Data Visualization with ggplot2

Outline

Today, we'll cover the following:

- Creating scatterplot using ggplot2 and aesthetic mappings
- Facets and geometric objects
- Statistical transformations and position adjustments

Part I: Scatterplot using ggplot2

Tidyverse

ggplot2 is one of the core members of tidyverse. To access it, we need to install and load the tidyverse:

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

```
library(tidyverse)
```

Note that you only need to install a package once, but you need to reload it every time you start a new session.

Creating a scatterplot

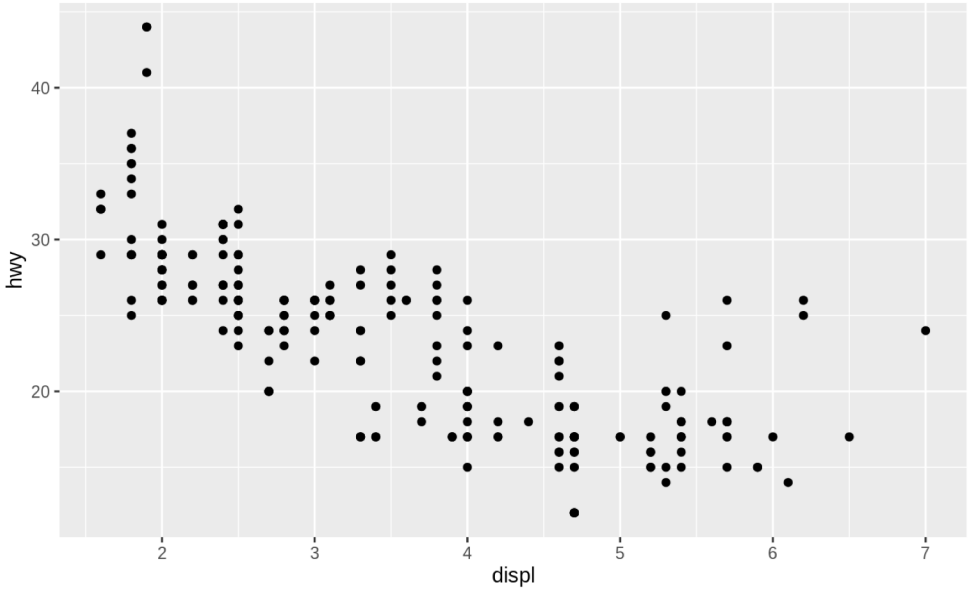
Suppose we want to know: Do cars with big engines use more fuel than cars with small engines?

We'll test our answer with the mpg data frame found in ggplot2. (`ggplot2::mpg`)

To plot mpg, run this code to put `displ` on the x-axis and `hwy` on the y-axis:

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

Creating a scatterplot



The plot shows a negative relationship between engine size(displ) and fuel efficiency(hwy).

In other words, cars with big engines use more fuel.

Aesthetic mappings

We can add a third variable to a two-dimensional scatterplot by mapping it to an **aesthetic**.

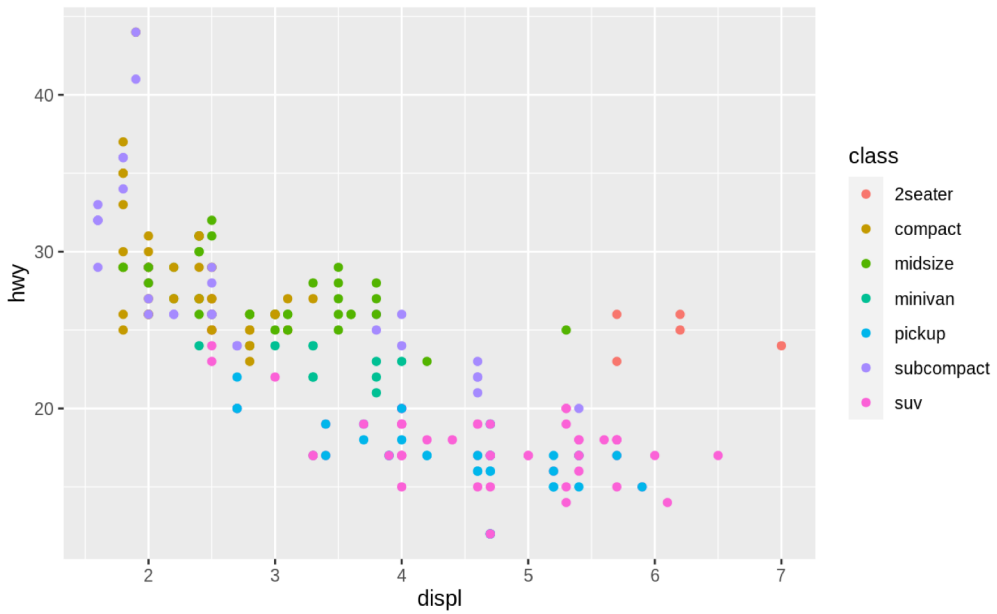
An **aesthetic** is a visual property of the objects in your plot. Aesthetics include things like the size, the shape, or the color of your points.

Aesthetic mappings

For example, for the mpg data frame, we can map the colors of points to the class variable to reveal the class of each car:

```
ggplot(data = mpg) +

    geom_point(mapping = aes(x = displ, y = hwy, color = class))
```

Aesthetic mappings



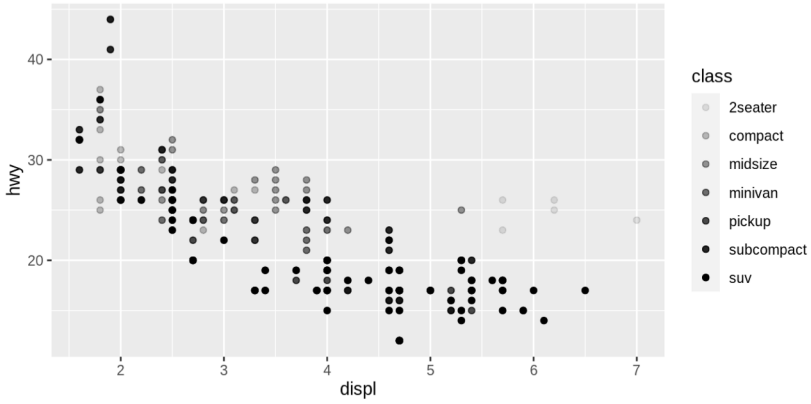
ggplot2 will automatically assign a unique level of the aesthetic (here a unique color) to each unique value of the variable.

It will also add a legend that explains which levels correspond to which values.

Aesthetic mappings

We could also map class to the alpha aesthetic, which controls the transparency of the points:

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



Part II: Facets and geometric objects

Facets

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

Facets

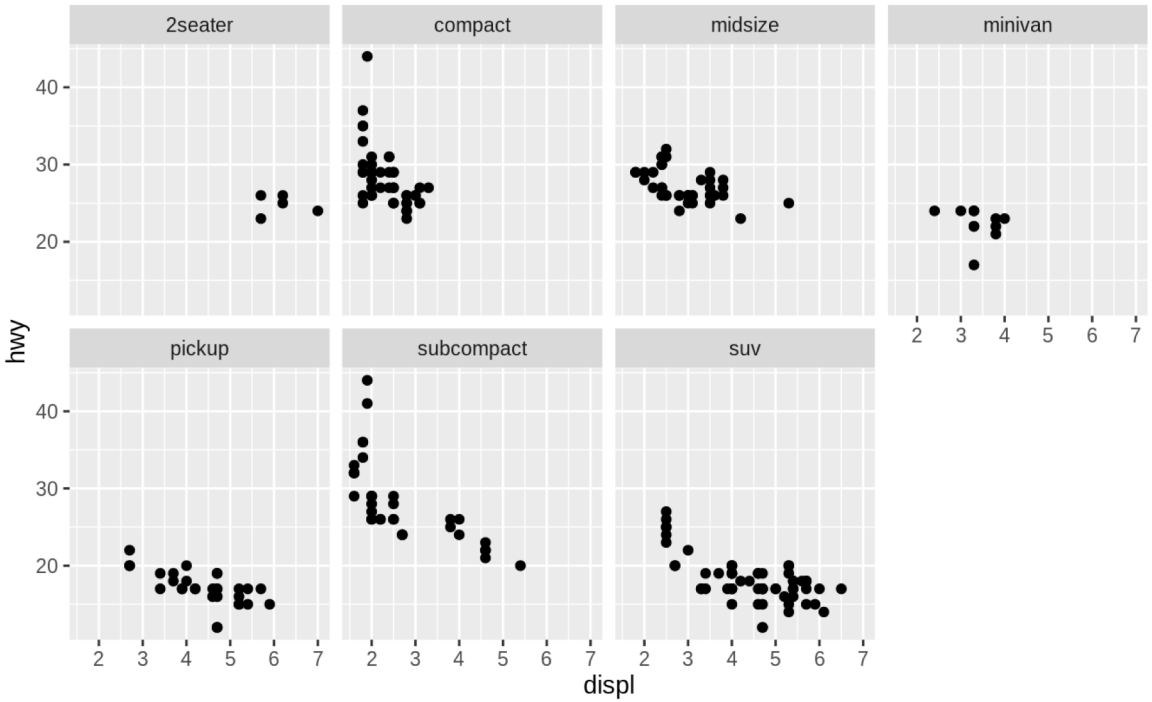
To facet plot by a single variable, use facet_wrap().

The first argument of facet_wrap() should be a formula, which you create with ~ followed by a variable name.The variable that you pass to facet_wrap() should be discrete.

For example, for mpg, we can facet plot by class:

```
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

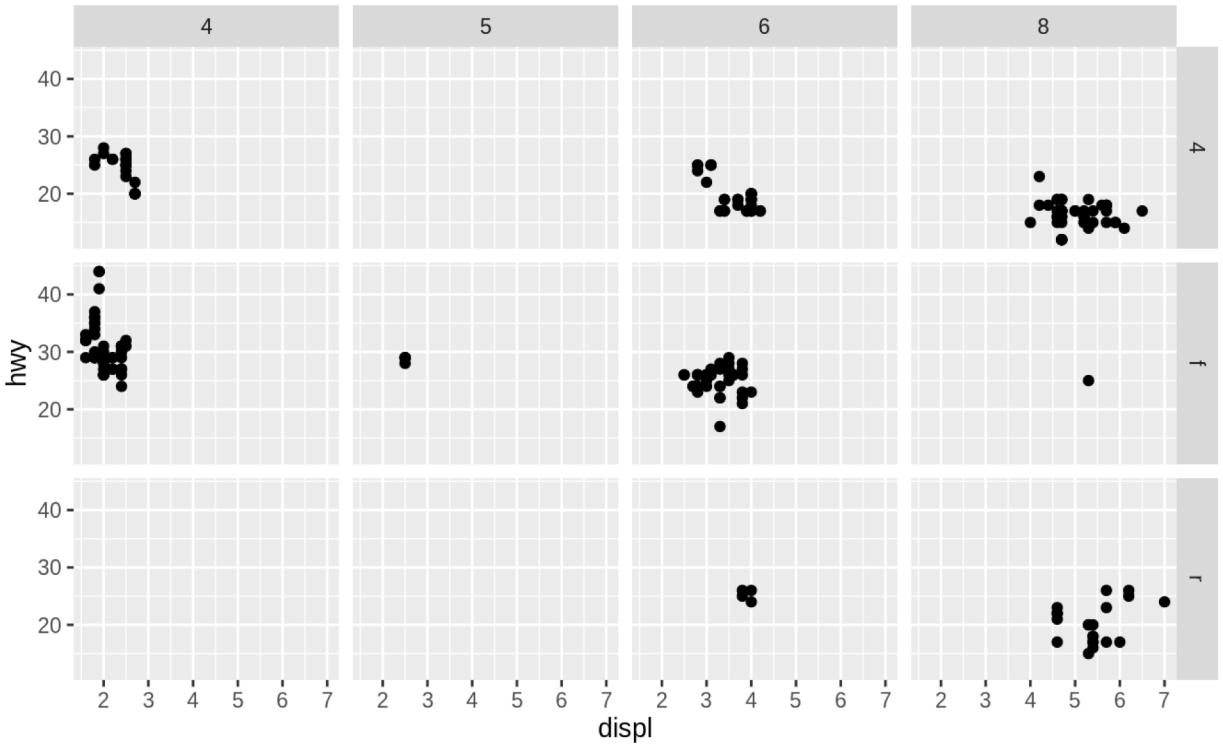
Facets

To facet plot on the combination of two variables, add facet_grid() to plot call.

The first argument of facet_grid() is also a formula. The formula should contain two variable names separated by a ~, with rows on the LHS and columns on the RHS:

```
ggplot(data = mpg) +

    geom_point(mapping = aes(x = displ, y = hwy)) +

    facet_grid(drv ~ cyl)
```

Facets



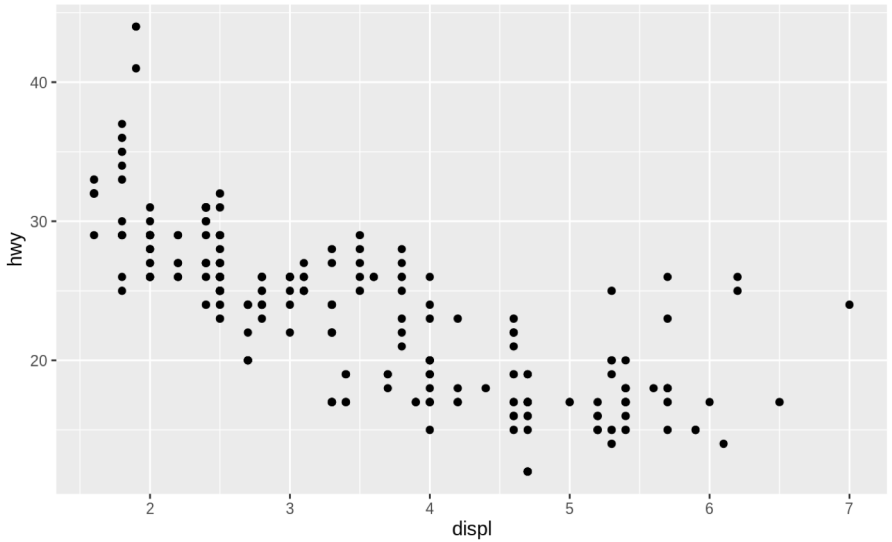
Facets

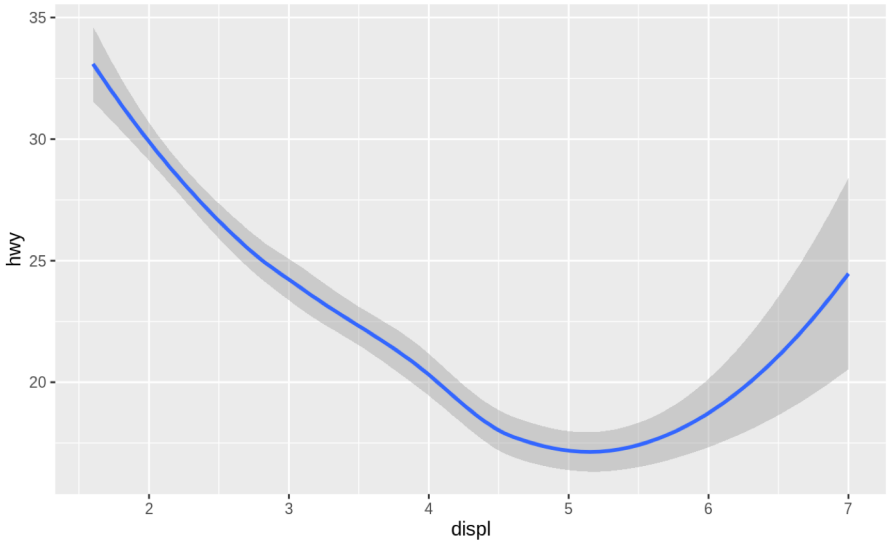
If you prefer to not facet in the rows or columns dimension, use a . instead of a variable name.

For example, if you want to not facet in the rows dimension:

```
ggplot(data = mpg) +

    geom_point(mapping = aes(x = displ, y = hwy)) +

    facet_grid(. ~ cyl)
```

Geometric objects





How are these two plots similar?

Geometric objects

Both plots contain same variables and describe the same data. However, each plot uses a different visual object to represent the data.

In ggplot2 syntax, we say that they use different **geoms**.

Bar charts use bar geoms, line charts use line geoms, boxplots use boxplot geoms, scatterplots use point geoms.

Geometric objects

To change the geom in your plot, change the geom function that you add to ggplot().

Back to the two plots, the left one uses the point geom, and the right one uses the smooth geom, a smooth line fitted to the data.

Geometric objects

To make these plots, you can use:

```
# left

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

# right

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

Geometric objects

Every geom function in ggplot2 takes a mapping argument. However, not every aesthetic works with every geom.

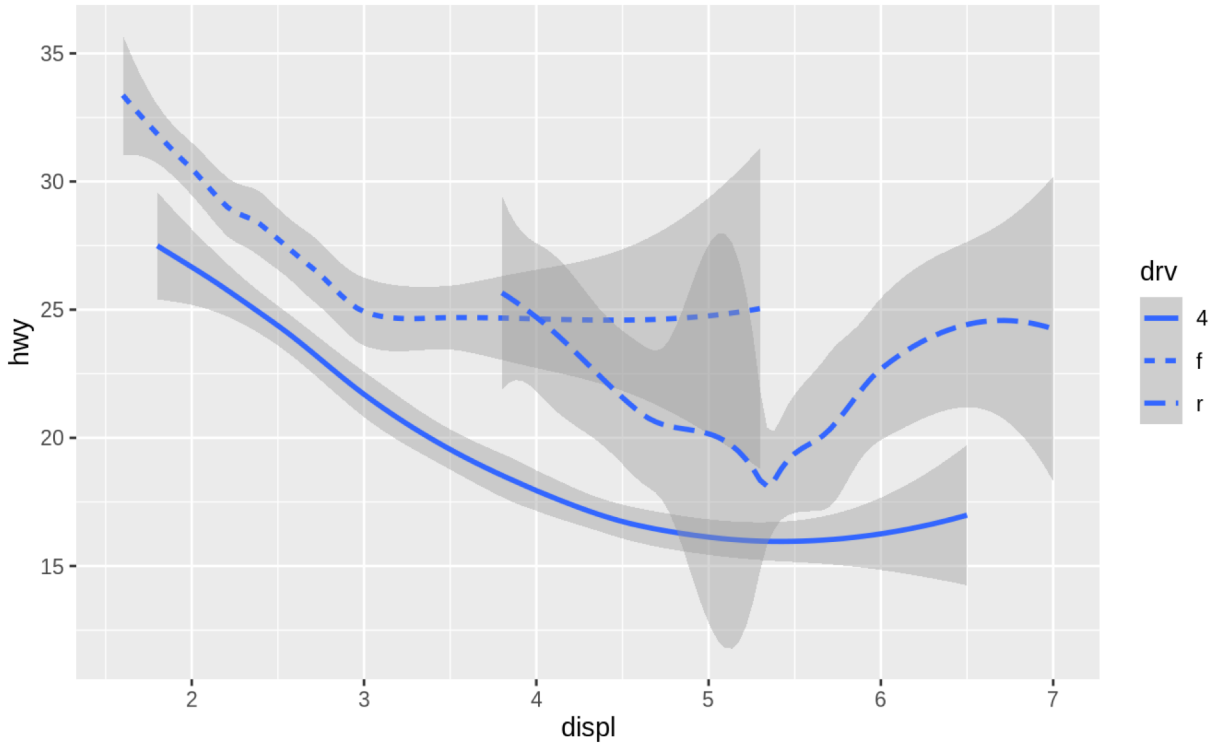
You could set the shape of a point, but you couldn't set the "shape" of a line.

On the other hand, you **could** set the linetype of a line.

For example, if you want to map `drv` to linetype:

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

Geometric objects



Geometric objects

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

```
ggplot(data = mpg) +

    geom_point(mapping = aes(x = displ, y = hwy)) +

    geom_smooth(mapping = aes(x = displ, y = hwy))
```

This, however, introduces some duplication in our code. Imagine if you wanted to change the y-axis to display `cty` instead of `hwy`. You'd need to change the variable in two places.

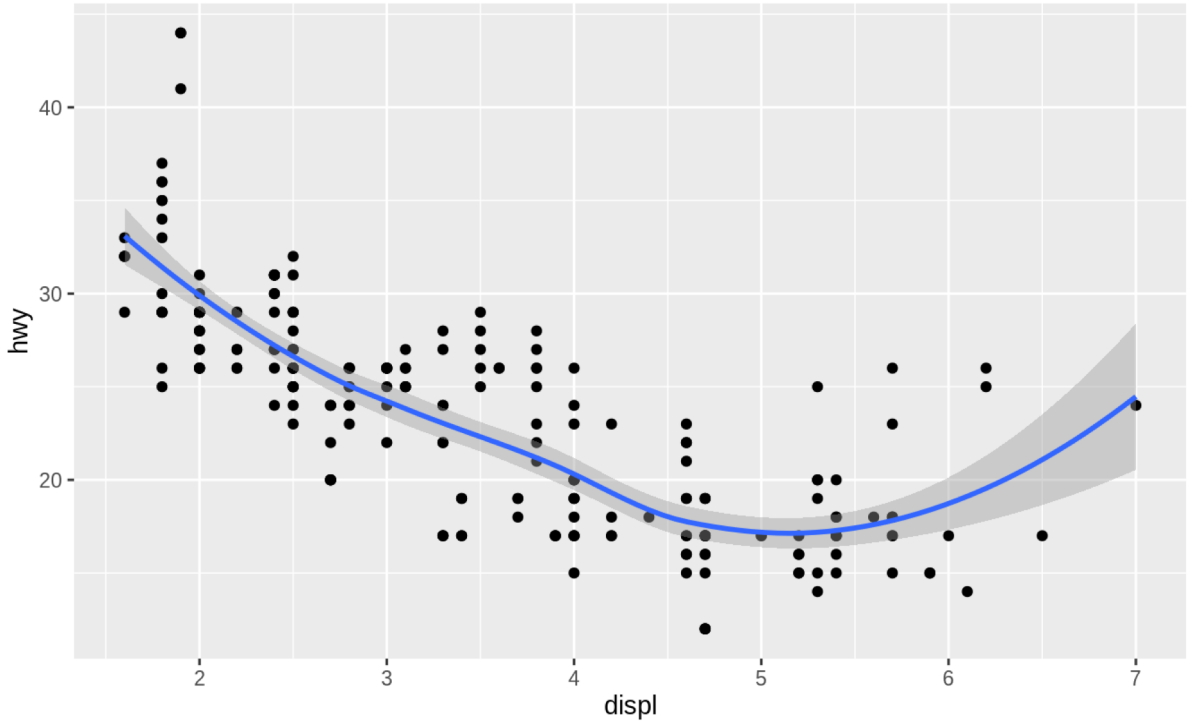
Geometric objects

You can avoid this type of repetition by passing a set of mappings to ggplot(). ggplot2 will treat these mappings as global mappings that apply to each geom in the graph.

The code below will produce the same plot as the previous code:

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

    geom_point() +

    geom_smooth()
```

Geometric objects



Geometric objects

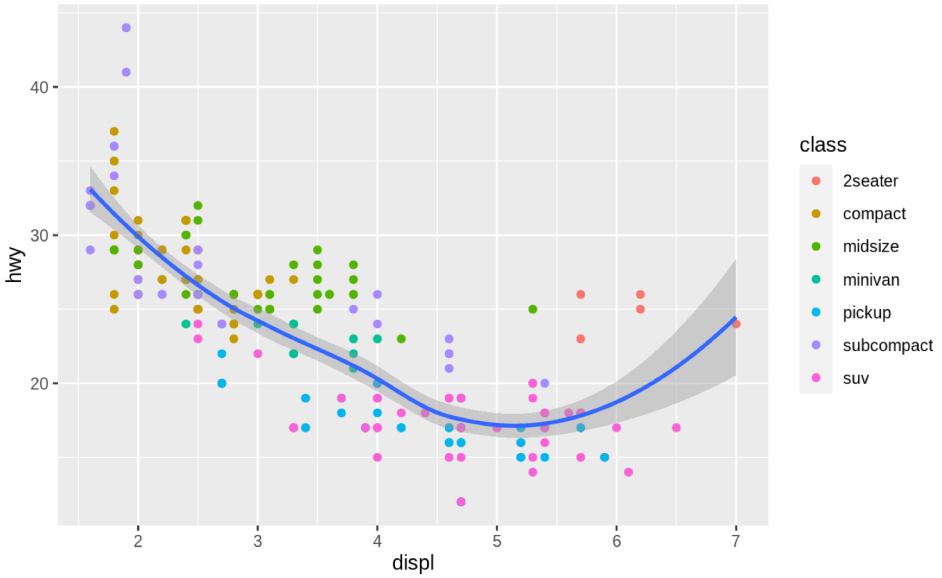
If you place mappings in a geom function, ggplot2 will treat them as local mappings for the layer. It will use these mappings to extend or overwrite the global mappings for that layer only.

This makes it possible to display different aesthetics in different layers.

Geometric objects

Example:

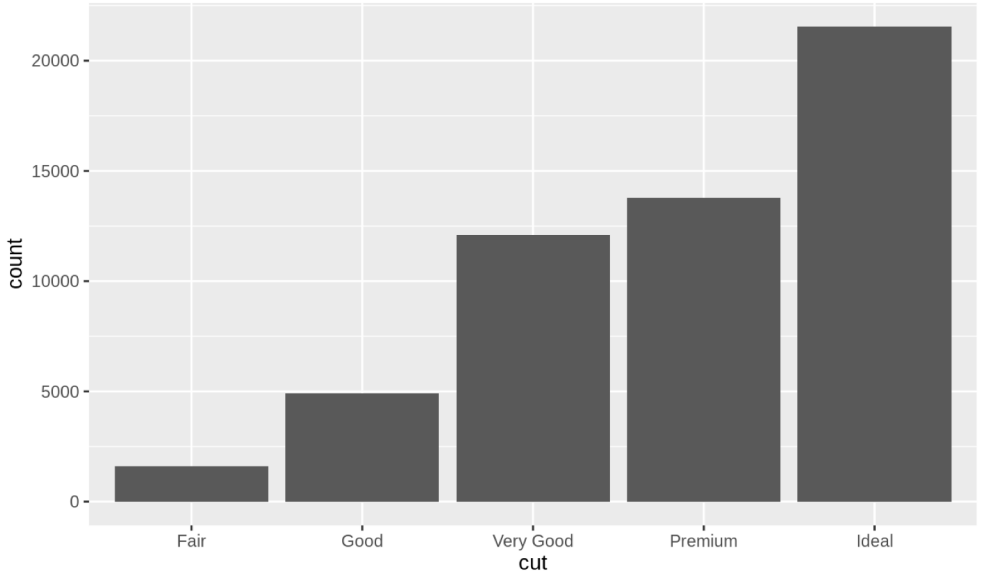
```
ggplot(data = mpg, mapping = aes(x =
displ, y = hwy)) +

geom_point(mapping = aes(color = class)
+

geom_smooth()
```



Part Ⅲ: Statistical transformation and position adjustments

Statistical transformation

Let’s look at a bar chart using the diamonds dataset founded in ggplot2.




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

Statistical transformation

On the y-axis, the chart 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 using **stat**, statistical transformation.

步页数

Every geom has a default stat. For example, for geom_bar(), the default value for stat is “count”.

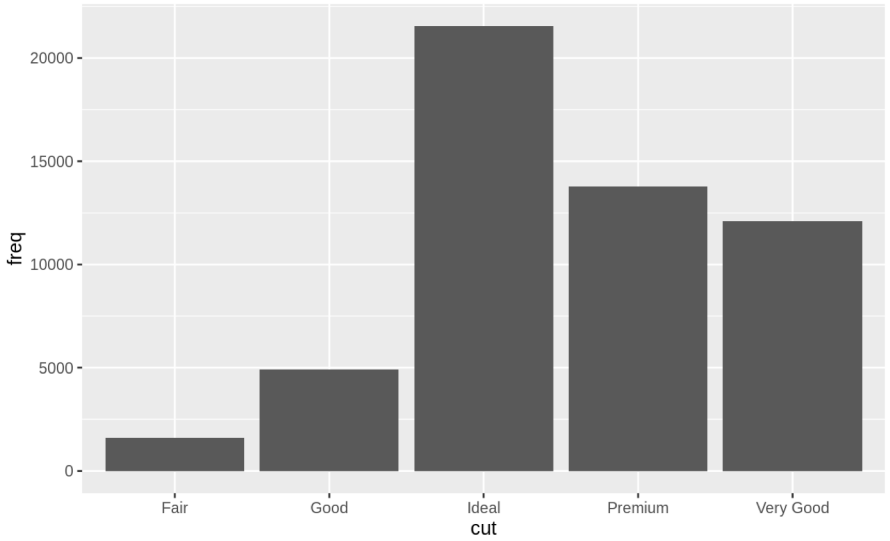
Statistical transformation

Typically, we can use geoms without worrying about statistical transformation. But there are three reasons we might need to use a stat explicitly:

1. We want to override the default stat. For example, we want to map the height of the bars to the raw values of a y variable. We can change the stat of geom_bar() from count (the default) to identity.

Statistical transformation

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

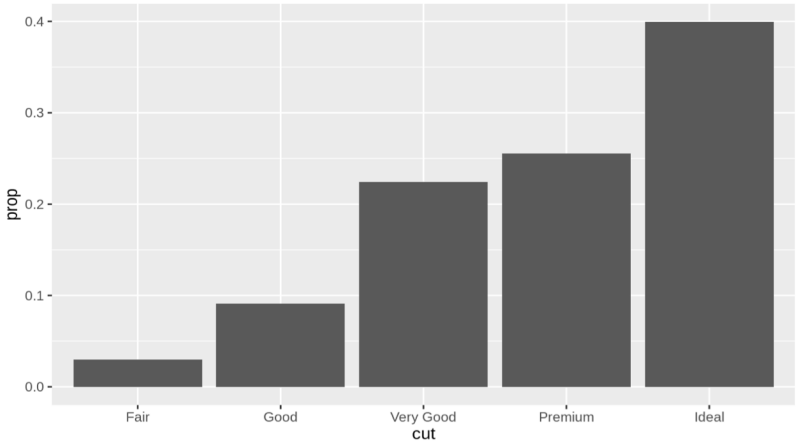



```
demo <-tribble(
~cut,          ~freq,
"Fair",        1610,
"Good",        4906,
"Very Good",   12082,
"Premium",     13791,
"Ideal",       21551)
```

Statistical transformation

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

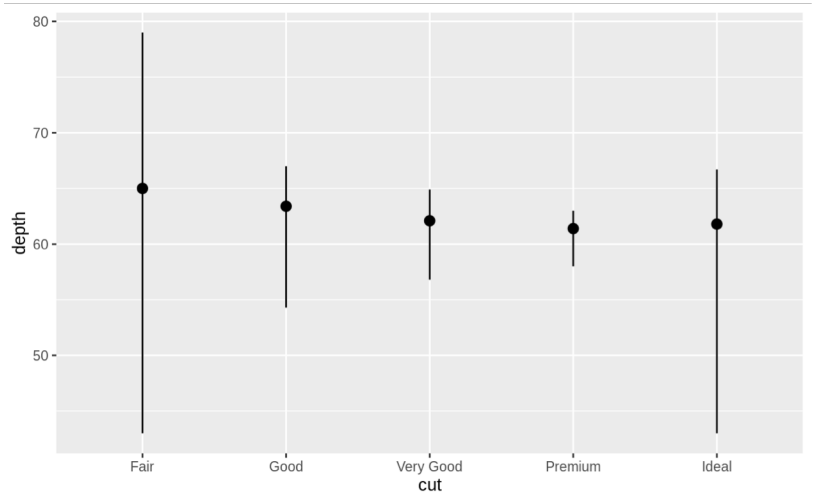
```
ggplot(data = diamonds) + geom_bar(mapping = aes(x = cut, y = stat(prop), group = 1))
```



Statistical transformation

3. We want to draw greater attention to the statistical transformation in the code. For example, we can use stat_summary(), which summarises the y values for each unique x value, to draw attention to the summary that we're computing:

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



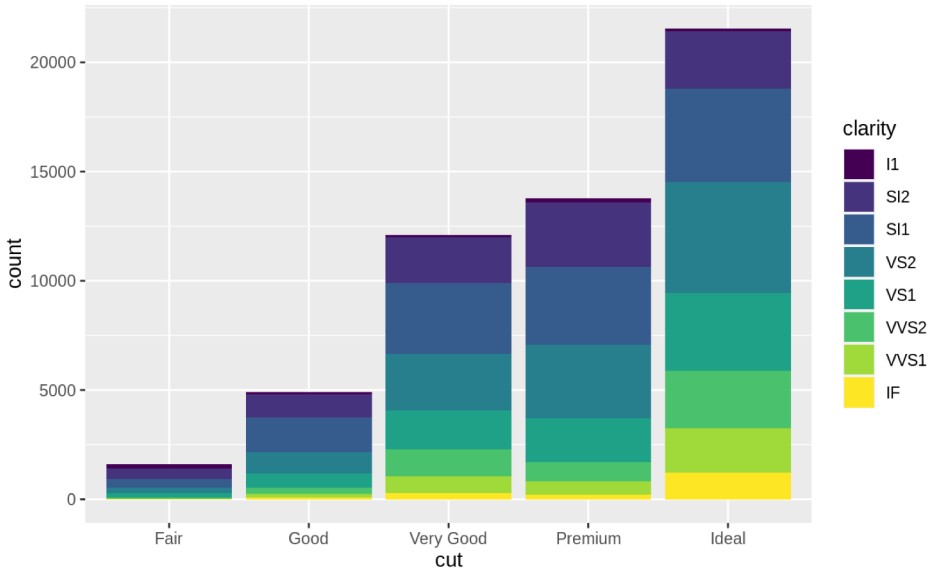
Position adjustments

We can color a bar chart using either color or fill aesthetics.

Using diamonds dataset, what will happen if we map the fill aesthetic to clarity?

Position adjustments

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



The bars are automatically stacked!

Position adjustments

The stacking is performed automatically by 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**".

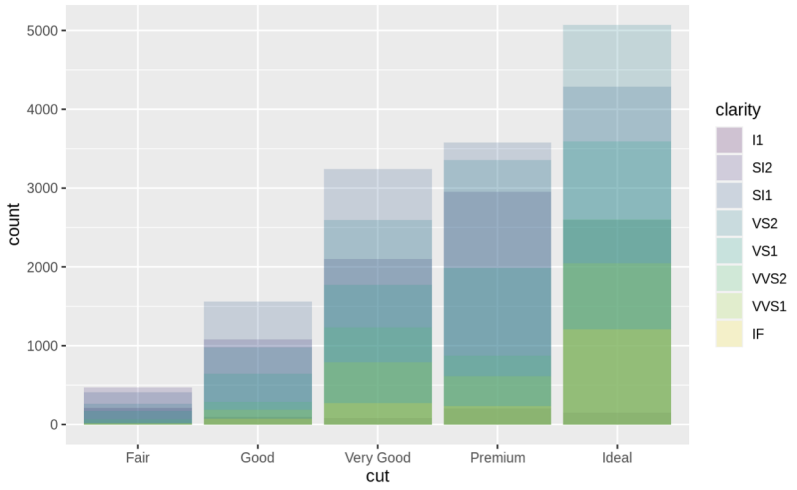
Position adjustments

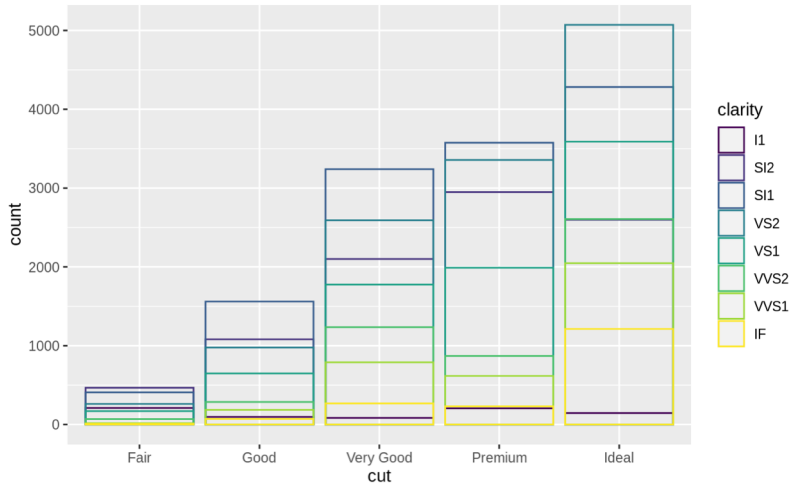
- position = "identity" will place each object exactly where it falls in the context of the graph.

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.

Position adjustments

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

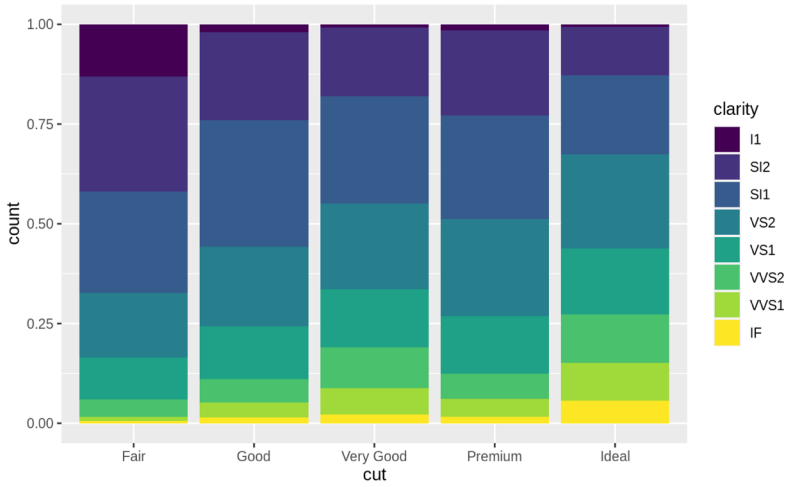




Position adjustments

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

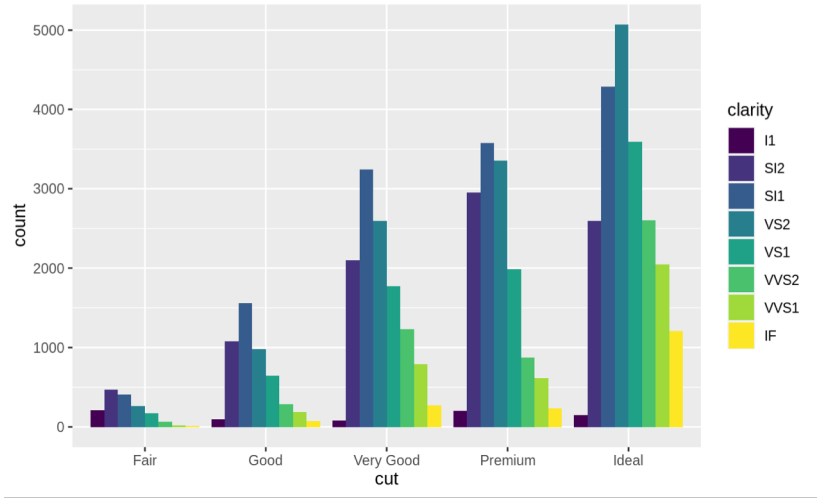
```
ggplot(data = diamonds) + geom_bar(mapping = aes(x = cut, fill = clarity), position = "fill")
```



Position adjustments

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

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



Position adjustments

There’s one other type of adjustment that is not useful for bar charts but can be very useful for scatterplots.

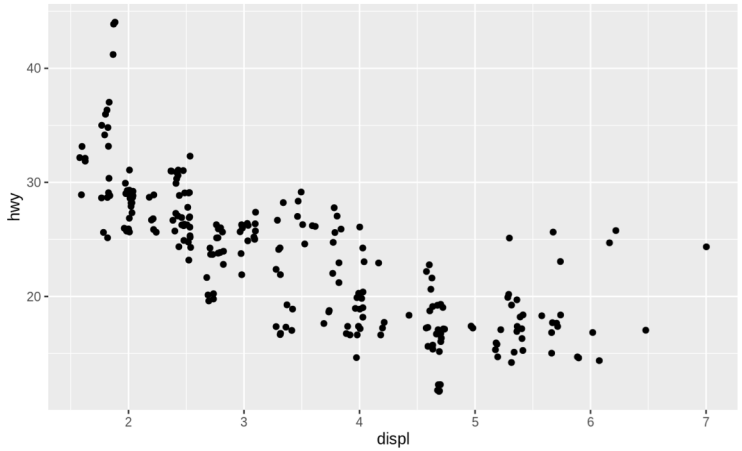
Sometimes, the values are so rounded so the points appear on a grid and many points overlap each other. This is **overplotting**.

You can avoid this gridding by setting the position adjustment to “**jitter**”.

Position adjustments

- position = "jitter" adds a small amount of random noise to each point. This spreads the points out because no two points are likely to receive the same amount of random noise.

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



Summary

Summary

From this lecture, we can summarize a code template which takes six parameters we've learned.

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

Summary

In practice, we rarely need to supply all parameters because ggplot2 will provide useful defaults for everything except the data, the mappings, and the geom function.