars to the raw values of a y variable.

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



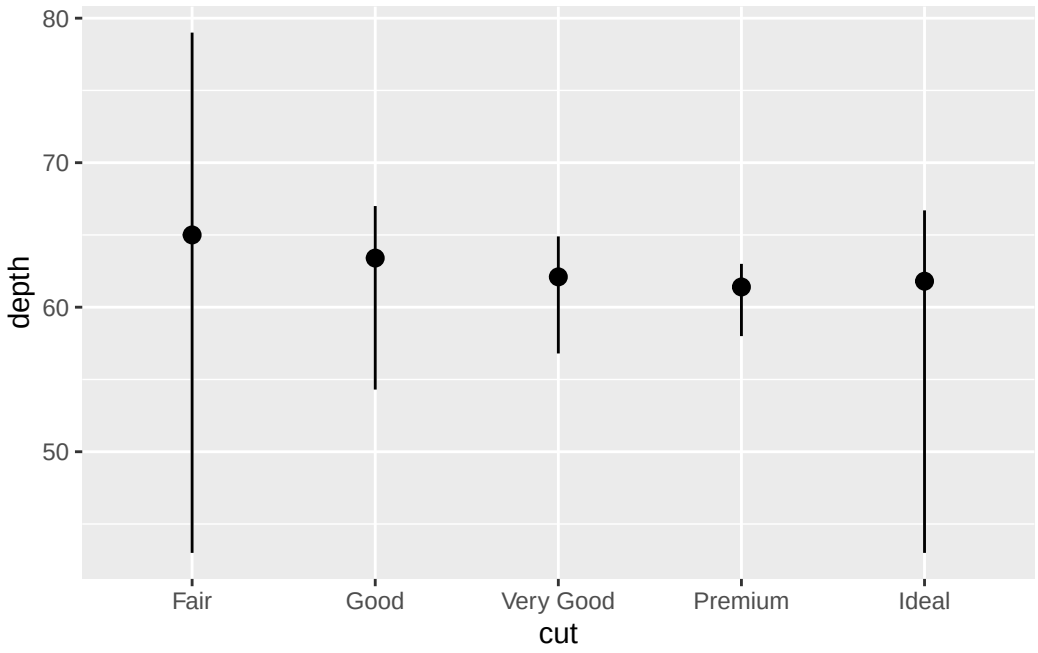
You might want to override the default mapping from transformed variables to aesthetics. For example, you might want to display a bar chart of proportions, rather than counts:

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



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
  )
```

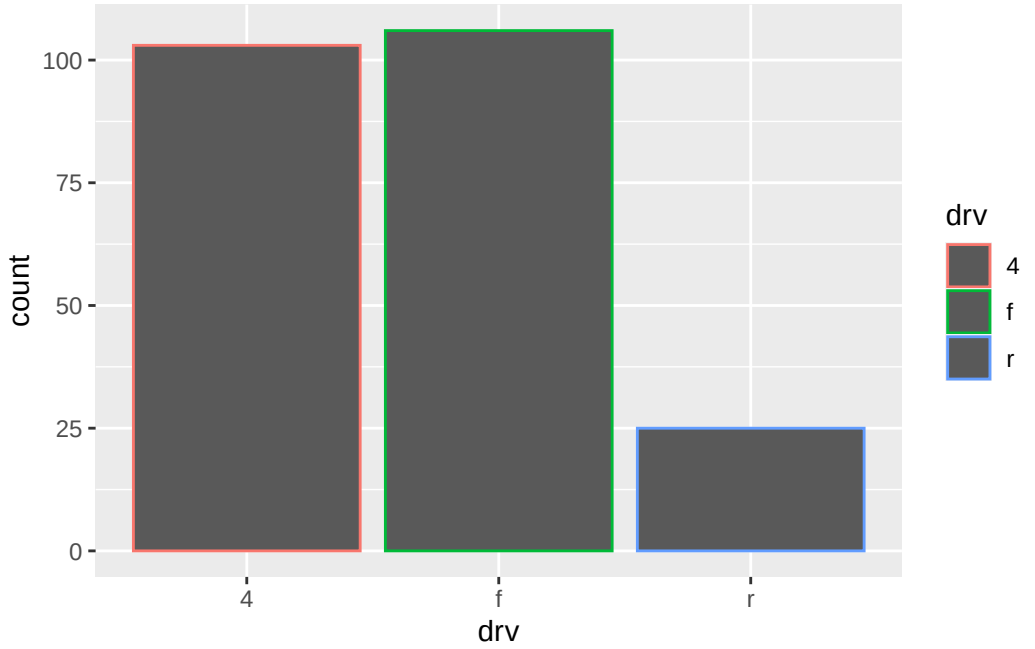


> **Note**
>
> ggplot2 provides more than 20 stats for you to use. Each stat is a function, so you can get help in the usual way, e.g., `?stat_bin`.

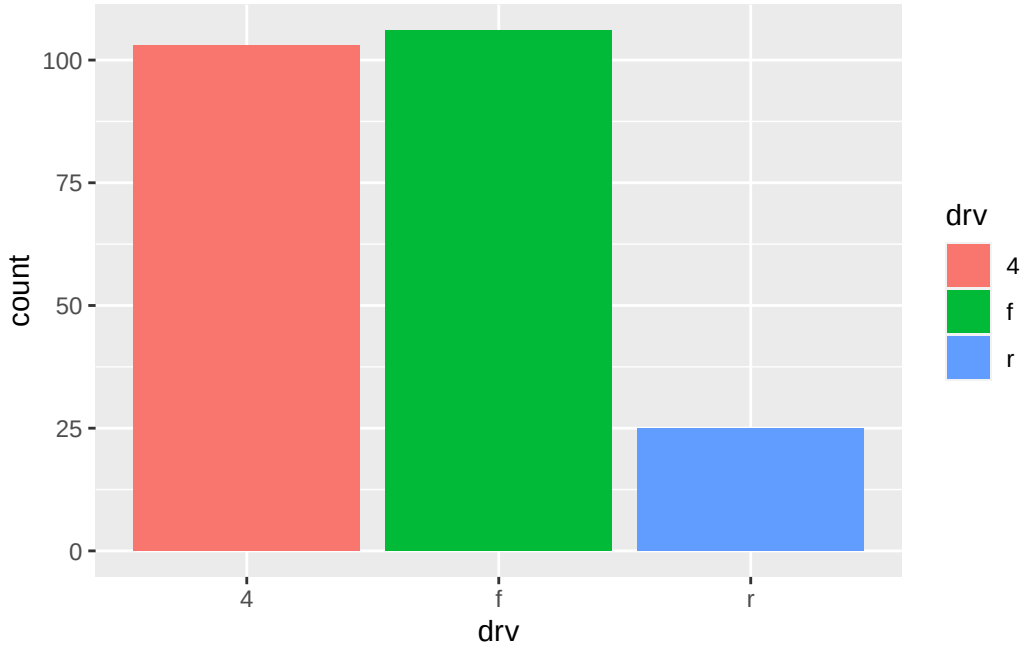
Position Adjustment

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:

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

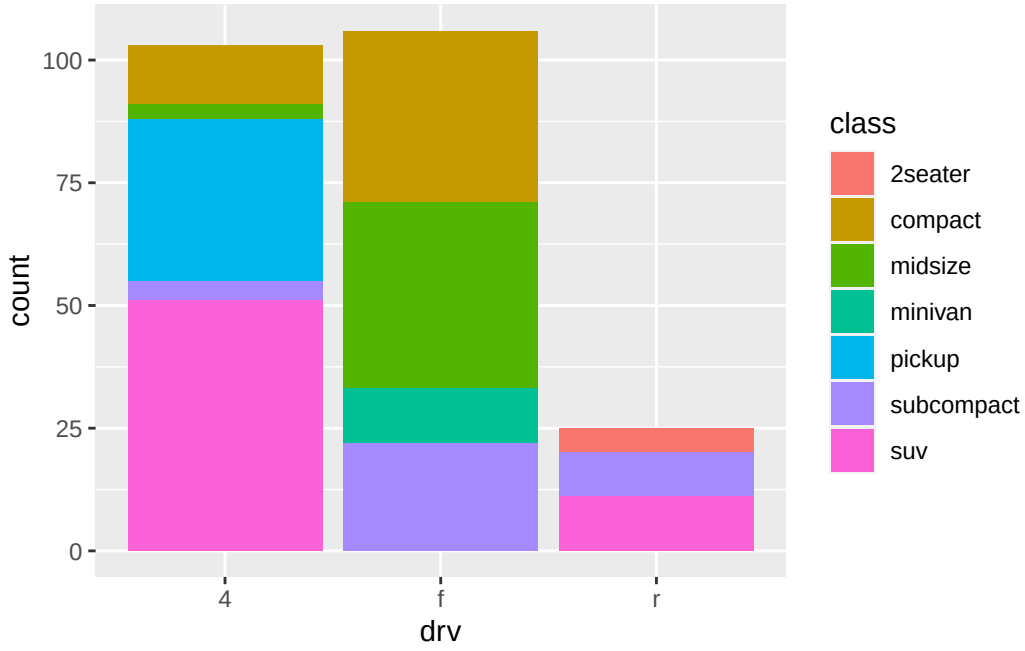



```
# Second
ggplot(mpg, aes(x = drv, fill = drv)) +
  geom_bar()
```



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()
```



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"`. Be curious, and give a try on them!

Overall:

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

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).